МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ
Факультет інформаційних технологій

С.В.БАРАН

ОСНОВИ WEB-ПРОГРАМУВАННЯ

НАВЧАЛЬНИЙ ПОСІБНИК

КРИВИЙ РІГ 2023

Рецензенти:
А.І. Купін, д.т.н., професор, завідувач кафедри комп'ютерних систем та мереж, Криворізький національний університет

І.О. Музика, к.т.н, доцент кафедри комп'ютерних систем та мереж, декан факультету інформаційних технологій, Криворізький національний університет

В.С. Лисенко, к.е.н., доцент кафедри інформатики та прикладного програмного забезпечення, Державний університет економіки і технологій

Рекомендовано до друку Вченою радою Державного університету економіки і технологій протокол № 10 від 30.03.2023р.

С.В.Баран. Основи web-програмування: Навчальний посібник. – Кривий Ріг: Державний університет економіки і технологій, 2023. –316 с.

Навчальний посібник містить практичні рекомендації для допомоги студентам у вивченні дисципліни " Основи web-програмування ".

Розглянуто базові питання й підходи до розробки web-додатків з використанням сучасних технологій. Розглянуто основи мови HTML, CSS та JavaScript для ефективної розробки динамічних web-сторінок.

ЗМІСТ

ВСТУП 5
РОЗДІЛ 1 СТРУКТУРА І ПРИНЦИПИ WEB 6
1.1. Сімейство протоколів TCP/IP 6
1.2. Мережа Internet. Послуги Інтернет 16
1.3. Служба WWW. Протокол HTTP 21
1.4. Еволюція Web 27
1.5. Сучасні технології web-програмування 28
Контрольні питання 38
РОЗДІЛ 2 ОРГАНІЗАЦІЯ WEB-ПРОЕКТІВ 39
2.1. Доменне ім'я 39
2.2. Вибір доменного імені 45
2.3. Хостинг 48
Контрольні питання 50
РОЗДІЛ 3 МОВА HTML 51
3.1. Поняття тегів і атрибутів 51
3.2. Базові теги розмітки гіпертексту 55
3.3. Списки та роздільники 60
3.4. Створення посилань 62
3.5. Графіка та мультимедіа 70
3.6. Таблиці 78
3.7. Структурування контенту. Теги <div> та <span> 86
3.8. Семантична розмітка в HTML5. Структура документу. 88
Контрольні питання 93
РОЗДІЛ 4 ФОРМИ ТА ЕЛЕМЕНТИ КЕРУВАННЯ HTML 94
4.1. Форми 94
4.2. Елементи керування "поля введення" 97
4.3. Поля введення в HTML5 100
4.4. Кнопочні елементи керування 104
4.5. Елемент керування "список" 108
Контрольні питання 110
РОЗДІЛ 5 КАСКАДНІ ТАБЛИЦІ СТИЛІВ (CSS) 112
5.1. Поняття CSS 112
5.2. Створення стилів. Селектори і властивості. 113
5.3. Способи задавання стилів 115
5.4. Принцип спадкування. Контекстні селектори 119
5.5. Задавання стилів за допомогою Класів 121
5.6. Задавання стилів за допомогою Ідентифікаторів 123
5.7. Спеціальні селектори 127
5.8. Правила каскадності та пріоритет стилів 138

5.9. Коментарі в CSS ... 141
5.10. Параметри шрифта ... 141
5.11. Параметри фону ... 149
5.12. Параметри абзаців, списків та відображення ... 151
5.13. Контейнери ... 155
5.14. Позиціонування елементів ... 159
5.15. Відступи, рамки та виділення ... 164
5.16. Параметри таблиць ... 169
5.17. CSS3. Градієнти ... 172
5.18. CSS3. Рамки та тіні ... 180
5.19. CSS3-переходи та фільтри ... 183
5.20. CSS3-трасформації ... 187
5.21. CSS3-анімація ... 190
5.22. CSS3 -медіазапити ... 196
5.23. CSS3 Flexbox ... 203
Контрольні питання ... 209
РОЗДІЛ 6 МОВА JAVASCRIPT ... 211
6.1. Створення сценаріїв на сторінці ... 211
6.2. Змінні. Типи даних. ... 212
6.3. Оператори. ... 220
6.4. Пріоритет операторів ... 222
6.5. Оператори порівняння ... 225
6.6. Перевірка умов ... 231
6.7. Цикли ... 234
6.8. Створення функцій ... 236
6.9. Область видимості змінної, замикання ... 244
Контрольні питання ... 249
РОЗДІЛ 7 РОБОТА З ОБ'ЄКТАМИ JAVASCRIPT ... 250
7.1. Об’єкти ... 250
7.2. Посилання на об’єкти. Копіювання об’єктів ... 258
7.3. Конструктори та оператор new ... 262
7.4. Symbol ... 265
7.5. Перетворення об’єктів в примітиви ... 267
7.6. Робота з рядками ... 270
7.7. Робота з масивами ... 272
7.8. Ітератори ... 282
7.9. Map та Set ... 284
7.10. Робота з датою та часом ... 289
7.11. Робота з функціями ... 292
7.12. Регулярні вирази ... 297
7.13. Обробка помилок ... 305
Контрольні питання ... 314
ВИСНОВКИ ... 315
СПИСОК ЛІТЕРАТУРИ ... 316

ВСТУП

Web-додаток - це прикладне програмне забезпечення, логіка якого розподілена між сервером та клієнтом, а обмін інформацією відбувається через мережу. Клієнтська частина реалізує інтерфейс користувача, а серверна - отримує і обробляє запити від клієнта, виконує обчислення, формує веб-сторінку і відправляє її клієнту згідно з протоколом HTTP.

Актуальність досліджень у галузі питань побудови web-додатків обумовлена тим, що даний вид програмного забезпечення:

- перспективний, як інструмент електронної комерції;
- надає широкі можливості соціальної взаємодії;
- у найближчому майбутньому може скласти реальну конкуренцію нативним додаткам мобільних операційних систем (Apple iOS, Google Android, Windows Phone).

Тому, метою дисципліни ознайомлення студентів з основами програмування на стороні клієнту та серверу з врахуванням особливостей браузерів та web-серверів.

Завдання дисципліни: вивчення принципів функціонування мережі Інтернет, основ адміністрування web-серверів, створення web-сайтів з використанням мов і технологій HTML, CSS, JavaScript.

Предмет дисципліни: використання сучасних технологій Інтернет-програмування для створення web-сайтів.

Організація вивчення дисципліни передбачає ознайомлення зі створенням web-додатків на основі використання сучасних засобів Інтернет-програмування та виконання на базах практики чи в комп’ютерному центрі практичних завдань. Практичні завдання орієнтовано на використання технологій та мов web-програмування: JavaScript, HTML, CSS.

Студент повинен знати:

– інструментарій Інтернет-програмування;
– встановлення, конфігурування та адміністрування web-серверів;
– специфічні особливості браузерів.

Студент повинен уміти:

– аналізувати та вибирати інструментальні засоби Інтернет-програмування;
– встановлювати необхідне програмне забезпечення для Інтернет-програмування;
– розробляти інтерактивні WEB-сторінки.

РОЗДІЛ 1
СТРУКТУРА І ПРИНЦИПИ WEB

1.1. Сімейство протоколів TCP/IP

Розрізняють логічну і фізичну моделі Інтернету. Під логічної, насамперед, розуміють Всесвітню павутину (World Wide Web), а під фізичною - комп'ютери, сервери і засоби передачі даних між ними. Програмне забезпечення складається із загального (системного) і прикладного (спеціального) програмного забезпечення. У зв’язку з тим, що безліч мереж ЕОМ мали (кожна окремо) оригінальні апаратні і програмні засоби, що не сполучаються один з одним, як в апаратному, так і в програмному відношенні, виникла необхідність у вирішенні глобальної проблеми – об’єднання окремих локальних різномовних мереж в єдину загальнодоступну мережу.

Для вирішення цієї проблеми були розроблені програмні засоби так звані протоколи обміну.

Протокол *— домовленості про сигнали, якими обмінюються комп’ютери під час встановлення зв’язку між собою і приймання чи передавання інформації.*

Найбільш популярним пакетом програмних засобів є протокол TCP/IP. Він є надзвичайно гнучким протоколом, що дозволяє пересилати файли і відправляти електронну пошту. що підтримує велику кількість мережевих інтерфейсів: Ethernet, ARCNET, FDDI, а також можливість цифрової передачі даних. Невід’ємною частиною TCP/IP є також протоколи динамічної маршрутизації, що враховують розподіл навантаження, визначення найкоротшого шляху і інші функції маршрутизації.

Сімейство протоколів TCP / IP працює на будь-яких моделях комп'ютерів, вироблених різними виробниками комп'ютерної техніки та працюють під управлінням різних операційних систем. Сімейства протоколів, такі як TCP/IP, це комбінації різних протоколів на різних рівнях. TCP/IP складається з рівнів, наведених у табл. 1.1.

Таблиця 1.1

Рівні TCP/IP

Рівень	**Назва рівня**	**Протоколи**
7	Прикладний	HTTP, SMTP, SNMP, FTP, Telnet, scp, SMB,NFS, RTSP, BGP
6	Представницький	XDR, ASN.1, AFP
5	Сеансовий	TLS, SSL, ISO 8327 / CCITT X.225, RPC, NetBIOS, ASP
4	Транспортний	TCP, UDP, RTP, SCTP, SPX, ATP, DCCP, GRE
3	Мережевий	IP, ICMP, IGMP, CLNP, OSPF, RIP, IPX, DDP
2	Канальний	Ethernet, Token ring, PPP, HDLC, X.25, Frame relay, ISDN, ATM, MPLS, Wi-Fi, ARP, RARP
1	Фізичний	електричні, оптоволоконні, радіозв'язок

Кожен рівень несе власне функціональне навантаження. Канальний рівень (link layer), ще його називають рівнем мережевого інтерфейсу зазвичай включає драйвер пристрою в операційній системі і відповідну мережеву інтерфейсну плату в комп'ютері. Разом вони забезпечують апаратну підтримку фізичного з'єднання з мережею.

Мережевий рівень (network layer), іноді званий рівнем міжмережевої взаємодії, відповідає за передачу пакетів по мережі. Маршрутизація пакетів здійснюється саме на цьому рівні. IP (Internet Protocol - протокол Internet), ICMP (Internet Control Message Protocol - протокол управління повідомленнями Internet) і IGMP (Internet Group Management Protocol - протокол управління групами Internet) забезпечують мережевий рівень в сімействі протоколів TCP/IP.

Транспортний рівень (transport layer) відповідає за передачу потоку даних між двома комп'ютерами і забезпечує роботу прикладного рівня, який знаходиться вище. У сімействі протоколів TCP/IP існує два транспортних протоколу: TCP (Transmission Control Protocol) і UDP (User Datagram Protocol). TCP здійснює надійну передачу даних між двома комп'ютерами. Він забезпечує поділ даних, що передаються від одного додатка до іншого, на пакети підходящого для мережевого рівня розміру, підтвердження прийнятих пакетів, установку тайм-аутів, протягом яких має прийти підтвердження на пакет, і так далі. Так як надійність передачі даних гарантується на транспортному рівні, на прикладному рівні ці деталі ігноруються. UDP надає більш простий сервіс для прикладного рівня. Він просто відсилає пакети, які називаються датаграми (datagram) від одного комп'ютера до іншого. При цьому немає ніякої гарантії, що датаграма дійде до пункту призначення. За надійність передачі даних, при використанні датаграмм відповідає прикладний рівень. Для кожного транспортного протоколу існують різні додатки, які їх використовують.

Мережевий і транспортні рівні — ядро сімейства TCP/IP, де основний протокол IP (Internet- Рrotoco1). IP надає адресний простір для міжмережевих взаємодій і керує маршрутизацією пакетів даних по мережах. **ARP** (Adress Resolution Protocol) — ще один протокол мережевого рівня, він допомагає мережевим пристроям визначати IP-адреси.

Для досягнення максимально можливого розміру пакету і налаштування параметрів передачі, окрім IP використовуються протоколи **TCP** (transmission control protokol) і **UDP** (user datagram protokol). TCP застосовується, коли потрібна стовідсоткова надійність передачі, а UDP — при менш жорстких вимогах.

Прикладний рівень (application layer) визначає деталі кожного конкретного додатка.

До найбільш важливих прикладних протоколів відносяться FTP (File Transfer Protokol), HTTP (Hypertext Transfer Protokol) для World Wide Web і SNMP (Simple Network Managment Protocol) для управління мережевими пристроями. Слід також відзначити службу DSN

(Domain Naming Service), що відповідає за перетворення числових IP-адресів в імена, які значно легше запам’ятати користувачам. На прикладному рівні діють і інші протоколи, регулюючі окремі аспекти роботи додатків. У їх числі протоколи для електронної пошти: SMTP (Simple Mail Transport Protocol), POP (Post Ofise Protocol), IMAP (Internet Mail Access Protocol) і MIME (Multimedia Internet Mail Extensions).

В Інтернеті для обміну файлами між файл-сервером і комп’ютером-клієнтом (в цій ролі може виступати Ваш власний персональний комп’ютер), використовується протокол FTP.

Поза сумнівом, зараз найпоширенішим інтерфейсом для Інтернету є Web, заснований на стандартній **мові розмітки гіпертексту HTML** (Hypeгtext Maгkup Language) і **протоколі передачі гіпертексту HTTP** (Hypeгtext Tгansfeг Pгotoco1). Браузер, встановлений на комп’ютері користувача, використовує HTML для того, щоб вирішити, в якому вигляді виводити на екран текст і графіку. HTTP, у свою чергу, визначає, як переслати файл (наприклад, документ HTML) від сервера клієнтові.

Електронна пошта - засіб комунікації в Інтернеті. Головними протоколами електронної пошти в Інтернеті є **SMTP** (Simple Mail Transport Protocol) і **POP** (Post Ofise Protocol). SMTP використовується для обміну поштою між серверами, а POP і новіший протокол **IMAP** (Internet Mail Access Protocol) — для обробки повідомлень.

Для організації зв'язку між двома комп'ютерами в глобальній мережі, кожний з них повинен мати унікальне позначення, яким є 32-бітна IP-адреса інтерфейсу. Звичайно така адреса записується як чотири числа, розділені «.», наприклад, 140.252.13.33. IP-адреса складається з двох компонентів: адреси мережі та адреси вузла (хоста). Визначити клас адреси, або клас мережі, можна по першому числу в адресі (табл. 1.2).

Таблиця 1.2

Типи мереж

Клас	**Перші біти**	**Діапазон**	**Распределение байт (С - мережа, Х- хост)**	**Число можливих адрес мережі**	**Число можливих адрес хостів**	**Маска підмережі**
A	0	**0**.0.0.0 - **127**.255.255.255	C.X.X.X	128	16 777 216	255.0.0.0
B	10	**128**.0.0.0 - **191**.255.255.255	C.C.X.X	16 384	65 536	255.255.0.0
C	110	**192**.0.0.0 - **223**.255.255.255	C.C.C.X	2 097 154	256	255.255.255.0
D	1110	**224**.0.0.0 - **239**.255.255.255	Групова адреса			
E	1111	**240**.0.0.0 - **247**.255.255.255	Зарезервовано			

Адреси з номером мережі 127 зарезервовані для тестової перевірки наявності зв’язку з собою (loop back) та перевірки функціонування міжпроцесорних зв’язків. Адреси мереж з номерами 224 і вище призначені для спеціальних протоколів і їх не можна використовувати. Позначення 192.168.X.X зарезервоване для локальних мереж. Значення 255 використовується в масках мереж. Маска мережі - це узагальнене представлення адреси вузла у мережі.

Так як кожен інтерфейс, підключений до мережі, повинен мати унікальну адресу, постає питання розподілу IP адрес в глобальній мережі Internet. Цим займається мережевий інформаційний центр (Internet Network Information Center або InterNIC). InterNIC призначає тільки мережеві ідентифікатори (ID). Призначенням ідентифікаторів хостів в мережі займаються системні адміністратори.

Існує три типи IP адрес: персональний адресу (unicast) - вказує на один хост, широкомовний адресу (broadcast) - вказує на всі хости у зазначеній мережі, і груповий адресу (multicast) - вказує на групу хостів, що належить до групи адресації.

Незважаючи на те що кожен мережевий інтерфейс комп'ютера має свою власну IP адресу, користувачі звикли працювати з іменами хостів. Існує розподілена світова база даних TCP/IP, звана системою імен доменів (DNS - Domain Name System), яка дозволяє встановити відповідність між IP адресами та іменами хостів. DNS - це комбінація з імен доменів (підмереж) на шляху до комп'ютера, розділених крапками. Наприклад адреса ippz.knu.dp.ua - адреса кафедри інформатики та прикладного програмного забезпечення (ippz) Криворізького національного університету (knu), який знаходиться в Дніпропетровській області (dp), в Україні (ua). За таким принципом побудовано більшість DNS-адрес. Для переведення DNS-адрес в IP-адреси призначені DNS-сервери (тут вже Domain Name Server).

Стандартний розмір IP заголовку у протоколі TCP/IP складає 20 байт, якщо не присутні опції.

<table>
<tr><td colspan="3">0 15</td><td colspan="2">16 31</td></tr>
<tr><td>4-біти версія</td><td>4-біти довжина заголовку</td><td>8-біт тип сервісу (TOS)</td><td colspan="2">16-біт повна довжина (в байтах)</td></tr>
<tr><td colspan="3">16-біт ідентифікація</td><td>3-біта флаги</td><td>13-біт зміщення заголовку</td></tr>
<tr><td colspan="2">8-біт час життя (TTL)</td><td>8-біт протокол</td><td colspan="2">16-біт контрольна сума заголовку</td></tr>
<tr><td colspan="5">32-біти IP-адреса джерела</td></tr>
<tr><td colspan="5">32-біти IP-адреса призначення</td></tr>
<tr><td colspan="5">опції (якщо є)</td></tr>
<tr><td colspan="5">дані</td></tr>
</table>

Рис. 1.1. IP датаграмма, поля IP заголовку.

Актуальні дві версії протоколу – 4 (IPv4) та 6 (IPv6). Довжина заголовка (header length) це кількість 32-бітних слів у заголовку, включаючи будь опції. Так як це 4-бітове поле, воно обмежує розмір заголовка в 60 байт.

4 біта TOS наступні: мінімальна затримка, максимальна пропускна здатність, максимальна надійність і мінімальна вартість. Тільки один з цих 4 біт може бути встановлений в одиницю одночасно.

Поле повної довжини (total length) містить повну довжину IP-датаграми в байтах. Завдяки цьому полю і полю довжини заголовку, ми знаємо, з якого місця починаються дані в IP-датаграмі і їх довжину. Так як це поле складається з 16 біт, максимальний розмір IP-датаграми складає 65535 байт. Незважаючи на те що існує можливість відправити датаграму розміром 65535 байт, більшість канальних рівнів поділять подібну датаграму на фрагменти. Більш того, від хоста не потрібно приймати датаграму розміром більше ніж 576 байт. TCP поділяє користувацькі дані на частині, тому це обмеження звичайно не робить впливу на TCP.

Поле ідентифікації (identification) унікально ідентифікує кожну датаграму, відправлену хостом. Значення, що зберігається в полі, зазвичай збільшується на одиницю з посилкою кожної датаграми.

Поле часу життя (TTL - time-to-live) містить максимальну кількість пересилань (маршутізаторов), через які може пройти датаграма. Це поле обмежує час життя датаграми. Значення встановлюється відправником (як правило 32 або 64) і зменшується на одиницю кожним маршрутизатором, який обробляє датаграму. Коли значення в полі досягає 0, датаграма видаляється, а відправник повідомляється про це за допомогою ICMP повідомлення. Подібний алгоритм запобігає зацикленню пакетів в петлях маршрутизації.

Контрольна сума заголовка (header checksum) розраховується тільки для IP заголовку. Вона не включає в себе дані, які слідують за заголовком. ICMP, IGMP, UDP і TCP мають контрольні суми у своїх власних заголовках, які охоплюють їх заголовки і дані.

Щоб розрахувати контрольну суму IP для вихідної датаграми, поле контрольної суми спочатку встановлюється в 0. Потім розраховується 16-бітна сума з порозрядним доповненням (One's complement, Заголовок цілком сприймається як послідовність 16-бітних слів). 16-бітове порозрядне доповнення цієї суми зберігається в полі контрольної суми. Коли IP датаграма приймається, обчислюється 16-бітна сума з порозрядним доповненням. Так як контрольна сума, розрахована приймачем, містить в собі контрольну суму, збережену відправником, контрольна сума приймача складається з бітів рівних 1, якщо в заголовку нічого не було змінено при передачі. Якщо в результаті не вийшло всі одиничні біти (помилка контрольної суми), IP відкидає прийняту датаграму. Повідомлення про помилку не генерується. Тепер завдання верхніх рівнів яким-небудь чином визначити, що датаграма відсутня, і забезпечити

повторну передачу. ICMP, IGMP, UDP і TCP використовують такий же алгоритм розрахунку контрольної суми.

Поле опцій (options) – це список додаткової інформації змінної довжини. В даний час опції можуть використовуватися наступним чином:

- для безпеки та обробки обмежень;
- запис маршруту (запис кожного маршруту і його IP-адреси);
- тимчасова марка (запис кожного маршруту, його IP-адреси і часу);
- вільна маршрутизація від джерела (указує список IP адрес, через які має пройти датаграма);
- жорстка маршрутизація від джерела (те ж саме, що і в попередньому пункті, однак IP датаграма повинна пройти тільки через зазначені в списку адреси).

Опції рідко використовуються і не всі хости або маршрутизатори підтримують всі опції.

Поле опцій завжди обмежено 32 бітами. Байти заповнення, значення яких дорівнює 0, додаються по необхідності. Завдяки цьому IP заголовок завжди кратний 32 бітам (як це і потрібно для поля довжини заголовку).

IP маршрутизація це досить простий процес, особливо з точки зору хоста. Якщо пункт призначення безпосередньо підключений до хоста (наприклад канал точка-точка) або хост включений між декількома мережами (Ethernet або Token ring), IP датаграма спрямовується безпосередньо в пункт призначення, інакше хост посилає датаграму на маршрутизатор за замовчуванням, тим самим надаючи маршрутизатору вирішувати як доставити датаграму в пункт призначення. Основна і фундаментальна різниця між хостом і маршрутизатором полягає в тому, що хост ніколи не перенаправляє датаграми з одного свого інтерфейсу на інший, тоді як маршрутизатор перенаправляє.

IP рівень має в пам'яті таблицю маршрутизації, яку він переглядає кожен раз при отриманні датаграми, яку необхідно перенаправити. Коли датаграма прийнята з мережевого інтерфейсу, IP, по-перше, перевіряє, чи не належить йому вказана IP адреса призначення або не є ця IP адреса широкомовною. Якщо це так, то датаграма доставляється в модуль протоколу, зазначений у полі протоколу в IP заголовку. Якщо датаграма не призначена для цього IP рівня, якщо IP рівень був налаштований для того щоб працювати як маршрутизатор, пакет перенаправляється, інакше датаграма мовчки знищується.

Кожен пункт таблиці маршрутизації містить таку інформацію:

1. IP адреса призначення. Це може бути як повна адреса хоста (host address) або адреса мережі (network address). Адреса хоста має нульове значення ідентифікатора хоста і вказує на один конкретний хост, тоді як адреса мережі має ідентифікатор хоста, встановлений в 0, і вказує на всі хости, включені в певну мережу (Ethernet, Token ring).

2. IP адреса маршрутизатора наступній пересилання (next-hop router), або, інакше кажучи, IP адресу безпосередньо підключеної мережі. Маршрутизатор наступного пересилання належить одній з безпосередньо підключених мереж, в яку ми можемо відправити датаграми для їх доставки. Маршрутизатор наступного пересилання це не кінцевий пункт призначення, проте він приймає датаграми, які ми надсилаємо, і перенаправляє їх у напрямку кінцевого пункту.

3. Прапори. Один прапор вказує, чи є IP адреса пункту призначення, адресою мережі або адресою хоста. Інший прапор вказує на те, чи є маршрутизатор наступного пересилання дійсно маршрутизатором або це безпосередньо підключений інтерфейс.

4. Вказівка на те, на якій мережевий інтерфейс мають бути передані датаграми для передачі.

IP маршрутизація здійснюється за принципом пересилання-за-пересиланням. Як ми можемо побачити з таблиці маршрутизації, IP не знає повний маршрут до пункту призначення (за винятком тих пунктів призначення, які безпосередньо підключені до хосту). Все що може надати IP маршрутизація - це IP адреса маршрутизатора наступного пересилання, на який посилається датаграма. При цьому робиться припущення, що маршрутизатор наступного пересилання ближче до пункту призначення, ніж посилає хост. Також робиться припущення, що маршрутизатор наступного пересилання безпосередньо підключений до хосту.

IP маршрутизація здійснює такі дії:

1. Здійснюється пошук в таблиці маршрутизації, при цьому шукається пункт, який співпаде з повною адресою пункту призначення (має співпасти ідентифікатор мережі та ідентифікатор хоста). Якщо пункт знайдений в таблиці маршрутизації, пакет посилається на вказаний маршрутизатор наступного пересилання чи на безпосередньо підключений інтерфейс (залежно від поля прапорів).

2. Здійснюється пошук в таблиці маршрутизації пункту, який співпаде, як мінімум, з ідентифікатором мережі призначення. Якщо пункт знайдений, пакет посилається на вказаний маршрутизатор наступного пересилання чи на безпосередньо підключений інтерфейс (залежно від поля прапорів).

3. У таблиці маршрутизації шукається пункт, позначений "за замовчуванням" (default). Якщо пункт знайдений, пакет відсилається на вказаний маршрутизатор за замовчуванням.

Якщо жоден із кроків не дав позитивного результату, датаграма вважається недоставленою. Якщо така датаграма була згенерована даними хостом, то зазвичай повертається помилка "хост недоступний" (host unreachable) або "мережа недоступна" (network unreachable). Цей код помилки повертається додатком, яке згенерувало датаграму.

На початку завжди здійснюється порівняння на збіг повної адреси хоста, після чого

здійснюється порівняння ідентифікатора мережі. Тільки в тому випадку, якщо результат обох порівнянь негативний, використовується маршрут за замовчуванням.

Ще одна фундаментальна характеристика IP маршрутизації полягає в можливості вказати маршрут до мережі, замість того, щоб вказувати маршрут до кожного окремо взятого хоста. Саме тому хости включені в Internet, наприклад, мають у своїх таблицях маршрутизації тисячі пунктів, замість того щоб утримувати в них не більше ніж мільйон пунктів.

Незважаючи на те, що TCP і UDP використовують один і той же мережевий рівень (IP), TCP надає додаткам абсолютно інші сервіси, ніж UDP. TCP надає заснований на з'єднанні надійний сервіс потоку байтів.

Термін "заснований на з'єднанні" (connection-oriented) означає, що два додатки, що використовують TCP (як правило, це клієнт і сервер), мають встановити TCP з'єднання один з одним, після чого у них з'являється можливість обмінюватися даними.

TCP забезпечує свою надійність завдяки наступному:

1. Дані від програми розбиваються на блоки певного розміру, які будуть відправлені. Блок інформації, який передається від TCP в IP, називається сегментом (segment).

2. Коли TCP посилає сегмент, він встановлює таймер, очікуючи, що з віддаленого кінця прийде підтвердження на цей сегмент. Якщо підтвердження не отримано після закінчення часу, сегмент передається повторно.

3. Коли TCP приймає дані від віддаленої сторони з'єднання, він відправляє підтвердження. Це підтвердження не надсилається негайно, а зазвичай затримується на частки секунди.

4. TCP здійснює розрахунок контрольної суми для свого заголовку і даних. Це контрольна сума, що розраховується на кінцях з'єднання, метою якої є виявити будь-які зміни даних у процесі передачі. Якщо сегмент прибуває з невірною контрольною сумою, TCP відкидає його і підтвердження не генерується.

5. Так як TCP сегменти передаються у вигляді IP датаграмм, а IP датаграми можуть прибувати безладно, також безладно можуть прибувати і TCP сегменти. Після отримання даних TCP може по необхідності змінити їх послідовність, в результаті додаток отримує дані в правильному порядку.

6. Так як IP датаграма може бути продубльована, TCP повинен відкидати продубльовані дані.

7. TCP здійснює контроль потоку даних. Кожна сторона TCP з'єднання має певний простір буфера. TCP на приймаючій стороні дозволяє віддаленій стороні посилати дані тільки в тому випадку, якщо одержувач може помістити їх в буфер. Це запобігає переповненню буферів повільних хостів швидкими хостами.

Між двома додатками по TCP з'єднанню здійснюється обмін потоком 8-бітових байтів. Автоматично TCP не вставляє записи маркерів. Це називається сервісом потоку байтів (byte stream service). Якщо додаток на одному кінці записав спочатку 10 байт, потім 20 байт і потім ще 50 байт, додаток на іншому кінці з'єднання не може сказати якого розміру був кожен запис. На іншому кінці ці 80 байт можуть бути зчитані, наприклад, за 4 рази по 20 байт за кожен раз. Один кінець з'єднання поміщає потік байтів в TCP, і точно так само ідентичний потік байт з'являється на іншому кінці.

TCP не інтерпретує вміст байтів. TCP поняття не має про те, чи відбувається обмін двійковими даними, ASCII символами, EBCDIC символами або чим-небудь ще. Ця інтерпретація потоку байтів здійснюється додатками на кожній стороні з'єднання.

Дані TCP інкапсулюються в IP датаграми (рис. 1.2).

Кожен TCP сегмент містить номер порту (port number) джерела і призначення, за допомогою яких ідентифікуються програми відправники і одержувачі. Ці два значення разом з IP адресою джерела і призначення в IP заголовку унікально ідентифікують кожне з'єднання. Комбінація IP адреси і номера порту іноді називається сокетом (socket).

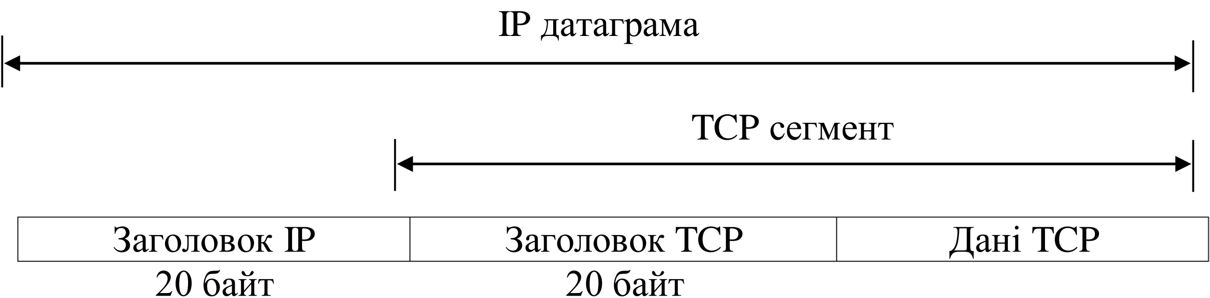


Рис. 1.2. Інкапсуляція TCP даних в IP-датаграму.

Структура заголовку наведена на рис. 1.3.

<table>
<tr><td colspan="8">0 15</td><td>16 31</td></tr>
<tr><td colspan="8">16-біт номер порта джерела</td><td>16-біт номер порта призначення</td></tr>
<tr><td colspan="9">32-біти номер послідовності</td></tr>
<tr><td colspan="9">32-біти номер підтвердження</td></tr>
<tr><td>4-біти довжина заголовку</td><td>6-біт зарезер-вовано</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>16-біт розмір вікна</td></tr>
<tr><td colspan="8">16-біт контрольна сума TCP</td><td>16-біт вказівка терміновості</td></tr>
<tr><td colspan="9">опції (якщо є)</td></tr>
<tr><td colspan="9">дані (якщо є)</td></tr>
</table>

Рис. 1.3. TCP заголовок

Номер послідовності (sequence number) ідентифікує байт в потоці даних від TCP-відправника до TCP-одержувача. Якщо ми представимо потік байтів в одному напрямку між двома додатками, TCP нумерує кожен байт номером послідовності. Номер послідовності являє собою 32-бітове беззнакове число, яке переходить через 0, по досягненню значення 2^{32} - 1.

За встановлення нового з'єднання відповідає «SYN». Поле номера послідовності (sequence number field) містить вихідний номер послідовності (ISN - initial sequence number), який вибирається хостом для даного з'єднання. Номер послідовності першого байта даних, який надсилається цим хостом, дорівнюватиме ISN плюс один, тому що прапор SYN - це номер послідовності.

Так як кожен байт, який бере участь в обміні, пронумерований, номер підтвердження (acknowledgment number) це наступний номер послідовності, який очікує отримати відправник підтвердження. Це номер послідовності плюс 1 останнього успішно прийнятого байта даних. Це поле приймається до розгляду, тільки якщо встановлено перемикач «ACK».

Відправка ACK не варто нічого (це означає, що на підтвердження не витрачається сегмент), тому що 32-бітове поле номера підтвердження завжди є частиною заголовку, так само як і прапор ACK. Коли з'єднання встановлено, це поле завжди встановлено і прапор ACK завжди дорівнює 1.

TCP надає для прикладного рівня повнодуплексний сервіс. Це означає, що дані можуть передаватися в кожному напрямку незалежно від іншого напрямку. Однак, на кожному кінці з'єднання необхідно відстежувати номер послідовності даних, переданих в кожному напрямку.

TCP може бути описаний як протокол із змінним вікном без селективних чи негативних підтверджень. Ми сказали, що в TCP немає селективних підтверджень, тому що номер підтвердження в TCP заголовку означає, що відправник успішно прийняв, всі байти за винятком цього байту. Таким чином, в даний час не існує можливості підтвердити окремо обрану частину потоку даних. Наприклад, якщо байти 1-1024 прийняті нормально, а наступний сегмент містить байти 2049-3072, які одержувач не може не може підтвердити цей новий сегмент. Все що він може - це послати «ACK» з номером підтвердження 1025. Також немає ніякого сенсу посилати негативні підтвердження на сегмент. Наприклад, якщо сегмент з байтами 1025-2048 прибув, проте була визначена помилка в контрольній сумі, все що може послати - це ACK з номером підтвердження рівним 1025.

Довжина заголовка (header length) містить довжину заголовку в 32-бітових словах. Це пояснюється тим, що довжина поля опцій змінна. З 4-бітовим полем у TCP є обмеження на довжину заголовку в 60 байт.

В TCP заголовку існують 6 бітів. Один або декілька з них можуть бути встановлені в одиницю в один і той же час: URG - терміновість (urgent pointer); ACK - номер підтвердження

необхідності прийняти в розгляд (acknowledgment); PSH – одержувач повинен передати ці дані додатку якомога швидше; RST – скинути з'єднання; SYN – синхронізуючий номер послідовності для встановлення з'єднання; FIN – відправник закінчує посилку даних.

Контроль потоку даних TCP здійснюється на кожному кінці з використанням розміру вікна (window size). Це кількість байт, що починається з вказаного в полі номера підтвердження, яке додаток збирається прийняти. Це 16-бітове поле обмежує розмір вікна в 65535 байт.

Контрольна сума (checksum) охоплює собою весь TCP сегмент: TCP заголовок і TCP дані. Це обов'язкова поле, яке має бути розраховане і збережено відправником, а потім перевірено одержувачем.

Терміновість (urgent pointer) дійсна тільки в тому випадку, якщо встановлено прапор URG. Цей покажчик є позитивним зміщенням, яке має бути додано до поля номера послідовності сегмента, щоб отримати номер послідовності останнього байта термінових даних. Режим терміновості TCP це спосіб, за допомогою якого відправник передає термінові дані.

Найбільш загальне поле опцій - це опція максимального розміру сегмента (MSS - maximum segment size). На кожному кінці з'єднання ця опція зазвичай вказується в першому сегменті, з якого починається обмін (сегмент з встановленим прапором SYN, який використовується для встановлення з'єднання). Вона вказує на максимальний розмір сегмента, який може бути прийнятий відправником.

TCP/IP надає надійний, орієнтований на з'єднання, і орієнтований на потік байтів сервіс транспортного рівня.

1.2. Мережа Internet. Послуги Інтернет

Internet *— всесвітня мережа; певна сукупність технічних засобів, стандартів і домовленостей, яка дає змогу підтримувати зв'язок між різними комп'ютерами у світі.*

Основою функціонування Internet є базовий протокол TCP/IP (Transmission Control Protocol/Internet Protocol).

На одному комп'ютері можуть одночасно функціонувати декілька програм-серверів, web-сервер, сервер електронної пошти. І, як правило, на комп'ютері користувача Internet одночасно працюють кілька програм-клієнтів, наприклад клієнтські програми: для роботи з електронною поштою і програма-браузер для переглядання гіпертекстових Web-документів.

Сервер Internet *— комп'ютер або програма, що надає послуги іншим комп'ютерам чи програмам.* ***Клієнт —*** *комп'ютер чи програма, що використовує ресурси серверу Internet.*

Сервери Internet встановлені в організаціях (сервіс-провайдерах), які надають комерційні мережеві послуги індивідуальним і колективним користувачам. Ці сервери називають ще *хостами* (від англ. host — господар), або *вузлами мережі.*

***Сервіс-провайдери** — установи, які надають комерційні послуги з підключення до Internet (Internet Service Provider, ISP).*

Сервіс-провайдер має в розпорядженні комп'ютерну мережу з постійним сполученням з Internet, у складі якої обов'язково є сервери доступу. З їх допомогою здійснюється підключення до Internet абонентів — окремих користувачів або користувачів локальних мереж цілих установ. Зв'язок між серверами доступу підтримується цілодобово завдяки постійно діючій системі магістралей, тунелів і розв'язок, якими безупинно рухається інформація.

Принципи влаштування Internet достатньо зрозумілі пересічному користувачеві. Кожній вузловий комп'ютер Internet має власну адресу. Як правило, вона розміщена справа від знаку @ (комерційне "at") у будь-якій адресі електронної пошти (E-mail). Така адреса виокремлює певний комп'ютер з величезної кількості вузлів Internet і дає змогу іншим комп'ютерам знайти його відповідно до встановленої ієрархії. Наприклад, в адресі електронної пошти serg@glas.apc.org адреса вузлового комп'ютера Internet визначається як glas.apc.org. Зліва від знаку @ міститься назва поштової скриньки — каталогу для електронних повідомлень. Поштова скринька (у нашому прикладі — serg) може бути розташована як на вузловому комп'ютері, так і на комп'ютері кінцевого користувача.

Кожна з частин адреси після символу @ називається *доменом.* Ієрархія доменів часто створюється за географічною ознакою. Так, електронна адреси vlad@icom.kiev.ua визначає комп'ютер з ім'ям icom, який знаходиться у Києві (в Україні) — kiev.ua. Домени верхнього рівня (у наших прикладах — org і ua) найчастіше вказують на регіональні ознаки в електронній адресі або на характер діяльності установи, яка має електронну пошту (або надає таку послугу). Домени верхнього рівня мають такі значення:

ua — Україна;

uk — Велікобрітанія;

de — Німеччина;

com — комерційні фірми;

net — мережеві організації;

edu — університети інші навчальні заклади;

.org — державні й суспільні установи.

Глобальна обчислювальна мережа Internet надає багато послуг: WWW, електронна пошта, FTP та ін.

***WWW (World Wide Web)** — Internet-послуга, призначена для гіпертекстового поєднання*

мультимедійних документів зі всього світу та організації надійних інформаційних зв'язків між ними, незалежних від фізичного розташування документів.

Вона потребує прямого сполучення з Internet, а також наявність спеціальних програм-браузерів для клієнта.

WWW – це своєрідна велика бібліотека Internet. Web-вузли, розташовані в різних куточках планети, нагадують книги з цієї бібліотеки, а Web-сторінки — сторінки цих книг. WWW-сторінки містять гіпертекст й ілюстрації до нього, об'єднані між собою посиланнями — зв'язками, які дають змогу легко переходити від одного матеріалу до іншого.

Гіпертекст (hypertext) *— метод надання інформації у вигляді тексту, окремі фрагменти якого з'єднані з допомогою посилань.*

Гіперпосилання (hyperlink), посилання *— спосіб зв'язку між різними компонентами інформації у WWW-системі.*

Гіпертекстові посилання активуються натисканням на обраному підкресленому слові або відокремленому малюнку. При цьому виконується перехід до іншого фрагмента поточного файлу чи документа, який може бути розташований на дисках серверу, територіальне дуже віддаленого від першого серверу.

У 1945 р. В. Буш — науковий радник президента США Г. Трумена, проаналізувавши способи подання інформації у вигляді звітів, доповідей, проектів, графіків тощо і зрозумівши їх неефективність, запропонував спосіб розташування інформації за принципом асоціативного мислення. На основі цього принципу було розроблено модель гіпотетичної машини МЕМЕКС. Через двадцять років Т. Нельсон реалізував цей принцип на ЕОМ і дав йому назву гіпертексту.

Web-сервер — комп'ютер у мережі, на якому встановлене серверне програмне забезпечення для оброблення запитів програм — браузерів, що використовує протокол HTTP.

Протокол HTTP (Hypertext Transfer Protocol) *— один з протоколів, який використовується в Internet і містить правила, за якими WWW-документи передаються з серверу до програми-браузера на комп'ютер користувача.*

На WWW-серверах можна знайти різноманітні дані: широкий спектр інформації з університетів і науково-дослідницьких організацій; правові довідкові системи; рекламу комерційних фірм з переліком товарів і послуг; електронні версії газет і журналів; фахову та розважальну інформацію тощо. Найпоширенішими є сервери, які надають наочну інформацію про свої організації: загальні дані про заклад і його історію; опис напрямків діяльності закладу; відомості про його працівників і керівництво; "координати" установи: поштову та електронну адреси, телефони; опис продукції або послуг. Існують також Internet-крамниці на основі WWW-серверів, які містять інформацію про товари і послуги. Ця інформація надається виробниками чи продавцями. Пересічний користувач Internet може зробити в електронних

демонстраційних залах замовлення на певний товар чи послугу, переглянувши каталог товарів і зовнішній вигляд самого товару.

Початкова сторінка WWW-вузла (homepage) *— перша WWW-сторінка, яку відкриває користувач, потрапивши до WWW-серверу; містить загальні відомості певного WWW-серверу.*

Щоб отримати доступ до WWW, насамперед необхідне on-line-сполучення. «Серфінг» користувача по WWW завжди починається з певного вузла. Початкова WWW-сторінка цього вузла є вхідним пунктом до системи World Wide Web. Кожна сторінка вузла, в тому числі й початкова, має свою адресу у форматі URL, наприклад: http://www.museum.com/index.html. У цій адресі поширення імені файла .htm або .html вказує на гіпертекстовий формат файлів, у яких зберігаються WWW-сторінки (HTML — Hypertext Markup Language — мова створення гіпертекстових файлів).

URL (Universal Resource Locator, уніфікований покажчик ресурсу) *— адреса інформаційного ресурсу в Internet, де вказаний протокол, за правилами якого передаються дані, ім’я серверу, на якому зберігається файл, а також може бути вказаний шлях до каталогу файла і безпосередньо ім’я.*

URL — спосіб компактної унікальної адресації для WWW-сторінок — http://; ресурсів Gopher — gopher://; ресурсів FTP — ftp://; груп новин — news://, або nntp:// тощо.

Для перегляду WWW-сторінок необхідно скористатися однією з програм-браузерів. Найпоширеніші нині браузери: FireFox, Microsoft Internet Explorer, Opera, Chrome.

WWW-сервери Internet надають доступ до своїх сторінок одночасно багатьом користувачам. Але не всі сервери мають рівні технічні можливості. Деякі з них не можуть обслуговувати велику кількість запитів від програм-браузерів з різних комп’ютерів.

Часто говорять про “мандри” WWW-системою і переходи до різних вузлів. При цьому виконуються переходи за посиланнями на нові сторінки і звернення до тем, про які раніше користувач відомостей не мав. Зараз інформаційні джерела WWW-системи настільки всеосяжні, що пересічному користувачу легко “заблукати” в ній у пошуках необхідних даних. Тому так само, як для звичайних публікацій створюються каталоги, довідники і реферативні журнали, для WWW-сторінок створюються сервери, які містять тільки посилання на інші сервери, зібрані за темами і супроводжені коментарями.

Інший спосіб знайти сторінки, які цікавлять, — скористатися одним із багатьох ***пошукових серверів***. Вони здійснюють пошук WWW-сторінок за певними ключовими словами. Такі сервери регулярно перевіряють світову систему WWW, накопичуючи дані про сторінки, які знову з’являються у системі. Результат пошуку користувач отримує у формі переліку посилань на документи, які містять слова із запиту.

Завдяки універсальності WWW-технології найвідоміші пошукові сервери нині перетворилися на інформаційні портали.

Інформаційний портал *— багатофункціональний сервер із зручним інтерфейсом і системою засобів, які полегшують користувачам навігацію в глобальній мережі; надає додаткові послуги: E-mail, Web-хостинг тощо.*

Користувачі WWW можуть не тільки переглядати чужі сторінки, а й готувати власні. Створення і публікація матеріалів для WWW-сторінок стало простим заняттям завдяки сучасним програмним засобам, і опублікована інформація швидко стає надбанням громадськості.

Для публікації сторінок в WWW насамперед необхідний вузол Internet зі спеціальним програмним забезпеченням. Деякі власники надають можливість усім бажаючим безкоштовно (або за дуже помірну плату) публікувати свої WWW-сторінки.

Web-хостинг *— надання господарями сайту місця для розміщення інформації.*

Сайт *—адреса розташування інформаційного ресурсу в Internet.*

Електронна пошта (E-mail). Це просте й дешеве вирішення проблем підтримки постійних контактів між людьми. E-mail дає змогу скласти текстове повідомлення на комп'ютері й відправити його з допомогою мережі іншому користувачеві. Електронною поштою можна також пересилати електронні таблиці в певному форматі, графічні файли, програми тощо.

Як правило, повідомлення електронної пошти складається з тексту в ASCII-коді. Різні рядки заголовка перед основною частиною електронного повідомлення мають назву Header і можуть містити такі дані:

- Sender (Отправитель): адреса відправника повідомлення;
- Date (Дата): дата його відправлення;
- Received (Получено): інформація про те, коли (за місцевим часом) і через які вузли пройшло повідомлення;
- Message-Id (ID сообщения): умовний цифровий код для ідентифікації певного електронного повідомлення, спроможний допомогти в наступних розшуках, якщо повідомлення не отримане адресатом;
- To (Кому): адресат з його адресою в електронній пошті;
- From (От): відправник з адресою в електронній пошті;
- Subject (Тема): тема повідомлення, яку визначає відправник.

Не завжди усі ці рядки видно. Часто сучасні поштові програми з графічним інтерфейсом «ховають» від користувача «малокорисні» для нього рядки Sender, Received, Message-Id.

FTP-послуга. ***FTP (File Transfer Protocol) —*** *Internet-сервіс для передавання файлів (в*

т. ч. програмних) мережею Internet.

До появи і швидкого розвитку World Wide Web загальнодоступні файлові архіви FTP-серверів були основним засобом накопичення і поширення серед користувачів Internet різної інформації — від програмного забезпечення у вигляді текстів програм і .exe-модулів до художньої літератури. Перші FTP-сервери виникли на «первинному» етапі розвитку Internet на базі протоколу передавання файлів між комп'ютерами (File Transfer Protocol).

Нині для безпосереднього звертання до FTP-серверів і копіювання файлів на диск свого комп'ютера найчастіше використовуються IP-підключення користувача до Internet і поширені програми перегляду WWW-сторінок, такі, як браузери Google Chrome, Internet Explorer, Opera, Mozzilla FireFox. Вони можуть працювати не тільки з протоколом передавання гіпертекстових файлів — HTTP, а й з протоколами FTP та іншими.

Великі вузли, на яких влаштовані WWW-сервери, підтримують також й FTP-сервери. Щоб зробити доступ до ресурсів FTP-серверів зручнішим, на багатьох серверах встановлено WWW-інтерфейс до каталогів FTP-серверів. Це означає, що загальну інформацію про файловий архів і опис його змісту користувач може отримати, працюючи з WWW, а потім, обравши на WWW-сторінці потрібний файл, скопіювати його з допомогою FTP-протоколу.

1.3. Служба WWW. Протокол HTTP

Служба WWW (World Wide Web) - основна служба в мережі Інтернет, що дозволяє отримувати доступ до інформації на будь-яких серверах, підключених до мережі. Служба WWW являє собою безліч незалежних, але взаємопов'язаних серверів і призначена для обміну текстовою, графічною, аудіо та відео інформацією. Працюючи з Web, користувач послідовно з'єднується з Web-серверами і отримує інформацію. WWW побудована за схемою "клієнт-сервер". В якості клієнта виступає браузер, який є також і інтерпретатором HTML. Як інтерпретатор, браузер в залежності від команд (тегів) виконує різні функції: розміщення тексту на екрані, обмін інформацією з сервером по мірі аналізу отриманого HTML-тексту та ін.

Web-сервер — комп'ютер у мережі, на якому встановлене серверне програмне забезпечення для оброблення запитів програм — браузерів, що використовує протокол HTTP. Web-сервер - це програмне забезпечення, яке відповідає за прийом запитів браузерів, пошук зазначених файлів і повернення їх вмісту. Web-сервер зберігає інформацію у вигляді текстових файлів, званих сторінками Web-серверу. Крім тексту, такі сторінки можуть містити посилання на інші сторінки, посилання на графічні зображення, аудіо- та відеоінформацію, різні об'єкти введення даних (поля, кнопки, форми і т. д.), а також інші об'єкти. Сторінки Web являють

собою деяку сполучну ланка між об'єктами різних типів.

Web-сервер є програмою, що запускається на підключеному до мережі комп'ютері і використовує протокол HTTP для передачі даних. У найпростішому вигляді така програма отримує по мережі HTTP-запит на певний ресурс, знаходить відповідний файл на локальному жорсткому диску і відправляє його по мережі клієнту. Web-сервери здатні динамічно формувати ресурси у відповідь на HTTP-запит.

Найбільш поширеним веб-сервером є Apache. Деякі інші відомі веб-сервери: IIS від компанії Microsoft, розповсюджуваний з ОС сімейства Windows; nginx; Google Web Server - веб-сервер, заснований на Apache і доопрацьований компанією Google та ін.

Для доступу до інформації, розташованої на web-серверах, користувачі застосовують спеціальні клієнтські програми - браузери.

Веб-оглядач, оглядач, браузер або браузер (від англ. Web browser) - прикладне програмне забезпечення для перегляду веб-сторінок; змісту веб-документів, комп'ютерних файлів і їх каталогів; управління веб-додатками; а також для вирішення інших завдань. У глобальній мережі браузери використовують для запиту, обробки, маніпулювання і відображення змісту веб-сайтів. Багато сучасних браузерів також можуть використовуватися для обміну файлами з серверами ftp, а також для безпосереднього перегляду змісту файлів багатьох графічних форматів (gif, jpeg, png, svg), аудіо-відео форматів (mp3, mpeg), текстових форматів (pdf, djvu) та інших файлів. Web-браузер - це програмне забезпечення для перегляду web-сайтів, тобто для запиту web-сторінок з WWW, для їх обробки і виведення, і для реалізації переходу від однієї сторінки до іншої. Браузер - комплексний додаток для обробки і виведення різних складових web-сторінки, і для надання інтерфейсу між web-сайтом і його відвідувачем. Браузер здатний попередньо обробляти дані, що відправляються на сервер, а також обробляти і представляти результати, отримані від сервера, в зручному для користувача вигляді.

Найбільш відомі такі браузери: Google Chrome, Microsoft Internent Explorer, Mozilla FireFox, Safari, Opera.

Протокол передачі гіпертексту HTTP (Hyper Text Transfer Protocol) - базується на TCP/IP і забезпечує доступ до документів на web-вузлах. Основне завдання протоколу полягає у встановленні зв'язку з web-сервером і забезпеченні доставки HTML-сторінок web-браузеру клієнта.

Транспортним протоколом для HTTP є протокол TCP, причому сервер HTTP (сервер Web) знаходиться в стані очікування з'єднання з боку клієнта стандартно по порту 80 TCP, а клієнт HTTP (браузер Web) є ініціатором з'єднання. Однією з найважливіших функцій сервера Web є надання доступу до частини локальної файлової системи. Для цього в настройках сервера вказується деяка директорія (каталог), яка є кореневою для даного сервера Web. щоб

опублікувати документ, тобто зробити його доступним користувачам, які відвідують даний сервер (здійснює з ним з'єднання по протоколу HTTP), потрібно скопіювати цей документ в кореневу директорію Web-сервера або в одну з її піддиректорій.

При з'єднанні по протоколу HTTP на сервері створюється процес з правами користувача, як правило, не існуючого реально, а спеціально створеного для перегляду ресурсів сервера. Налаштовуючи права та дозволи даного користувача, можна управляти доступом до ресурсів Web.

Взаємодія між клієнтом і сервером Web здійснюється шляхом обміну повідомленнями. Повідомлення HTTP діляться на запити клієнта сервера і відповіді сервера клієнту.

Запити та відповіді виглядають наступним чином:

початкова рядок

заголовок 1

заголовок 2

...

заголовок N

CR LF (порожній рядок)

тіло повідомлення (може бути відсутнім).

HTTP-заголовки

Формат початкового рядка (start-line) клієнта і сервера розрізняються. Заголовки бувають чотирьох видів:

1. Загальні заголовки (general-headers), які можуть бути присутніми як в запиті, так і у відповіді.

2. Заголовки запитів (request-headers), які можуть бути присутніми тільки в запиті.

3. Заголовки відповідей (response-headers), які можуть бути присутніми тільки в відповіді.

4. Заголовки об'єкта (entity-headers), які відносяться до тіла повідомлення і описують його вміст.

Кожен заголовок складається з назви, символу двокрапки ":" і значення.

В тілі повідомлення міститься власне передана інформація. Тіло повідомлення являє собою послідовність октетів (байтів). Тіло повідомлення може бути закодовано, наприклад, для зменшення обсягу переданої інформації, при цьому спосіб кодування вказується в заголовку об'єкта Content-Encoding.

Запит від клієнта до сервера складається з рядка запиту (request-line), заголовків (загальних, запитів, об'єкта) і, можливо, тіла повідомлення.

Рядок запиту:

<Команда HTTP>

<Ідентифікатор запитуваного ресурсу>

<Версія HTTP>

Основні команди протоколу HTTP наступні:

OPTIONS - запит інформації про опції з'єднання (наприклад, методах, типах документів, кодуваннях), які підтримує сервер для ресурсу, що запитується. Якщо ідентифікатор запитуваного ресурсу - зірочка ("*"), то запит призначенийдля звернення до серверу в цілому.

GET - Дозволяє отримати інформацію, пов'язану із ресурсом, що запитується. Якщо ідентифікатор ресурсу вказує на документ, то сервер повертає вміст цього документа (вміст файлу). Якщо запитуваний ресурс є додатком (програмою), що формує в процесі своєї роботи деякі дані, то в тілі повідомлення відповіді повертаються ці дані. Якщо ідентифікатор запитуваного ресурсу вказує на директорію (каталог, папку), то, в залежності від налаштувань сервера, може бути повернуто або вміст директорії (список файлів), або вміст одного з файлів, що знаходиться в цій директорії (як правило, index.html). У разі запиту папки її ім'я може зазначатися як з символом "/" на кінці, так і без нього. При відсутності на кінці ідентифікатора ресурсу даного символу сервер видає одна з відповідей з перенаправленням (з кодами статусу 301 або 302).

Різновидами команди GET є "умовний GET" ("conditional GET") і "частковий GET" ("partial GET"). Умовний GET запрошувати передачу об'єкта, тільки якщо він задовольняє умовам, описаним в наведених заголовках. Частковий GET запрошує передачу тільки частини об'єкта.

HEAD - Ідентична команді GET, за винятком того, що сервер не повертає у відповіді тіло повідомлення.

POST - Використовується для запиту, при якому сервер приймає дані, включені в тіло повідомлення (об'єкт) запиту, і відправляє їх на обробку додатком, вказаною як запитуваний ресурс.

PUT - Тіло повідомлення, яке передається в запиті, зберігається на сервері, причому ідентифікатор запитуваного ресурсу буде ідентифікатором збереженого документа.

DELETE - Запит на видалення ресурсу, що має запитуваний ідентифікатор.

TRACE - Використовується для тестування або діагностики. Одержувач запиту (сервер Web) відправляє отримане повідомлення назад клієнтові як тіло повідомлення відповіді.

Наступний приклад запиту отримує веб-сторінку з каталогу «/wiki/HTTP», що знаходиться на сервері «uk.wikipedia.org»:

GET /wiki/HTTP HTTP/1.1

Host: uk.wikipedia.org

User-Agent: firefox/5.0 (Linux; Debian 5.0.8; en-US; rv:1.8.1.7) Gecko/20070914 Firefox/2.0.0.7

Connection: close

Після отримання та інтерпретації повідомлення запиту, сервер відповідає повідомленням HTTP відповіді.

Перший рядок відповіді - це рядок стану (Status-Line):

<Версія HTTP>

<Код стану>

<Пояснююча фраза>

Код стану (Status-Code) - це цілочисельний трьохрозрядний код результату розуміння і задоволення запиту. Код стану призначений для обробкипрограмним забезпеченням, а фраза призначена для користувачів.

Перша цифра коду стану визначає клас відповіді. Останні дві цифри не мають певної ролі в класифікації. Є 5 значень першої цифри:

· 1xx: Інформаційні коди - запит отриманий, продовжується обробка.

· 2xx: Успішні коди - дія була успішно отримано, зрозуміла і оброблена.

· 3xx: Коди перенаправлення - для виконання запиту повинні бути зроблені подальші дії.

· 4xx: Коди помилок клієнта - запит має помилку синтаксису або не може бути виконаний.

· 5xx: Коди помилок сервера - сервер не в змозі виконати допустимий запит.

За рядком стану слідують заголовки (загальні, відповіді і об'єкта) і, можливо, тіло повідомлення.

Приклад HTTP-відповіді сервера на HTTP-запит приведений вище:

HTTP/1.1 200 OK

Server: Apache

Content-Language: uk

Content-Type: text/html; charset=utf-8

Content-Length: 1234

<HTML>

<HEAD>…</HEAD>

<BODY>…Текст сторінки…</BODY>

</HTML>

Internet — на 90 відсотків англомовне середовище. Однак у мережі можна працювати і з

кирилицею, тобто підтримується передавання повідомлень українською, російською й іншими слов'янськими мовами. Крім того, існує багатомовне програмне забезпечення для роботи з мережею. Але користувачі Internet, які звертаються до серверів за інформацією слов'янськими мовами, стикаються з проблемою, якої не існує під час роботи з англомовними документами.

Річ у тім, що існує декілька систем кодування символів кирилиці, тобто декілька різних кодових таблиць відповідності символу і його числового коду. Кожний символ кодується одним байтом, що дає змогу надати 256 символів і керуючих кодів. Якщо для латинських символів, які використовуються в англійській мові, існує загальноприйнятий стандарт ASCII (числові коди від 65 до 90 і від 97 до 122), то символи національних абеток (в тому числі української та російської) надаються кодами другої частини таблиці (номери кодів від 128 до 255). Існують такі стандарти кодування кирилиці: KOI8; CP-1251 — кодова сторінка (code page) для кирилиці у Windows; ISO — стандарт передавання кирилиці між вузлами мережі, встановлений міжнародною організацією зі стандартизації (International Standartization Organization).

Єдиний покажчик ресурсів (англ. Uniform Resource Locator, URL) – однаковий локатор (визначник місцезнаходження) ресурсу. Раніше називався Universal Resource Locator - універсальний покажчик ресурсу. URL служить стандартизованим способом запису адреси ресурсу в мережі Інтернет.

Спочатку локатор URL був розроблений як система для максимально природного вказівки на місцезнаходження ресурсів в мережі. Локатор повинен був бути легко розширюваним і використовувати лише обмежений набір ASCII-символів (наприклад, пробіл ніколи не застосовується в URL). У зв'язку з цим, виникла наступна традиційна форма запису URL:

<схема>://<логин>:<пароль>@<хост>:<порт>/?<параметры>#<якорь>

У цьому записі:

Схема – схема звернення до ресурсу; в більшості випадків мається на увазі мережевий протокол.

Логін – ім'я користувача, яке використовується для доступу до ресурсу

Пароль – пароль зазначеного користувача

Хост – повністю прописане доменне ім'я хоста в системі DNS або IP-адреса хоста у формі чотирьох груп десяткових чисел, розділених крапками; числа - цілі в інтервалі від 0 до 255.

Порт – порт хоста для підключення.

URL-шлях – уточнююча інформація про місце знаходження ресурсу; залежить від протоколу.

Параметри – врядок запиту з переданими на сервер (методом GET) параметрами.

Роздільник параметрів - знак &.

приклад:?параметр_1=значение_1&параметр_2=значение_2&параметр3=значение_3

Якір – ідентифікатор «якоря», що посилається на деяку частину (розділ) документа, що відкривається.

1.4. Еволюція Web

Основні тенденції Веб 1.0 включали турботи про проблеми безпеки і приватності в односторонньому потоці інформації, через веб-сайти, що містять матеріал «тільки для читання». Характерним для Веб 1.0 також були комп'ютерна неграмотність широких мас і поширеність повільних типів підключення до Інтернету, на додачу до обмежень самого Інтернету.

Типові принципи Веб 1.0:

- статичні сторінки замість генерованого користувачами динамічного контенту;
- бідна гіпертекстова розмітка;
- використання фреймів;
- використання специфічних тегів HTML;
- гостьові книги, форуми або чати;
- вказування конкретної роздільної здатності монітору, при якій дизайн сайту відображається коректно;
- вкрай рідкісне і непопулярне використання стилів CSS при оформленні сторінок сайту.

Web 2.0 базується на декількох старих, однак по новому осмислених технологіях: AJAX; JSON; SVG; RSS; XPath; Canvas. Ці технології дозволили винести веб на якісно новий рівень, однак потрібно усвідомлювати, що самі по собі дані технології не є революційними, революцію Web 2.0 зробили методики використання даних технологій.

Найбільш поширеною версією трактування терміна Веб 3.0 є ідентифікація його як Семантичної Павутини (Semantic Web). Головна думка цієї концепції базується на впровадженні мета-мови, що описує зміст сайтів для організації автоматичного обміну між серверами.

Семантична павутина (Semantic Web) - частина глобальної концепції розвитку мережі Інтернет, метою якої є реалізація можливості машинної обробки інформації, доступної у Всесвітній павутині. Основний акцент концепції робиться на роботі з метаданими, що однозначно характеризують властивості і зміст ресурсів Всесвітньої павутини, замість використовуваного в даний час текстового аналізу документів. У семантичній павутині

передбачається повсюдне використання, по-перше, універсальних ідентифікаторів ресурсів (URI), а по-друге - онтологій і мов опису метаданих.

Ця концепція була прийнята і просувається Консорціумом W3. Для її впровадження передбачається створення мережі документів, що містять метадані про ресурси Всесвітньої павутини та існуючої паралельно з ними. Тоді як самі ресурси призначені для сприйняття людиною, метадані використовуються машинами (пошуковими роботами та іншими інтелектуальними агентами) для проведення однозначних логічних висновків про властивості цих ресурсів.

По суті, Веб 3.0 використовує технологічну базу Веб 2.0:

- AJAX - завантаження даних без перезавантаження самої веб-сторінки;
- RIA (Adobe Flex, JavaFX, Microsoft Silverlight);
- XML (eXtensible Markup Language) - мова розмітки даних, є зведенням загальних синтаксичних правил;
- RSS (Really Simple Syndication) - сімейство XML-форматів, призначених для опису стрічок новин, анонсів статей, змін в блогах і т. П .;
- Теги - відображення тегів у вигляді хмари, що значно спрощує визначення користувачем найбільш актуальної інформації;
- блогові структура інформації - стрічкова подача інформації, де потік йде по спадаючій зверху-вниз, а метод сортування задає користувач.

Головна ідея Веб 3.0, полягає в тому, щоб користувач, який до цього одноосібно був залучений в процес формування контенту, відтепер творить колективно, крім інших користувачів, були експерти різних напрямків, причому статус користувача може бути змінений на експертний, так само, як і форма співпраці творця контенту та порталу. Експерт повинен виступити своєрідним модератором контенту, що публікується. По суті, не виключається і можливість платної основи для співробітництва, але набагато більш важливим моментом є поява в порталах формату Веб 3.0 «колективного розуму» (wisdom of the crowds). Веб 3.0 передбачає появу вузькоспеціалізованих ресурсів, де буде проведена агрегація всіх необхідних користувачеві сервісів та інструментів професійної соціальної складової і здійснюватиметься публікація експертно-модеруємого контенту.

1.5. Сучасні технології web-програмування

Веб-програмування (Веб-розробка) - це бурхливо розділ програмування, що розвивається і орієнтований на розробку динамічних Інтернет-додатків.

Мови веб-програмування діляться на дві групи: клієнтські і серверні.

Клієнтські мови обробляються на стороні користувача (в основному в браузері). Відповідно, обробка скрипта залежить від браузера користувача, і користувач має повноваження налаштувати свій браузер так, щоб той взагалі ігнорував скрипти. При цьому якщо браузер старий, він може не підтримувати ту чи іншу мову або версію мови, на яку спирався розробник. З сучасними браузерами таких проблем виникати не повинно, до того ж мови програмування не так вже й часто кардинально оновлюються (раз на кілька років) і кращі з них давно відомі. Код клієнтського скрипта може подивитися кожен, вибравши в меню свого браузера «Вихідний код сторінки».

Перевага клієнтського мови полягає в тому, що обробка скриптів на такій мові може виконуватися без відправки документа на сервер. Програма відразу перевірить правильне заповнення форми перед відправкою, і, якщо необхідно, виведе помилку. Звідси ж випливає і те обмеження, що за допомогою клієнтського мови програмування ніщо не може бути записано на сервері.

Найпоширенішим з клієнтських мов є JavaScript, розробниками якого є компанія Netscape спільно з компанією Sun Microsystems. Крім того використовуються також такі технології як AJAX, Adobe Flash, Microsoft Silverlight та ін.

Серверні мови програмування відкривають перед програмістом великі простори в діяльності.

Коли користувач робить запит на якусь сторінку (переходить на неї по посиланню, або вводить адресу в адресному рядку свого браузера), то викликана сторінка спочатку обробляється на сервері (тобто виконуються всі програми, пов'язані зі сторінкою) і тільки потім повертається до відвідувача по мережі у вигляді файлу. Цей файл може мати розширення: HTML, PHP, ASP, Perl, SSI, XML, DHTML, XHTML.

Робота програм вже повністю залежна від сервера, на якому розташований сайт, і від того, яка версія тієї чи іншої мови підтримується.

Важливою стороною роботи серверних мов є Система управління базами даних (СУБД). Це, по суті, теж сервер, на якому в певному користувачем порядку зберігається різна необхідна інформація, яка може бути викликана в будь-який момент. Популярними серед систем управління базами даних є:

- IBM DB2;
- Microsoft SQL Server;
- MySQL;
- Oracle;
- PostgreSQL;
- SQLite.

Розглянемо докладніше найбільш відомі мови і технології.

HTML (HyperText Markup Language - «мова розмітки гіпертексту») - стандартна мова розмітки документів у Всесвітній павутині. Більшість веб-сторінок створюються за допомогою мови HTML. Мова HTML інтерпретується браузером і відображається у вигляді документа, в зручній для людини формі. HTML є додатком SGML (стандартної узагальненої мови розмітки) і відповідає міжнародному стандарту ISO 8879.

Мова HTML була розроблена британським ученим Тімом Бернерс-Лі приблизно в 1991-1992 роках в стінах Європейської ради з ядерних досліджень в Женеві (CERN). HTML створювалася як мова для обміну науковою і технічною документацією, придатна для використання людьми, які не є фахівцями в галузі верстки. HTML успішно справлявся з проблемою складності SGML шляхом визначення невеликого набору структурних і семантичних елементів - дескрипторів. Дескриптори також часто називають «тегами». За допомогою HTML можна легко створити відносно простий, але красиво оформлений документ. Крім спрощення структури документа, в HTML внесена підтримка гіпертексту. Мультимедійні можливості були додані пізніше. Спочатку мова HTML була задумана і створена як засіб структурування та форматування документів без їх прив'язки до засобів відтворення (відображення).

Текстові документи, що містять код на мові HTML (такі документи традиційно мають розширення .html або .htm), обробляються спеціальними додатками, які відображають документ у його форматованому вигляді. Такі додатки, звані «браузерами» або «Інтернет-оглядачами», зазвичай надають користувачеві зручний інтерфейс для запиту веб-сторінок, їх перегляду (і виведення на інші зовнішні пристрої) і, при необхідності, відправки введених користувачем даних на сервер.

XHTML (Extensible Hypertext Markup Language - розширювана мова розмітки гіпертексту) - мова розмітки веб-сторінок, за можливостями можна порівняти з HTML, створена на базі XML. Як і HTML, XHTML відповідає специфікації SGML, оскільки XML є її підмножиною.

Стандарт XHTML побудований не як самодостатній опис мови, а як перелік відмінностей між HTML 4.01 і XHTML:

1. Всі елементи повинні бути закриті. Теги, які не мають тегу, що закриває (наприклад, <img> або <br>) повинні мати на кінці / (наприклад, <br />).

2. Булеві атрибути записуються в розгорнутій формі. Наприклад, слід писати <option selected = "selected"> або <td nowrap = "nowrap">.

3. Імена тегів і атрибутів повинні бути записані малими літерами (наприклад, замість <IMG ALT = "" />).

4. XHTML набагато суворіше ставиться до помилок в коді; <I & скрізь, навіть в URL, повинні заміщатися & lt; і & amp; відповідно. За рекомендацією W3C браузери, зустрівши помилку в XHTML, повинні повідомити про неї і не обробляти документ.

5. Кодуванням за замовчуванням є UTF-8 (на відміну від HTML, де кодуванням за замовчуванням є ISO 8859-1).

Для XHTML сторінок рекомендується задавати MIME-тип - application / xhtml + xml, але це не є обов'язковим, більше того - браузер Internet Explorer 8 і молодші версії, не зможуть обробляти сторінку, тому з XHTML 1.0 традиційно використовується MIME-тип для HTML - text / html.

CSS (Cascading Style Sheets - каскадні таблиці стилів) - технологія опису зовнішнього вигляду документа, написаного мовою розмітки. Переважно використовується як засіб оформлення веб-сторінок у форматі HTML і XHTML, але може застосовуватися з будь-якими видами документів у форматі XML, включаючи SVG і XUL.

Термін «каскадні таблиці стилів» був запропонований Хокон Віум Лі в 1994 році. Спільно з Бертом Босом він став розвивати CSS.

CSS використовується творцями веб-сторінок для завдання кольорів, шрифтів, розташування та інших аспектів представлення документа. Основною метою розробки CSS було розділення вмісту (написаного на HTML або іншій мові розмітки) і представлення документа (написаного на CSS). Це розділення може збільшити доступність документа, надати велику гнучкість і можливість управління його уявленням, а також зменшити складність і повторюваність в структурному вмісті. Крім того, CSS дозволяє представляти один і той же документ в різних стилях або методах виведення, таких як екранне уявлення, друк, читання голосом (спеціальним голосовим браузером або програмою читання з екрану) та ін.

Найбільш повно підтримують стандарт CSS є браузери, що працюють на Gecko (Mozilla Firefox і ін.) І WebKit (Arora, Google Chrome, Safari), а також браузер Opera. Що стосується Internet Explorer, то тільки восьмий його версія повністю підтримує CSS 2.1 і частково - CSS 3.

Переваги застосування CSS:

- кілька дизайнів сторінки для різних пристроїв перегляду;
- зменшення часу завантаження сторінок сайту за рахунок перенесення правил представлення даних в окремий CSS-файл;
- простота подальшої зміни дизайну;
- додаткові можливості оформлення, наприклад, за допомогою CSS-верстки можна зробити блок тексту, який решту тексту буде обтікати або зробити так, щоб меню було завжди видно при прокручуванні сторінки.

- Недоліки застосування CSS:
- різне відображення верстки в різних браузерах (особливо застарілих), які по-різному інтерпретують одні й ті ж дані CSS;
- часто зустрічається необхідність на практиці виправляти не тільки один CSS-файл, але й теги HTML і серверний код, які складним і ненаглядним способом пов'язані з селекторами CSS.

XML (eXtensible Markup Language - розширювана мова розмітки) - рекомендована Консорціумом Всесвітньої павутини мова розмітки, що фактично є зведенням загальних синтаксичних правил. XML - текстовий формат, призначений для зберігання структурованих даних (замість існуючих файлів баз даних), для обміну інформацією між програмами, а також для створення на його основі більш спеціалізованих мов розмітки (наприклад, XHTML), іноді званих словниками. XML є спрощеною підмножиною мови SGML. Роком народження XML можна вважати 1996 рік, в кінці якого з'явився чорновий варіант специфікації мови, або 1998, коли ця специфікація була затверджена.

Метою створення XML було забезпечення сумісності при передачі структурованих даних між різними системами обробки інформації, особливо при передачі таких даних через Інтернет. Словники, засновані на XML (наприклад, RDF, RSS, MathML, XHTML, SVG), самі по собі формально описані, що дозволяє програмно змінювати і перевіряти документи на основі цих словників, не знаючи їх семантики, тобто, не знаючи смислового значення елементів. Важливою особливістю XML також є застосування так званих просторів імен (namespace).

До переваг використання XML можна віднести:

- XML - мова розмітки, що дозволяє відобразити двійкові дані в текст, що читається людиною і аналізуємий комп'ютером;
- XML підтримує Юнікод;
- у форматі XML можуть бути описані такі структури даних як записи, списки і дерева;
- XML - це самодокументований формат, який описує структуру і імена полів так само як і значення полів;
- XML має строго певний синтаксис і вимоги до аналізу, що дозволяє йому залишатися простим, ефективним і несуперечливим;
- XML - формат, заснований на міжнародних стандартах;
- Ієрархічна структура XML підходить для опису практично будь-яких типів документів, крім аудіо і відео мультимедійних потоків, растрових зображень, мережевих структур даних і двійкових даних;
- XML являє собою простий текст, вільний від ліцензування та будь-яких обмежень;

- XML не залежить від платформи;
- XML є підмножиною SGML;
- вже накопичений великий досвід роботи з мовою і створені спеціалізовані додатки;
- XML не накладає вимог на розташування символів у рядку;
- на відміну від бінарних форматів, XML містить метадані про імена, типи і класи об'єктів;
- XML має реалізації парсерів для всіх сучасних мов програмування;
- XML підтримується на низькому апаратному, мікропрограмному та програмному рівнях в сучасних апаратних рішеннях.

До недоліків XML можна віднести:

- синтаксис XML надмірний:
- розмір XML документа істотно більше бінарного представлення тих же даних;
- надмірність XML може вплинути на ефективність додатку (зростає вартість зберігання, обробки і передачі даних);
- для великої кількості завдань не потрібна вся міць синтаксису XML і можна використовувати значно простіші рішення;
- неоднозначність моделювання, тобто немає загальноприйнятої методології для моделювання даних в XML;
- XML не містить вбудованої в мову підтримки типів даних;
- ієрархічна модель даних, пропонована XML, обмежена в порівнянні з реляційною моделлю і об'єктно-орієнтованими графами і мережевою моделлю даних;
- простори імен XML складно використовувати і їх складно реалізовувати в XML-парсер.

Найбільш поширені три способи перетворення XML-документа:

1. Застосування стилів CSS.
2. Застосування перетворення XSLT.
3. Написання на мові програмування обробника XML-документу.

Без використання CSS або XSL XML-документ відображається як простий текст в більшості Веб-браузерів. Деякі браузери, такі як Internet Explorer, Mozilla і Mozilla Firefox відображають структуру документа у вигляді дерева, дозволяючи згортати і розгортати вузли за допомогою натискань клавіші миші.

Об'єктно-орієнтована скриптова мова програмування **JavaScript** (спочатку названа LiveScript його творцем, Бренданом Ваше, і розгорнута у складі браузера Netscape Navigator) була вперше представлена публіці в 1995 році.

JavaScript зазвичай використовується як вбудована мова для програмного доступу до

об'єктів додатків. Найбільш широке застосування знаходить в браузерах як мова сценаріїв для додання інтерактивності веб-сторінкам.

Основні архітектурні риси: динамічна типізація, слабка типізація, автоматичне керування пам'яттю, Прототипне програмування, функції як об'єкти першого класу.

На JavaScript вплинули багато мов, при розробці була мета зробити мову схожу на Java, але при цьому легку для використання непрограмістів.

JavaScript має низку властивостей об'єктно-орієнтованої мови, але реалізоване в мові прототипування обумовлює відмінності в роботі з об'єктами в порівнянні з традиційними об'єктно-орієнтованими мовами. Крім того, JavaScript має ряд властивостей, властивих функціональним мовам, - функції як об'єкти першого класу, об'єкти як списки, анонімні функції, замикання - що додає мові додаткову гнучкість.

Структурно JavaScript можна представити у вигляді об'єднання трьох чітко помітних один від одного частин: ядро (ECMAScript); об'єктна модель браузера (Browser Object Model або BOM); об'єктна модель документа (Document Object Model або DOM).

Якщо розглядати JavaScript у відмінних від браузера оточеннях, то об'єктна модель браузера і об'єктна модель документа можуть не підтримуватися. Об'єктну модель документа іноді розглядають як окрему від JavaScript сутність, що узгоджується з визначенням DOM як незалежної від мови інтерфейса документу.

Для додавання JavaScript-коду на сторінку, можна використовувати теги <script> </ script>.

Область застосування JavaScript дуже широка:

- в клієнтської частини веб-додатків; в AJAX; в технології Comet; в браузерних операційних системах; для створення невеликих програм, що розміщуються в закладки браузера;
- додатки, написані на JavaScript, можуть виконуватися на серверах, що використовують Java 6 і пізніших версій, що використовується для побудови серверних додатків, що дозволяють обробляти JavaScript на стороні сервера;
- в якості мови розробки мобільних додатків (на платформі Mojo SDK в Palm webOS); • для реалізації віджетів, так і для реалізації движків віджетів (Apple_Dashboard, Microsoft Gadgets, Google Desktop Gadgets, Klipfolio Dashboard);
- для написання прикладного ПО (57% вихідного коду Mozilla Firefox написано на JavaScript);
- в якості скриптового мови доступу до об'єктів додатків; в офісних додатках для автоматизації рутинних дій, написання макросів, організації доступу з боку веб-служб; в Excel Services 2010 додалися два нових інтерфейси програмування додатків: REST API і JavaScript

Object Model (JSOM).

Для забезпечення високого рівня абстракції і досягнення прийнятної ступеня крос-браузерності при розробці веб-додатків використовуються бібліотеки JavaScript. Вони являють собою набір багаторазово використовуваних об'єктів і функцій. Серед відомих JavaScript бібліотек можна відзначити Adobe life, Dojo Toolkit, Extjs, jQuery, Mootools, Prototype, Qooxdoo.

На сьогоднішній день підтримку JavaScript забезпечують сучасні версії найпопулярніших браузерів.

AJAX (Asynchronous Javascript and XML - «асинхронний JavaScript і XML») - підхід до побудови інтерактивних користувацьких інтерфейсів веб-додатків, що полягає в «фоновому» обміні даними браузера з веб-сервером. В результаті, при оновленні даних веб-сторінка не перезавантажується повністю і веб-додатки стають швидшими і зручнішими.

Вперше термін AJAX був публічно використаний в 2005 році в статті Джесі Джеймса Гарретта (Jesse James Garrett) «Новий підхід до веб-додатків». Гарретт придумав термін, коли йому довелося якось назвати новий набір технологій, запропонований ним клієнту.

AJAX - не самостійна технологія, а концепція використання декількох суміжних технологій. AJAX базується на двох основних принципах: використання технології динамічного звернення до сервера «на льоту», без перезавантаження всієї сторінки повністю, наприклад: з використанням XMLHttpRequest (основний об'єкт) або через динамічне створіння дочірніх фреймів або через динамічне створіння тега <script>; використання DHTML для динамічної зміни змісту сторінки;

Як формат передачі даних зазвичай використовуються JSON або XML.

До переваг AJAX можна віднести:

- використання AJAX дозволяє значно скоротити трафік при роботі з веб-додатком;
- AJAX дозволяє дещо знизити навантаження на сервер;
- прискорення реакції інтерфейсу.

До недоліків AJAX можна віднести:

- відсутність інтеграції зі стандартними інструментами браузера;
- динамічно завантажувати вміст зазвичай недоступно пошуковикам;
- старі методи обліку статистики сайтів стають неактуальними.

PHP (PHP: Hypertext Preprocessor - «PHP: препроцесор гіпертексту», Personal Home Page Tools - «Інструменти для створення персональних веб-сторінок») - скриптова мова програмування загального призначення, інтенсивно застосовується для розробки веб-додатків. В даний час підтримується переважною більшістю хостинг-провайдерів і є одним з лідерів серед мов програмування, що застосовуються для створення динамічних веб-сайтів.

Мова і його інтерпретатор розробляються групою ентузіастів в рамках проекту з відкритим кодом. Проект не є вільним і розповсюджується під власною ліцензією.

В області програмування для Інтернету PHP - одна з найпопулярніших скриптових мов завдяки своїй простоті, швидкості виконання, багатій функціональності, багатоплатформеності і розповсюдженню початкових кодів на основі ліцензії PHP. Популярність в області побудови веб-сайтів визначається наявністю великого набору вбудованих засобів для розробки веб-додатків. До найбільших сайтів, які використовують PHP, відносяться Facebook (який, однак, використовує транслятор коду HipHop з PHP на C ++ з метою оптимізації), ВКонтакте, Wikipedia.

На даний момент існує єдина реалізація PHP, жодна стороння компанія не підтримує модулів, відмінних від офіційної збірки. Такий стан речей, з одного боку, дозволяє швидко впроваджувати і поширювати нововведення серед спільноти розробників, з іншого боку, розробляти мову програмування в умовах відсутності стандарту, так як єдина реалізація забезпечує його по факту. В таких умовах великого значення набуває версія інтерпретатора, що визначає поточну функціональність (зворотна сумісність між версіями інтерпретатора не дотримується строго).

Perl - високорівнева динамічний мова програмування загального призначення, створений в 1987 р Ларрі Уоллом, лінгвістом за освітою. Назва мови являє собою абревіатуру, яка розшифровується як Practical Extraction and Report Language «практична мова для вилучення даних і складання звітів».

Згідно Ларрі Уолл, Perl має два девізи. Перший - «There's more than one way to do it» («Є більше ніж один шлях зробити це», також відомий як TMTOWTDI); другий - «Easy things should be easy and hard things should be possible» («Прості речі повинні бути простими, а складні речі - можливими»).

Основною особливістю мови вважаються його багаті можливості для роботи з текстом, у тому числі реалізовані за допомогою регулярних виразів. Перл успадкував багато властивостей від мов C, shell script, awk.

ASP.NET - технологія створення веб-додатків і веб-сервісів від компанії Microsoft. Вона є складовою частиною платформи Microsoft .NET і розвитком старішої технології Microsoft ASP.

Хоча ASP.NET бере свою назву від старої технології Microsoft ASP, вона значно від неї відрізняється. Microsoft повністю перебудувала ASP.NET, грунтуючись на Common Language Runtime (CLR), який є основою всіх додатків Microsoft .NET. Розробники можуть писати код для ASP.NET, використовуючи практично будь-які мови програмування, в тому числі, і ті, які входять у комплект .NET Framework (C #, Visual Basic.NET, і JScript .NET). ASP.NET має

перевагу в швидкості в порівнянні зі скриптовими технологіями, так як при першому зверненні код компілюється і поміщається в спеціальний кеш, і згодом виконується, не вимагаючи витрат часу на парсінг, оптимізацію, і т. Д.

Разом з тим слід враховувати, що зазначена перевага не завжди може бути реалізована Це пов'язано з тим, що на швидкість роботи реального проекту впливають безліч чинників. В першу чергу це кваліфікація керівників розробки і виконавців: повільні алгоритми легко зводять нанівець незначну перевагу скомпільованого коду перед інтерпретацією серверних скриптів.

Переваги ASP.NET перед ASP:

- компільований код виконується швидше, більшість помилок знаходиться ще на стадії розробки;
- значно поліпшена обробка помилок часу виконання, з використанням блоків try..catch;
- елементи управління (controls) дозволяють виділяти часто використовувані шаблони, такі як меню сайту;
- використання компонентів, які вже застосовуються в Windows-додатках, наприклад, таких як елементи управління і події;
- широкий набір елементів управління і бібліотек класів дозволяє швидше розробляти програми;
- ASP.NET спирається на багатомовні можливості .NET, що дозволяє писати код сторінок на VB.NET, Visual C #, J # і т. д .;
- можливість кешування всієї сторінки або її частини для збільшення продуктивності;
- можливість кешування даних, що використовуються на сторінці;
- можливість поділу візуальної частини та бізнес-логіки по різних файлах («code behind»);
- розширювана модель обробки запитів;
- розширена модель подій;
- розширювана модель серверних елементів управління;
- наявність master-сторінок для завдання шаблонів оформлення сторінок;
- підтримка CRUD операцій при роботі з таблицями через GridView;
- вбудована підтримка AJAX.

Контрольні питання

1. Що таке протокол?
2. Які протоколи Ви знаєте?
3. Структура протоколу http.
4. Які послуги Інтернет Ви знаєте?
5. Яка структура протоколу TCP/IP?
6. Які поштові протоколи Ви знаєте?
7. Яке призначення FTP?

РОЗДІЛ 2
ОРГАНІЗАЦІЯ WEB-ПРОЕКТІВ

2.1. Доменне ім’я

Доменна система імен (англ. Domain Name System, DNS) — розподілена система перетворення імені хоста (комп'ютера або іншого мережевого пристрою) в IP-адресу. Кожен комп'ютер в Інтернеті має свою власну унікальну адресу — число, яке складається з чотирьох байтів. Уся система імен в Інтернеті — ієрархічна. Це зроблено для того, щоб не підтримувати одне централізоване джерело.

Повне доменне (від англ. domain) ім'я машини (FQDN, Fully Qualified Domain Name) можна розбити на дві частини — ім'я області-домена та власне ім'я машини. Наприклад, m30.ziet.zhitomir.ua — повне доменне ім'я машини m30, яка перебуває у домені ziet.zhitomir.ua.

За порядок у доменах, як правило, відповідає певний комп'ютер, користувачі-адміністратори якого слідкують за тим, щоб не було, наприклад, різних машин з однаковими IP-адресами. Наприклад, відповідальність за область-домен ziet.zhitomir.ua покладається на машину alpha.ziet.zhitomir.ua Ця влада делегується зверху вниз від машини ns.lucky.net, яка відповідає за домен zhitomir.ua. В свою чергу, відповідальність за область ua делегована машині зверху від так званих кореневих серверів (root server). Всю цю систему можна уявити у вигляді перевернутого дерева.

Необхідно розрізняти доменне ім'я, та поштову адресу. В поштовій адресі повинен бути знак «@», який розділяє поштову адресу на доменне ім'я та ім'я поштової скриньки. Після того, які кількість машин значно збільшилася, така схема перестала ефективно працювати і програмісти університету штату Каліфорнія в Берклі спроектували і написали програму BIND (Berkeley Internet Name Domain), яка відповідає на запити машин користувачів, які стосувалися імен та IP-адресу.

Служба імен DNS (Domain Name System) — це розподілена база даних доволі простої структури. Для початкового знайомства можна вважати, що це кілька таблиць, у яких записано:

яку IP-адресу має машина з певним іменем;

яке ім'я має машина з визначеною адресою;

що це за комп'ютер і яка операційна система встановлена на ньому;

куди потрібно направляти електронну пошту для користувачів цієї машини;

які псевдоніми є у даної машини.

Для прикладу розглянемо випадок, коли користувач посилає пошту з машини

polesye.zhitomir.ua користувачу за адресою rozhik@ziet.zhitomir.ua (знак «@» носить назву commercial «at» sign). При встановленні на машину протоколів TCP/IP системний адміністратор вказує IP-адресу комп'ютера — найближчого серверу імен. Поштова програма подає цьому найближчому серверу запит: «Куди посилати пошту для ziet.zhitomir.ua» Якщо найближчий сервер не може відповісти, то він, в свою чергу, посилає запит до більш «старшого» серверу. Нарешті, стає зрозумілим, що всю пошту для області ziet.zhitomir.ua необхідно відправляти на машину alpha.ziet.zhitomir.ua або relay2.lucky.net. Разом з цим відповіді містять ще адресу цієї машини. Поштова програма зв'язується з цим комп'ютером (використовуючи не ім'я, а адресу) та передає йому пошту.

Всі ці переговори та відправка пошти, як правило, відбувається протягом кількох секунд і користувач не помічає цього. Якщо машина ziet.zhitomir.ua недоступна то тоді пошта на час, в якій неможливо зв'язатися з машиною ziet.zhitomir.ua (наприклад під час профілактики каналу зв'язку) чекає своєї черги на пересилку на машині relay2.lucky.net. Це характерна для Internet-програм поведінка. У більшості випадків у програмах користувачів намагаються дізнатися лише одне — яка IP-адреса у машини з відповідним іменем.

Існує три основні типи серверів DNS, які відрізняються покладеними на них завданнями: основний сервер DNS; резервний (вторинний) сервер DNS; кешуючий сервер DNS. Основний сервер DNS управляє зоною повноважень. Якщо потрібно додати/видалити домен або вузол або якось інакше модифікувати зону, зміни потрібно проводити на основному сервері DNS. Через певний час, який залежить від настройок сервера, основний сервер передасть зону резервному серверу DNS. Дане явище називається трансфером зони. Що ж до резервних серверів, то повинен бути хоч би один резервний сервер DNS. Тому є декілька причин: якщо клієнтів багато, то наявність резервного сервера DNS дозволить знизити навантаження на основний сервер DNS і прискорити доступ фізично віддалених від основного сервера клієнтів до бази даних доменних імен.

DNS володіє наступними характеристиками:

Розподільність адміністрування. Відповідальність за різні частини ієрархічної структури несуть різні люди або організації. Розподільність зберігання інформації. Кожен вузол мережі в обов'язковому порядку повинен зберігати тільки ті дані, які входять в його зону відповідальності та (можливо) адреси кореневих DNS-серверів.

Кешування інформації. Вузол може зберігати деяку кількість даних не зі своєї зони відповідальності, для зменшення навантаження на мережу. Ієрархічна структура, в якій всі вузли об'єднані в дерево, і кожен вузол може або самостійно визначає роботу нижчестоящих вузлів, або делегувати (передавати) їх іншим вузлам.

Резервування. За зберігання та обслуговування своїх вузлів (зон) відповідають (зазвичай)

кілька серверів, розділені як фізично, так і логічно, що забезпечує збереження даних і продовження роботи навіть у разі збою одного з вузлів. DNS важлива для роботи Інтернету, так як для з'єднання з вузлом необхідна інформація про його IP-адресу, а для людей простіше запам'ятовувати буквені (зазвичай осмислені) адреси, ніж послідовність цифр IP-адреси. У деяких випадках це дозволяє використовувати віртуальні сервери, наприклад, HTTP-сервери, розрізняючи їх по імені запиту. Спочатку перетворення між доменними та IP-адресами вироблялося з використанням спеціального текстового файлу hosts, який складався централізовано й автоматично розсилався на кожну з машин у своїй локальній мережі. З ростом Мережі виникла необхідність в ефективному, автоматизованому механізмі, яким і стала DNS. DNS була розроблена Полом Мокапетрісом в 1983 році.

Ключовими поняттями DNS є: Домен (англ.domain - область) - частина простору ієрархічних імен мережі Інтернет, що обслуговується групою серверів доменних імен (DNS-серверів) та централізовано адмініструється. DNS-сервери зберігають інформацію про вузли, імена яких належать домену і виконують трансляцію їх імен в адреси. Кожний домен має унікальне ім'я, а кожен комп'ютер, підключений до Інтернету, має, як правило, доменне ім'я. Домени мають між собою ієрархічні відношення. Два домени, що розташовані на сусідніх рівнях ієрархії, називаються відповідно доменом вищого та нижчого рівнів. Домени найвищого (верхнього) рівня можуть бути сформовані за організаційним або географічним ознаками. Домени, сформовані за географічним ознаками, об'єднують вузли, що належать конкретній державі. За географічними ознаками об'єднуються в основному комп'ютери, що містяться на території США.

Піддомен (англ. subdomain) — підлеглий домен (наприклад, wikipedia.org — піддомен домену org, а uk.wikipedia.org — домену wikipedia.org). Теоретично такий розподіл може досягати глибини в 127 рівнів, а кожна мітка може містити до 63 символів, поки загальна довжина разом з крапками не досягне 254 символів. Але на практиці реєстратори доменних імен використовують більш суворі обмеження. Наприклад, якщо у вас є домен виду mydomain.ua, ви можете створити для нього різні піддомени виду mysite1.mydomain.ua, mysite2.mydomain.ua і т. д.

Ресурсний запис — одиниця зберігання і передачі інформації в DNS. Кожний ресурсний запис має ім'я (тобто прив'язаний до певного доменного імені, вузлу в дереві імен), тип і поле даних, формат і зміст якого залежить від типу. Зона — частина дерева доменних імен (включаючи ресурсні записи), що розміщується як єдине ціле на деякому сервері доменних імен (DNS-сервері, див. нижче), а частіше — одночасно на декількох серверах (див. нижче). Метою виділення частини дерева в окрему зону є передача відповідальності (див. нижче) за відповідний домен іншій особі або організації. Це називається делегуванням (див. нижче). Як

зв'язкова частина дерева, зона всередині теж являє собою дерево. Якщо розглядати простір імен DNS як структуру із зон, а не окремих вузлів/імен, теж виходить дерево; виправдано говорити про батьківських і дочірніх зонах, про старших і підлеглих. На практиці, більшість зон 0-го і 1-го рівня ('.', ua, com, …) складаються з єдиного вузла, якому безпосередньо підпорядковуються дочірні зони. У великих корпоративних доменах (2-го і більше рівнів) іноді зустрічається утворення додаткових підпорядкованих рівнів без виділення їх у дочірні зони. Делегування — операція передачі відповідальності за частину дерева доменних імен іншій особі або організації. За рахунок делегування в DNS забезпечується розподільність, адміністрування та зберігання. Технічно делегування виражається у виділенні цієї частини дерева в окрему зону, і розміщенні цієї зони на DNS-сервері (див. нижче), керованому цією особою чи організацією. При цьому в батьківську зону включаються «склеюючі» ресурсні записи (NS і A), що містять покажчики на DNS-сервера дочірньої зони, а вся інша інформація, що відноситься до дочірньої зоні, зберігається вже на DNS-серверах дочірньої зони. DNS-сервер — програма, призначена для відповідей на DNS-запити за відповідним протоколом. Також DNS-сервером можуть називати хост, на якому запущено відповідну програму.

DNS-клієнт (від англ. Domain Name System-client - доменних імен система - клієнт) - програма або модуль в програмі, що забезпечує з'єднання із DNS-сервером для визначення IP-адреси по його доменному імені. Авторитетність (англ. authoritative) — ознака розміщення зони на DNS-сервері. Відповіді DNS-сервера можуть бути двох типів: авторитетні (коли сервер заявляє, що сам відповідає за зону) і неавторитетні (англ. Non-authoritative), коли сервер обробляє запит, і повертає відповідь інших серверів. У деяких випадках замість передачі запиту далі DNS-сервер може повернути вже відоме йому (за запитами раніше) значення (режим кешування). DNS-запит (англ. DNS query) — запит від клієнта (або сервера) сервера. Запит може бути рекурсивним або нерекурсивний. Система DNS містить ієрархію DNS-серверів, відповідну ієрархії зон. Кожна зона підтримується як мінімум одним авторитетним сервером DNS (від англ. Authoritative — авторитетний), на якому розташована інформація про домен. Ім'я та IP-адресу не тотожні — одна IP-адреса може мати безліч імен, що дозволяє підтримувати на одному комп'ютері безліч веб-сайтів (це називається віртуальний хостинг). Зворотне теж справедливо — одному імені може бути зіставлено безліч IP-адрес: це дозволяє створювати балансування навантаження. Для підвищення стійкості системи використовується безліч серверів, що містять ідентичну інформацію, а в протоколі є засоби, що дозволяють підтримувати синхронність інформації, розташованої на різних серверах. Існує 13 кореневих серверів, їх адреси практично не змінюються. Протокол DNS використовує для роботи TCP-або UDP-порт 53 для відповідей на запити. Традиційно запити та відповіді відправляються у вигляді однієї UDP дейтаграми. TCP використовується для

AXFR-запитів.

Система імен DNS - це ієрархічна деревоподібна система. У цьому дереві існує корінь - він позначається «.» (root). Список кореневих серверів повинен бути у кожного сервера: він міститься у файлі named.ca. Цей файл може називається і по-іншому - залежно від налаштувань сервера. Існує певна кількість доменів верхнього рівня. Найбільш відомі: com, gov, net, org та інші (у тому числі і домени країн - ru, ua, fr та ін.). Нехай користувач вводить у вікні браузера адрес http :// server . Проте адресація в локальній мережі (так само як і в Інтернет) побудована на основі IP-протоколу. Тому для того, щоб встановити з'єднання з комп'ютером server комп'ютеру користувача необхідно знати його IP-адресу, тому операційна система користувача намагається вирішити (перевести) ім'я комп'ютера в IP-адресу. З цією метою вона спочатку використовує свої стандартні засоби (той же файл hosts), а потім звертається до служби DNS. Розглянемо тепер інтернет-адресу www.yahoo.com (насправді абсолютно неважливо це інтернет-адреса або адреса в локальній мережі - все те ж саме). Сервер DNS спочатку намагається вирішити ім'я даного комп'ютера, використовуючи свій власний кеш імен. Якщо необхідне ім'я комп'ютера в нім відсутнє, то сервер DNS звертається до одного з кореневих серверів DNS. Запит обробляється рекурсивно: кореневий сервер звертається до сервера, який відповідає за домен com, а той, у свою чергу, до сервера DNS домена yahoo.com. Сервер DNS домена yahoo.com повертає IP-адресу комп'ютера www - 64.58.76.222 або всі адреси, які зіставлені цьому імені (багато мережевих операційних систем дозволяють одному імені зіставляти декілька IP-адрес). А офіційне ім'я комп'ютера www.yahoo.com - www. yahoo . akadns. net Схеми запитів DNS-імен Нерекурсивна процедура:

1. DNS-клієнт звертається до кореневого DNS-сервера з вказівкою повного доменного імені.

2.DNS-сервер відповідає клієнту, вказуючи адресу наступного DNS-сервера, який виконує обслуговування домену верхнього рівня, заданого в наступній старшій частині імені.

3. DNS-клієнт виконує запит наступного DNS-сервера, який його надсилає до DNS-сервера потрібного піддомена і т.д., доти, доки не буде знайдено DNS-сервер, який повністю відповідає запитуваному імені IP-адреси.

Сервер дає кінцеву відповідь клієнту. Рекурсивна процедура:

1. DNS-клієнт запитує локальний DNS-сервер, який обслуговує піддомен, якому належить клієнт.

2.Далі, якщо локальний DNS-сервер відповідь знає, то повертає її клієнту, в протилежному випадку виконує ітеративні запити до кореневого сервера до тих пір, поки не отримає відповідь. Після отримання відповіді сервер передає її клієнту.

Таким чином, при рекурсивній процедурі клієнт фактично передоручає роботу власному

серверу. Для прискорення пошуку IP-адрес DNS-сервери часто застосовують кешування (на час від годин до декількох днів) відповідей, які проходять через них. Записи DNS Записи DNS, або Ресурсні записи (англ. Resource Records, RR) - одиниці зберігання і передачі інформації в DNS. Кожний ресурсний запис складається з наступних полів: ім'я (NAME) - доменне ім'я, до якого прив'язана або яким «належить» даний ресурсний запис; TTL (Time To Live) - допустимий час зберігання даного ресурсного запису в кеші не відповідального DNS-сервера; тип (TYPE) ресурсного запису - визначає формат і призначення даного ресурсного запису; клас (CLASS) ресурсної записи; теоретично вважається, що DNS може використовуватися не тільки з TCP / IP, але і з іншими типами мереж, код в поле клас визначає тип мережі; довжина поля даних (RDLEN); поле даних (RDATA), формат та зміст якого залежить від типу запису.

Найбільш важливі типи DNS-записів:

Запис A(address record) або запис адреси зв'язує ім'я хоста з адресою IP. Наприклад, запит A-запису на ім'я referrals.icann.org поверне його IP адресу - 192.0.34.164;

Запис AAAA (IPv6 address record) зв'язує ім'я хоста з адресою протоколу IPv6. Наприклад, запит AAAA-запису на ім'я K.ROOT-SERVERS.NET поверне його IPv6 адресу - 2001:7 fd :: 1;

Запис CNAME (canonical name record) або канонічний запис імені (псевдонім) використовується для перенаправлення на інше ім'я;

Запис MX (mail exchange) або поштовий обмінник вказує сервер(и) обміну поштою для даного домену. Запис NS (name server) вказує на DNS-сервер для даного домену;

Запис PTR (pointer) або запис покажчика зв'язує IP хоста з його канонічним ім'ям. Запит в домені in-addr.arpa на IP хоста в reverse формі поверне ім'я (FQDN) даного хоста. Наприклад, (на момент написання), для IP адреси 192.0.34.164: запит запису PTR 164.34.0.192.in-addr.arpa поверне його канонічне ім'я referrals.icann.org. З метою зменшення обсягу небажаної кореспонденції (спаму) багато серверів-одержувачів електронної пошти можуть перевіряти наявність PTR запису для хоста, з якого відбувається відправлення. У цьому випадку PTR запис для IP адреси повинен відповідати імені відправляючого поштового сервера, яким він представляється в процесі SMTP-сесії.

Запис SOA (Start of Authority) або початковий запис зони вказує, на якому сервері зберігається еталонна інформація про даний домен, містить контактну інформацію особи, відповідальної за дану зону, таймінги (параметри часу) кешування зонної інформації та взаємодію DNS-серверів. SRV-запис (server selection) вказує на сервери для сервісів, використовується зокрема для Jabber і Active Directory.

2.2. Вибір доменного імені

Після створення сайту постає питання реєстрації домену, або доменного імені. В Інтернеті адреси прийнято поділяти на домени за тематичною (організаційною) чи географічною (національною) належністю. Географічні зони можуть поділяться на організаційні (com.ua, gov.ua, org.ua тощо).

Перш ніж зареєструвати домен необхідно визначиться, в якій доменній зоні реєструватися. Якщо компанія орієнтована на український ринок, то необхідно реєструватися в одній з українських організаційних зон (.com.ua, biz.ua, info.ua) або в одній з українських обласних зон (kiev.ua, lviv.ua, zp.ua). Кожен обласний центр має свою доменну зону.

Домени першого рівня у свою чергу поділяються на домени другого рівня (наприклад, в адресі www.company.kiev.ua - .ua є зоною першого рівня, яка вказує, що компанія знаходиться в Україні, .kiev - це зона другого рівня, яка вказує, що це київська кампанія).

Компанія може зареєструвати домен як в обласній, так і в організаційній українській доменній зоні. Якщо в компанії немає філій в інших областях України, то можна зареєструвати домен в обласній зоні, тоді у користувачів компанія асоціюватиметься з містом в якому знаходиться компанія, якщо компанія має філії, то краще реєструватися в українській організаційній зоні, тоді користувачі будуть сприймати компанію як всеукраїнську. Якщо компанія має зареєстровану торгову марку, тоді можна зареєструвати домен «торговамарка.ua».

Як правило, українські компанії реєструють одне доменне ім'я в певній українській доменній зоні, хоча адрес в одного сайту може бути множина. Якщо для сайту потрібно зареєструвати кілька доменних адрес (company.ua, company.com.ua, company.kiev.ua, company.com), то варто одне з доменних імен вибрати як основне. Воно буде використовуватися в логотипі, рекламі, візитках, бланках тощо, решта доменів буде скеровуватися на «основний» домен.

Правила вибору доменного імені:

1. Доменне ім'я повинне бути коротким.
2. Доменне ім'я повинно запам'ятовуватися.
3. Доменне ім'я не повинне допускати різночитань.
4. Доменне ім'я може складатися з букв латинського алфавіту, цифр і символу «-»(дефіс).

Скорочення (абревіатури) є прийнятними для тих випадків, коли назва сайту складається з трьох і більше слів або коли підсумкова назва, що утворюється при сполученні всіх слів назви, виявляється дуже довгою або важко вимовною.

Доменне ім'я не повинне допускати різночитань.

Доменна адреса може бути транскрипцією латинськими буквами кириличних назв і тут можуть бути неприємні сюрпризи. Наприклад, компанія «Киев Пассажирский» в транскрипції може виглядати як KievPassazhirsky і як KievPassazhirskiy, а в транскрипції з української як KyivPasazhirsky. Деякі користувачі можуть помилитися і писати як з двома «s» так і з однією, тому для такої компанії краще підібрати щось просте, наприклад, www.kpas.ua.

Доменне ім'я може складатися з букв латинського алфавіту, цифр і символу "-" (дефіс), починатися і закінчуватися буквою латинського алфавіту або цифрою. Довжина імені кожного домену (між розділовими точками) не може перевищувати 63 знаки. Мінімальна довжина доменного імені (без врахування розділових точок і назви домену) прийнята рівною 2 (для зон .biz і .info - 3 символи).

Поради з вибору і використання доменних імен:

1. Реєструйте кілька доменів
2. Використовуйте ключові слова в назві домену.
3. Одночасно реєструйте домен в національній та організаційній зонах (kiev.ua, com.ua).
4. Використовуйте «чужі» географічні зони (.tv, .co, .cc, .to).
5. Використовуйте міжнародні домени (.com, .biz, .info).
6. Використовуйте назву поєднання організаційних або географічних зон і назву домену.
7. В назві домену краще уникати дефіси.
8. Не використовуйте як основний доменне ім'я, що схоже на домен конкурента.
9. Реєструйте домени, що звільняються.

Якщо назва компанії українською та російською мовою пишеться по-різному, то варто зареєструвати кілька доменів в різних варіантах написання. Таким чином, будь-яке написання назви компанії приведе користувача на сайт.

За багатьма дослідженнями деякі користувачі вгадують адресу, тобто беруть назву того, що хочуть знайти і додають .com, відповідно українські користувачі додають .com.ua або .kiev.ua. Тому, якщо компанія продає монітори, то окрім домену company.com.ua, бажано зареєструвати домен monitor.com.ua чи monitor.kiev.ua. Також, не варто забувати про назви доменів в множині, наприклад, monitors.com.ua чи monitors.kiev.ua.

Крім того, в пошукових системах за відповідними ключовими словами сайт буде вищим в результатах пошуку, оскільки в назві домену присутнє ключове слово.

Як правило, визначившись з назвою домену багато київських компаній, реєструють домен в зоні .com.ua, або .kiev.ua, хоча, слід реєструвати доменне ім'я в обох зонах, оскільки

компанія конкурент може зареєструвати друге доменне ім'я і скерувати його на свій сайт. Користувачі часто плутають .com.ua і .kiev.ua (так вже історично склалося, що Київ став флагманом розвитку Інтернет в Україні, тому зони .com.ua і .kiev.ua для українського користувача стали рівнозначними) тому реєструйте домен в обох зонах, тоді навіть допустившись помилки користувач все одно потрапить на сайт.

1. Не реєструйте домен, якщо він зайнятий хоча б в одній із зон .com.ua або .kiev.ua, компанією, яка займається тим же бізнесом.

2. Якщо компанія конкурент зареєструвала домен лише в одній із зон .com.ua або .kiev.ua, то необхідно зареєструвати цей домен в іншій зоні і скерувати його на свій основний домен, тоді користувачі помилково зайдуть на ваш сайт і можливо знайдуть там те, що шукали.

3. Якщо кампанія є власником торгової марки, то слід зареєструвати домени в трьох зонах: .ua, .com.ua і .kiev.ua.

4. Якщо в компанії є філії в обласних центрах, то слід зареєструвати домени і у відповідних українських обласних доменних зонах.

Домен .ag (держава Антигуа) як альтернатива домену .com може особливо зацікавити акціонерні компанії орієнтовані на німецький ринок. В німецькій мові скорочення Ag позначає «акціонерне суспільство» (Die Aktiengesellschaft) і широко використовується як позначення типу компанії, подібно до англійських скорочень Inc. і Corp.

Компанія VeriSign купила права на управління доменною зоною .tv (маленька острівна держава Тувалу) і тепер адмініструє домен .tv - телевізійним, і тепер реєстрація домену в цій зоні вельми приваблива для телекомпаній і сайтів, що мають телевізійну скерованість.

Національний домен Колумбії .co добре асоціюється із словами company та commercial і може, таким чином, стати альтернативою для переповненої зони .com.

Слід пам'ятати, що користувачі розділяють сайти за географічною і організаційною ознакою, тому, перш ніж зареєструвати домен в «чужій» доменній зоні - необхідно все ретельно зважити. Реєстрація в «чужій» доменній зоні буде виправданою, якщо отримана адреса добре запам'ятовується і асоціюється з родом діяльності компанії.

У поєднанні із закінченням (організаційним доменом) назва домену може дати несподівано добрий результат. Наприклад, kuvyr.com - така адреса дуже добре запам'ятовується, оскільки домен kuvyr спільно із зоною .com читається як «кувырком». Також добре запам'ятовуються словосполучення daleko.edu (далеко еду), доменне ім'я daleko само по собі добре запам'ятовується, але спільно із закінченням .edu (освітня доменна зона) - запам'ятовується ще краще. Такий домен міг би підійти як основний для туристичної фірми, хоча в даному випадку потрібно буде переконати реєстратора, що домен реєструється для освітнього сайту, оскільки .edu це організаційна зона для освітніх сайтів.

Якщо компанія працює на міжнародному ринку, то не слід обмежуватися лише українським доменом - зареєструйте домен в зоні .com, .biz або .info. Скеровуйте міжнародні домени на англійську версію сайту, а українські на українську (або російську - якщо немає української версії). Відповідно для роботи на міжнародному ринку використовуйте міжнародний домен (у візитках, рекламі, бланках), а на внутрішньому ринку український домен. Тоді, іноземні користувачі відразу потраплять на англійську версію сайту, а українські відповідно на українську версію.

Дефіси рідко зустрічаються, тому, краще обмежити їх використання в назвах доменів, хоча цього не завжди можна уникнути, оскільки в зоні .com всі благозвучні домени вже зайняті.

Користувачі набирають ту адреса, яка краще запам'ятовується, і можуть потрапити на сайт конкурента. Виключенням можуть бути випадки, коли назви компаній конкурентів є схожими. Тоді, слід подумати про використання в назві основного домену ключового слова.

Багато доменів в пошукових системах вже мають певний авторитет і тому, покупка такого домену, від якого з певних причин відмовився колишній власник, може бути корисною для просування вашого основного сайту.

Перевірити чи є вільним обраний домен можна на сайтах:

1. whois.com.ua (для перевірки доменів в українських обласних зонах, в зоні .ua, .com.ua, та інших українських доменних зонах).
2. www.networksolutions.com або www.register.com (для перевірки в міжнародних зонах)
3. Скористатися перевіркою на сайті любого українського хостинг-провайдера.

Підтримку більшості доменів необхідно оплачувати щорічно. За рік (сплачений термін обчислюється з моменту реєстрації домену) необхідно оплатити підтримку наступного року, у разі несплати домен відключається і може бути зареєстрований іншим користувачем.

2.3. Хостинг

Хо́стинг (англ. hosting) — послуга, що надає дисковий простір для розміщення фізичної інформації на сервері, що постійно перебуває в мережі Інтернет.

Зазвичай під поняттям послуги хостингу мають на увазі, як мінімум, послугу розміщення файлів сайту на сервері, на якому запущене ПЗ, необхідне для обробки запитів до цих файлів (веб-сервер). Як правило, до послуг хостингу вже входить надання місця для поштової кореспонденції, баз даних, DNS файлового сховища тощо, а також підтримка функціонування відповідних сервісів, однак вони можуть надаватися і окремо. Розрізняють безкоштовний та

платний хостинг. Безкоштовні «хостери» заробляють на тому, що розміщують рекламу на своїх сайтах.

Повнофункціональний хостинг:

1. Віртуальний хостинг — надається місце на диску для розміщення веб-сайтів, середовище виконання веб-сервісів єдине для багатьох користувачів, ресурси розподілені між усіма користувачами на одному сервері, де може розміщуватись від 50 до 1000 користувачів.

2. Віртуальний виділений сервер (VPS або VDS) — послуга, в рамках якої користувачеві надається так званий віртуальний виділений сервер. У плані управління операційною системою здебільшого вона відповідає фізичному виділеному серверу. Зокрема: права адміністратора, root-доступ, власні IP-адреси, порти, правила фільтрування і таблиці маршрутизації.

3. Виділений сервер — надається сервер цілком. Використовується для реалізації нестандартних завдань (сервісів), а також розміщення «важких» веб-проектів, які не можуть спів-існувати на одному сервері з іншими проектами і вимагають під себе всі ресурси сервера.

Віртуальний хостинг-це одна з найбільш поширених послуг. Цей вид хостингу є найдешевшим і доступним широкому колу людей. Сенс віртуального хостингу полягає в одночасному використанні різними людьми загальних ресурсів сервера або частини сервера, а саме: дискового простору, пам'яті, бази даних та процесорного часу. За цією схемою кожному користувачеві на сервері виділяються певна дискова квота (залежно від тарифного плану) і певні ресурси сервера, які він може використовувати.

Тобто якщо взяти за приклад абстрактний комп'ютер і розділити жорсткий диск ПК на сотню розділів, віддавши їх окремо взятим користувачам, то в підсумку ви отримаєте той самий віртуальний хостинг з сотнею клієнтів. всі користувачі вашого комп'ютера будуть використовувати різні розділи, але всі люди будуть користуватися одними системними ресурсами ПК. У цьому випадку ступінь завантаженості, або ступінь використання системних ресурсів комп'ютера одним користувачем може відбиватися і на інших. Щоб цього не відбувалося, провайдери використовують систему обмежень ресурсів для кожного клієнта, але все одно всі клієнти, розташовані на одному Web сервері, залежать один від одного. Якщо один проект буде завантажувати системні ресурси сервера і провайдер не встигне відреагувати, то ваш проект, розташований на тому ж сервері, матиме більший час відгуку завантаження сторінок або навіть недоступний в певні проміжки часу.

Другий за рангом вид послуг - це віртуальний виділений сервер. Для позначення цього виду послуг призначена абревіатура VPS (Virtual Private Server) або VDS (Virtual Dedicated Server). Цей вид хостингу набагато цікавіше і практично не має проблем віртуального хостингу. Ідея Віртального виділеного сервера полягає в наступному. Усередині великого і

потужного серверу створюється кілька віртуальних серверів (фактично розділів), де кожен із серверів забезпечений своєю операційною системою, супутнім програмним забезпеченням і фіксованими системними ресурсами. Завдяки цьому цей вид хостингу і називається віртуальним виділеним сервером, оскільки є виділеної частиною самого сервера. Користувач віртуального виділеного сервера можете розподіляти ресурси системи для кожного конкретно взятого сайту, яких може бути скільки завгодно (в межах дискової квоти). Більше того, можна створювати власних користувачів і виділяти цим користувачам певні ресурси системи (диск, пам'ять, процесорний час, бази даних і т. д.). Що стосується фіксованих ресурсів, то вони дійсно фіксовані, але якщо вам виділено, скажімо, 512 Мбайт пам'яті і процесор 1200 ГГц, то ви гарантовано отримаєте ці ресурси. У випадку з віртуальним хостингом ці параметри будуть ділитися між усіма клієнтами одного майданчика.

Виділений сервер - цей вид послуг використовується більшою мірою великими організаціями або дуже великими проектами (наприклад, від 10000- ... унікальних відвідувачів на добу, що неможливо реалізувати в рамках віртуального виділеного сервера. Виділений сервер, на відміну від віртуального виділеного сервера, - це і є той самий великий і потужний сервер, який цілком і повністю буде належати вам. Провайдер послуг просто здасть вам цей сервер в оренду. Що ви будете робити з цим сервером, яке програмне забезпечення встановлювати - це вже ваша особиста справа. Крім виділеного сервера, ви можете орендувати у провайдера місце під свій сервер.

Крім перерахованих вище видів послуг, варто обов'язково згадати про реселлерів. Це людина, підприємець чи організація, які орендують у провайдера послуг або віртуальний виділений сервер, або виділений сервер і організовують свій хостинг зі своїми тарифами і послугами.

Контрольні питання

1. Які домени першого рівня Ви знаєте?
2. Для чого потрібна IP адреса?
3. Які способи розміщення ссайту Ви знаєте?
4. Призначення хостингу?
5. Що таке VPS?

РОЗДІЛ 3
МОВА HTML

3.1. Поняття тегів і атрибутів

Найпростішим компонентом Web є HTML. Це проста мова форматування документів, що відображаються у Web-браузері. Найважливіше завдання браузера – це відображення документів у відповідності із HTML-тегами. Як і будь яка інша мова програмування, HTML передбачає деяку стандартну структуру побудови програми – у даному випадку html-документу. Така структура описує навіть не послідовність команд, а почерговість слідування ряду обов’язкових блоків, які містять безпосередньо програмний код. На відміну від мов програмування, директиви HTML мають власне найменування – „теги”. Загальний синтаксис тегів виглядає так: <ТЕГ>. Усі об’єкти, які не знаходяться між символами “<” та „>”, інтерпретатор сприймає як текст, відображуючи їх на екрані „як вони є”.

У HTML є ще одна значна особливість, що відрізняє його від інших мов програмування: практично усі теги даної мови (за виключенням деяких) – парні, тобто є „відкриваючий” і „закриваючий” теги. Синтаксис останнього: </ТЕГ>. Все, що розташовано між „відкриваючим” і „закриваючим” тегом, обробляється інтерпретатором згідно з алгоритмом, який присвоєно даному конкретному тегу. В загальному вигляді рядок програми виглядає так: <ТЕГ>Значення, що обробляється</ТЕГ>. Дана властивість дозволяє використовувати принцип вложення одного тега в інший:

<ТЕГ1>

<ТЕГ2>Значення, що обробляється</ТЕГ2>

</ТЕГ1>

На Web-сторінці повинен бути тег, який вказує браузеру на те, що даний документ є документом HTML. Це теги <HTML> та </HTML>. Заголовок HTML містить інформацію про сам документ, а іноді і спеціальні директиви, що підказують браузеру правила, за якими слід обробляти код сторінки. Тег заголовку: <HEAD>. Тіло документу, що описується тегами <BODY> та </BODY>, включає в себе весь основний код розмітки сторінки, який і визначає відображення html-документу у клієнтській області вікна браузера.

Секція тіла Web-сторінки знаходиться усередині парного тега <BODY>. Вона описує сам вміст Web-сторінки, те, що буде виведено на екран.

А у парному тегу <HEAD> знаходиться секція заголовка Web-сторінки. У цю секцію поміщають відомості про параметри Web-сторінки, що не відображаються на екрані та призначені виключно для Web-браузера.

І заголовок, і тіло Web-сторінки знаходяться всередині парного тегу <HTML>, який розташований на найвищому (нульовому) рівні вкладеності та не має батька. Будь-яка Web-сторінка має бути правильно відформатована: мати секції заголовка та тіла та всі відповідні їм теги. Тільки в такому разі вона буде вважатися коректним з погляду стандартів HTML.

Найпростіший код сторінки можна уявити наступним чином:

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="UTF-8">
   <title>Web-сторінка</title>
</head>
<body bgcolor="blue">
<!--Це перша сторінка-->
&quot;Кафедра інформатики та прикладного програмного забезпечення &copy;&quot;
</body>
</html>
```

Тег <TITLE> визначає рядок, який буде відображений у заголовку вікна.

Таким чином, тег – це деяка команда HTML, яка вказує інтерпретатору браузера, яким чином він повинен обробляти відповідне кожній конкретній директиві значення. Багато тегів HTML розпізнають спеціальні ключові слова, які називаються атрибутами тега. Тег може мати атрибути, а може і не мати їх(наприклад, тег <HTML>). В загальному вигляді синтаксис запису тегу у сукупності з його атрибутами виглядає наступним чином:

<ТЕГ ім’я-атрибуту-1=”значення” ім’я-атрибуту-2=”значення” ... ім’я-атрибуту-n=”значення”>

Наприклад, встановлення кольору сторінки:

<BODY BGCOLOR=”blue”>

Значення атрибуту можна не брати в лапки ””, але коли значення складається, наприклад, з двох слів, відокремлених пробілом, то це необхідно для того, щоб інтерпретатор не подумав, що друге слово є іншим атрибутом. У нашому випадку можна записати: <BODY BGCOLOR=blue>, а от наступний запис є некоректним: <INPUT TYPE=”BUTTON” VALUE=Натисни мене >, оскільки слово „мене” інтерпретатор буде намагатися розпізнати як новий атрибут тегу <INPUT>. Якщо у значенні атрибуту необхідно використати лапки як символ, то у такому випадку доречним буде використати одинарні лапки: <INPUT TYPE=”BUTTON” VALUE=”Натисни ‘знову’ мене”> або <INPUT TYPE=”BUTTON” VALUE=’Натисни “знову” мене’>.

Згідно стандартів специфікації мови HTML дозволяється не брати значення у лапки в таких випадках:

1. Коли є тількі строчні або заглавні символи латинського алфавіту і відсутні інші символи.

2. Значення складається лише із цифр від 0 до 9.

HTML є нечуттєвим до регістру: <HEAD> або <head> або <HeaD> - для нього однаково.

Порядок прямування тегів, що закривають, повинен бути зворотним тому, в якому слідують відкриваючі теги. Інакше кажучи, теги з усім їх вмістом повинні повністю вкладатися в інші теги. Приклад:

<P>Вітаємо на нашому веб-сайті всіх, хто займається

<EM><STRONG>Web-дизайном</EM></STRONG>! Тут Ви зможете знайти

Тег, в який безпосередньо вкладений цей тег, називається батьківським, або батьком. У свою чергу, тег, вкладений у цей тег, називається дочірнім, або нащадком. Так, для тега <EM> у наведеному прикладі тег <P> - батьківський, а тег <STRONG> - дочірній.

Будь-який тег може мати скільки завгодно дочірніх тегів, але тільки один батьківський (що, втім, зрозуміло — не може він бути безпосередньо вкладений одночасно в два теги).

Елемент Web-сторінки, в який вкладено елемент, створюваний даним тегом, називається батьківським, чи батьком. А елемент Web-сторінки, який вкладений у цей елемент, — дочірнім, чи нащадком. Те саме, що й у випадку тегів.

Рівень вкладеності того чи іншого тега показує кількість тегів, які він послідовно вкладено. Якщо прийняти за точку відліку тег <P>, то тег <EM> матиме перший рівень вкладеності, тому що він вкладений безпосередньо в тег <P>. Тег <STRONG> буде мати другий рівень вкладеності, оскільки він вкладений в тег <EM>, а той, у свою чергу, - у тег <P>. У складних же Web-сторінках рівень вкладеності інших тегів може становити кілька десятків. Рівень вкладеності тегів у HTML-коді позначають за допомогою відступів, які ставлять ліворуч від відповідного тега та створюють за допомогою пробілів. На відображення Web-сторінки вони не впливають, оскільки форматування, які накладені в ісходному коді не впливають на інтерпретацію, але є і виключення - наприклад, тег PRE.

Іноді при розробці сторінок виникає необхідність використовувати в тексті символи, що зарезервовані для означення елементів коду HTML. Наприклад, „<", ">" та ін. Іншими словами, необхідно, щоб вони у вікні браузера відображались „як вони є", тобто як текст, але інтерпретатор сприймає їх як частину команд. Для цього були введені так звані escape-послідовності або літерали, які передбачають заміну службових символів спеціальними командами. Дані послідовності починаються з символу „&" і закінчуються „ ; ". Найбільш

розповсюджені команди наведені у табл. 3.1.

Таблиця 3.1

Символи escape-послідовності

Символ	Значення	Команда
<	Символ „менше ніж”	<
>	Символ „більше ніж”	>
“	Прямі лапки	"
&	Амперсанд	&
©	Символ копірайта	©
®	Символ зареєстрованої торгової марки	®
	Нерозривний пробіл	

Наприклад, в результаті такого запису:

"Кафедра інформатики та прикладного програмного забезпечення ©"

отримаємо:

”Кафедра інформатики та прикладного програмного забезпечення ©”

HTML також дозволяє вставити в текст будь-який символ, який підтримується кодуванням Unicode, просто вказавши його код. Для цього передбачено літерал виду:

&#<десятковий код символу>;

Наприклад, символ “©” можна вставити таким чином:

©

де 0169 - десятковий код цього символу.

При підготовці Web-сторінок слід також пам’ятати, що введене користувачем за допомогою клавіатури форматування тексту, що включає в себе пробіли, відступи табуляції та переведення рядків, ігнорується браузером при інтерпретації.

Код HTML може містити коментарі:

<!--Це коментар. Все, що розміщено в ньому ігнорується браузером. -->

Розділ заголовку (<HEAD>) містить наступні основні елементи: теги <TITLE>, <META>, <BASE>. META-теги не відображаються на екрані, але містять корисну інформацію про зміст документу: опис, ключові слова, що використовуються пошуковими системами. Існують два класи META-тегів. Синтаксис першого:

<META NAME= ім’я елементу метаінформації CONTENT= зміст інформації >

Так, наприклад, для зазначення ключових слів та термінів (перелік ключових слів – через „ , ”), які використовуються пошуковими машинами, слід помістити на сторінці наступний запис:

<META Name="keywords" CONTENT=" Ключові слова ">

Для зазначення короткого опису сторінки:

<META Name="description" CONTENT =" Опис сторінки ">

Для зазначення імені автора сторінки та інформації про нього:

<META Name="autor" CONTENT ="П.І.П., E-Mail...">

Для зазначення авторських прав:

<META Name="copyright" Content=" Фірма...">

Другий клас META-тегів, використовує атрибут HTTP-EQUIV:

<META HTTP-EQUIV= ім'я елементу метаінформації CONTENT= зміст інформації >

Ця директива використовується для надання web-сторінці деяких специфічних властивостей. Так, наприклад, для автоматичного визначення браузером кодування сторінки використовується такий запис:

<META HTTP-EQUIV="Content-Type" CONTENT ="text/html; charset=utf-8">

<!-- В HTML5 -->

<meta charset="utf-8">

Цей заголовок повідомляє про те, що Web-сервер "вважає", що надсилає вам HTML-документ і що цей документ використовує кодування символів "UTF-8".

Якщо недосвідчений Web-майстер скопіює сторінку, в якій є наступний тег, то при завантаженні даної сторінки браузер автоматично буде переходити до зазначеної URL-адреси кожні N секунд (замість N необхідно поставити ціле число):

<META HTTP-EQUIV="Refresh" CONTENT="N; url=http://адреса сторінки ">

Тег <BASE> задає базову адресу. Наприклад, якщо у заголовку на сторінці є запис:

<BASE href="https://duet.edu.ua/examples/">,

а в тілі сторінці (<BODY>) розміщене зображення: <IMG SRC="images/bar.gif">

Це призведе до формування наступної адреси зображення: https://duet.edu.ua/examples/images/bar.gif

Метатег <!DOCTYPE> задає, по-перше, версію мови HTML, де написана Web-сторінка, а по-друге, різновид цієї версії. Так, існують метатеги <!DOCTYPE>, що вказують на HTML 5, "суворий" і "перехідний" різновиди HTML 4.01 (це попередня версія мови HTML, що ще діє на даний момент) та мова XHTML (відгалуження HTML, що має дещо інший синтаксис).

Так от, метатег <!DOCTYPE html>, який ми поставили на початок нашої Web-сторінки, вказує на HTML 5.

3.2. Базові теги розмітки гіпертексту

Абзац

Базові теги HTML дозволяють форматувати текст. Перша з можливостей – це означення

абзацу. текст завжди розбивають на абзаци. Невеликі, що включають пов'язані за змістом речення, вони доносять авторський текст поступово: від простого до складного.

Абзац HTML відокремлюється невеликим відступом від попереднього та наступного елементів сторінки. Якщо абзац повністю поміщається по ширині у батьківський елемент Web-сторінки, він буде виведений в один рядок; інакше Web-браузер розіб'є його текст на кілька коротких рядків. Абзац – це незалежний елемент Web-сторінки, який відображається окремо від інших елементів. Такі елементи Web-сторінки називаються блоковими, або блоками.

Синтаксис:

<P ALIGN=АРГУМЕНТ>Текст абзацу</P>

Аргументом атрибуту ALIGN можуть бути такі значення:

RIGHT	Вирівнювання тексту в рядку за правою межею вікна або стовпця таблиці
LEFT	Вирівнювання тексту в рядку за лівою межею вікна або стовпця таблиці
CENTER	Вирівнювання тексту в рядку по центру вікна або стовпця таблиці
JUSTIFY	Вирівнювання тексту в рядку по ширині вікна або стовпця таблиці

Таким чином, запис <P ALIGN=CENTER> призводить до того, що рядки будуть вирівнюватись по центру.

У наступному прикладі ілюструється робота з вищеописаними тегами:

```
<!DOCTYPE html>
<html><head>
   <meta charset="UTF-8">
   <title>Абзаци та заголовки</title>
</head>
<body><h2>Розділ. Інформаційні системи та технології</h2>
<p align="justify">
   Інформатика - сукупність наукових спрямувань,
   що вивчають властивості інформації, способи
   її представлення та автоматичної обробки.
   Інформатика - це область людської діяльності,
   пов'язаної з процесом перетворення інформації
   за допомогою комп'ютерів та їх взаємодією із середовищем застосування.
</p>
</body></html>
```

Заголовки

Крім абзаців, великий текст для зручності читання та пошуку в ньому потрібного фрагмента зазвичай ділять більші частини: параграфи, глави, розділи. HTML не надає засобів для такого структурування тексту. Але він дозволяє створити заголовки, які ділять текст на частини, принаймні візуально.

Перш за все, усвідомимо, що в HTML є поняття рівня заголовка, що вказує, наскільки більшу частину тексту відкриває цей заголовок. Усього таких рівнів шість, і позначаються числами від 1 до 6.

Заголовок першого рівня (1) відкриває найбільшу частину тексту. Як правило, це заголовок усієї Web-сторінки. Web-браузер виводить заголовок першого рівня найбільшим шрифтом.

Заголовок другого рівня (2) відкриває дрібнішу частину тексту. Зазвичай це великий розділ. Web-браузер виводить заголовок другого рівня меншим шрифтом, ніж заголовок першого рівня.

Заголовок третього рівня (3) відкриває ще дрібнішу частину тексту; зазвичай розділ у розділі. Web-браузер виводить такий заголовок ще меншим шрифтом.

Заголовки четвертого, п'ятого та шостого рівнів (4 6) відкривають окремі параграфи, великі, дрібніші та найдрібніші відповідно. Web-оглядач виводить заголовки четвертого та п'ятого рівня ще меншим шрифтом, а заголовок шостого рівня — тим самим шрифтом, що й звичайні абзаци лише напівжирним.

На Web-сторінках невеликого та середнього розміру зазвичай застосовують заголовки першого, другого та третього рівня. Найменші рівні використовуються тільки на дуже великих Web-сторінках, що містять складно структурований текст.

Для створення заголовку використовуються теги виду <H*n*>, де *n* - номер від 1 до 6, який задає розмір шрифту. Крім того, до заголовків, як і до абзаців, можна застосовувати атрибут ALIGN.

Наприклад:

<H1>Головний заголовок сторінки</H1>

<H2>Заголовок розділу</H2>

<H3>Заголовок підрозділу</H3>

<H4>Заголовок великого абзацу</H4>

<H5>Заголовок більш меншого абзацу/H5>

<H6>Заголовок невеликого абзацу</H6>

Заголовок також відноситься до блокових елементів Web-браузер при виведенні заголовків слідує тим самим правилам, що і для абзацу.

Теги для виділення тексту

Звичайно, при написанні сторінок є можливість виділення тексту (жирним, курсивом та ін.). Теги такої розмітки наведені у таблиці 3.2.

Наприклад, тегом <CODE> звичайно виділяються невеликі фрагменти програмного коду. Зазвичай це робиться моноширним шрифтом. Якщо фрагмент програми займає декілька строчок використовують тег розмітки <PRE>. Цей тег відрізняється тим, що текст, який знаходиться в ньому зберігає своє форматування, тобто пробіли та переведення рядків, таке, як і в редакторі (рис. 3.1).

Таблиця 3.2

Основні теги для виділення тексту

Тег	Опис
<ABBR>	Абревіатура. Виводиться підкресленим
<CITE>	Невелика цитата. Виводиться курсивом
<CODE>	Невеликий фрагмент програмного коду (однорозмірний шрифт)
<DEL>	Видалений текст зі сторінки. Виводиться закресленим
<DFN>	Новий термін. Виводиться курсивом
<EM> або <I>	Курсив
<INS>	Новий текст на сторінці. Виводиться підкресленим
<Q>	Невелика цитата. Виводиться звичайним шрифтом
<SAMP> або <KBD>	Дані, що виводяться будь-якою програмою. Моноширний шрифт
<STRONG> або <B>	Жирний текст
<SUB>	Нижній індекс
<SUP>	Верхній індекс
<U>	Підкреслений
<VAR>	Ім'я змінної у програмі. Виводиться курсивом

HTML передбачає виділення тексту досить багато тегів (табл. 3.2), які мають дві особливості: усі вони парні; служать для виділення частин тексту блокових елементів (абзаців, заголовків, пунктів списків, текст фіксованого форматування).

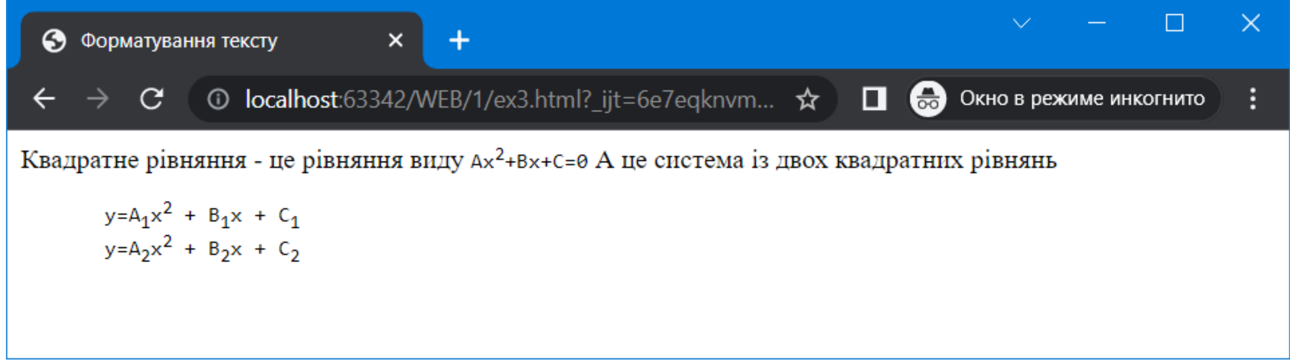


Рис. 3.1. Приклад використання тегів виділення тексту

Реалізація прикладу, показаного на рис.1.1, можлива за допомогою коду html:

```
Квадратне рівняння - це рівняння виду
<code>Ax<sup>2</sup>+Bx+C=0</code>
А це система із двох квадратних рівнянь
<pre>
    y=A<sub>1</sub>x<sup>2</sup> + B<sub>1</sub>x + C<sub>1</sub>
    y=A<sub>2</sub>x<sup>2</sup> + B<sub>2</sub>x + C<sub>2</sub>
</pre>
```

Спеціально для випадків, коли текст має бути виведений саме так, як набрано, зі збереженням всіх пробілів та переносів рядків, мова HTML передбачає парний тег <PRE>. Такий текст називається текстом фіксованого формату.

Правила відображення тексту фіксованого формату:

- для виведення використовується моноширинний шрифт (у моноширинного шрифту всі символи мають однакову ширину, на відміну від пропорційних, у яких ширина символів різна);
- усі прогалини та переноси рядків зберігаються при виведенні;
- якщо рядок тексту фіксованого формату не міститься у вікні браузера за шириною, він ні в якому разі не переноситься. Через це може виникнути потреба у горизонтальному прокручуванні Web-сторінки (що, взагалі-то, поганий стиль Web-дизайну);
- можлива наявність HTML-тегів для виділення тексту та гіперпосилань.

Текст фіксованого формату також є блоковим елементом.

Часто на Web-сторінках вказують контактні дані їх творців (поштові та електронні адреси, телефони та ін.). Для цього HTML передбачає спеціальний тег <ADDRESS>. Він поводиться так само, як тег абзацу, але його вміст виводиться курсивом:

Для того, щоб у текстовому блоці броузер зробив переведення рядка необхідно використати тег <BR>:

<ADDRESS>Усі права захищені
2023 рік.</ADDRESS>

Розрив рядків слід застосовувати економно, лише у випадках, коли потрібно розділити один абзац на кілька логічно пов'язаних частин. Раніше недосвідчені Web-дизайнери з його допомогою часто розбивали текст на абзаци, але зараз це поганий тон Web-дизайну.

При зміні ширини вікна браузера текст автоматично переноситься на нові рядки, не виходячи за межі вікна. Для того, щоб вказати браузеру місце, де допускається зробити перенос, існує тег <WBR>.

Для організації цитат призначений тег <BLOCKQUOTE>:

<BLOCKQUOTE>Цитування</BLOCKQUOTE>

3.3. Списки та роздільники

Списки використовуються для того, щоб надати читачеві перелік будь-яких позицій, пронумерованих чи непронумерованих, – пунктів списку. Список з пронумерованими пунктами так і називається нумерованим, а з непронумерованими - маркованим. У маркованих списках пункти позначаються особливим значком - маркером, який ставиться ліворуч від пункту списку.

Марковані списки зазвичай служать для простого перерахування будь-яких позицій, порядок дотримання яких не є важливим. Якщо ж слід звернути увагу читача на те, що позиції повинні йти одна за одною саме в тому порядку, в якому вони перераховані, слід застосувати нумерований список.

Web-браузер виводить список з відступом зліва. Відстань між пунктами списку він робить меншими, ніж у випадку абзаців або заголовків. Також він сам розставляє необхідні маркери чи нумерацію.

Будь-який список у HTML створюється у два етапи. Спочатку пишуть рядки, які стануть пунктами списку, і кожен із цих рядків поміщають усередину парного тега <LI>. Потім всі ці пункти поміщають всередину парного тега <UL> (якщо створюється маркований список) або <OL> (при створенні нумерованого списку) - ці теги визначають сам список.

Маркувальний список представляється такою конструкцією:

<UL TYPE=”параметр”>

<LI>Елемент 1</LI><LI>Елемент 2</LI>…<LI>Елемент N</LI>

</UL>

Параметр TYPE набуває одного з наступних значень:

1. disc – заповнені кола.
2. circle – пусті кола.

3. square – заповнені квадрати.

Нумерований список дозволяє відображати впорядковану інформацію:

<OL TYPE=”параметр” START=”значення”>

<LI>Елемент 1</LI><LI>Елемент 2</LI>…<LI>Елемент N</LI>

</OL>

Значення параметру TYPE:

1. Якщо дорівнює „1” – звичайні арабські числа 1,2, ...

2. Якщо „I” – римські у верхньому регістрі I, II, ...

3. Якщо „i” – римські у нижньому регістрі i, ii, ...

4. Якщо „A” – символьна маркіровка у верхньому регістрі: A, B, …

5. Якщо „a” – символьна маркіровка у нижньому регістрі: a, b, …

Параметр “START” означає, з якого числа необхідно почати нумерацію.

Слід відмітити, що тег <LI> може включати ті ж атрибути, що і теги самих списків. Важливо лише слідкувати за тим, щоб вказані в тегу <LI> параметри атрибутів співпадали з типом списку, що використовується.

Списки можна розміщувати один в одного, створюючи вкладені списки. Робиться це наступним чином. Спочатку в "зовнішньому" списку (в який повинен бути поміщений вкладений) шукають пункт, після якого повинен бути вкладений перелік. Потім HTML-код, що створює вкладений список, поміщають у розрив між текстом цього пункту та його тегом, що закриває </LI>. Якщо вкладений список повинен поміщатися на початку "зовнішнього" списку, його слід вставити між відкриваючим тегом <LI> першого пункту "зовнішнього" списку та його текстом.

Для того, щоб візуально відокремити одні об’єкти HTML від інших використовуються роздільники, які визначаються тегом <HR>. Синтаксис:

<HR ALIGN=”параметр” WIDTH=”значення” SIZE=”значення” COLOR=”значення”>

Атрибут “WIDTH” визначає довжину лінії, яка задається або в абсолютному вигляді, або у відсотках до ширини вікна браузера. Атрибут SIZE задає товщину лінії у пік селах.

В наступному прикладі наведено використання вищеописаних тегів списку:

```
<h2>Технологія HTML</h2>
<ol type="I" start="2">
  <li>Огляд HTML</li>
  <ul>
    <li>Поняття тегів і атрибутів</li>
    <li>Базові теги розмітки гіпертексту</li>
    <li>Списки та роздільники</li>
```

```
    </ul>
    <hr width="80%" size=3>
    <li>Елементи керування</li>
    <ul type="square">
      <li>Введення-виведення</li>
      <li>Списки</li>
      <li>Active X</li>
    </ul>
</ol>
```

Результат наведено на рис. 3.2.

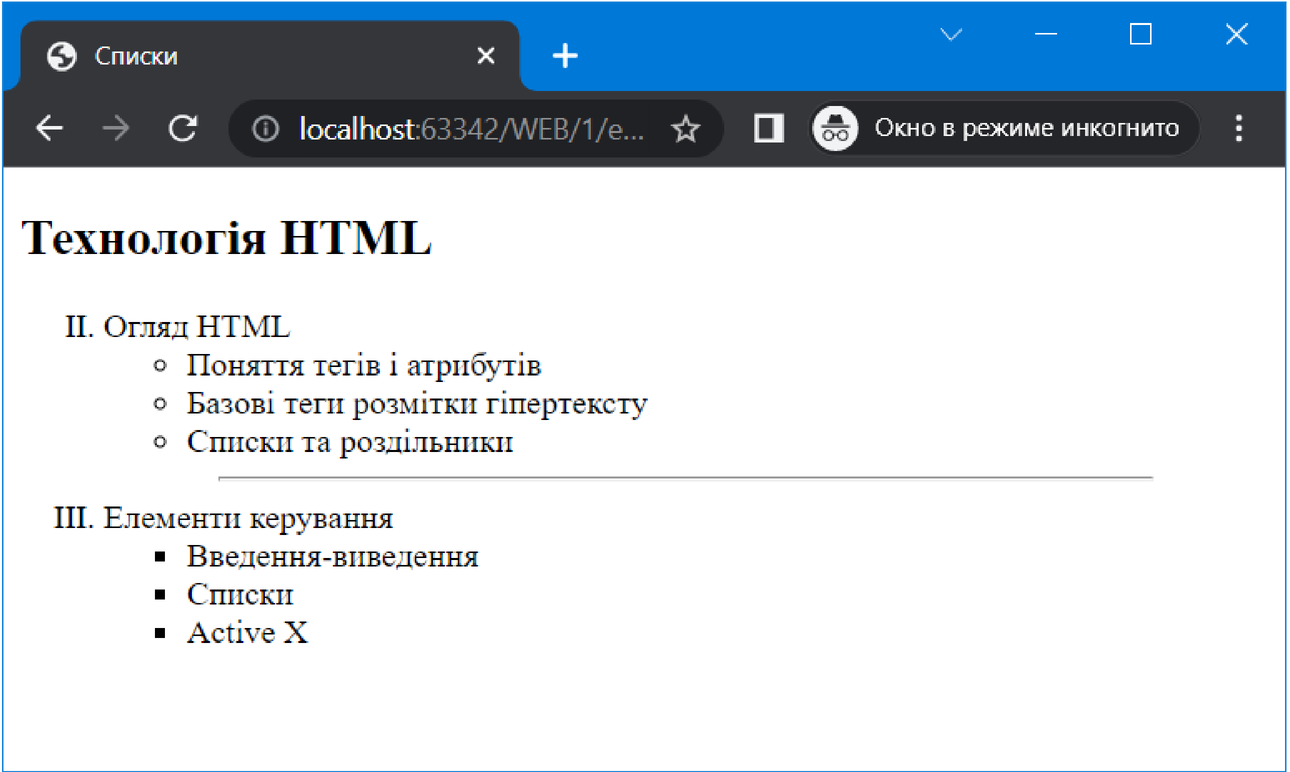


Рис. 3.2. Приклад використання тегів списків

Як видно з коду, роздільник має довжину 80% від ширини екрану і розміщується по центру вікна.

Списки та їх пункти відносяться до блокових елементів Webсторінки, і при їх виведенні на екран Web-браузер керується тими ж правилами, що і при виведенні абзаців та заголовків.

3.4. Створення посилань

Гіперпосилання встановлюють зв’язок між певним елементом поточного документу

HTML та іншою web-сторінкою або незалежним об’єктом – файлом чи зображенням. При натисканні кнопки миші на посиланні браузер робить запит і виводить інший документ (який знаходиться на даному чи іншому сервері). Як посилання може виступати не тільки частина тексту, а й графічний елемент. Посилання можна поділити на 4 категорії.

Посилання на документи

Реалізувати посилання на іншу сторінку можна шляхом використання тегу <A>. Синтаксис:

<A HREF=”адреса” TARGET=”параметр” TITLE=”альтернативний текст”>Текст посилання</A>

Атрибут HREF вказує на адресу сторінки, з якою необхідно встановити зв’язок. Способи запису адреси такі:

1. Адреса може бути представлена у вигляді повної адреси. Наприклад:

<A HREF=”https://duet.edu.ua/examples/index.html”>Приклади </A>.

Більшість посилань звичайно вказують на інші документи, що знаходяться на тому ж сервері, що і головна сторінка. Ці посилання містять відносні посилання на документи:

2. Якщо сторінка (наприклад, “1.html”) знаходиться в одній папці з поточною, в цьому випадку достатньо вказати її назву:

<A HREF=”1.html”>Приклади </A>.

3. Якщо сторінка (наприклад, “2.html”) знаходиться в папці (наприклад, details), яка в свою чергу знаходиться в одній папці з поточною сторінкою, то має місце запис:

<A HREF=” details/2.html”>Опис </A>.

4. Якщо сторінка (наприклад, “3.html”) знаходиться в папці (наприклад, product), яка, в свою чергу, знаходиться в кореневій директорії того ж сайту, що і поточна сторінка, то має місце запис:

<A HREF=” /product/3.html”>Опис продукції </A>.

5. Якщо сторінка (наприклад, “4.html”) знаходиться у папці (наприклад, product), яка в свою чергу знаходиться в тій же папці того ж сайту, що і папка поточної сторінки, то має місце запис:

<A HREF=” ../product/4.html”>Опис продукції </A>.

Атрибут TARGET містить директиви, які описують правила відкриття елементу у браузері. TARGET може приймати значення: “_blank” – посилання відкривається щоразу в новому вікні; “_self” – в цьому ж вікні. Якщо TARGET має інші значення (в нашому випадку „example”), то в такому разі він задає найменування вікна, в якому необхідно відкрити посилання. Якщо вікно із зазначеним ім’ям існує, то сторінка завантажується в нього. Інакше створюється нове вікно браузера із зазначеним ім’ям, в якому відкривається сторінка.

Атрибут TITLE необхідний для створення альтернативного тексту – підказки.

Слід також зазначити, що одне посилання не можна вкладати в інше.

Наступний приклад ілюструє організацію посилань.

```
<p align="center">Приклади Web-сторінок</p><hr>
<ul>
  <li><a href="ex1.html" target="example">Приклад 1</a></li>
  <li><a href="ex2.html" target="example">Приклад 2</a></li>
....
  <li><a href="ex6.html" target="example">Приклад 6</a></li>
</ul>
<p align="center">
  <a href="help/about.html" target="_blank" title="Короткий опис прикладів">Опис</a>
</p>
```

Як видно з прикладу, на сторінці є посилання на інші сторінки, які містять усі вищенаведені приклади. Якщо, не існує вікна з іменем "example" при натисканні на будь-яке посилання, то вікно браузера з таким умовним іменем (що не відображається на екрані) створиться, і при виборі наступних прикладів вони будуть завантажуватися в новостворене вікно. При підведенні курсору миші до посилання „Опис" з'явиться підказка, визначена атрибутом TITLE. Web-сторінка з описом прикладів буде постійно завантажуватись у нове вікно. Інтерфейс реалізації прикладу наведено на наступному рис. 3.3.

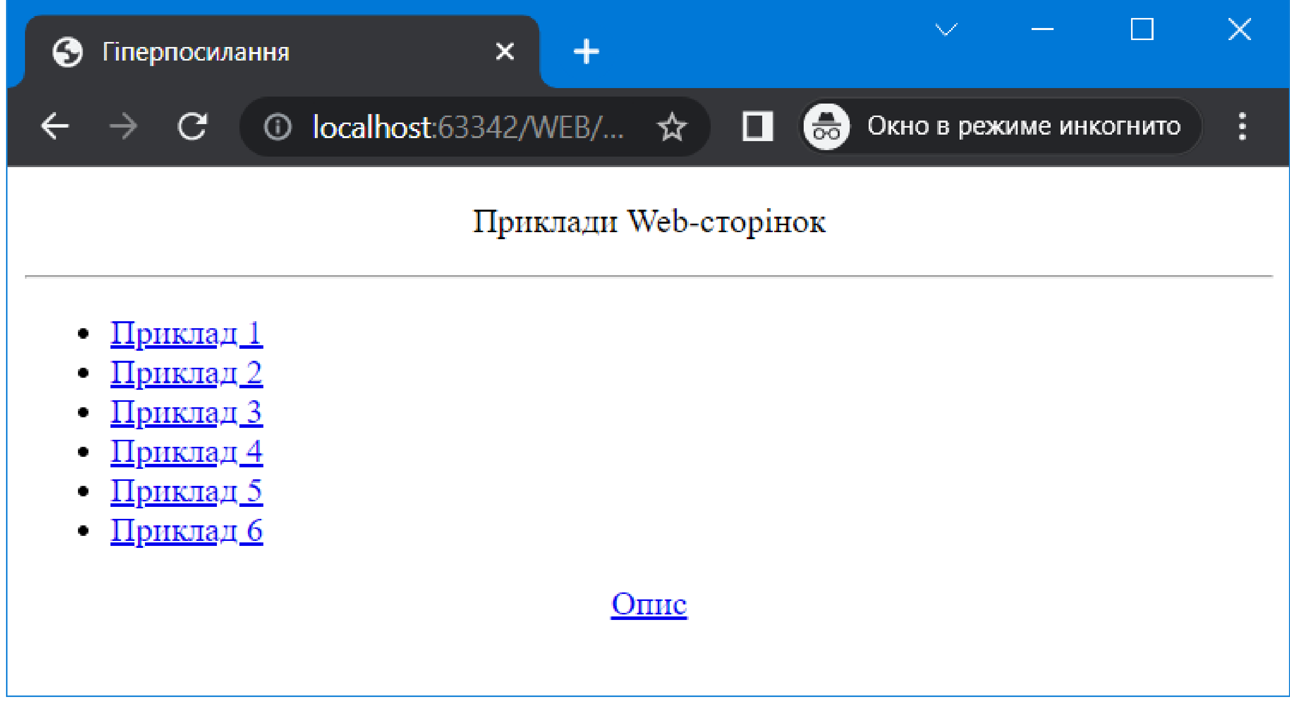


Рис. 3.3. Приклад створення посилань

Правила відображення гіперпосилань web-браузером:

- звичайні гіперпосилання виділяються синім кольором;
- гіперпосилання, за якими користувач вже «ходив» (відвідане раніше гіперпосилання), виводяться темно-червоним кольором;
- гіперпосилання, на якій відвідувач в даний момент натискає (активне гіперпосилання), виводиться яскраво-червоним кольором;
- текст будь-яких гіперпосиланнь підкреслюється;
- при розташуванні курсору миші на гіперпосиланні web-браузер змінює його зображення.

Посилання на розділи

Іноді виникає необхідність розмістити на web-сторінці посилання не на інший документ, а на розташований в межах цієї ж сторінки розділ або ділянку тексту. При активізації подібного посилання браузер перебудовує скролінг (прокручує вікно) таким чином, щоб необхідний розділ або фраза опинилися у межах клієнтської частини вікна. Процес створення посилання на розділ можна умовно поділити на два етапи. Перший полягає у підготовці „закладки” – спеціальної мітки з унікальним в межах даного документу ім’ям, що присвоюється „закладці” атрибутом ID. Якорі створюють за допомогою тега <A>, як і гіперпосилання. Тільки в даному випадку атрибут HREF у ньому бути не повинен. Замість нього в тег <A> поміщають обов'язковий атрибут ID, що задає унікальне в межах поточної web-сторінки ім'я створюваного якоря.

Для цього у тій частині тексту, де необхідно розмістити мітку, застосовується команда:

<A ID=”ім’я закладки”>Ключове слово або заголовок розділу</A>

Другий етап – створення власне посилання за допомогою команди:

<A HREF=”#ім’я закладки”>Текст посилання</A>

Необхідно пам’ятати, що для імені „закладки” краще використовувати латинські символи. Якщо необхідно розмістити посилання на розділ якогось іншого документу, необхідно перед ім’ям „закладки” вказати повну URL-адресу документу. Приклад:

```
<body>
  <p align="center">Приклади Web-сторінок</p>
  <hr>
  <ul>
    <li><a href="#ex1">Приклад 1</a></li>
    <li><a href="#ex2">Приклад 2</a></li>
...
    <li><a href="#ex7">Приклад 7</a></li>
```


<p align="center">Приклад 1
Опис прикладу 1</p>

<p align="center">Приклад 2
Опис прикладу 2</p>

...

<p align="center">Приклад 7
Опис прикладу 7</p>

</body>

Мається також можливість створити гіперпосилання, яке нікуди не вказує ("пусте" гіперпосилання). Для цього достатньо задати в якості значення атрибуту HREF=”#” :

<A HREF="#">Натиснення нікуди не веде</A>

При переході на таке гіперпосилання нічого не відбудеться.

Посилання на адресу електронної пошти

При роботі в Internet часто зустрічаються посилання, при активізації яких автоматично запускається встановлена на комп’ютері поштова програма, і на екрані формується вже готовий до відправки бланк електронного листа з заповненим адресним полем, а іноді і з заповненим полем „Тема листа” (Subject). Подібні посилання теж можна реалізувати за допомогою тегу <A>. Це здійснюється таким чином:

<A HREF=”mailto: адреса електронної пошти?cc=адреса1, адреса2, ... , адресаn&subject=тема листа”>Текст посилання</A>

Наприклад:

<A HREF=”mailto: noname@gmail.com&subject=HTML”>Чекаю листів!</A>

Директива „mailto:” вказує на основну адресу електронної пошти, куди слід відправляти автоматично створений лист. Мінімально необхідне значення атрибуту HREF це зазначення описаної директиви, після якої слід помістити адресу e-mail. Директива “?cc=” дозволяє визначити адреси електронної пошти користувачів, яким буде розіслано копію листа. Вони вказуються підряд через кому „ , ”. За допомогою функції “&subject=” можна вказувати тематику листа.

Посилання на номери телефонів

Пам'ятайте, що смартфони, з яких люди заходять на ваш сайт, дозволяють використовувати посилання для здійснення телефонних викликів. Ви можете позбавити відвідувачів зайвої дії, дозволивши їм набрати номер телефону, вказаний на вашому сайті, просто торкнувшись його на сторінці. Синтаксис, що використовується для форми з атрибутом tel: дуже простий:

Звони нам безкоштовно(800) 555-1212

При дотику до посилання користувачі смартфонів побачать вікно повідомлення із проханням підтвердити, що вони хочуть зателефонувати за цим номером. Ця функція

підтримується в більшості мобільних пристроїв.

Посилання на файлові об’єкти

Посилання на деякий файловий об’єкт має на увазі організацію зв’язку документу HTML з будь-яким файлом, що зберігається на сервері, наприклад, архівом ZIP. При натисканні кнопки миші на таке посилання браузер автоматично видає на екран діалогове вікно і просить зазначити чи завантажувати файл для зберігання його на локальний комп’ютер. Наведемо приклад організації посилання на файл:

<A HREF=”https://duet.edu.ua/example/archive.zip”>Архів</A>

Іноді для організації зв’язку з файловим об’єктом у запису адреси посилання використовується префікс „file://”, але такий підхід застосовується, в основному, для посилань на файловий об’єкт, розташований на комп’ютері в межах локальної мережі:

<A HREF=”file://C:\Doc\archive.zip”>Архів</A>

Якщо завантаження файлу планується здійснювати по протоколу FTP, слід використовувати префікс “ftp://”:

<A HREF=”ftp://duet.edu.ua/example/archive.zip”> Архів</A>

Формування адреси посилання (HREF) відбувається згідно із правилами формування посилань на сторінці. Ці правила було описано вище.

Додаткові можливості гіперпосилань

Мова HTML пропонує нам деякі додаткові можливості для створення гіперпосилань. Їх застосовують нечасто, але іноді вони корисні. Перш за все, ми можемо вказати для гіперпосилання "гарячу" клавішу. Якщо користувач натискає цю клавішу, утримуючи також клавішу <Alt>, веб-браузер дозволяє виконати перехід по гіперпосиланню.

Для вказівки "гарячої" клавіші передбачений необов'язковий атрибут ACCESSKEY тегу <A>. Значення цього атрибуту - латинська буква, відповідає потрібній клавіші:

<A HREF="http://www.somesite.com/pages/page125.html"

ACCESSKEY="d">Сторінка №125</A>

Тут ми вказали для гіперпосилання "гарячу" клавішу <D>. І, щоб перейти по ній, користувачеві буде достатньо нажати комбінацію клавіш <Alt>+<D>.

По гіперпосиланням також можна переходити (активізувати фокус) за допомогою клавіатури. Гіперпосилання, що має фокус введення, виділяється тонкою чорною штриховою рамкою. Якщо нажати клавішу <Enter>, Web-браузер виконає перехід по гіперпосиланню, що має в даний момент фокус введення. Якщо нажати клавішу <Tab>, Web-браузер перенесе фокус введення на наступне гіперпосилання. Якщо нажати комбінацію клавіш <Shift>+<Tab>, Web-браузер перенесе фокус введення на попереднє гіперпосилання.

Порядок, в якому виконується перенесення фокусу введення з одного гіперпосилання на

інше при натисненні клавіші <Tab> або <Shift>+<Tab>, так і називається - порядок обходу. За замовчуванням він складається з порядком, у якому гіперпосилання визначені в HTML-коді веб-сторінки. Але можна визначити власний порядок обходу за допомогою атрибуту TABINDEX тегу <A>. Його значення — ціле число від -32 767 до 32 767 — номер в порядку обходу.

Якщо вказано позитивний номер, саме він буде визначати порядок обходу. Іншими словами, спочатку фокус введення отримає гіперпосилання з номером 1, потім — з номером 2, далі - з номером 3 і т.д. Якщо вказаний номер, рівний нулю, обхід буде реалізований у порядку, в якому гіперпосилання визначене в HTML-коді веб-сторінки. Фактично нуль - значення атрибуту тегу TABINDEX за умовчанням. Якщо вказано негативний номер, дане гіперпосилання взагалі виключається із порядку обходу. Неможливо буде працювати за допомогою клавіатури - можна буде тільки виконати дії за допомогоюмиші.

Приклад:

```
<P><A HREF="page1.htm" TABINDEX="3">Сторінка 1</A></P>
<P><A HREF="page2.htm" TABINDEX="2"> Сторінка 2</A></P>
<P><A HREF="page3.htm" TABINDEX="1"> Сторінка 3</A></P>
```

Цей HTML-код створює три гіперпосилання зі "зворотнім" порядком обходу. Спочаткуфокус введення отримає гіперпосилання «Сторінка 3», потім — «Сторінка 2» і нарешті - «Сторінка 1».

Зображення-гіперпосилання

Мова HTML дозволяє використовувати як вміст гіперпосилання будь-який фрагмент будь-якого блокового елемента, зокрема і графічне зображення, тобто створити зображення-гіперпосилання.

Для створення зображення гіперпосилання достатньо помістити всередину тега <A> тег <IMG>:

```
<A HREF="http://www.w3.org"><IMG SRC="w3logo.gif"></A>
```

Цей HTML-код створює зображення-гіперпосилання, що вказує на Web-сайт організації W3C. А як зображення обрано логотип цієї організації.

Правила виведення зображень-гіперпосилань Web-браузером:

- зображення-гіперпосилання оточується рамкою, що має відповідний гіперпосиланню колір: синій - для невідвіданої, темно-червоний - для відвіданої і т.д.;
- при розміщенні курсору миші на зображення-гіперпосилання Web-браузер змінює його форму на "вказівний перст", як і у випадку текстового гіперпосилання.

Рамка навколо зображення-гіперпосилання найчастіше виглядає непрезентабельно, тому її зазвичай прибирають.

Зображення-мапи

А ще HTML дозволяє перетворити на гіперпосилання частину графічного зображення. Більше того, ми можемо розбити зображення на частини, кожна з яких буде гіперпосиланням, що вказує на свою інтернет-адресу. Такі зображення називають зображеннями-мапами, а її частини-гіперпосилання - "гарячими" областями.

За допомогою зображень-мап часто створюють заголовки Web-сайтів, фрагменти якого є гіперпосиланнями, що вказують на певну Web-сторінку. Пік популярності зображень-мап давно пройшов, проте їх ще можна досить часто зустріти.

Зображення-мапу створюють у три етапи. Перший етап – створення самого зображення за допомогою добре нам знайомого тега <IMG>:

<IMG SRC="map.gif">

Другий етап - створення мапи, особливого елемента Web-сторінки, який описує набір "гарячих" областей зображення-мапи. Сама мапа на Web-сторінці не відображається.

Мапу створюють за допомогою парного тега <MAP>:

<MAP NAME="<ім'я мапи>"></MAP>

За допомогою обов'язкового атрибуту NAME тега <MAP> задається унікальне в межах Web-сторінки ім'я мапи. Воно однозначно ідентифікує цю карту, може містити тільки латинські літери, цифри та знаки підкреслення та починатися повинно з літери:

<MAP NAME="samplemap"></MAP>

Після створення мапи слід прив'язати її до створеного на першому етапі зображення. Для цього ми застосуємо обов'язковий у даному випадку атрибут USEMAP тегу <IMG>. Його значення - ім'я мапи, що прив'язується до зображення, причому на початку цього імені обов'язково слід поставити символ “#”. (В імені, заданому атрибутом NAME тега <MAP>, символ “#” не повинен бути присутнім.):

<IMG SRC="map.gif" USEMAP="#samplemap">

На третьому етапі створюють описи самих "гарячих" областей у мапі. Їх поміщають усередину відповідного тега <MAP> і формують за допомогою одинарних тегів

<AREA>:

<AREA [SHAPE="rect|circle|poly"] COORDS="<набір параметрів>"
HREF="<інтернет-адреса гіперпосилання>"|NOHREF
TARGET="<мета гіперпосилання>">

Необов'язковий атрибут SHAPE задає форму гарячої області. Обов'язковий атрибут COORDS перераховує координати, необхідні для побудови цієї області.

Значення атрибуту SHAPE:

- "rect" - прямокутна "гаряча" область. Атрибут COORDS у цьому випадку

записується у вигляді COORDS="<X1>,<Y1>,<X2>,<Y2>", де X1 та Y1 - координати верхнього лівого, а X2 і Y2 - правого нижнього кута прямокутника. До речі, якщо атрибут SHAPE відсутній, створюється прямокутна область;

- "circle" - кругла "гаряча" область. В цьому випадку атрибут COORDS має вигляд COORDS="<X центру>,<Y центру>,<радіус>";
- "poly" - "гаряча" область у вигляді багатокутника. Атрибут COORDS повинен мати вигляд COORDS="<X1>,<Y1>,<X2>,<Y3>,<X3>,<Y3>...", де Xn та Yn — координати відповідної вершини багатокутника.

Атрибут HREF задає інтернет-адресу гіперпосилання. Він може бути замінений атрибутом без значення NOHREF, що задає область, не пов'язану з жодною інтернет-адресою. HTML-код, який створює зображення-карту:

```
<IMG SRC="map.gif" USEMAP="#samplemap">
<MAP NAME="samplemap">
<AREA SHAPE="circle" COORDS="50,50,30" HREF="page1.html">
<AREA SHAPE="circle" COORDS="50,150,30" HREF="page2.html">
<AREA SHAPE="poly" COORDS="100,50,100,100,150,50,100,50" NOHREF>
<AREA SHAPE="rect" COORDS="0,100,30,100" HREF="appendix.html" TARGET="_blank">
</MAP>
```

Тут ми створили дві круглі "гарячі" області, що вказують на Web-сторінки page1.html і page2.html, багатокутну область, яка не посилається нікуди, і прямокутну область, яка посилається на Web-сторінку appendix.html. Причому остання "гаряча" область при натисканні на ній відкриє Web-сторінку в новому вікні Web-браузеру.

3.5. Графіка та мультимедіа

Насамперед, графічні зображення та мультимедійні дані зберігаються в окремих файлах. А в HTML-коді Web-сторінок за допомогою спеціальних тегів прописують посилання на ці файли, власне, їх інтернет-адреси. Зустрівши таке тег-посилання, Web-браузер запитує у Web-сервера відповідний файл із зображенням, аудіо- або відеороликом і виводить його на Web-сторінку в місце, де зустрівся даний тег.

Графічні зображення, аудіо- та відеоролики та взагалі будь-які дані, що зберігаються в окремих від Web-сторінки файлах, називаються вбудованими елементами Web-сторінок.

Формати інтернет-графіки

На даний момент існує кілька десятків форматів зображень. Але Web-браузери

підтримують далеко не всі. У WWW зараз використовуються лише три формати: GIF, JPEG, PNG. Слід зазначити, що ці формати підтримують стиснення графічної інформації. Завдяки стисненню розмір графічного файлу сильно зменшується, і тому він передається по мережі швидше, ніж не стиснутий файл.

Формат GIF (Graphics Interchange Format, формат обміну графікою) був розроблений ще у 1987 році та модернізований у 1989 році. На даний момент він вважається застарілим, але все ще поширеним.

Переваг у нього досить багато. По-перше, GIF дозволяє задати для зображення "прозорий" колір. По-друге, в одному GIF-файлі можна зберігати відразу кілька зображень, фактично справжній фільм (анімований GIF). По-третє, через особливості стиснення, що застосовується в ньому, він відмінно підходить для зберігання штрихових зображень (мап, схем, малюнків олівцем та ін.).

Недолік формату GIF лише один — він зовсім не годиться для зберігання напівтонових зображень (фотографій, картин тощо). Це пов'язано з тим, що GIF-зображення можуть включати всього 256 кольорів, і втратою якості при стисненні.

GIF використовується для зберігання елементів оформлення Web-сторінок (усіляких лінійок, маркерів списків тощо) та штрихових зображень.

Формат JPEG (Joint Photographic Experts Group, Об'єднана група експертів з фотографії) був розроблений у 1993 році спеціально для зберігання напівтонових зображень. Активно застосовується і досі.

JPEG, на відміну від GIF, не обмежує кількість кольорів зображення, а реалізоване в ньому стиснення найкраще підходить для напівтонових зображень. Однак він погано справляється зі штриховою графікою, не підтримує "прозорий" колір та анімацію.

Формат PNG (Portable Network Graphics, мережева графіка, що переміщується) з'явився на світ у 1996 році. Він розроблявся як заміна застарілому і не дуже зручному GIF, а також певною мірою JPEG.

До переваг формату PNG можна віднести можливість зберігання як штрихових, так і напівтонових зображень та підтримку напівпрозорості. Нестача всього один і некритичний - неможливість зберігання анімації.

Файли GIF і PNG мають розширення gif і png, а файли JPEG - jpg, jpe або jpeg.

Вставка графічних зображень

При відправленні сторінок клієнту разом із сторінкою, відправляються і графічні зображення, які повинні бути присутніми на ній.

Інтеграція графіки в html-документ здійснюється за допомогою тегу <IMG>, синтаксис якого в загальному вигляді можна представити таким чином:

<IMG SRC=”адреса зображення” ALIGN=”значення” WIDTH=”значення” HEIGHT=”значення” BORDER=”значення” ALT=”Текст” VSPACE=”значення” HSPACE=”значення”>

Приклад:

```
<body>
  <h1>Інтернет-магазин</h1><hr>
  <p align="center">
    <img src="img/home.jpg" alt="Інтернет-магазин" title="Інтернет-магазин">
  </p>
  <hr>
  <p align="center">Приклад картинки, яка розташована по центру вікна</p>
</body>
```

<IMG> — вбудований елемент (inline). Це означає, що він повинен бути частиною блокового елемента, наприклад абзацу.

У тега <IMG> є два спеціальних параметри, які визначають висоту і ширину зображення. Якщо задати один з них, наприклад

<IMG SRC="img/home.jpg" WIDTH=100>,

то рисунок пропорційно зменшиться (або збільшиться – все залежить від початкових розмірів картинки) так, що його ширина буде дорівнювати 100 пікс. Аналогічно можна використовувати параметр HEIGHT. Але слід зазначити, що змінюється розмір того, що виводиться на екран, а не розмір картинки у байтах, оскільки дана операція виконується вже після завантаження зображення з сервера, і зменшення робить сам інтерпретатор браузера. Переваги вищеназваних параметрів полягають в тому, що можна заздалегідь задати габарити зображення. Тобто, можна побудувати сторінки таким чином, що будь які зміни файла зображення не впливатимуть на її зовнішній вигляд.

Якщо у браузері клієнта вимкнений показ зображень, картинка не з’явиться на екрані. Таким чином, може бути втрачена інформативність сторінки. На такий випадок є атрибут, що дозволяє вивести замість зображення надпис як альтернативу. Атрибут так і називається ALT. Наприклад:

<IMG SRC="img/home.jpg" ALT=”Інтернет-магазин”>

Якщо у користувача невелика швидкість зв’язку, а розмір файла з картинкою великий, можна під час завантаження основної картинки, організувати показ меншої за допомогою зміни значення атрибуту LOWSRC:

<IMG SRC="img/home.jpg" LOWSRC=”img/homel.jpg”>

Відокремити картинку від тексту можна, помістивши її у рамку. Товщина рамки навколо картинки визначається параметром BORDER:

<IMG SRC="img/home.jpg" BORDER=1>

Можна задати вертикальний та горизонтальний відступи картинки від тексту в пікселах за допомогою атрибутів VSPACE та HSPACE:

<IMG SRC="img/home.jpg" BORDER=1 VSPACE=20 HSPACE=10>

Для того, щоб при натисканні на картинку викликалась інша сторінка, достатньо тег <IMG> помістити в тег <A>.

<A HREF="desc/aboutex.htm">

<IMG SRC="img/home.jpg" WIDTH=100 HEIGHT=20 BORDER=1 ALT="Сайт Інтернет-магазину"></A>

Мультимедіа

Форматів мультимедійних файлів існує не менше форматів файлів графічних. Як і у випадку з інтернет-графікою, Web-браузери підтримують далеко не всі мультимедійні формати, а небагато. Але Web-браузеру мало підтримувати лише сам формат мультимедійних файлів. Він має бути "знaqjvbq" і з форматом кодування записаної в ньому аудіо- та (або) відеоінформації. У світі мультимедіа так - різні файли одного формату можуть зберігати інформацію, закодовану різними форматами. Більше того, аудіо- та відеодоріжки мультимедійного файлу практично завжди кодуються різними форматами.

Практично всі формати кодування мультимедійних даних підтримують їх стиснення. Завдяки цьому розмір мультимедійних файлів значно зменшується, що добре позначається на швидкості їх передачі через мережу.

В Web-дизайні використовуються та підтримуються Web-браузерами такі формати:

1. WAV (WAVe, хвиля) - був розроблений Microsoft на початку 90-х років минулого століття для зберігання аудіоданих і застосовується для цієї мети досі. Файли такого формату мають розширення wav.

2. OGG - був розроблений організацією Xiph.org для зберігання аудіо- та відеоінформації. Файли цього формату мають розширення ogg (універсальне розширення), oga (аудіофайли) та ogv (відеофайли); останні два розширення трапляються рідко.

3. MP4 - було розроблено організацією Motion Picture Expert Group (Експертна група з питань рухомого зображення; також відома як MPEG) у 1998 році для зберігання аудіо- та відеоданих. Файли цього формат мають розширення mp4.

4. QuickTime - формат дуже старий, він старший навіть за WAV. Був розроблений Apple в 1989 році для зберігання аудіо- та відеоданих. Файли такого формату мають розширення mov.

Тепер розглянемо формати кодування аудіо та відео, що підтримуються сучасними Web-браузерами:

1. PCM (Pulse-Coded Modulation, імпульсно-кодова модуляція) — найпростіший та найстаріший формат кодування. Він навіть не підтримує стиснення інформації. Для кодування аудіоданих.

2. Vorbis – більш сучасний формат кодування. Був представлений організацією Xiph.org (розробником формату файлу OGG) у 2002 році. Використовується для кодування аудіоданих.

3. AAC (Advanced Audio Coding, розвинене кодування аудіо) - був розроблений організацією Motion Picture Expert Group в 1997 році. Застосовується для кодування аудіоданих.

4. Theora - він також розроблений організацією Xiph.org. Використовується для кодування відео.

5. H.264 - теж дуже "молодий". Був представлений організаціями Motion Picture Expert Group та Video Coding Experts Group (Група експертів з кодуванню відео) у 2003 році. Призначений для кодування відео.

Майже всі ці формати відкриті. Винятки - формат файлів QuickTime, що належить Apple, і формат кодування H.264, захищений більш ніж сотнею патентів.

Типи MIME

По мережі передаються різні дані: Web-сторінки, графічні зображення, аудіо- і відеофайли, архіви та ін. Ці дані призначені різним програмам. До того ж, з різними даними, програма, яка прийняла їх, може вчинити по-різному. Так, Web-браузер при отриманні Web-сторінки або графічного зображення відобразить їх на екрані, а при отриманні архіву або файлу, що виконується — відкриє або збереже його на диску.

Всім переданим по мережі даним присвоюється особливе позначення, що однозначно вказує на їхню природу, - тип MIME (Multipurpose Internet Mail Extensions, багатоцільові розширення пошти Інтернету). Тип MIME надає даним програма, що їх відправляє, наприклад, Web-сервер. А програма, що приймає (той же Web-браузер) за типом MIME прийнятих даних визначає, чи підтримує вона ці дані, і якщо підтримує, то що з ними робити.

Щоразу, коли ваш Web-браузер запитує сторінку, Web-сервер спочатку відсилає йому заголовки (headers), і потім — фактичну розмітку сторінки (page markup). Зазвичай ці заголовки невидимі, незважаючи на те, що насправді вони є засобами Web-розробки, які стануть видимими, якщо ви у цьому зацікавлені. Заголовки відіграють важливу роль, тому що вони повідомляють вашому браузеру, як слід інтерпретувати розмітку сторінки, яка буде

згодом отримана браузером. Найбільш важливий заголовок називається Content-Type і виглядає так: Content-Type: text/html

Рядок text/html називається "типом вмісту" (або "типом контенту", або "MIME-типом") сторінки. Цей заголовок одноосібно визначає тип ресурсу, який є сторінкою, і, отже, вказує, як ця сторінка має бути візуалізована. Зображення мають власні типи MIME (наприклад, image/jpeg для зображень формату JPEG, image/png для зображень формату PNG і т. д.). JavaScript мають власний тип MIME. Каскадні сторінки стилів (CSS) також мають власний тип MIME. Усе має власний тип MIME. Всесвітня павутина працює завдяки типу MIME.

З метою забезпечення зворотної сумісності з продуктами, датованими до 1993 року, деякі популярні браузери за певних обставин ігнорують заголовок Content-Type. Цей прийом називається "сніффінгом контенту" (content sniffing) або "розпізнавання браузера" (browser sniffing). Але як загальне емпіричне правило, все, що ви коли-небудь переглядали через Web, включаючи сторінки HTML, зображення, скрипти, відео, PDF-файли, і все, що має URL, надавалася вам із зазначенням конкретного типу MIME у заголовку Content-Type.

Так, веб-сторінка має тип MIME text/html. Графічне зображення формату GIF мають тип MIME image/gif. Тип MIME файлу, що виконується, - application/x-msdownload, а архіву ZIP - application/x-zip-compressed. Свої типи MIME мають мультимедійні файли.

У табл. 3.3 пераховані типи форматів MIME мультимедійних файлів.

Таблиця 3.3

Типи MIME для мультемедіа файлів

Формат	**Тип MIME**
WAV	audio/wave audio/wav audio/x-wav audio/x-pn-wav
OGG	audio/ogg (для аудіофайлів) video/ogg (для відео) application/ogg
MP4	video/mp4
QuickTime	video/quicktime

Web-браузери працюють з дуже обмеженим набором форматів мультимедійних файлів із кількох десятків існуючих. Більш того, різні Web-браузери підтримують різні формати. Тому Web-браузер повинен визначити, чи він підтримує формат отриманого файлу, тобто варто його взагалі завантажувати.

Вставка аудіо

Для вставки на Web-сторінку аудіо мова HTML 5 передбачає парний тег <AUDIO>. Веб-адреса файлу, в якій зберігається цей аудіоролик, вказують за допомогою атрибуту SRC цього тегу:

<AUDIO SRC="sound.wav"></AUDIO>

Зустрівши тег <AUDIO>, Web-браузер може відразу завантажити та відтворити аудіофайл, тільки завантажити його без відтворення чи взагалі нічого не робити.

Також він може вивести на Web-сторінку елементи управління, за допомогою яких відвідувач запускає відтворення аудіофайлу, зупиняє його, прокручує вперед або назад і регулює гучність. Все це налаштовується за допомогою різних атрибутів тегу <AUDIO>.

Тег <AUDIO> створює блоковий елемент веб-сторінки. Отже, ми не зможемо вставити аудіоролик на Web-сторінку як частину абзацу.

За умовчанням веб-браузер не відтворюватиме аудіо. Щоб він це зробив, у тегу <AUDIO> потрібно вказати спеціальний атрибут AUTOPLAY. Це дійсно особливий атрибут: він не має значення - достатньо однієї його присутності в тегу, щоб він почав діяти:

<P>Зараз ви почуєте звук!</P>

<AUDIO SRC="sound.ogg" AUTOPLAY></AUDIO>

За замовчуванням аудіо ніяк не відображається на Web-сторінці. Але якщо в тегу <AUDIO> поставити атрибут без значення CONTROLS, Web-браузер виведе де проставлений тег <AUDIO>, елементи управління відтворенням аудіо. Вони включають кнопку запуску та призупинення відтворення, шкалу відтворення та регулятор гучності:

<P>Натисніть кнопку відтворення, щоб почути звук.</P>

<AUDIO SRC="sound.ogg" CONTROLS></AUDIO>

Атрибут без значення AUTOBUFFER має сенс вказувати в тегу <AUDIO> тільки якщо там відсутній атрибут AUTOPLAY. Якщо він вказаний, браузер відразу після завантаження веб-сторінки почне завантажувати файл аудіо - це дозволить виключити затримку файлу перед початком його відтворення.

Вставка відео

Для вставки на Web-сторінку відео призначено парний тег <VIDEO>. Інтернет-адреса відеофайлу вказується за допомогою знайомого нам атрибуту SRC цього тегу:

<VIDEO SRC="film.ogg"></VIDEO>

Зустрівши цей тег, Web-браузер виведе панель перегляду вмісту відео на місті тегу. Залежно від зазначених нами в тегу <VIDEO> атрибутів, він може відразу завантажити та відтворити відео, тільки завантажити його без відтворення або взагалі нічого не робити.

Також він може вивести на Web-сторінку елементи керування відтворенням відео.

Як і тег <AUDIO>, тег <VIDEO> створює блоковий елемент Web-сторінки та підтримує атрибути AUTOPLAY, CONTROLS та AUTOBUFFER:

<VIDEO SRC="film.ogg" AUTOPLAY CONTROLS></VIDEO>

Якщо відтворення відео ще не запущено, на панелі перегляду буде виведено перший його кадр або взагалі нічого (конкретна поведінка залежить від Web-браузеру). Але ми можемо вказати графічне зображення, яке буде туди виведено як заставку. Для цього є атрибут POSTER тегу <VIDEO>; його значення вказує на інтернет-адресу потрібного графічного файлу:

<VIDEO SRC="film.ogg" CONTROLS POSTER="filmposter.jpg"></VIDEO>

Тут ми вказали для відео заставку, яка буде виведена на панелі перегляду перед початком відтворення. Заставка зберігається у файлі filmposter.jpg.

Раніше ми дізналися, що набір мультимедійних форматів у різних Web-браузерів різниться. І може статися так, що аудіо-або відеоролик, який ми помістили на Web-сторінку, виявиться не знайомим якомусь Web-браузеру. Спеціально для таких випадків HTML 5 передбачає можливість вказати в одному тегу <AUDIO> або <VIDEO> відразу кілька мультимедійних файлів. Web-оглядач сам вибере з них той, формат який він підтримує.

Якщо ми збираємось вказати відразу кілька мультимедійних файлів в одному тегу <AUDIO> або <VIDEO>, то маємо зробити дві речі:

1. Прибрати з тега <AUDIO> або <VIDEO> вказівку на мультимедійний файл, тобто атрибут SRC та його значення.

2. Помістити всередині тега <AUDIO> або <VIDEO> набір тегів <SOURCE>, кожен з яких визначатиме інтернет-адресу одного мультимедійного файлу.

Одинарний тег <SOURCE> вказує як інтернет-адресу мультимедійного файлу, так і та його тип MIME. Для цього призначені атрибути SRC та TYPE даного тегу відповідно:

```
<VIDEO>
<SOURCE SRC="film.ogg" TYPE="video/ogg">
<SOURCE SRC="film.mov" TYPE="video/quicktime">
</VIDEO>
```

Даний тег <VIDEO> визначає відразу два відеофайли, що зберігають один і той же фільм: один – формату OGG, інший – формату QuickTime. Web-браузер, що підтримує формат OGG, завантажить і відтворить перший файл, а Web-браузер, що підтримує QuickTime, - другий файл.

Зазначимо, що тип MIME відеофайлу (і, відповідно, атрибут TYPE тегу <SOURCE>) можна опустити. Але тоді Web-браузеру доведеться завантажити початок файлу, щоб з'ясувати, чи підтримує він формат цього файлу. А це зайве і зовсім непотрібне навантаження

на мережу. Так що тип MIME краще вказувати завжди.

Якщо Web-браузер взагалі не підтримує теги <AUDIO> та <VIDEO>, то в такому випадку він їх проігнорує і нічого на Web-сторінку не виведе. Однак ми можемо вказати текст заміни, в якому описати проблему і запропонувати будь-який шлях її вирішення (наприклад, змінити Web-браузер). Такий текст заміни ставлять усередині тегу <AUDIO> або <VIDEO> після всіх тегів <SOURCE> (якщо вони є).

3.6. Таблиці

Структура представлення таблиць співпадає з класичним представленням, скажімо таким, як у текстовому редакторі Microsoft Word, тобто включає в себе рядки, у кожному з яких є комірки, які визначають стовпці таблиці. Комірки таблиці можуть містити будь-які теги, тобто елементи: посилання, зображення та ін.

Таблиці HTML створюються в чотири етапи. На першому етапі в HTML-коді за допомогою парного тега формують саму таблицю:

<TABLE></TABLE>

Таблиця HTML є блоковим елементом Web-сторінки. Це означає, що вона розміщується окремо від решти всіх блокових елементів: абзаців, заголовків, великих цитат, аудіо- і відео. З другого краю етапі формують рядки таблиці. Для цього передбачені парні теги <TR>; кожен такий тег створює окремий рядок. Теги <TR> поміщають усередину тегу <TABLE>:

<TABLE> <TR></TR> <TR> </TR> <TR> </TR> </TABLE>

На третьому етапі створюють комірки таблиці, для чого використовують парні теги <TD> та <TH>. Тег <TD> створює звичайну комірку, а тег <TH> - комірку заголовка, в якій буде поміщатися "шапка" відповідного стовпця таблиці. Теги <TD> та <TH> поміщають у теги <TR>, що створюють рядки таблиці, в яких повинні знаходитися ці осередки:

<TABLE>

<TR> <TH></TH> <TH></TH> <TH></TH> </TR>

<TR> <TD></TD> <TD></TD> <TD></TD> </TR>

<TR> <TD></TD> <TD></TD> <TD></TD> </TR>

</TABLE>

На четвертому, останньому етапі вказують вміст комірок, який поміщають у відповідні теги <TD> та <TH>:

<TABLE>

<TR>

<TH>Стовпець 1</TH><TH>Стовпець 2</TH><TH>Стовпець 3</TH>

```
</TR>
<TR>
<TD>Комірка 1.1</TD><TD>Комірка 1.2</TD><TD>Комірка 1.3</TD>
</TR>
<TR>
<TD>Комірка 2.1</TD><TD>Комірка 2.2</TD><TD>Комірка 2.3</TD>
</TR>
</TABLE>
```

Якщо нам потрібно помістити в комірку таблиці простий текст, ми можемо просто вставити його у відповідний тег <TD> або <TH>. При цьому брати їх у теги, що створюють блокові елементи, необов'язково.

Якщо нам потрібно якось оформити вміст комірок, ми застосуємо вивчені вже теги. Наприклад, ми можемо надати номерам комірок особливої важливості, скориставшись тегом <EM>; в результаті їх буде виведено курсивом:

```
<TABLE>
<TR>
<TD>Комірка <EM>1.1</EM></TD>
<TD>Комірка <EM>1.2</EM></TD>
<TD>Комірка <EM>1.3</EM></TD>
</TR>
</TABLE>
```

Ще ми можемо помістити в комірку графічне зображення:

```
<TD><IMG SRC="picture.jpg" ALT="Малюнок у комірці таблиці"></TD>
```

Але часто буває необхідно іноді помістити в комірку таблиці великий текст, іноді що складається з кількох абзаців. У такому разі знадобляться знайомі нам по розділі 2 теги, які створюють блокові елементи сторінки. Теги <TD> та <TH> це дозволяють:

```
<TD><H4>Це великий текст</H4>
<P>Це початок великого тексту, що являє собою вміст комірки
таблиці.</P>
<P>Це продовження великого тексту, що є вмістом комірки таблиці.</P>
<P><IMG SRC="picture.jpg" ALT="Ілюстрація до великого тексту"></P>
<P>А це <STRONG>довгоочікуване закінчення</STRONG> великого тексту.</P>
</TD>
```

В цілому синтаксис таблиць можна уявити наступним чином:

```
<TABLE ALIGN=”значення” WIDTH=”значення” BORDER=”значення”
```

CELLSPACING=”значення” CELLPADING=”значення” BGCOLOR=”значення”>

<CAPTION ALIGN=”значення”> Заголовок таблиці в цілому</CAPTION>

<TR ALIGN=”значення” VALIGN=”значення” BGCOLOR=”значення”>

<TH ALIGN=”значення” VALIGN=”значення” BGCOLOR=”значення” COLSPAN=” ціле число” ROWSPAN=”ціле число” WIDTH=”значення” HEIGHT=”значення” NOWRAP>Заголовок стовпця</TD>

</TR>

<TR ALIGN=”значення” VALIGN=”значення” BGCOLOR=”значення”>

<TD ALIGN=”значення” VALIGN=”значення” BGCOLOR=”значення” COLSPAN=” ціле число” ROWSPAN=”ціле число” WIDTH=”значення” HEIGHT=”значення” NOWRAP>Зміст комірки</TD>

</TR>

</TABLE>

<TR>-визначає початок нової строки таблиці; <TH> - звичайно використовується для означення заголовку стовпця таблиці (комірка таблиці); <TD> - комірка певного запису таблиці. Різниця між <TD> та <TH> полягає лише в тому, що вони мають різні базові значення деяких атрибутів. Тег <TH> має власні базові параметри – це центрування тексту посередині комірки та виведення тексту жирним шрифтом.

Атрибути CELLSPACING та CELLPADING визначають, відповідно, відстань між комірками, які йдуть послідовно одна за одною, та простір між змістом кожної комірки і її межами. Використання атрибуту CELLPADING задає додатковий простір навколо змісту комірки. Атрибути ALIGN та VALIGN задають, відповідно, горизонтальне та вертикальне вирівнювання в комірці. З таким атрибутом, як ALIGN, вже було знайомство, а от можливі найбільш розповсюджені значення атрибуту VALIGN наступні: TOP (притиснення тексту до верхньої межі комірки), MIDDLE (центрування тексту), BOTTOM (притиснення тексту до нижньої межі комірки). Згадані параметри мають місце і в тегу <TABLE> і визначають поведінку таблиці відносно її місця у вікні (горизонтальне або вертикальне вирівнювання таблиці). Атрибут BORDER вказує товщину рамок стовпців та комірок таблиці.

Тег <TD> оперує такими атрибутами: COLSPAN – кількість стовпців, які об’єднує комірка; ROWSPAN – кількість рядків, які об’єднує комірка. NOWRAP забороняє перенесення по словам у комірці (при зміні розмірів вікна), якщо строчка повністю не поміщається в комірці. Це пов’язано із тим, що браузер при зміні розмірів вікна пропорційно зменшує розміри стовпців, якщо явно не вказано ширину в атрибуті WIDTH. Якщо стоїть даний атрибут, то при неможливості зміни розмірів таблиці (так щоб вона вміщувалась у вікно), браузер перебудовує прокрутку.

Прості таблиці

Приклад простої таблиці (рис. 3.4):

```
<table align="center" bgcolor="aqua" border="1" width="80%">
  <caption align="center">Робота з таблицями</caption>
  <tr>
    <th>Стовбець 1</th>
    <th>Стовбець 2</th>
    <th>Стовбець 3</th>
  </tr>
  <tr align="left" bgcolor="white">
    <td align="center">Рядок 1 Стовбець 1</td>
    <td>Рядок 1 Стовбець 2</td>
    <td>Рядок 1 Стовбець 3</td>
  </tr>
  <tr align="left" height="100">
    <td align="center">Рядок 2 Стовбець 1</td>
    <td valign="bottom">Рядок 2 Стовбець 2</td>
    <td valign="top" bgcolor="red">Рядок 2 Стовбець 3</td>
  </tr>
  <tr>
    <td valign=middle align=right nowrap>Рядок 3 Стовбець 1</td>
  </tr>
</table>
```

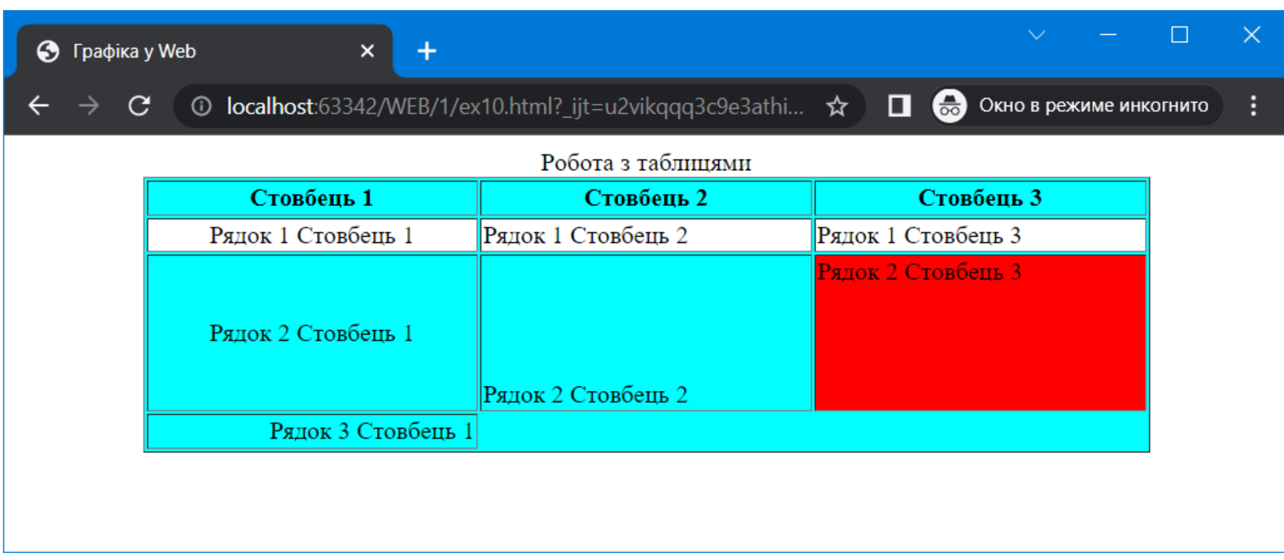


Рис. 3.4. Результат роботи з проситими таблицями

Як видно з коду та результату, розмір усієї таблиці – 80% від ширини клієнтської частини вікна. Всі рядки таблиці мають фоновий колір „aqua”, окрім тих рядків та комірок, для яких колір задано явно (другий рядок та третя комірка третього рядка). Задавання атрибуту ALIGN=CENTER тегу <TABLE> дозволяє вирівнювати саму таблицю по центру. Що стосується загальних атрибутів тегів <TABLE>, <TR>, <TD>, то пріоритетність значень одних і тих же атрибутів розповсюджується від окремого до загального, тобто, наприклад пріоритетність значення атрибуту BGCOLOR тегу <TD> вища, ніж тегу <TR>, де, в свою чергу, – вище ніж у тегу <TABLE>.

Значення атрибуту BGCOLOR – це числовий код кольору або символьна строка, що означає заданий колір. Відомо, що для виводу кольорового зображення на екран використовується палітра RGB (Red, Green, Blue). Будь-який колір можна отримати змішуванням у різних пропорціях “Червоного”, “Зеленого” та “Синього”. Для означення кольору в HTML прийнято використовувати шістнадцятиричний числовий код або символьну помітку. Так, наприклад білий колір - це повністю насичений “Червоний”, “Зелений” і “Синій”, тобто числове значення кожного з них дорівнює 255, що можна записати числом у шістнадцятиричному форматі FFFFFF. В HTML це записується так: “#FFFFFF”. Білому кольору також відповідає символьна строчка: „white”. Отже, білий текст запишеться: <TD BGCOLOR=”#FFFFFF”> або <TD BGCOLOR=”white”>. А, наприклад, задавання тексту жовтого кольору запишеться так: <TD BGCOLOR=”#FFFF00”> або <TD BGCOLOR=”yellow”>. Слід також зазначити, що краще використовувати шістнадцятиричний формат, оскільки не всі броузери однаково „розуміють” строкове означення кольору.

HTML-код, що створює таблиці, може здатися дещо громіздким. Але це плата за виняткову гнучкість таблиць HTML. Ми можемо помістити до таблиці будь-який вміст: абзаци, заголовки, зображення, аудіо- та відео і навіть інші таблиці.

Тепер настав час розглянути правила, якими керуються Web-браузери при виведенні таблиць на екран:

1. Таблиця є блоковим елементом Web-сторінки.
2. Розміри таблиці та її комірок робляться такими, щоб повністю вмістити їх контент.
3. Між кордонами окремих комірок і між кордоном кожної комірки та її вмістом робиться невеликий відступ.
4. Текст комірок заголовку виводиться напівжирним шрифтом і вирівнюється по центру.
5. Рамки навколо всієї таблиці та навколо окремих її комірок не малюються.

І ще кілька правил, згідно з якими створюється HTML-код таблиць. Якщо їх порушити, Web-браузер відобразить таблицю некоректно або не виведе її взагалі:

1. Тег <TR> може знаходитися тільки всередині тегу <TABLE>. Будь-який інший вміст тегу <TABLE> (крім заголовка та секцій таблиці) буде проігноровано.

2. Теги <TD> та <TH> можуть знаходитися тільки всередині тегу <TR>. Будь-який інший вміст тегу <TR> буде проігноровано.

3. Вміст таблиці може знаходитися лише у тегах <TD> та <TH>.

4. Комірки таблиці повинні мати хоч якісь вміст, інакше Web-оглядач може їх взагалі не відобразити. Якщо ж якась порожня комірка, в неї слід помістити нерозривний пробіл (HTML-літерал:).

Заголовок та секції таблиці

Насамперед, за допомогою парного тега <CAPTION> ми можемо дати таблиці заголовок. Заголовок таблиці виводиться з неї, а текст його вирівнюється по центру таблиці. За бажання ми можемо його якось оформити, скориставшись знайомими тегами:

```
<CAPTION><STRONG>Це таблиця<STRONG></CAPTION>
```

Зазвичай тег <CAPTION> міститься відразу після тега <TABLE> — так логічніше. Але не має значення, де в HTML-коді таблиці він присутній - заголовок все одно буде поміщений Web-браузером над таблицею.

Крім того, ми можемо логічно розбити таблицю HTML на три значущі частини - секції таблиці:

секцію заголовка, в якій знаходиться рядок з комірками заголовку, що формує її "header";

секцію тіла, де знаходяться рядки таблиці, що становлять основні дані;

секцію завершення з рядками, що формують "footer" таблиці (зазвичай мають у своєму розпорядженні підсумкові дані і різні примітки).

Секцію заголовка таблиці формує тег <THEAD>, секцію тіла - <TBODY>, а секцію завершення - <TFOOT>. Всі ці теги парні, розміщуються безпосередньо в тегу <TABLE> і містять теги <TR>, що формують рядки таблиці, які входять до відповідної секції:

```
<TABLE>
<THEAD>
<TR>
<TH>Стовпець 1</TH>
<TH>Стовпець 2</TH>
<TH>Стовпець 3</TH>
</TR>
</THEAD>
<TBODY>
<TR>
```

```
<TD>Комірка 1.1</TD>
<TD>Комірка 1.2</TD>
<TD>Комірка 1.3</TD>
</TR>
<TR>
<TD>Комірка 2.1</TD>
<TD>Комірка 2.2</TD>
<TD>Комірка 2.3</TD>
</TR>
</TBODY>
<TFOOT>
<TR>
<TD>Підсумок по 2.1</TD>
<TD>Підсумок по 2.2</TD>
<TD>Підсумок по 2.3</TD>
</TR>
</TFOOT>
</TABLE>
```

Секції таблиці Web-браузер ніяк не відображає і не виділяє на Web-сторінці. Вони просто поділяють таблицю на три логічні частини. Однак ми можемо поставити для тегів, що формують секції таблиці, якесь уявлення, яке керуватиме їх відображенням.

Об'єднання комірок таблиць

Іноді при створенні web-сайту на сторінці необхідно представити таблицю, деякі комірки якої займають кілька рядків або стовпців по вертикалі або горизонталі. З цією метою використовуються атрибути COLSPAN та ROWSPAN, змінюючи параметри яких досягають бажаного результату.

Щоб об'єднати кілька осередків по горизонталі в одну, потрібно виконати такі кроки:

1. Знайти в коді HTML тег <TD> (<TH>), що відповідає першій з об'єднуваних комірок (зліва направо).

2. Вписати в нього атрибут COLSPAN і привласнити йому кількість комірок, що об'єднуються, вважаючи і найпершу з них.

3. Видалити теги <TD> (<TH>), що створюють інші комірки, що об'єднуються, даного рядка.

Об'єднати комірки по вертикалі трохи складніше. Ось кроки, які потрібні:

1. Знайти в коді HTML рядок (тег <TR>), в якому знаходиться перша з комірок, що

об'єднуються (якщо вважати рядки зверху вниз).

2. Знайти в коді цього рядка тег <TD> (<TH>), що відповідає першій з комірок, що об'єднуються.

3. Вписати в нього атрибут ROWSPAN і присвоїти йому кількість комірок, що об'єднуються, вважаючи і найпершу з них.

4. Переглянути наступні рядки та видалити з них теги <TD> (<TH>), що створюють інші комірки, що об'єднуються.

Наприклад, отримати результат показаний на рис. 3.5 можна за допомогою нижченаведеного коду:

```
<table align="center" bgcolor="aqua" border=2 width=90% cellspacing="0">
  <caption align="center">Сировина для виробництва агломерату</caption>
  <tr>
    <th rowspan="2">Вид сировини</th>
    <th rowspan="2">Найменування</th>
    <th colspan="2">Ціна</th>
    <th rowspan="2">Кількість</th>
  </tr>
  <tr>
    <th>грн.</th>
    <th>дол.</th>
  </tr>
  <tr>
    <td rowspan="2">Основна</td>
    <td><a href="help/p1.htm">Концентрат</a></td>
    <td align="right">320</td>
    <td align="right">8</td>
    <td align="right">200 т.</td>
  </tr>
  <tr>
    <td><a href="help/p2.htm">Аглоруда</a></td>
    <td align="right">160</td>
    <td align="right">4</td>
    <td align="right">100 т.</td>
  </tr>
  <tr>
```

```
        <td>Інша</td>
        <td>Електроенергія</td>
        <td align="right">1000</td>
        <td align="right">25</td>
        <td align="right">200 кВт</td>
    </tr>
</table>
```

Таблиці: об'єднання комірок

localhost:63342/WEB/1/ex11.html?_ijt=qfbbhdk4qk6vss1a...

Окно в режиме инкогнито

Сировина для виробництва агломерату

Вид сировини	Найменування	Ціна		Кількість
		грн.	дол.	
Основна	Концентрат	320	8	200 т.
	Аглоруда	160	4	100 т.
Інша	Електроенергія	1000	25	200 кВт

Рис. 3.5. Результат роботи з таблицями, в яких комірки об'єднують декілька рядків або стовпців

Як видно з коду, третя комірка об'єднує два стовпці (3 та 4), де вказується ціна у гривнях та ціна у доларах для кожної з сировини. Оскільки продукція „концентрат" та „агломерат" об'єднуються у групу основної сировини, доречно об'єднати перші комірки другого та третього рядку в одну.

3.7. Структурування контенту. Теги <div> та <span>

Елемент div використовується визначення логічного поділу контенту або елементів на сторінці. Він вказує, що разом елементи утворюють певну смислову одиницю і мають розглядатися CSS та JavaScript як одна одиниця. Відзначаючи пов'язані елементи як розділ div і надаючи йому унікальний ідентифікатор id або вказуючи, що це частина елемента class ви надаєте контекст для елементів на сторінці.

У цьому прикладі елемент div використовується як контейнер, щоб згрупувати зображення та два абзаци:

```
<div class="listing">
```

```
<img src="mcdonald.gif" alt="">
<p><cite>Створення Web-сайтів. Основне керівництво</cite>,
<p>З книги ви дізнаєтеся, як створювати веб-сторінки та багато чого інше.</p>
</d iv>
```

Поміщаючи ці елементи div, автор зазначає, що вони концептуально пов'язані. Це також дозволяє застосовувати стилі до елементів <p> в межах class=”listing” інакше, ніж до інших абзаців на сторінці.

Нижче наведено приклад іншого звичного використання елемента div , що вживається, щоб розбити сторінку на частини з метою компонування макету. У цьому прикладі заголовок і кілька абзаців укладені елемент div і визначені як розділ “news”:

```
<div class=”news”>
<h1>Новини цього тижня</h1>
<p>Ми продовжували працювати над...</p>
<p>І останнє, але не в останню чергу...</p>
</d iv>
```

Існує можливість вкладати елементи div всередину інших елементів d iv, але цу треба робити обдумано. Ви повинні завжди прагнути, щоб розмітка була якомога простіше, тому додайте елемент div тільки якщо він необхідний для логічної структури, додавання стилю чи сценарію.

У специфікації HTML5, верстальники все рідше вдаються до використання загальних елементів div.

Елемент span пропонує ті ж переваги, що і div. Також він використовується для вбудованих елементів. Оскільки елементи span вбудовані, вони можуть містити тільки текст та інші вбудовані елементи (іншими словами ви не можете помістити туди заголовки, списки, елементи групування контенту і т.д.). Для правильного розуміння звернемося до деяких прикладів.

Немає елемента telephone , але ми можемо використовувати елемент span для надання значення телефонним номерам. У цьому прикладі кожен номер телефону розмічено як span та класифіковано як ”phone”.

```
<ul>
<li>Володимир: <span class="phone">999-8282</span></li>
<li>Сергій: <span class="phone">888-4849</span></li>
<li>Оля: <span class="phone">567-8123</span></li>
</ul>
```

Видно, як елементи span додають значення контенту, який інакше міг би бути

випадковою послідовністю цифр. Крім цього елемент span дозволяє застосовувати той самий стиль до всіх телефонних номерів на сайті (наприклад, гарантувати, що вони ніколи не будуть розділені на два рядки за допомогою визначення white-space: nowrap каскадних таблиць стилів). Так інформація стає розпізнаваною не тільки для людей, а й для комп'ютерних програм, які знають, що зробити з телефонною інформацією. Фактично деякі значення (у тому числі «phone») були прийняті до стандарту системи розмітки, відомої як «Мікроформати», яка підвищує корисність контенту для програм.

3.8. Семантична розмітка в HTML5. Структура документу.

Один із типів змісту, що включає багато нових елементів керування HTML5, відповідає за розбивку документу на окремі частини. Так новий стандарт визначає семантичну розмітку.

До появи специфікації HTML5 немає можливості згрупувати ці фрагменти у більші частини, крім уклавши їх у універсальний загальний елемент (div). У мові HTML5 були введені нові елементи, що надають семантичне значення розділам звичайної веб-сторінки або програми, включаючи розділи (section), статті (article), навігацію (nav), опосередковано пов'язаний контент (aside), верхні (header) та нижні (footer) колонтитули.

Довгі документи простіше сприймати, коли їхній контент розділений на дрібніші частини. Наприклад, книги поділяються на розділи, в газетах є розділи місцевих новин, спортивна колонка, гумор і т. д. Щоб розділити довгі веб-документи на тематичні секції, використовуючи елемент section.

У розділах зазвичай є заголовок (всередині елемента section) та будь-який інший контент, для угруповання якого є розумна причина.

Існує безліч застосувань елемента section, від розподілу цілої сторінки на основні розділи до визначення тематичних розділів у межах однієї статті.

У наступному прикладі документ з інформацією про ресурси за типографікою було поділено на дві частини за типом ресурсів^

```
<section>
<h2>Книги</h2><ul><li>...</li></ul>
</section>
<section><h2>Онлайн-посібники</h2>
<p>Це найкращі навчальні посібники у Всесвітньому павутинні.</p>
<ul><li>...</li></ul >
</section>
```

HTML5 визначає і нові семантичні елементи, перелічені у табл. 3.4.

Таблиця 3.4.

Нові теги семантичної розмітки в HTML5

Елемент	**Опис**
<section>	Елемент section представляє розділ типового документа чи програми. Розділ (section) являє собою тематичне угруповання інформаційного вмісту, як правило, з заголовком. Домашня сторінка Web-сайту може бути розбита на такі розділи, як вступ, новини, контактна інформація.
<nav>	Елемент nav є розділом сторінки, яка посилається на інші сторінки або на інші частини в межах тієї ж сторінки: це розділ навігації. Не всі групи посилань на сторінці повинні являти собою елементи nav — тільки розділи, що складаються з основних навігаційних блоків, підходять для використання елементами nav. Зокрема, нижні колонтитули зазвичай містять короткий список посилань на загальні сторінки сайту, такі, як умови обслуговування, домашня сторінка, сторінка з угодою про авторські права і т. д. Елемент нижнього колонтитулу (footer) сам по собі вже достатній для таких випадків, і жодної потреби в елементах nav немає.
<article>	Елемент article представляє компонент сторінки, що складається з самодостатньої композиції в документі, на сторінці, у додатку, на сайті, і призначається для незалежного поширення та багаторазового використання. Це може бути, наприклад, повідомлення на форумі, стаття в газеті або журналі, запис у блозі, коментар, надісланий користувачем, інтерактивний віджет (widget) або гаджет (gadget), або будь-який інший незалежний елемент або інформаційне наповнення.
<aside>	Елемент aside є розділом сторінки, що складається з інформаційного наповнення, яке має непряме ставлення до контенту, що оточує цей елемент, і може розглядатися окремо від нього. Такі розділи в друкованому вигляді часто представляються як поля або бічні панелі. Елемент може застосовуватися для груп елементів nav, а також для іншого інформаційного наповнення, яке вважається окремим від основного на сторінці
<hgroup>	Елемент hgroup є заголовком розділу. Елемент використовується для групування набору елементів h1-h6, у тих випадках, коли заголовок має безліч рівнів, включаючи підзаголовки, альтернативні заголовки або рядки тегів.
<header>	Елемент header представляє групу допоміжних інструментів — для введення або навігаційних. Елемент header зазвичай призначається для того, щоб містити заголовок розділу (Елемент h1-h6 або елемент hgroup), але ця вимога не є обов'язковою. Елемент header також може використовуватися для "обгортання" змісту розділу, форми пошуку або будь-яких логотипів.
<footer>	Елемент footer є заголовком для найближчого дочірнього розділу або кореневого елемента розділу. Нижній колонтитул (footer) зазвичай містить інформацію про свій розділ, зокрема відомості про автора тексту, посилання на інші документи, пов'язані з цим, інформацію про авторські права тощо. Нижні колонтитули не обов'язково повинні розміщуватися в кінці розділу, хоча зазвичай це буває саме так.
<time>	Елемент time час за 24-годинною шкалою, або точну дату за григоріанським календарем (за новим стилем), зазвичай із зазначенням часу та часового поясу
<mark>	Елемент mark представляє фрагмент тексту в документі, позначений кольором або виділений підсвічуванням для довідкових цілей

Використовуйте елемент article для незалежних робіт, які можуть використовуватись окремо або багаторазово в іншому контексті (наприклад, при синдикації). Це корисно для журнальних та газетних статей, повідомлень у блогах, коментарів чи інших елементів, які можна витягти для зовнішнього використання.

Щоб підвищити інтерес, можна розбити довгий елемент на кілька розділів, як показано тут:

```
<article>
<h1>Знайомство зі шрифтом Helvetica</hl>
<section>
<h2>історія виникнення шрифту Helvetica</h2>
<p>...</p>
</section>
<section>
<h2>Helvetica сьогодні</I2>
<p>...</p>
</section>
</article>
```

І навпаки, елемент section веб-документа може складатися з кількох статей:

```
<section>
<article>
<h1>Свіжий погляд на шрифт Futura</hl>
<p>...</p>
</article>
<article>
<h1>Знайомимось зі шрифтом Humanist</hl>
<p>...</p>
</article>
</section>
```

Елементи section та article легко переплутати, особливо тому, що їх можна вкладати одна в одну. Пам'ятайте, що якщо контент автономний і може з'являтися поза цим контекстом, його краще розмітити як article.

Елемент aside визначає контент, опосередковано пов'язаний із навколишнім вмістом. Це еквівалент бічної врізки у друку, але елемент не можна було назвати sidebar тому, що слово «бічний» (side) вказує на положення, а не призначення. Проте бічна врізка хороший наочний приклад того, як використовується aside. Його можна застосувати для розміщення цитат,

довідкової інформації, елементів списку, виносок або іншого матеріалу, пов'язаного з контентом (але не має особливого значення).

У цьому прикладі елемент aside використовується для створення списку:

посилань, пов'язаних із основною статтею.

```
<h1>Веб-друкарня</h1>
<p>У 1997 році існували конкуруючі між собою формати шрифтів та інструменти для їх створення...</p>
<p>Тепер у нас є кілька способів використання чудових шрифтів на веб-сторінках...</p>
<aside>
<h2>Ресурсbі веб-шрифтів</h2>
<ul>
<li><a href=”http://typekit.com/”>Сервіс Typekit</a></li>
<li><a href=”http://www.google.com/webfonts”>I1рифти Google</a><//li>
</ul>
</aside>
```

Елемент aside не відображається за замовчуванням, тому вам необхідно перетворити його на блоковий елемент і налаштувати зовнішній вигляд і положення за допомогою таблиць стилів.

Новий елемент nav надає розробникам семантичний спосіб вказати навігацію сайту:

```
<nav>
<ul>
<li><a href="">Serif</a>/li>
<li><a href="">Sans-serif</a>/li>
<li><a href="">Script</a>/li>
<li><a href="">Display</a>/li>
<li><a href="">Dingbats</a>/li>
</ul>
</nav>
```

Однак, не всі списки посилань слід оточувати тегами nav. У специфікації чітко вказано, що його потрібно застосовувати для посилань, що забезпечують основну навігацію по всьому сайту, довгому розділі або статті.

Елемент nav може бути корисним з точки зору доступності. Як тільки програми екранного доступу та інші пристрої стануть сумісні зі специфікацією HTML5, користувачі можуть легко виявити або пропустити розділ навігації без тривалих пошуків.

Так як верстальники веб-сторінок протягом багатьох років розмічали в документах

розділи верхнього та нижнього колонтитула, не викликало сумнівів, що повноцінні елементи для цих дій виявляться корисними.

Елемент header використовується для розміщення вступного матеріалу, який зазвичай розташований на початку веб-сторінки або у верхній частині розділу чи статті. Немає певних вимог до змісту елемента header; допустимо додавати все, що підходить як вступ до сторінки або розділу.

У наступному прикладі верхній колонтитул елемента містить логотип, назву сайту та засоби навігації:

```
<header>
<img src="/images/logo.png">
<h2>Останні новини</h2>
<nav>
<ul>
<li><a href="">Головна</a></li>
<li><a href="">Блог</a></li>
<li><a href="">Магазин</a></li>
</ul>
/nav>
</header>
... Контент сторінки ...
```

При використанні в окремій публікації елемент header може включати назву статті, ім'я автора і дату публікації, як показано тут:

```
<article>
<header>
<h1>Додаткові відомості про WOFF</hl>
<p>Дженніфер Роббінс,
<time datetime="ll-ll-2011" pubdate>ll листопада, 2011</time></p>
</header>
<p>...тут починається текст статті...</p>
</article>
```

Елемент footer використовується для вказівки типу інформації, яка зазвичай повідомляється в кінці сторінки або статті, наприклад, ім'я автора, інформація про авторські права, що стосуються статті документи або елементи навігації, footer може застосовуватися до всього документу або бути пов'язаним з певним розділом або статтею.

Якщо нижній колонтитул знаходиться прямо всередині елемента body до або після решти

контенту цього елемента, то він застосовується до всеї сторінки. Якщо він розташований всередині елемента розділу (section, article, nav або aside), елемент footer аналізується як нижній колонтитул лише цього розділу. Зазначимо, що, хоч його і називають «нижнім колонтитулом», немає жодних вимог, які свідчать, що він повинен бути вказаний на останньому місці у документі чи елементі розділу.

У цьому прикладі ми бачимо типову інформацію, вказану у нижній частині статті або повідомлення у блозі, розмічену як елемент footer:

```
<article>
<header>
<h1>Додаткові відомості про WOFF</hl>
<p>Дженніфер Роббінс,
<time datetime="ll-ll-2011" pubdate>ll листопада, 2011</time></p>
</header>
<p>...тут починається текст статті...</p>
<footer>
<p><small>Всі права захищені &copy;2013 Дженнифер Роббинс.</small></p>
<nav>
<ul>
<li><a href="">Попередня</a></li>
<li><a href="">Наступна</a></li>
</ul>
</nav>
</footer>
</article>
```

Контрольні питання

1. Для чого потрібна мова HTML?
2. Що таке тег?
3. Що таке атрибут?
4. Як створити посилання в документі html?
5. Які базові теги розмітки Ви знаєте?
6. Які теги списків Ви знаєте?
7. Які теги забезпечують створення таблиць?

РОЗДІЛ 4
ФОРМИ ТА ЕЛЕМЕНТИ КЕРУВАННЯ HTML

4.1. Форми

Форми застосовуються для передавання даних від html-документу інтерактивним елементам сайту, наприклад сценаріям PHP, ASP.NET та ін. Помістивши у форму будь-які значення, відвідувач серверу натискає на відповідну кнопку, після чого введена ним інформація передається сценарію на сервері, який приймає управління процесом обробки даних.

Ось основна схема роботи серверної програми:

1. Відвідувач вводить в елементи керування, розташовані у Web-формі на Web-сторінці, необхідні дані.
2. Ввівши дані, відвідувач натискає розташовану в тій же Web-формі спеціальну кнопку - кнопку надсилання даних.
3. Web-форма кодує введені в неї дані та відправляє їх серверному додатку, розташованому за вказаною інтернет-адресою.
4. Web-сервер перехоплює надіслані дані, запускає серверну програму та передає дані йому.
5. Серверний додаток обробляє отримані дані.
6. Серверний додаток формує Web-сторінку з результатами обробки даних відвідувача та передає її Web-серверу.
7. Web-сервер отримує сформовану серверним додатком Web-сторінку та відправляє її відвідувачу.

Для того щоб успішно підготувати введені відвідувачем дані та надіслати їх серверному додатку, Web-форма має "знати" значення трьох параметрів:

1. Інтернет-адреси серверної програми.
2. Метод надсилання даних, що вказує вид, у якому дані будуть надіслані. Таких методів HTML підтримує два. Метод GET формує із введених відвідувачем даних <ім'я елемента управління>=<введені до нього дані> (кожен елемент управління обов'язково повинен мати унікальне в межах Web-форми ім'я.) Ці пари додаються праворуч до інтернет-адреси серверного додатка, відокремлюючись від нього символом ? (знак запитання); самі пари поділяються символами & (амперсанд). Отримана таким чином інтернет-адреса відправляється Web-серверу, який витягує з нього інтернет-адреси серверної програми та самі дані.

Метод POST також формує з введених даних пари виду <ім'я елемента управління>=<введені до нього дані>. Але відправляє він їх не до складу інтернет-адреси, а слідом за ним, як додаткові дані.

3. Метод кодування даних. Він актуальний тільки при надсиланні даних методом POST; Для методу GET його можна не вказувати.

Форми вставляють на веб-сторінки за допомогою елемента form. Він являє собою контейнер для всього вмісту форми, включаючи такі елементи, як текстові поля та кнопки, а також блокові елементи (наприклад, p та списки). Однак він не може містити в собі інший елемент form.

Загальний синтаксис директиви форми можна представити наступним чином:

<FORM ACTION=”посилання” METHOD=”значення”>

Зміст форми, що включає всі елементи керування, що використовуються

</FORM>

Значення атрибуту ACTION задає адресу, на яку буде відправлено дані. Якщо Web-форма служить для введення даних, призначених для обробки Web-сценарієм, як значення цього атрибуту тега вказують "порожню" інтернет-адресу «#».

Значення атрибуту METHOD встановлює метод передавання даних із форми на сервер: „GET” – строка запиту відображується у полі „адреса” браузеру або „POST” –не відображується. Якщо значення атрибуту METHOD не задано, то використовується метод “GET”.

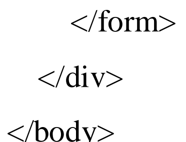
Кожен елемент керування, який розміщується у формі, для відправлення серверу введених в ньому користувачем даних повинен мати атрибут NAME. Тоді загальний синтаксис запиту, який формує браузер виглядатиме наступним чином:

адреса(вказана в атрибуті ACTION)?name1=значення1&name2=значення2...,

де name1, значення1 – значення атрибуту name1 та введені дані певного елементу керування. Все, що розміщено після символу „?” – строка запиту.

Наступний приклад виводить форму авторизації в браузері (рис. 4.1):

```
<body>
  <div>
    <form method="get" action="save.php">
      <p><label for="firstname">Ім'я:<input type="text" id="firstname" name="firstname"></label></p>
      <p><label for="lastname">Прізвище:<input type="text" id="lastname" name="lastname"></label></p>
      <p><input type="submit" value="Зберегти"></p>
```



```
        </form>
    </div>
</body>
```

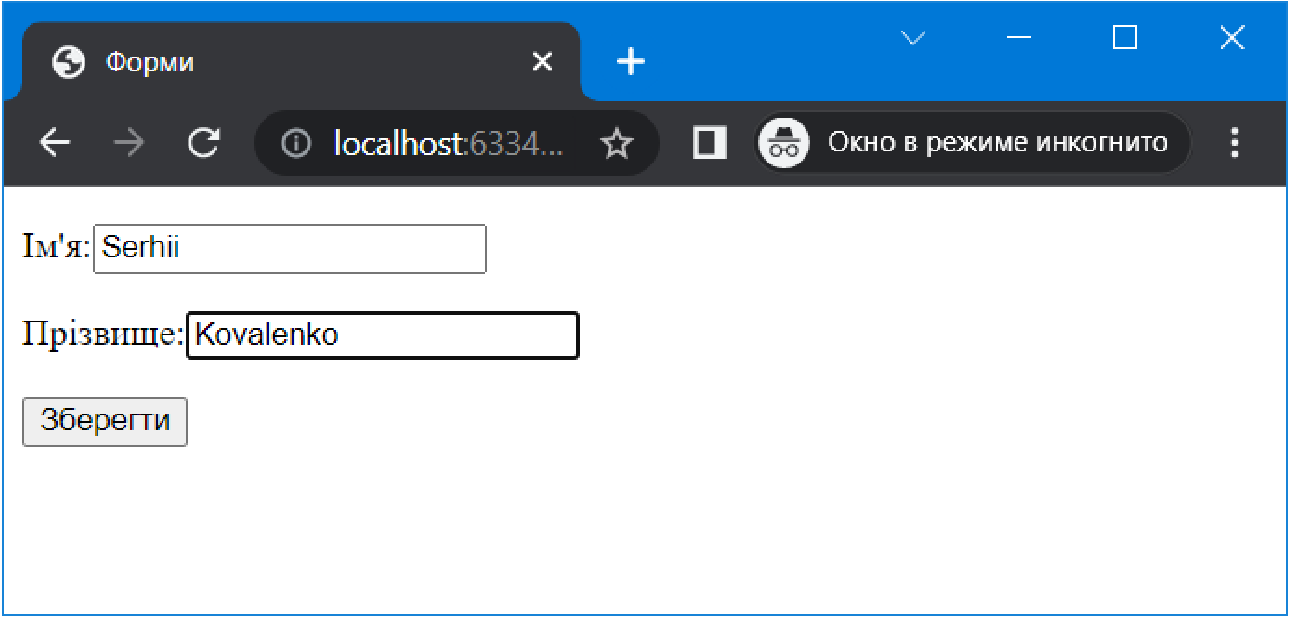


Рис. 4.1. Приклад форми

Як видно з прикладу, на сторінці розташовані два елементи керування „поля введення”, які визначаються тегом: <INPUT TYPE="TEXT"> і мають значення атрибутів NAME відповідно firstname та lastname. На сторінці у формі також розташована кнопка, яка забезпечується тегом: <INPUT TYPE="SUBMIT">. Саме тип “SUBMIT” вказує на те, що при натисканні цієї кнопки дані будуть передані для обробки на адресу, вказану в значенні атрибуту ACTION тегу <FORM>. Тобто, при натисканні кнопки у випадку, який показано на рис. 4.1., браузер сформує такий запит до серверу:

http://localhost save.php?firstname=Serhii&lastname=Kovalenko

Черговість включення до запиту значень введених користувачем в елементи керування визначається порядком розташування елементів на сторінці. У даному випадку, першим йде елемент із значенням атрибуту NAME “firstname”, а другим “lastname”.

Ця можливість забезпечує обернений зв’язок користувача із сервером. Web-майстру необхідно організувати на певній сторінці введення даних, а за допомогою сценарію – їх обробку, оскільки в сценарії є можливість отримувати передані у запиті значення.

Слід також зазначити, що в одному документі може бути більше однієї форми. браузер формує запит тільки із значень тих атрибутів, в яких можна вводити або обирати значення. Елементи керування не обов’язково мають бути розташованими саме у формі. Але розміщення їх у тегу <FORM> забезпечує „автоматичність” формування запиту до серверу.

4.2. Елементи керування “поля введення”

Даний клас елементів керування забезпечує користувачу введення текстової інформації. Елементи даного класу такі:

1. TEXT – одностročне поле введення. Синтаксис:

<INPUT TYPE=”TEXT” NAME=”значення” VALUE=”значення” SIZE=”значення” MAXLENGTH=”значення” TABINDEX=”номер у порядку обходу” DISABLED READONLY AUTOCOMPLETE="on|off" AUTOFOCUS PLACEHOLDER=”значення” REQUIRED PATTERN=”регулярний вираз”>

Атрибут VALUE задає початкове значення. Після введення користувачем тексту атрибут VALUE буде містити нове значення. Атрибут SIZE встановлює розмір елементу керування на формі як кількість символів, атрибут MAXLENGTH – максимальна кількість символів, які користувач може вводити в елемент керування. Наявність атрибуту DISABLED, забезпечує неможливість зміни інформації в елементі керування, тобто елемент буде недоступним і не зможе отримати фокус введення. READONLY забезпечує неможливість зміни даних в елементі керування, тобто на відміну від атрибуту DISABLED він надає користувачу можливість перенесення фокусу (дані можна скопіювати у буфер обміну але змінити - ні). TABINDEX задає число, яке визначає порядок обходу елементів при натисненні клавіші TAB. Тобто, від найменшого індексу до найбільшого.

Перше вдосконалення, яке HTML5 вносить у Web-форми - це можливість використання тексту, що заміщає (placeholder) в полі введення. Заміщаючий текст відображається всередині поля введення, поки поле є порожнім і не має фокусу введення. Як тільки користувач активізує цн поле або перейдете в нього натисканням клавіші <Tab>, текст, що заміщає, зникає.

Атрибут autofocus (HTML5) робить саме те, що можна припустити його назві: як тільки сторінка завантажується, він передає фокус введення в конкретне поле введення. Але, оскільки це не скрипт, а лише розмітка, ця поведінка буде несуперечливою на всіх Web-сайтах.

При наявності атрибуту required браузер відправить дані на сервер лише при умові, що клієнт заповнив дане поле.

При значенні атрибуту autocomplete=”off’ (атрибут введений в HTML5) браузер не буде відобаражти для елемента списоку вибору раніше введених даних.і

2. PASSWORD. Синтаксис:

<INPUT TYPE=”PASSWORD” NAME=”значення” VALUE=”значення” SIZE=”значення” MAXLENGTH=”значення” TABINDEX=”значення” DISABLED READONLY>

Як видно, ця конструкція повністю аналогічна попередній, за винятком того, що

інформація, яка вводиться користувачем, відображується на екрані символами „*”, щоб приховати текст від стороннього глядача.

3. TEXTAREA аналогічний елементу TEXT, але дозволяє вводити декілька текстових строк. Для розміщення даного елементу на сторінці використовується тег <TEXTAREA>, синтаксис якого наступний:

<TEXTAREA NAME=”значення” ROWS=”значення” COLS=”значення” TABINDEX=”значення” DISABLED READONLY>Початковий текст для редагування

</TEXTAREA>

Оскільки даний тег дозволяє вводити багатострочний текст, то він парний, тобто має тег закриття. Атрибути ROWS та COLS задають розмірність елементу на сторінці (кількість символів). Слід зазначити, що символ переведення рядка зберігається і при відображенні в елементі керування. Якщо включити теги HTML в документ, то вони з’являться як текст в елементі керування.

Приклад форми (рис. 4.2).

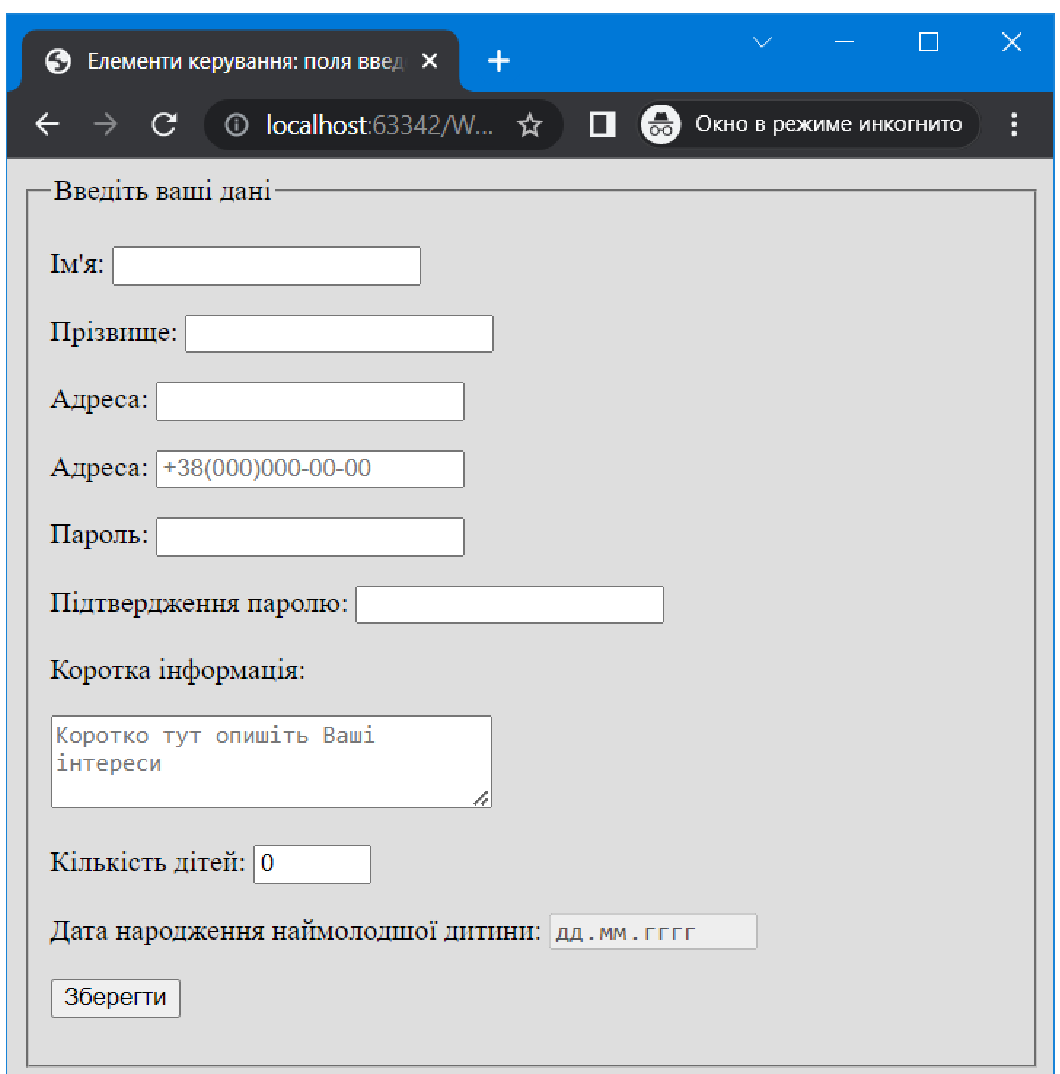


Рис. 4.2. Робота з полями введення

```
<body bgcolor="#dedede">
  <form>
    <fieldset>
      <legend>Введіть ваші дані</legend>
      <p><label for="firstname">Ім'я: <input type="text" id="firstname" name="firstname" required autofocus></label></p>
      <p><label for="lastname">Прізвище: <input type="text" id="lastname" name="lastname" required></label></p>
      <p><label for="address">Адреса: <input type="text" id="address" name="address"></label></p>
      <p><label for="phone">Адреса: <input type="text" id="phone" name="phone" placeholder="+38(000)000-00-00" required></label></p>
      <p><label for="password">Пароль: <input type="password" id="password" name="password" required></label></p>
      <p><label for="passwordrepeat">Підтвердження паролю: <input type="password" id="passwordrepeat" name="passwordrepeat" required></label></p>
      <p>Коротка інформація:</p>
      <p><textarea rows="3" cols="30" name="description" placeholder="Коротко тут опишіть Ваші інтереси"></textarea></p>
      <p><label for="children">Кількість дітей: <input type="number" id="children" name="children" value="0" min="0" max="20"></label></p>
      <p><label for="childdate">Дата народження наймолодшої дитини: <input type="date" id="childdate" name="childdate" disabled></label></p>
      <p><input type="submit" value="Зберегти"></p>
    </fieldset>
  </form>
</body>
```

Останній елемент управління типу date недоступний для користувача. За допомогою javascript можна зробити його активним при зміні попереднього текстового поля, що буде відрізнятися від 0. Тег <FIELDSET> забезпечує рамку навколо елементів керування, а <LEGEND> - надпис у лівому-верхньому куті рамки.

Крім того, тут використовуються також нові теги HTML5 для введення чисел (type=”number”) та дати (type=”date”).

4.3. Поля введення в HTML5.

До появи HTML5 вводити адреси електронної пошти, номери телефонів, URL-адреси або пошукові терміни можна було тільки за допомогою універсального поля введення тексту. У HTML5 значення полів введення email, tel , url і search підказують браузеру, якого роду інформацію чекати. Ці нові поля використовують ті ж самі атрибути, що і універсальне, описане раніше (name, maxlength , size , і value), а також низку нових атрибутів HTML5.

Поле введення значення для пошуку з'явилося у HTML 5. Воно нічим не відрізняється від звичайного поля введення за тим винятком, що з введеного значення автоматично видаляються переведення рядків. Поле введення значення для пошуку також створюється за допомогою непарного тегк

<INPUT>:

<INPUT TYPE="search" [VALUE="<початкове значення>"]

[SIZE="<розмір>"] [MAXLENGTH="<максимальна кількість символів>"]

[TABINDEX="<номер у порядку обходу>"] [ACCESSKEY="<швидка клавіша>"]

[DISABLED] [READONLY] [AUTOFOCUS]>

Значення "search" атрибуту тега TYPE вказує Web-браузеру створити поле введення значення для пошуку:

<FORM ACTION="#">

<P>Знайти: <INPUT TYPE="search" ID="keyword" NAME="keyword" SIZE="40"></P>

</FORM>

До нових полів введення відносяться також поля контактної інформації (адреси електронної пошти, телефону, та web-адреси):

<form>

<p><label>Email: <input type="email" id=”email” name=”email”></label></p>

<p><label>Web-сайт: <input type="url" id=”website” name=”website”></label></p>

<p><label>Телефон: <input type="tel" id=”phone” name=” phone”></label></p>

<p><input type="submit" value="Прийняти"></p>

</form>

Web-адреси, які суворі прихильники дотримання стандартів частіше називають URL, є ще одним типом спеціалізованого тексту. Синтаксис Web-адрес обмежений відповідними стандартами Інтернету. Для їх введення і валідації був введений відповідний тип:

<input type=”url”>

Якщо ви коли-небудь бронювали номер у готелі або авіаквитки у Всесвітньому павутинні, то, безсумнівно, користувалися невеликим віджетом календаря для вибору дати.

Швидше за все, маленький календар було створено за допомогою JavaScript. Мова HTML5 представляє шість нових типів введення даних, які перетворюють віджети для вибору дати та часу в одну із стандартних вбудованих можливостей візуалізації браузера (таку ж, як здатність відображати прапорці, списки та інші сучасні елементи).

Нові елементи введення, пов'язані із зазначенням дати та часу, є наступними:

<input type="date" name="ім'я" value = " 2013-01-14">

Створює елемент введення дати, такий як календар, що спливає для вказання дати (рік, місяць, день). Початкове значення має бути представлено у форматі міжнародної організації зі стандартизації, International Organization for Standardization, ISO (YYYY-MM-DD).

<input type="time" name= "ім'я" value = "03:13:00">

Створює елемент введення для вказівки часу (годин, хвилин, секунд, частки секунди) без часового поясу. Значення вказується наступним чином: hh:mm:ss.

<input type=”datetime" name="ім'я" value="2004-01-14T03:13:00-5:00">

Створює елемент введення поєднання дати та часу, який містить інформацію про часовий пояс. Значення є датою та час у форматі ISO з часового поясу щодо Універсального астрономічного часу, як і елемента time в розділі 5 (YYYY-MM-DDThh:mm:ssTZD).

<input type="datetime-local" name="ім'я" value = "2004-01-14T03:13:00">

Створює елемент введення поєднання дати та часу без урахування часового поясу (YYYY-MM-DDThh:mm:ss).

<input type="month" name="ім'я" value="2004-01">

Створює елемент введення певного місяця (YYYY-MM).

<input type="week" name ="ім'я" value="2004-IV2">

Створює елемент введення дати для позначення певного тижня року з використанням нумерації формату ISO (YYYY-W#).

У браузерах, які не підтримують ці елементи, поля введення дати та часу відображаються у вигляді робочого текстового поля.

Числові дані можна вводити за допомогою типів введення number та range. Для типу введення number браузер може надати віджет лічильника, щоб дати можливість вибрати конкретне числове значення (браузери, які не підтримують цей тип введення, відображатимуть текстове поле). Тип введення range, як правило, відображається у вигляді повзункового регулятора, який дозволяє користувачеві вибрати значення в межах вказаного діапазону.

<label>Кількість гостей <input type="number" name="guests" min="1" max="6"></label>

<label>Задоволеність (0 до 10)<input type="range" name="satis" min="0" max="10" step="1"></label>

Обом типу введення number і range притаманні атрибути min і max для вказівки

мінімальних та максимальних допустимих значень введення (Знову ж таки, браузер може перевірити, що введені користувачем значення знаходяться в межах допустимих).

Атрибути min та max необов'язкові, і ви можете також встановити один із них, опустивши інший.

Атрибут step дозволяє розробникам вказувати прийнятні збільшення числового введення. За замовчуванням встановлено значення 1. Якщо встановити його рівним .5, це дозволить використовувати значення 1, 1,5, 2, 2,5 і так далі; якщо 100 - 100, 200, 300, і так далі. Крім того, можна встановити значення any атрибута step, вказавши, що як збільшення приймаються будь-які величини.

Знову ж таки, браузери, які не підтримують ці нові типи введення, замість них відображають звичайне текстове поле — чудовий запасний варіант.

Мета елемента вибору кольору — створення палітри кольорів, що з'являється, схожої на ті, які використовуються в графічних редакторах для візуального вибору значення кольору. Значення наведено у шістнадцятковому форматі RGB (# RRGGBB):

```
<label>Улюблений колір: <input type="color" name="favorite"></label>
```

Елемент progress надає користувачам зворотний зв'язок, повідомляючи про стан поточних процесів, таких як завантаження файлу:

```
Відсоток завантаження: <progress max="100" name="fave">0</progress>
```

Meter схожий на елемент progress, але він завжди є вимірюванням у відомому діапазоні значень:

```
<meter min="0" max="100" name= "download" >50%</meter>
```

Приклад нових полів HTML5 (рис. 4.3):

```
<body bgcolor="#dedede">
  <form>
    <fieldset>
      <legend>Введіть ваші дані</legend>
      <p><label>Ім'я:<input type="text" id="firstname" name="firstname" pattern="[a-z]+"></label></p>
      <p><label>Email: <input type="email" id="email" name="email"></label></p>
      <p><label>Web-сайт: <input type="url" id="website" name="website"></label></p>
      <p><label>Телефон: <input type="tel" id="phone" name="phone"></label></p>
      <p><label>День народження: <input type="date" id="birthday" name="birthday"></label></p>
      <p><label>Кількість гостей: <input type="number" name="guests" min="1" max="6"></label></p>
```

```
            <p><label>Задоволеність (0 до 10):<input type="range" name="satis" min="0" max="10" step="1"></label></p>
            <p><label>Улюблений колір: <input type="color" name="favorite"></label></p>
            <p><input type="submit" value="Прийняти"></p>
        </fieldset>
    </form>
</body>
```

Рис. 4.3. Робота з полями введення

Як видно з рис. 4.3 і коду для валідації даних в HTML5 введено атрибут pattern, в значенні якого задається регулярний вираз для перевірки. У нашому випадку вираз “[a-z]+” означає, що ім’я можна вказати тільки маленькими буквами латинського алфавіту. Таких букв потрібно більше 0, що сигналізує квантіфікатор «+».

4.4. Кнопочні елементи керування

На формі можна розмістити кнопочні елементи керування таких типів:

1. SUBMIT, який забезпечує пересилання змісту всіх елементів керування форми на сервер (значення обробляються на сервері, адреса (URL) якого задається атрибутом ACTION тегу <FORM>). Синтаксис:

<INPUT TYPE="SUBMIT" VALUE="значення" DISABLED>

Атрибут VALUE задає надпис, який з'явиться на кнопці.

2. RESET – забезпечує „скидання" введених значень в усіх елементах керування на формі. Вона не передає ніяких значень на сервер. Основний синтаксис:

<INPUT TYPE="RESET" VALUE="значення" DISABLED>

3. BUTTON. Загальний тип кнопки. На відміну від попередніх у нього не має базової дії. Дію, яка відбувається при натисканні кнопки, описується розробником у функції тегу <SCRIPT>, яка реагує на подію натисканні кнопки. Опис цього виходить за рамки даної роботи, тому цій темі буде висвітлена в іншій. Основний синтаксис:

<INPUT TYPE="BUTTON" VALUE="значення" DISABLED>

4. CHECKBOX – прапорець, який працює як кнопка. Елемент представляє собою просту форму вибору. Він приймає одне з двох станів: „відмічено" – „не відмічено". Синтаксис:

<INPUT TYPE="CHECKBOX" DISABLED CHECKED NAME="значення">

Елемент керування приймає значення 0 („неправда") або 1 („істина"). Наявність атрибуту CHECKED визначає те, що при появі прапорець буде встановлений, тобто набуде значення 1.

5. RADIOBUTTON – перемикач – використовується для задавання списку опцій, подібно елементу CHECKBOX, але при цьому може бути вибраний тільки один з елементів списку. Щоразу, коли відвідувач обирає нову опцію, попередня відміняється. Синтаксис:

<INPUT TYPE="RADIO" DISABLED CHECKED NAME="значення" VALUE="значення">

Якщо кожний елемент управління CHECKBOX має унікальне ім'я (NAME), то група елементів RadioButton повинна мати одне. Це дозволяє браузеру визначити, скільки таких елементів управління входить в одну групу, і який елемент встановлений для кожної з груп. Щоб визначити, який з елементів групи буде встановлений першим, використовується атрибут CHECKED. Параметр VALUE забезпечує при виборі елементу включення його значення у запит. Наприклад, є 4 елементи групи, кожний з яких має ім'я "rb1". Значення VALUE першого – 1, другого 2 і т.д. Якщо обраний 2 елемент, то у запиті буде параметр: rb1=2.

6. FILE генерує на екрані кнопку, при натисканні на яку на екрані з'являється Провідник Windows, що дозволяє обрати файл на локальному комп'ютері для відправки на сервер. Даний

елемент використовується, в основному, у формах відправки повідомлень електронної пошти з „прив'язаним" до неї файлом, а також для завантаження зображень на сервер. Зазвичай поряд із кнопкою відображається текстове поле, куди автоматично заноситься ім'я файлу з маршрутом. Синтаксис:

<INPUT TYPE="FILE" NAME="значення">

7. IMAGE створює кнопку відправки, аналогічну елементу SUBMIT, але з використанням графічного зображення. Адреса зображення вказується стандартно в значенні атрибуту SRC. При цьому сам елемент може містити власні атрибути, що застосовуються у тегу <IMG>, в тому числі ALIGN, ALT та ін. Синтаксис:

<INPUT TYPE="IMAGE" SRC="URL адреса" NAME="значення">

Результат інтерпретації коду із зазначеними елементами показано на рис. 4.4.

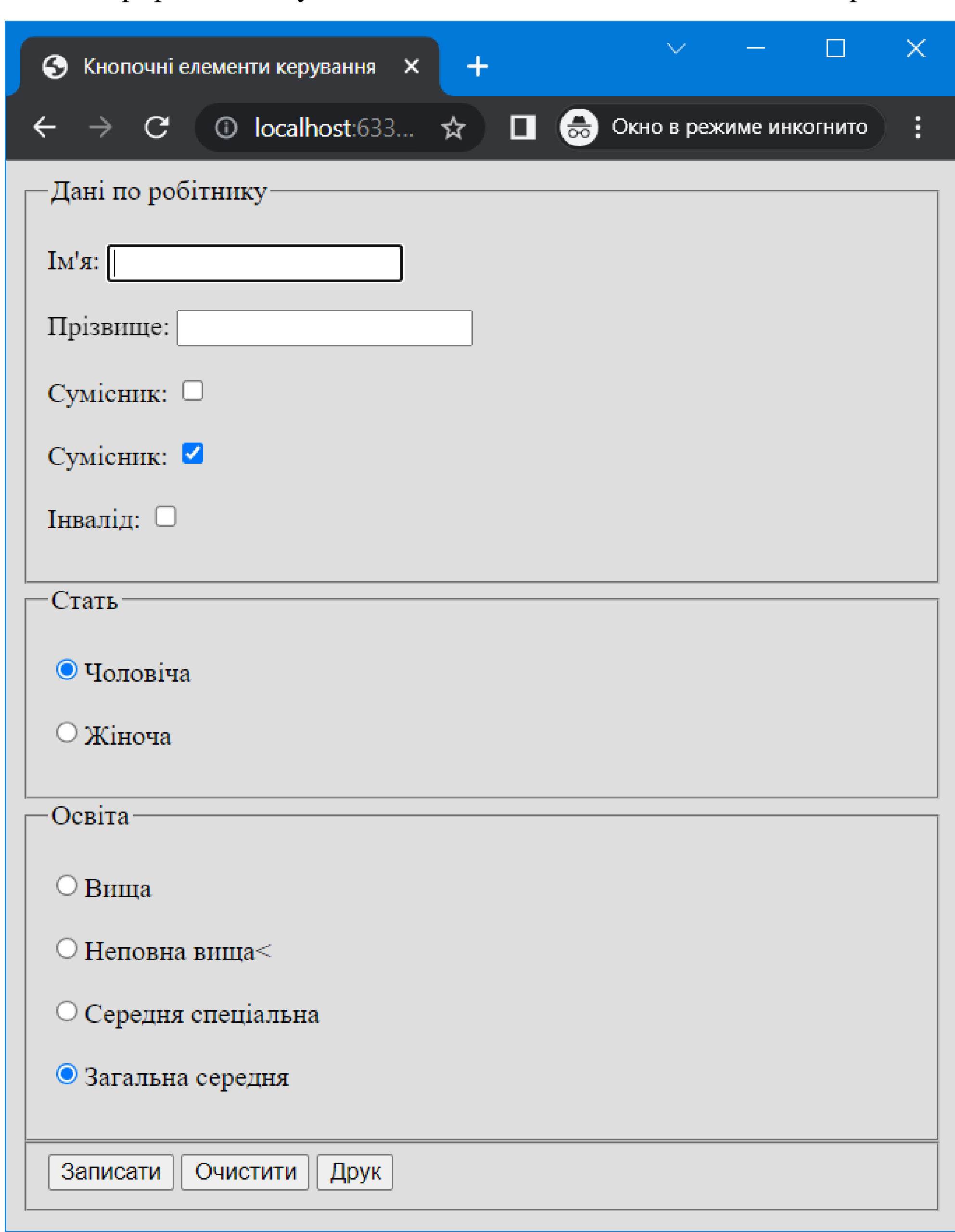


Рис. 4.4. Робота з кнопочними елементами керування

Код сторінки:

```
<body bgcolor="#dedede">
  <form action="#">
    <fieldset>
      <legend>Дані по робітнику</legend>
      <p><label>Ім'я: <input type="text" id="firstname" name="firstname" required autofocus></label></p>
      <p><label>Прізвище: <input type="text" id="lastname" name="lastname" required></label></p>
      <p><label>Сумісник: <input type="checkbox" name="parttimeworker"></label></p>
      <p><label>Сумісник: <input type="checkbox" name="laborunion" checked></label></p>
      <p><label>Інвалід: <input type="checkbox" name="invalid"></label></p>
    </fieldset>
    <fieldset>
      <legend>Стать</legend>
      <p><input type="radio" name="sex" value="1" checked>Чоловіча</p>
      <p><input type="RADIO" name="sex" value="2">Жіноча</p>
    </fieldset>
    <fieldset>
      <legend>Освіта</legend>
      <p><input type="radio" name="education" value="1">Вища</p>
      <p><input type="radio" name="education" value="2">Неповна вища<</p>
      <p><input type="radio" name="education" value="3">Середня спеціальна</p>
      <p><input type="radio" name="education" value="4" CHECKED>Загальна середня</p>
    </fieldset>
    <fieldset>
      <input type="submit" value="Записати">
      <input type="reset" value="Очистити">
      <input type="button" value="Друк">
    </fieldset>
  </form>
</body>
```

Як видно з коду HTML та його інтерпретації, на формі розміщено три прапорці, другий

з яких має початкове значення „вибраний”. Є також дві групи перемикачів, у першій з них значення атрибуту NAME дорівнює “sex”, а в другій – “education”. Саме це є ознакою залучення певних перемикачів до групи. Значення VALUE кожного з перемикачів сигналізує, чому буде дорівнювати “sex” або “education” (дивлячись, до якої групи належить перемикач). В залежності від обраної радіокнопки у блоці стать параметр буде відісланий на сервер з відповідним значенням, що відповідає значенню атрибуту VALUE обраного перемикача. Так при виборі чоловічої статі на сервер буде відправлено sex=1 як один із параметрів.

Веб-форми дозволяють надсилати на сервер не лише дані. Можна також завантажувати файли з комп'ютера користувача. Наприклад, друкарня може приймати через веб-форму графічні макети як додаток до замовлень на виготовлення візитних карток. Редакція журналу за допомогою веб-форми може приймати фотографії на фотоконкурс.

За допомогою елемента вибору файлу користувач може вибрати документ, який потім буде відправлено на сервер разом з рештою даних з полів форми. Додати його можна за допомогою добре знайомого нам елемента input, в якому атрибут type приймає значення "file".

Наведена нижче розмітка демонструє елемент вибору файлу:

<form action="/client.php" method="POST" enctype="multipart/form-data">

<label>Завантажте фотографію <input type=”file” name=”photo” size=”28” accept=".jpg, .jpeg, .png" ></label>

</form>

У різних браузерах поле вибору файлу може відображатись по-різному.

Це може бути текстове поле з кнопкою «Огляд» для вибору файлу на жорсткому диску, або кнопка з іншою назвою та супроводжуючим написом, знайдений файл.

Важливо, що для форми, яка містить елемент вибору файлу, в якості типу кодування (enctype) слід вказувати значення multipart/form-data і надсилати дані методом POST.

У цьому прикладі атрибут size визначає довжину поля файлу в символах (хоча її також можна задавати за допомогою правила CSS), якщо у браузері відображається поле.

Буває, що необхідно надіслати програмі-обробнику інформацію, яка не виходить від користувача. У цих випадках застосовують приховані елементи форми, які передають на сервер необхідну інформацію разом з іншими даними в момент їх надсилання, але при цьому невидимі під час перегляду форми в браузері.

Щоб приховати елемент форми, слід атрибуту type елемента input привласнити значення hidden. Єдине його завдання – передати пару ім'я/значення на сервер у момент надсилання даних форми. Наприклад, щоб обмежити розмір файлу що може бути завантажений на сервер можна використати таке:

<input type="hidden" name="MAX_FILE_SIZE" value="30000">

4.5. Елемент керування "список"

Елемент керування select є списком із декількох пунктів. Користувач може нічого не вибрати, або вибрати один чи кілька пунктів із списку. Список обмежений парою тегів <SELECT>. Кожен елемент списку описується окремим тегом <OPTION>. Синтаксис тегів:

```
<SELECT NAME=”ім’я” MULTIPLE SIZE=”ціле число”>
<OPTION VALUE=”значення 1”>Найменування пункту 1</OPTION>
<OPTION VALUE=”значення 2”>Найменування пункту 2</OPTION>
<OPTION VALUE=”значення 3>Найменування пункту 3</OPTION>


…
<OPTION VALUE=”значення N”>Найменування пункту N</OPTION>
</SELECT>
```

Атрибут SIZE задає кількість видимих рядків. Якщо його опустити, то список буде представлений одним рядком, а відвідувачу необхідно буде використовувати клавіші керування курсором або мишу, щоб переглянути доступні пункти. Якщо список містить більше пунктів, ніж вміщається в його рамках, то вертикальна смуга прокрутки автоматично надає допомогу відвідувачу у виборі необхідного елементу.

Атрибут MULTIPLE вказує, чи може відвідувач сторінки обрати кілька пунктів списку, використовуючи клавіші Shift або Ctrl і одночасно вибираючи пункти зі списку. Якщо не вказати даний атрибут, то при виборі нового елементу попередній відміняється.

У тега <OPTION> є атрибут VALUE, який задає значення, що попаде у строку запиту, при виборі даного елементу. Для початкового вибору однієї або більше опцій використовується атрибут SELECTED.

Необхідно також підкреслити, що між тегами <SELECT> та </SELECT> і <OPTION> та </OPTION> не можна задавати інших тегів, тобто картинок, кнопок чи інших елементів керування.

Елемент <datalist>, що з'явився в HTML5, використовується лише в формах. Він дозволяє завчасно побудувати список пунктів, які в подальшому будуть пропонуватися у якості заповнення полів введення.

```
<datalist id=”mydata”>
<option value=”123” label=”phone1”>
<option value=”234” label=”phone2”>
</datalist>
<input type=”tel” name=”phone” list=”mydata”>
```

Приклад:

```
<body bgcolor="#dedede">
  <form action="#">
    <fieldset>
      <legend>Оплата</legend>
      <p>Спосіб оплати:
        <select name="payment" size="4">
          <option value="1" selected>Google Pay</option>
          <option value="2">LiqPay</option>
          <option value="3">Картка</option>
          <option value="4">Оплата за реквізитами</option>
        </select>
      </p>
    </fieldset>
    <fieldset>
      <legend>Доставка</legend>
      <p>Спосіб доставки
        <select name="delivery">
          <option value="1">Доставка кур'єром</option>
          <option value="2" selected>Нова пошта</option>
          <option value="3">Укрпошта</option>
        </select>
      </p>
      <p>Місто:
        <input type="text" name="city" list="cities">
        <datalist id="cities">
          <option value="1" label="Київ">
          <option value="2" label="Кривий Ріг">
          <option value="3" label="Дніпропетровськ">
          <option value="4" label="Херсон">
          <option value="5" label="Кіровоград">
          <option value="6" label="Миколаїв">
        </datalist>
      </p>
    </fieldset>
```

```
        <fieldset>
            <input type="submit" value="Оформити">
            <input type="reset" value="Очистити">
        </fieldset>
    </form>
</body>
```

Результат інтерпретації коду показано на рис. 4.5.

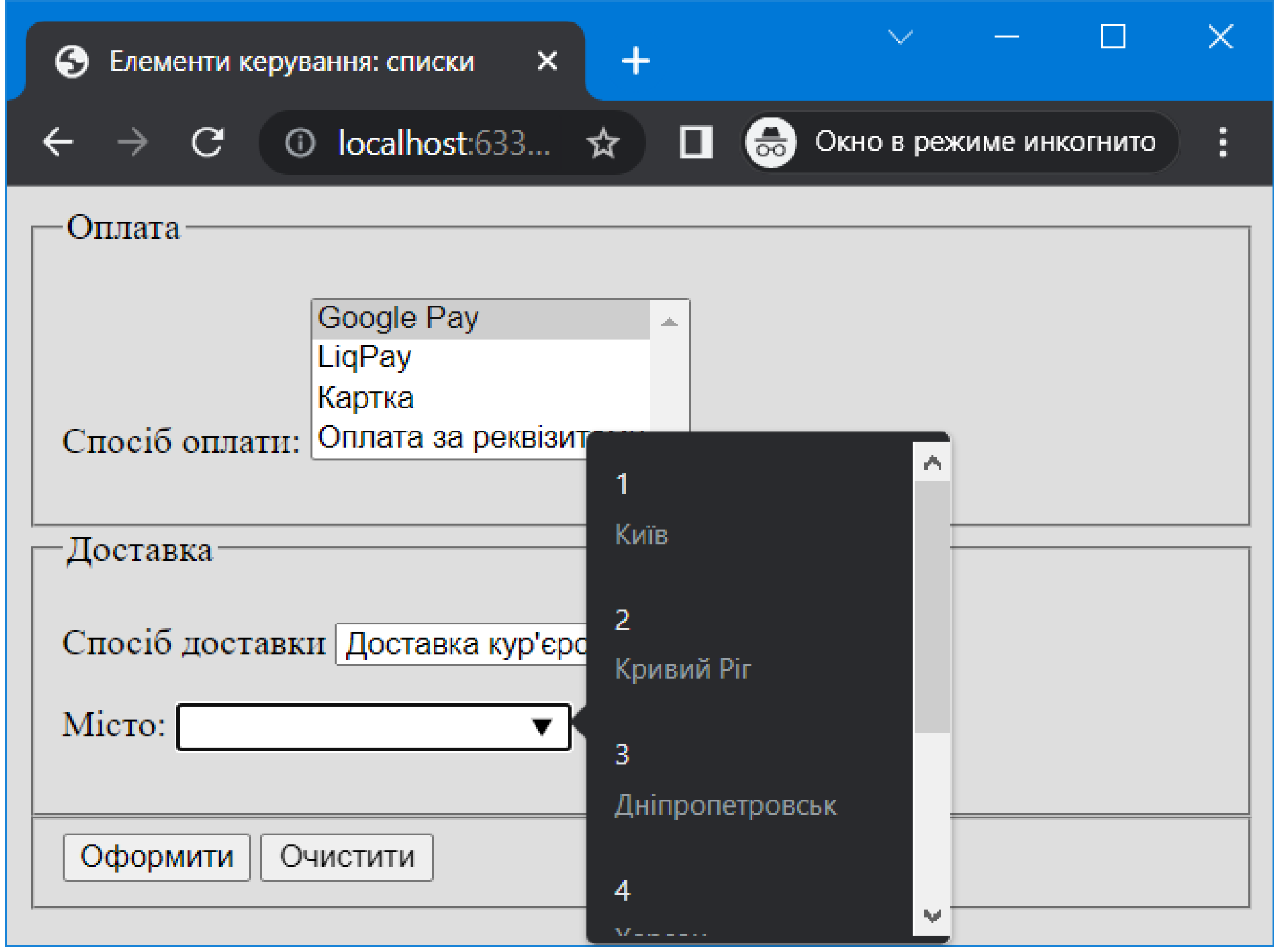


Рис. 4.5. Робота з елементами керування типу „Список”

Як видно з коду та рис. 4.5, на формі знаходяться два елементи керування типу список, перший – багаторядкового типу, другий – однорядкового. Перший може показати одночасно чотири опції, що задає параметр SIZE. При появі сторінки в першому списку обраний перший елемент (оскільки саме він має атрибут SELECTED), у другому – другий.

Контрольні питання

1. Призначення форм HTML?

2. Як створити поле введення?
3. Як створити кнопку?
4. Як створити радіокнопки?
5. Як передати файл на сервер?
6. Як організувати списки на сторінці?
7. Які атрибути форми Ви знаєте?

РОЗДІЛ 5
КАСКАДНІ ТАБЛИЦІ СТИЛІВ (CSS)

5.1. Поняття CSS

Мовою HTML стилі тегів визначаються за допомогою надання певних значень відповідним атрибутам (шрифти, вирівнювання та ін.). Але, якщо на сторінці, наприклад, є багато абзаців, які необхідно вирівняти по центру? В такому випадку потрібно абзацу задавати значення атрибуту ALIGN: <P ALIGN=CENTER>. Таким чином, необхідно це робити для кожного абзацу, хоча усі вони мають однаковий стиль вирівнювання. Проблема в цьому випадку полягає в тому, що чим більше абзаців, тим більший буде програмний код. Навіть гірше – зі збільшенням кількості тегів зростає і обсяг файлу html-документу. Це впливає на швидкість його завантаження з серверу та складність подальшого програмування, оскільки текст html-коду із кожним новим записом все важче сприймається.

В такому випадку, перед розробниками стандарту HTML постала проблема: чи можна описати усі параметри тегів, що використовуються на сторінці, лише один раз. Це необхідно було зробити, але так, щоб не вплинути на принципи стандарту HTML і не змінити їх. Необхідність вирішення цієї проблеми диктувалась і зростанням кількості параметрів (наприклад, розробнику необхідно змінити товщину не всієї рамки навколо таблиці, а лише лівої чи правої межі). Для вирішення цих проблем і було створено новий стандарт CSS (Cascading Style Sheets – Каскадні таблиці стилів).

CSS, на відміну від HTML використовує дещо інший алгоритм опису елементів сторінки. Лише один раз можна вказати властивості кожного елементу в текстовому файлі з розширенням .css (наприклад mystyle.css) та підключити цей файл до html-документу. В такому випадку браузер зчитує значення параметрів (розмір шрифта, колір та ін.) кожного тегу вже зі створеного css-файлу. Більш того, оскільки стилі описуються в окремому файлі, то його можна підключити до будь-якої кількості web-сторінок, що дозволить уникнути необхідності призначати властивості кожному конкретному об’єкту (тегу). Є ще одна перевага такого підходу: якщо необхідно змінити стиль оформлення будь-якого елементу усіх web-сторінок, достатньо виправити один або декілька рядків в одному файлі – файлі зі стилями. Таким чином, можна виділити три основні переваги застосування таблиці стилів:

1. Можна створити таблицю стилів і застосувати її до одного або декількох документів. Ця універсальність надає змогу змінювати вигляд усіх сторінок, змінюючи лише таблицю стилів у файлі.

2. Надання більшого контролю керування текстом: з’явилися нові властивості, за

допомогою яких можна створювати спливаючі меню, розміщати один текст над іншим (причому в незалежності від їх послідовності у коді HTML), створювати ефекти тіні для тексту та ін.

3. Таблиці стилів, на відміну від інших методів управління зовнішнім виглядом, дозволяють зберігати в різних місцях текст та інформацію про те, як він повинен виглядати, що дозволяє зменшити розміри файлів документів.

Технологія каскадних таблиць стилів (Cascading Style Sheets, CSS) призначена для створення представлення Web-сторінок чи навіть таблиць стилів. Таблиця стилів містить набір правил (стилів), що описують оформлення самої Webсторінки та окремих її фрагментів. Ці правила визначають колір тексту та вирівнювання абзацу, відступи між графічним зображенням, наявність та параметри рамки у таблиці, колір фону Web-сторінки та багато іншого.

Кожен стиль має бути прив'язаний до відповідного елемента Web-сторінки (або самої Web-сторінці). Після прив'язки параметри, що описуються вибраним стилем, починають застосовуватися до даного елементу. Прив'язка може бути явна, коли ми самі вказуємо, який стиль якого елемента Web-сторінки прив'язаний, або неявна, коли стиль автоматично прив'язується до всіх елементів Web-сторінки, створених за допомогою певного тегу.

Таблиця стилів може зберігатися прямо в HTML-коді Web-сторінки або в окремому файлі.

5.2. Створення стилів. Селектори і властивості.

Таблиця стилів складена з однієї і більше інструкцій (званих правилами або наборами правил), які описують як елемент або група елементів мають відображатися. Кожне правило вибирає елемент і повідомляє, як він має виглядати.

Відповідно до термінології CSS існує дві головні частини правила - це селектор, що встановлює елемент або елементи, на які треба впливати, і визначення, що надає інструкції уявлення. Визначення, своєю чергою, складено з властивості та його значення, розділених двокрапкою:

```
<селектор> {
<властивість стилю 1>: <значення 1>;
<властивість стилю 2>: <значення 2>;
. . .
<властивість стилю n-1>: <значення n-1>;
<властивість стилю n>: <значення n>
```

}

Ось основні правила створення стилю:

- визначення стилю включає селектор та список властивостей стилю з їхніми значеннями;
- селектор використовується для прив'язки стилю до елемента Web-сторінки, на який він має поширювати свою дію. Фактично селектор однозначно ідентифікує цей стиль;
- за селектором, через пробіл, вказують список властивостей стилю та їх значень, обрамлених у фігурні дужки.
- властивість стилю представляє один із параметрів елемента Web-сторінки: колір шрифту, вирівнювання тексту, величину відступу, товщину рамки та ін. Значення властивості стилю вказують після нього через символ: (двокрапка). У деяких випадках значення властивості стилю укладають у пари <властивість стилю>:<значення> відокремлюють один від одного символом «;» (точка з комою);
- між останньою парою <властивість стилю>:<значення> і фігурною, що закриває дужкою символ «;» не ставлять, інакше деякі Web-браузери можуть неправильно обробити визначення стилю;
- визначення різних стилів поділяють пробілами або переведеннями рядків.

Наприклад:

P { color: #0000FF }

де P – це селектор. Він є ім'ям тега <P>;

color – це властивість стилю. Він задає колір тексту;

#0000FF – це значення властивості стилю color. Він представляє код синього кольору, записаний у форматі RGB.

Коли Web-браузер зчитує описаний стиль, він автоматично застосує його до всіх абзаців Web-сторінки (теги <P>). Це типовий приклад неявної прив'язки стилю. Стиль, що ми розглянули, називається стилем перевизначення тегу. Як селектор тут вказано ім'я HTML-тегу, що перевизначається цим стилем без символів < та >.

Ось стиль перевизначення тега <EM>:

EM { color: #00FF00; font-weight: bold }

Будь-який текст тегу <EM>, Web-браузер виведе зеленим напівжирним шрифтом. Властивість стилю font-weight задає ступінь "жирності" шрифту,

а його значення bold – напівжирний шрифт.

А це стиль перевизначення тега <BODY>:

BODY { background-color: #000000; color: #FFFFFF }

Він задає для всієї Web-сторінки білий колір тексту (RGB-код #FFFFFF) та чорний колір

фону (RGB код #000000). Властивість стилю background-color, як ми вже зрозуміли, задає фоновий колір.

Універсальний селектор

Відповідає будь-якому HTML-елементу. Наприклад, * {margin: 0;} обнуляє зовнішні відступи для всіх елементів сайту. Також селектор може використовуватися в комбінації із псевдокласом або псевдоелементом: *:after {CSS-стилі}, *:checked {CSS-стилі}.

5.3. Способи задавання стилів

Існують такі способи задавання стилю тегів:

1. Розташування інформації про стилі всередині тегів. Для цієї мети введений атрибут STYLE. У них немає селектора, оскільки вони розміщуються безпосередньо в потрібному тегу. Селектор у цьому випадку не потрібен. У них немає фігурних дужок, оскільки немає необхідності відокремлювати список атрибутів стилю від селектору. Вбудований в тег стиль може бути прив'язаний лише до одного тегу - того, в якому він знаходиться. Визначення такого стилю вказується як значення атрибута стилю:

<ТЕГ STYLE=”властивість1: значення1; властивість2: значення2;... властивістьN: значенняN”>

У значенні атрибуту “STYLE” міститься перерахування властивостей та їх значень. Наприклад: <P STYLE=”font-size: 14pt; color: #000080;”>. Все, що розташовано в значенні атрибуту, інтерпретується браузером на основі спеціального алгоритму. Даний атрибут введено для збереження можливості задавання стилю окремому тегу та використання нових властивостей при форматуванні. Приклад:

```
<body style="background: url(img/back.gif)">
  <div style="text-align: center;margin-bottom:10px;">
    <a style="padding:5px;font-size: 10pt;width: 110px;display: inline-block;
          color: #ffffff;background-color: #748c94;
          line-height: 20px;text-decoration: none" href="#">Продукція</a>
    <a style="padding:5px;font-size: 10pt;width: 110px;display: inline-block;
          color: #ffffff;background-color: #748c94;
          line-height: 20px;text-decoration: none"  href="#">Новини</a>
    <a style="padding:5px;font-size: 10pt;width: 110px;display: inline-block;
          color: #ffffff;background-color: #748c94;
          line-height: 20px;text-decoration: none" href="#">Контакти</a>
  </div>
```

```
<div style="width:456px;margin: auto;text-align: center;background-color:#a3dcc8;padding:5px 0">
  <a href="#">Інтернет-магазин</a>
</div>
<div style="width:456px;margin: auto">
   <img style="width:100%" src="img/home.jpg">
</div>
</body>
```

Результат показано на наступному рис. 5.1.

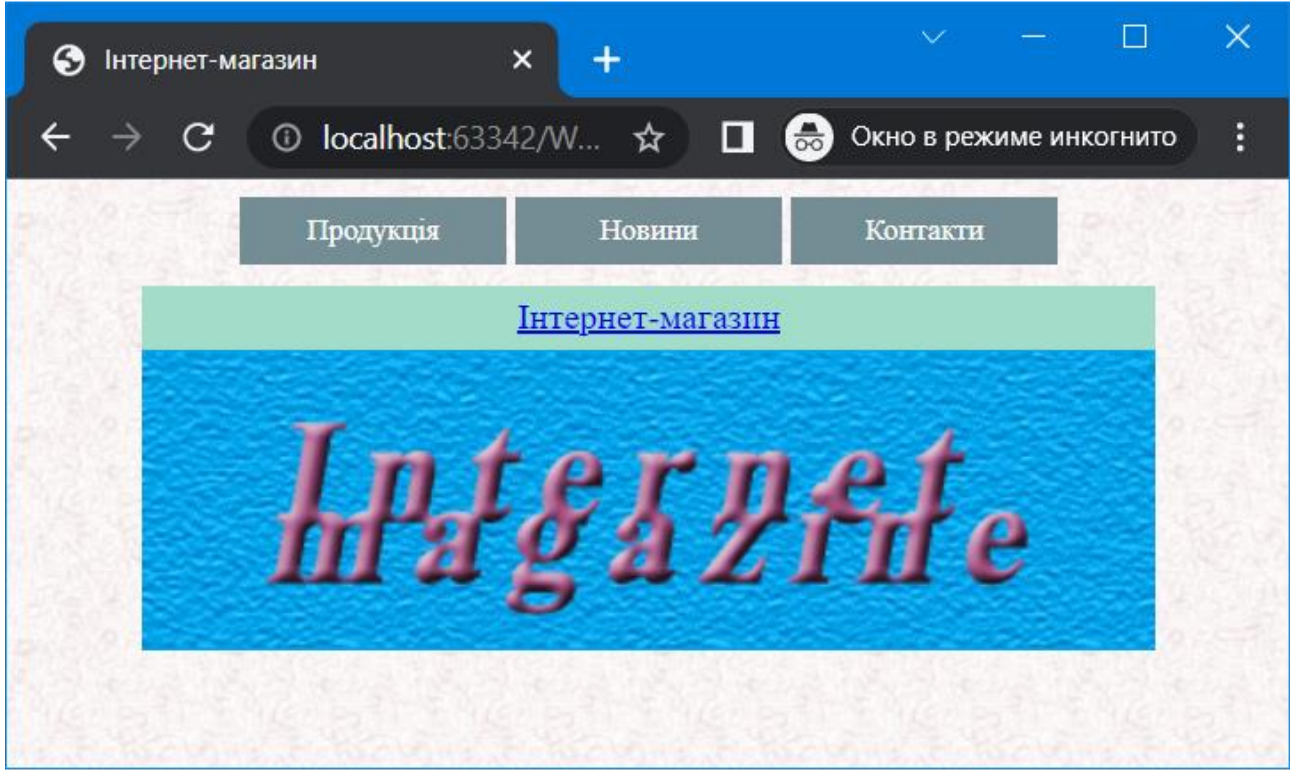


Рис. 5.1. Використання стилів при створенні сторінки

Як видно з коду та рис.5.1, на сторінці розташовані посилання, що мають однаковий стиль оформлення, у тому числі ширину (110 пікселів) та висоту (30 пікселів). Властивість text-decoration задає ефекти тексту: підкреслення, перекреслення. У даному випадку посилання не мають жодного з цих ефектів, що забезпечує значення „none". Слід відмітити, що у тега:

Як видно у даному випадку, одні і тіж властивості використовуються для кожного тегу <A>, і досить неефективно, оскільки при зміні стилю оформлення необхідно змінювати властивості стилю для кожного тегу. Це можна виправити використавши наступний спосіб задавання стилів.

2. Використання для задавання стилів тегу <STYLE>. В цьому випадку у заголовку <HEAD> використовується наступний синтаксис:

<STYLE type=”text/css”> … інформація про стилі … </STYLE>

У такому разі інформація про стилі тегів записується наступним чином:

ТЕГ1 [, ТЕГ2 [, ТЕГN]] { властивість1: значення1; властивість2: значення2;... властивістьN: значенняN }

Така директива CSS називається **селектором**. Все, що міститься між {} прийнято називати визначником селектора. Кожне визначення містить перерахування властивостей і значень. Наприклад, щоб задати колір фону усіх заголовків таблиць, необхідно написати так:

TH {background-color: #A3DCC8;}

А щоб встановити чорний колір для заголовків таблиць і комірок таблиць та розмір шрифту 14 пунктів:

TH, TD {color: #000000; font-size: 14pt;}

Таким записом можна зазначити властивості для групи тегів.

Щоб встановити шрифт Verdana для трьох типів заголовків необхідно записати так:

H1, H2, H3 {font-family: Verdana;}

Якщо клієнт не має доступу до цього шрифту, то буде застосовано базовий шрифт, який використовується браузером.

Такий спосіб задавання стилю надає йому універсальності, адже можна задати стиль один раз для багатьох тегів. Це забезпечує також зменшення розміру Web-сторінки.

Розглянемо модифікацію попереднього прикладу:

```
< <!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Інтернет-магазин</title>
 <style type="text/css">
 <!--
 a {
  padding:5px;font-size: 10pt;width: 110px;display: inline-block;
  color: #ffffff;background-color: #748c94;
  line-height: 20px;text-decoration: none
 }
 h1, h2, h3 {color: #800000; font-size: 15pt}
 -->
 </style>
</head>
```

```
<body style="background: url(img/back.gif)">
  <div style="text-align: center;margin-bottom:10px;">
    <a href="#">Продукція</a>
    <a href="#">Новини</a>
    <a href="#">Контакти</a>
  </div>
  <div style="width:456px;margin: auto;text-align: center;background-color:#a3dcc8;padding:5px 0">
    <h1>Інтернет-магазин</h1>
    <h2>Інтернет-магазин</h2>
    <h3>Інтернет-магазин</h3>
  </div>
  <div style="width:456px;margin: auto">
     <img style="width:100%" src="img/home.jpg">
  </div>
</body></html>
```

Тепер немає потреби задавати стиль для кожного тегу <A>, оскільки в даному випадку вигляд усіх посилань буде однаковим за рахунок використання селектора. Така об'ява у тегу стилю: a { padding:5px;font-size: 10pt;...} примушує браузер замінити певні базові параметри, які він використовував для тегу <A> на вказані селектором. Крім того, за рахунок надання одного й того ж шрифту тегам <H1>, <H2> та <H3>, вони перестали відрізнятися. Тричі буде виведено текст „Інтернет-магазин" з одним і тим же форматуванням.

Старі браузери не підтримують інтерпретацію стилів, оскільки в них не закладений алгоритм надання тегам стилів. Вони будуть намагатися вивести стилі просто як текст. Для цього всі селектори необхідно узяти в текст коментарію <!--…-->. Оскільки інтерпретатор сучасних браузерів має такий алгоритм, то, незважаючи на присутність тегу коментарю, він його проігнорує, але це справедливо в даному випадку тільки для тегу <STYLE>.

Вищеописаний спосіб забезпечує єдине оформлення, але в межах однієї сторінки. Єдиний інтерфейс сторінок можна організувати за допомогою наступного способу.

3. Задавання стилю в окремому файлі стилів (CSS-файлі). Усі стилі, що задані в атрибуті <STYLE> можна винести в окремий текстовий файл з розширенням .css (наприклад, mystyle.css). Підключення файлу здійснюється так (<HEAD>):

```
<link rel="stylesheet" type="text/css" href="mystyle.css">
```

Такий спосіб забезпечує універсальність таблиці стилів та застосування її до документу або групи документів. Можна змінювати інтерфейс сторінок, змінюючи таблицю стилів.

5.4. Принцип спадкування. Контекстні селектори

Спадкування дозволяє не описувати властивості усіх можливих тегів. Якщо ми не задаємо властивості для тега <EM>, він спадкує властивості тегу, в якому знаходиться:

<H3>Частина 3. <EM>Таблиці стилів.</EM> Спадкування</H3>

Якщо таблиця стилів визначає для усіх заголовків H3 зелений колір: H3 {color:#00ff00;}, та в ній нічого не має про тег <EM>, то слова „Таблиці стилів" будуть зеленими, як і весь текст тегу <H3>. <EM> успадкує стилі від „батька", тобто від тегу <H3>, в якому він сам знаходиться. І тільки, коли спеціально для тегу <EM> задати стиль: EM {color:#0000ff;}, то текст, який в ньому знаходиться, буде синім.

Інший приклад. Включимо в css-файл наступний запис:

```
body{
        background-color: #00ff00;
        color: #ff0000;
        font-size: 10pt;
}
```

Оскільки усі елементи web-сторінок є дочірніми директивами тегу <BODY>, то всі вони автоматично успадкують його властивості, тобто матимуть зелений колір фону, червоний – переднього плану та розмір шрифту 10 пунктів.

Селектори можуть теж здійснювати спадкування. Наприклад, теги <A> та <TD> повинні мати однакові значення кольору та розміру шрифту, але різні значення кольору фону. Перший варіант задавання стилю такий:

```
A{font-size: 10px;color:#ff0000; background-color:#0000ff;}
TD{font-size: 10px;color: #ff0000; background-color:#000000;}
```

другий варіант має наступний запис:

```
A, TD{font-size: 10px;color:#ff0000;}
A{ background-color: #0000ff;}
TD{background-color: #000000;}
```

В цьому випадку другий та третій селектори доповняться властивостями першого (font-size та color). Другий підхід можна здійснювати за умови, що необхідно створити єдине оформлення щодо розміру шрифту та кольору для посилань та комірок таблиць. В іншому випадку привабливішим є перший варіант.

Контекстні селектори

Інша корисна особливість спадкування полягає в тому, що воно може бути використане для застосування стилів контекстно. Наприклад:

H3 {color:green;}

EM {color:blue;}

Якщо необхідно, щоб колір переднього плану тексту тегу <EM> був жовтим у випадку знаходження цього тегу між тегами <H3> </H3> – і тільки в них, тоді:

H3 EM {color:yellow;}

Такі оголошення називаються **контекстними селекторами**, тому що вони визначають значення властивостей в залежності від контексту. Можна також визначити значення для різних контекстів в одній строчці. Наприклад, строчка: H3 EM, H2 I {color: yellow} призведе до такого ж результату, що і дві інші: H3 EM {color: yellow} H2 I {color: yellow}.

Приклад:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Інтернет-магазин</title>
  <style type="text/css">
    <!--
    body {
      background-color: gold;
      color: red;
      font-size: 10pt;
    }
    h2,h3 {color:green;}
    h3 i {font-size: 18pt;}
    h2 {font-size: 20pt;}
    -->
  </style>
</head>
<body>
  <h3>Це H3</h3>
  <i>Це стандартний курсив</i>
  <h2>Це зелений колір, розмір шрифту 20pt<br>
  <b>Це теж зелений та розмір 20pt</b></h2>
  <h3>Це зелений колір<br>
  <i>Це розмір шрифту 18 pt</i></h3>
```

</body>

</html>

Як видно з прикладу, текст тегів H2 та H3 буде виведено зеленим кольором, а завдяки об’яві селектора: H2 {font-size: 20pt;}, текст тегу H2 буде виведений з використанням розміру шрифту 20 пунктів. Текст тегу <I>, який знаходиться у тегу <H3>, буде виводитись шрифтом розміром у 18 пунктів, оскільки є об’ява: H3 I {font-size: 18pt;}.

5.5. Задавання стилів за допомогою Класів

CSS є достатньо гнучкою структурою, що дозволяє описувати не тільки параметри для будь-яких елементів таблиці, але назначати різноманітні властивості одним і тим же елементам. Припустимо, в коді html-документу є декілька варіантів розташування тексту: частина абзаців позиціонується по ширині сторінки, а частина – по лівому краю. І в тому і в іншому випадку використовується тег <P>. Якщо йому назначити певні властивості, як селектору, то всі уривки тексту будуть представлятися однаково. Для запобігання цього є спеціальні кодові об’єкти, що називаються **класами**.

Приклад:

.redtext { color: #FF0000 }

Він визначає червоний колір тексту (RGB-код #FF0000). Але як селектор використовується явно не ім'я тегу - HTML-тегу <REDTEXT> не існує.

Це інший різновид стилю CSS – стильовий клас. Він може бути прив'язаний до будь-якому тегу. Як селектор тут вказують ім'я стильового класу, яке його однозначно ідентифікує. Ім'я стильового класу має складатися з букв латинського алфавіту, цифр та дефісів, причому починатися має з літери. У визначенні стильового класу його ім'я обов'язково передує символ точки — це ознака того, що визначається саме стильовий клас.

Стильовий клас вимагає явної прив'язки до тега. Для цього є атрибут CLASS, підтримуваний майже всіма тегами. Як значення цього атрибуту вказують ім'я потрібного стильового класу без символу точки:

<P CLASS="redtext">Увага!</P>

Так прив'язується створений стильовий клас redtext до абзацу.

Існує два методи використання класів:

1. Використання унікального імені. Клас можна представити як придумане розробником унікальне ім’я з будь-яким переліком властивостей, що являє собою точку та записане ім’я з використанням виключно символів латинського алфавіту та цифр. Синтаксис:

ТЕГ.ім’я класу {властивість1: значення; ... властивістьN: значення}

Наприклад, у файлі CSS є запис:

P.myclass {text-align: justify;}

Тепер, якщо викликати у коді сторінки тег форматування абзацу з вказанням імені класу, то даному елементу будуть надані значення, вказані в селекторі:

<BODY>

<P CLASS=”myclass”>Текст відформатований згідно з директивами CSS</P>

<P>Текст з базовими параметрами</P>

</BODY>

Тепер перший абзац буде вирівнюватись по ширині, а другий як – визначено браузером.

2. Без прив’язки до елементу. Синтаксис:

.ім’я класу {властивість1: значення; ... властивістьN: значення}

Наприклад: .myclass {color: green;}

В такому разі властивості цього класу можна присвоїти будь-якому тегу:

<P CLASS=”myclass”>Текст абзацу</P>

<H3 CLASS=”myclass”>Заголовок</H3>

Як значення атрибуту CLASS можна вказати відразу кілька імен стильових класів, розділивши їх пробілами. У такому разі дія стильових класів як би складається:

.attention { color: #FF0000; font-style: italic }

.bigtext { font-size: large }

. . .

<P><STRONG CLASS="attention bigtext">Стильовий клас вимагає явної прив'язування атрибутом тегу CLASS!</STRONG></P>

До тегу <STRONG> прив’язано відразу два стильові класи: attention і bigtext. В результаті вміст цього тега буде виведено жирним курсивним шрифтом червоного кольору і великого розміру. Про пріоритетність застосування стилів мова піде трохи пізніше.

Правила, встановлені за стандартом CSS під час написання селекторів:

- селекторами можуть бути назви тегів, імена класів стилів та ідентифікаторів;
- список селекторів переpаховують зліва направо і позначають рівень вкладеності відповідних тегів, що збільшується при русі зліва направо: теги, зазначені зправа повинні бути вкладені в теги, які стоять ліворуч;
- якщо назва тегу поєднується з назвою класу стилю або ідентифікатору означає, що даного тегу повинно бути вказано це ім’я класу або ідентифікатор;
- селектори розділені пробілами;
- стиль прив’язаний до тегу, що знаходиться найправіше в описі селектору.

Наприклад:

P EM { color: #0000FF }

Тег <em> повинен бути всередині тегу <p>. Стиль буде прив'язаний до тегу <em>.

P.mini { color: #FF0000;font-size: smaller }

Тег <p> поєднується з назвою "mini" класу стилю. Отже, стиль буде застосований до будь -якого тегу <p>, для якого вказано назву стилю "mini":

<P class = "mini"> Маленький червоний текст</p>

І още один приклад комбінованого стилю:

P.sel strong {color: #ff0000}

Цей стиль буде застосований до тегу <strong>, розташованого всередині тегу <p>, до якого додається клас стилю SEL.

<P class = "sel"> <strong> Цей текст буде червоним. </p>

Стандарт CSS дозволяє визначити відразу кілька однакових стилів, перерахувавши їх селектори через кому:

H1, .redtext, P EM <STRONG> { color: #FF0000 }

Тут ми створили відразу три однакові стилі: стиль перевизначення тега <H1>, стильовий клас redtext та комбінований стиль контекстного селектору P EM. Всі вони задають червоний колір шрифту.

5.6. Задавання стилів за допомогою Ідентифікаторів

Ідентифікатор, або його ще називають Іменований стиль, багато в чому схожий на стильовий клас. Селектор цього стилю також має ім'я, яке його однозначно ідентифікує, та прив'язується він до тегу також явним чином. А далі починаються відмінності:

- у визначенні іменованого стилю перед його ім'ям ставлять символ # ("решітка"). Він повідомляє Web-браузеру, що перед ним іменований стиль.
- прив'язку іменованого стилю до тегу реалізують через атрибут ID, який також підтримується практично всіма тегами. Як значення цього атрибуту вказують ім'я потрібного іменованого стилю, вже без символу #.
- значення атрибуту тегу ID має бути унікальним у межах Web-сторінки. Іншими словами, в HTML-коді Web-сторінки може бути присутнім лише один тег із заданим значенням атрибуту ID. Тому іменовані стилі використовують, якщо будь-який стиль слід прив'язати до одного-єдиного елементу веб-сторінки.

На відміну від селекторів та класів, **ідентифікатори** являють собою кодові об'єкти CSS, які дозволяють назначати властивості окремим компонентам HTML без використання інших стандартних методів. У файлі CSS ідентифікатор визначається символом "#", а у документі

HTML – атрибутом ID. Загальний синтаксис:

ТЕГ#ім’я ідентифікатора {властивість: значення;}

Наприклад, у файлі CSS є запис:

```
P#myp {text-align: center;}
```

В коді сторінки цей ідентифікатор можна викликати з використанням атрибуту ID:

```
<P ID=”myp”>Текст</P>
```

Очевидно, що тег <P>, який містить атрибут ID із значенням імені ідентифікатора, успадкує властивості останнього.

Ідентифікатори можна також вказувати без назви тегу. Так, строка:

```
#myid {color: red;}
```

створює ідентифікатор (аналогічно класу), який може використовувати будь-який елемент HTML:

```
<H1 ID=”myid”>Текст заголовку</H1>
<H3 ID=”myid”>Текст заголовку</H3>
```

Приклад для класів та ідентифікаторів. Зміст файлу CSS зі стилями:

```
body {
  background: url(img/back.gif)
}

.info, #info {text-align: center; margin:0}

th { background-color: #a3dcc8 }

table#products {
  font-size: 10pt;
  font-weight: normal;
  font-style: italic;
  background-color: #add8e6;
  border: 1px solid #800000;
  border-collapse: collapse;
  margin:0px auto;
}

table#products td, table#products th
```

```
{
  border-top-width: 2px;
  border-left-width: 2px;
  border-bottom-width: 0px;
  border-right-width: 0px;
  border-color: white;
  border-style: solid;
}

td.groups{
  background-color: aqua;
}

td#best {
  background-color: #800000;
  color: #ffffff;
}
```

Далі наведемо HTML файл:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Інтернет-магазин</title>
    <link rel="stylesheet" type="text/css" href="my.css">
</head>
<body>
    <h5 class="info">Курс долара: 40 грн.</h5>
    <h3 class="info">Ціни без НДС</h3>
    <h3 id="info">Оптовим покупцям - знижки</h3>

    <table id="products">
        <caption>Товари магазину</caption>
        <tr>
            <th>Найменування</th>
            <th>Продавець</th>
```

```
    <th>Ціна</th>
    <th>Гарантія</th>
</tr>
<tr><td colspan=4 class="groups">Телевізори</td></tr>
<tr>
    <td>Panasonic</TD>
    <td>Comfy</TD>
    <td>300$</TD>
    <td>1 рік</TD>
</tr>
<tr>
    <td>Samsung</TD>
    <td>Rozetka</TD>
    <td>250$</TD>
    <TD id="best">10 років</TD>
</tr>
<tr><td colspan=4 class="groups">Холодильники</td></tr>
<tr>
    <td>LG</TD>
    <td>Comfy</td>
    <td>40096,75 грн.</td>
    <td>1 рік</td>
</tr>
<tr>
    <td>Panasonic</td>
    <td>Comfy</td>
    <td>400$</td>
    <td>3 роки</td>
</tr>
<tr>
    <td>Samsung</td>
    <td>Foxtrot</td>
    <td>270$</td>
    <td>2 роки</td>
</tr>
```

```
        <tr>
            <td>Sony</td>
            <td>Comfy</td>
            <td>30000 грн.</td>
            <td>2 роки</td>
        </tr>
    </table>
</body>
</html>
```

Як видно з коду, до таблиці застосовано клас products, а до її комірок – ні. Але необхідний вигляд комірок та заголовків забезпечується об’явою TABLE#products TD, TABLE# products TH. Це означає застосувати описаний стиль для усіх <TH> та <TD> таблиці з ідентифікатором «products», яка на сторінці повинна бути єдиною. При цьому колір фону <TH> таблиці залишиться таким же, що й об’явлений для усіх <TH> таблиць, тобто:

```
TH {  background-color: #A3DCC8;}
```

Назви груп товарів будуть виводитись іншим кольором, оскільки вони мають атрибут CLASS.

5.7. Спеціальні селектори

Спеціальний селектор - це селектор, що неявно прив'язує стиль елементу Web-сторінки відповідно до складнішого, ніж ім'я тегу, критерію. Таким критерієм може бути, наприклад, порядковий номер елементу батьківського елементу, вказівку на певну частину вмісту абзацу (перший рядок або першу літеру), стан гіперпосилання (відвідана вона чи ні) та ін. Зазвичай спеціальні селектори використовують у комбінованих стилях, щоб зробити їх конкретнішими.

Комбінатори

Комбінатори - різновид спеціальних селекторів, що прив'язує стиль до елементу Web-сторінки на підставі його розташування щодо інших елементів.

Комбінатор “+” дозволяє прив'язати стиль до елемента Web-сторінки, що безпосередньо наступає за іншим елементом. При цьому обидва дочірні елементи повинні бути вкладеними в один і той же батьківський:

```
<елемент 1> + <елемент 2> {<стиль>}
```

Стиль буде прив'язаний до елементу 2, який повинен безпосередньо слідувати за елементом 1. Приклад:

```
H6 + PRE { font-size: smaller }
```

. . .

<H6>Це заголовок</H6>

<PRE>Цей текст буде набраний зменшеним шрифтом.</PRE>

<P>А цей — звичайним шрифтом.</P>

<H6>Це заголовок</H6>

<P>І цей — звичайним шрифтом.</P>

<PRE>І цей — звичайним шрифтом.</PRE>

Стиль, буде застосований тільки до першого тексту фіксованого форматування, оскільки він безпосередньо слідує за заголовком шостого рівня.

Комбінатор "~" (тильда) дозволяє прив'язати стиль до елемента Web-сторінки, наступного за іншим елементом і, можливо, відокремленого від нього іншими елементами. При цьому обидва дочірні елементи повинні бути вкладеними в один і той же батьківський:

<елемент 1> ~ <елемент 2> { <стиль> }

Стиль буде прив'язаний до елемента 2, який повинен слідувати за елементом 1. При цьому елемент 2 може бути відокремлений від елемента 1 іншими елементами.

Приклад:

H6 ~ PRE { font-size: smaller }

. . .

<H6>Це заголовок</H6>

<PRE>Цей текст буде набраний зменшеним шрифтом.</PRE>

<P>А цей — звичайним шрифтом.</P>

<H6>Це заголовок</H6>

<P>І цей — звичайним шрифтом.</P>

<PRE>А цей — зменшеним шрифтом.</PRE>

Стиль буде застосований до обох текстів фіксованого формату: і тому, що безпосередньо слідує за заголовком шостого рівня, і тому, що відокремлений від заголовка абзацом.

Комбінатор ">" дозволяє прив'язати стиль до елемента Web-сторінки, безпосередньо вкладеного в інший елемент:

<елемент 1> > <елемент 2> { <стиль> }

Стиль буде прив'язаний до елемента 2, який безпосередньо вкладено елемент 1.

Приклад:

BLOCKQUOTE > P { font-style: italic }

. . .

<BLOCKQUOTE>

<P>Цей абзац буде набраний курсивом.</P>

<DIV>

<P>А цей абзац є звичайним шрифтом.</P>

</DIV>

</BLOCKQUOTE>

Стиль буде застосований лише до абзацу, безпосередньо вкладеного у велику цитату. На другий абзац, послідовно вкладений у велику цитату та блоковий контейнер, цей стиль не діятиме.

Комбінатор <пробіл> дозволяє прив'язати стиль до елемента Web-сторінки, вкладеного в інший елемент, причому не обов'язково безпосередньо:

<елемент 1> <елемент 2> { <стиль> }

Стиль буде прив'язаний до елемента 2, який так чи інакше вкладений в елемент 1. При цьому елемент 2 може бути вкладений в інший елемент, вкладений елемент 1, або навіть кілька таких елементів послідовно.

Приклад:

BLOCKQUOTE P { font-style: italic }

. . .

<BLOCKQUOTE>

<P>Цей абзац буде набраний курсивом.</P>

<DIV>

<P>Цей абзац — теж.</P>

</DIV>

</BLOCKQUOTE>

Стиль буде застосований до обох абзаців: і вкладеному безпосередньо у велику цитату, і тому, що послідовно вкладено у велику цитату та блоковий контейнер.

Селектори за атрибутами тегу

Селектори за атрибутами тегу - це спеціальні селектори, що прив'язують стиль до тегу на підставі, чи є в ньому певний атрибут чи не має, або чи має він певне значення.

Селектори за атрибутами тегу використовуються не власними силами, а лише в сукупності зі стилями перевизначення тегу або комбінованими стилями. Їх записують відразу після основного селектору без жодних прогалин і беруть у квадратні дужки.

Селектор виду

<основний селектор>[<ім'я атрибута тега>] { <стиль> }

прив'язує стиль до елементів, теги яких мають атрибут із зазначеним ім'ям.

Приклад:

TD[ROWSPAN] { background-color: grey }

Цей стиль буде прив'язаний до комірок таблиці, теги яких мають атрибут ROWSPAN, тобто до комірок, об'єднаних по горизонталі.

Селектор виду

<основний селектор>[<ім'я атрибута тега>=<значення>] { <стиль> }

прив'язує стиль до елементів, теги яких мають атрибут із зазначеним ім'ям та значенням.

Приклад:

TD[ROWSPAN=2] { background-color: grey }

Цей стиль буде прив'язаний до комірок таблиці, теги яких мають атрибут ROWSPAN зі значенням 2, тобто до подвійних комірок, об'єднаних по горизонталі.

Селектори виду

<основний селектор>[<ім'я атрибута тега>~=<список значень, розділених пробілами>] { <стиль> }

і

<основний селектор>[<ім'я атрибута тега>|=<список значень, розділених комами>] { <стиль> }

прив'язує стиль до елементів, теги яких мають атрибут із зазначеним ім'ям та значенням, що збігається з одним із зазначених у списку.

Приклад:

TD[ROWSPAN~=2 3] { background-color: grey }

TD[ROWSPAN|=2,3] { border: thin dotted black }

Ці стилі будуть прив'язані до осередків таблиці, теги яких мають атрибут ROWSPAN зі значеннями 2 і 3, тобто до подвійних і потрійних осередків, об'єднаних по горизонталі.

Селектор виду

<основний селектор>[<ім'я атрибуту тега>^=<підстрока>] { <стиль> }

прив'язує стиль до елементів, теги яких мають атрибут із зазначеним ім'ям та значенням, що починається із зазначеної строки.

Приклад:

IMG [SRC ^ = http://www.pictures.com] { margin: 5px }

Цей стиль буде прив'язаний до графічних зображень, теги яких мають атрибут SRC зі значенням, що починається з "http://www.pictures.com", тобто зображень, взятим з Web-сайту http://www.pictures.com.

Селектор виду

<основний селектор>[<ім'я атрибута тега>$=<підстрока>] { <стиль> }

прив'язує стиль до елементів, теги яких мають атрибут із зазначеним ім'ям та значенням, що закінчується зазначеною строкою.

Приклад:

IMG[SRC$=gif] { margin: 10px }

Цей стиль буде прив'язаний до графічних зображень, теги яких мають атрибут SRC зі значенням, що закінчується "gif", тобто до зображень формату GIF.

Селектор виду

<основний селектор>[<ім'я атрибута тега>*=<підстрока>] { <стиль> }

прив'язує стиль до елементів, теги яких мають атрибут із зазначеним ім'ям та значенням, що включає зазначену строку.

Приклад:

IMG[SRC*=/picts/] { margin: 10px }

Цей стиль буде прив'язаний до графічних зображень, теги яких мають атрибут SRC зі значенням, що включає "/picts/", тобто до зображень, взятих із папки picts Web-сайту, звідки вони завантажені.

Псевдоелементи

Псевдоелементи - різновид спеціальних селекторів, що прив'язують стиль до певного фрагмента елементу веб-сторінки. Таким фрагментом може бути перший символ або перший рядок абзацу.

Псевдоелементи використовуються не власними силами, а лише в сукупності з іншими. Їх записують відразу після основного селектору без жодних прогалин:

<основний селектор>::<псевдоелемент> {<стиль>}

Псевдоелемент ::first-letter прив'язує стиль до першої літери тексту в елементі Web-сторінки, якщо їй не передує вбудований елемент, який не є текстом, наприклад, зображення.

Приклад:

P::first-letter { font-size: larger }

Цей стиль буде прив'язаний до першої літери абзацу.

Псевдоелемент ::first-line прив'язує стиль до першого рядка тексту в елементі Web-сторінки:

P::first-line { text-transform: uppercase }

Цей стиль буде прив'язаний до першого рядка абзацу.

Псевдоелементи використовуються для додавання вмісту, що генерується за допомогою властивості content:

:first-letter - вибирає першу букву кожного абзацу, застосовується лише до блокових елементів;

:first-line - Вибирає перший рядок тексту елемента, застосовується тільки до блокових елементів;

:before — вставляє вміст, що генерується перед елементом;

:after — додає вміст, що генерується після елемента.

Псевдокласи

Псевдокласи - найпотужніший різновид спеціальних селекторів. Вони дозволяють прив'язати стиль до елементів Web-сторінки на основі їх стану (наведений на них курсор миші чи ні) та розташування в батьківському елементі.

Псевдокласи також використовуються не власними силами, а лише в сукупності з іншими стилями. Їх записують відразу після основного селектора без жодних прогалин:

<основний селектор>:<псевдоклас> {<стиль>}

Псевдокласи характеризують елементи з наступними властивостями:

:link - не відвідане посилання;

:visited - Відвідане посилання;

:hover - будь-який елемент, яким проводять курсором миші;

:focus - інтерактивний елемент, до якого перейшли за допомогою клавіатури або активували мишею;

:active — елемент, активований користувачем;

:valid - поля форми, вміст яких пройшов перевірку у браузері на відповідність зазначеному типу даних;

:invalid - поля форми, вміст яких не відповідає зазначеному типу даних;

: enabled - всі активні поля форм;

:disabled - заблоковані поля форм, тобто, що знаходяться в неактивному стані;

:in-range - поля форми, значення яких перебувають у заданому діапазоні;

:out-of-range - поля форми, значення яких не входять до встановленого діапазону;

:lang() — елементи з текстом зазначеною мовою;

:not(селектор) - елементи, які не містять зазначений селектор - клас, ідентифікатор, назва або тип поля форми - :not([type="submit"]);

:target - елемент із символом #, на який посилаються в документі;

:checked - виділені (вибрані користувачем) елементи форми.

Псевдокласи, своєю чергою, самі діляться на групи.

Псевдокласи гіперпосилань

Псевдокласи гіперпосилань потрібні для прив'язки стилів до гіперпосилань на основі їх стану: є гіперпосилання відвіданим або невідвіданим, клацає на ній відвідувач мишею чи тільки навів її у курсор миші та інших.

Усі псевдокласи гіперпосилань, доступні у стандарті CSS 3:

:link - невідвідане гіперпосилання;

:visited - відвідане гіперпосилання;

:active - гіперпосилання, на якому відвідувач зараз клацає мишею;

:focus - гіперпосилання, що має фокус введення;

:hover - гіперпосилання, на яке наведено курсор миші.

Псевдокласи гіперпосилань застосовуються разом із стилями, що задають параметри для гіперпосилань. Це можуть бути стилі перевизначення тега <A> або комбіновані стилі, створені на основі стилю перевизначення тегу <A>:

A:link { text-decoration: none }

A:visited { text-decoration: overline }

A:active { text-decoration: underline }

A:focus { text-decoration: underline }

A:hover { text-decoration: underline }

У цьому прикладі псевдокласи гіперпосилань діють разом із стилями перевизначення тега <A>. В результаті наведені стилі будуть застосовані до всіх гіперпосилання на Web-сторінці.

Приклад:

A.special:link { color: darkred }

A.special:visited { color: darkviolet }

A.special:active { color: red }

A.special:focus { color: red }

A.special:hover { color: red }

У цьому прикладі псевдокласи гіперпосилань поєднані з комбінованими стилями, що поєднують стиль перевизначення тега <A> та стильовий клас special. Вони будуть застосовані тільки до гіперпосилань, до яких був прив'язаний зазначений стильовий клас.

Псевдокласи гіперпосилань можна комбінувати, записуючи їх один за одним:

A:visited:hover { text-decoration: underline }

Цей стиль буде застосований до відвіданого гіперпосилання, над яким знаходиться курсор миші.

Структурні псевдокласи

Структурні псевдокласи дозволяють прив'язати стиль до елемента Web-сторінки на основі його розташування у батьківському елементі.

Структурні псевдокласи відбирають дочірні елементи відповідно до параметра, зазначеного у круглих дужках:

:nth-child(odd) - непарні дочірні елементи;

:nth-child(even) - парні дочірні елементи;

:nth-child(3n) - кожен третій елемент серед дочірніх;

:nth-child(3n+2) - вибирає кожен третій елемент, починаючи з другого дочірнього елемента (+2);

:nth-child(n+2) - вибирає всі елементи, починаючи з другого;

:nth-child(3) - вибирає третій дочірній елемент;

:nth-last-child() — у списку дочірніх елементів вибирає елемент із зазначеним місцезнаходженням, аналогічно з :nth-child(), але починаючи з останнього у зворотний бік;

:first-child - дозволяє оформити тільки перший дочірній елемент;

:last-child - дозволяє форматувати останній дочірній елемент;

:only-child - вибирає елемент, що є єдиним дочірнім елементом;

:empty - вибирає елементи, які не мають дочірніх елементів;

:root - вибирає елемент, що є кореневим у документі - елемент HTML.

Псевдокласи :first-child та :last-child прив'язують стиль до елементу Web-сторінки, який є, відповідно, першим та останнім дочірнім елементом свого батька:

```
UL:first-child { font-weight: bold }
```

Цей стиль буде застосований до пункту, що є першим дочірнім елементом свого батька-списку, тобто до першого пункту списку.

Приклад:

```
TD:last-child { color: green }
```

Цей стиль буде застосовано до останнього дочірнього елемента кожного рядка таблиці - до її останнього осередку.

Приклад:

```
#cmain P:first-child { font-weight: bold }
. . .
<DIV ID="cmain">
<P>Цей абзац буде набраний зеленим кольором.</P>
<BLOCKQUOTE>
<P>Цей абзац — теж.</P>
</BLOCKQUOTE>
<BLOCKQUOTE>
<P>І цей теж.</P>
<P>А цей — ні.</P>
</BLOCKQUOTE>
</DIV>
```

Стиль буде застосований до першого абзацу, єдиному абзацу у першій великій цитаті та першому абзацу у другій великій цитаті. Справа в тому, що останні два абзаци, до яких буде застосовано стиль, що відраховуються щодо своїх батьків — великих цитат — і в них є першими.

А якщо ми змінимо цей стиль ось так:

```
#cmain > P:first-child { font-weight: bold }
```

він буде застосований тільки до першого абзацу, безпосередньо вкладеного в контейнері cmain. Адже ми вказали комбінатор, який наказує, щоб елемент, до якого застосовується стиль, повинен бути безпосередньо вкладений у свого батька.

Селектор структурних псевдокласів типу вказують на конкретний тип дочірнього елемента:

:nth-of-type() - вибирає елементи за аналогією з :nth-child(), при цьому бере до уваги лише тип елемента;

:first-of-type - вибирає перший дочірній елемент даного типу;

:last-of-type - вибирає останній елемент даного типу;

:nth-last-of-type() - вибирає елемент заданого типу у списку елементів відповідно до зазначеного розташування, починаючи з кінця;

:only-of-type - вибирає єдиний елемент зазначеного типу серед дочірніх елементів батьківського елемента

Псевдоклас only-of-type прив'язує стиль до елементу Web-сторінки, який є єдиним дочірнім елементом, сформованим за допомогою цього тегу, у своєму батькові.

Приклад:

```
P:only-of-type { color: green }
. . .
<BLOCKQUOTE>
<P>Цей абзац буде набраний зеленим кольором.</P>
</BLOCKQUOTE>
<BLOCKQUOTE>
<P>І цей абзац буде набраний зеленим кольором.</P>
<ADDRESS>А цей текст — ні.</ADDRESS>
</BLOCKQUOTE>
<BLOCKQUOTE>
<P>І цей — ні.</P>
<P>І цей — ні.</P>
</BLOCKQUOTE>
```

Стиль буде застосований до абзаців, вкладених у першу та другу великі цитати, оскільки ці абзаци є єдиними елементами, сформованими з допомогою тегу <P>. До абзаців, вкладених у третю велику цитату, стиль не буде застосований.

Псевдоклас :nth-child дозволяє прив'язати стиль до елементів Web-сторінки, вибравши їх за порядковими номерами, під якими ці елементи визначені у своєму батьку:

<основний селектор>:nth-child(<a>n+<b>) { <стиль> }

Після імені даного псевдокласу в дужках вказують формулу, за якою обчислюються номери елементів, до яких буде застосований стиль. Ця формула має два параметри, що задаються Web-дизайнером: a та b. Їх значення повинні бути невід'ємними цілими числами.

Розглянемо, як виконується прив'язка стилю, що включає псевдоклас: nth-child.

Спочатку Web-браузер зчитує CSS-код стилю і, керуючись його селектором, знаходить елементи Web-сторінки, до яких, можливо, буде застосовано цей стиль. Також Web-браузер визначає батьків цих елементів.

Після цього Web-браузер розбиває всі знайдені елементи на групи. Кількість елементів у кожній групі визначається параметром a наведеної формули.

Після розбиття Web-браузер обчислює кількість груп.

Далі Web-браузер послідовно підставляє у формулу замість n номера груп, що вийшли, починаючи з нуля. В результаті кожного проходу обчислення виходить номер елементу, до якого застосовується стиль.

Наприклад створимо таблицю з п'яти рядків і застосуємо до неї такий стиль:

TR:nth-child(2n+1) { text-align: center }

У стилі ми вказали, що група має містити два дочірні елементи. Тоді Web-браузер розіб'є рядки таблиці на дві групи, по два рядки в кожній.

Web-браузер підставить у формулу n 0. Після обчислення вийде значення 1. Web-браузер застосує цей стиль до першого рядка таблиці.

Web-браузер підставить у формулу n рівне 1. Після обчислення вийде значення 3. Web-браузер застосує цей стиль до третього рядка таблиці.

Web-браузер підставить у формулу n рівне 2. Після обчислення вийде значення 5. Web-браузер застосує цей стиль до п'ятого рядка таблиці. Оскільки 2 - загальна кількість груп, то на цьому обчислення закінчаться.

В результаті цей стиль буде застосований до кожного непарного рядка нашої таблиці.

Якщо ми створимо такий стиль:

TR:nth-child(2n+0) { text-align: center }

то він буде застосований до другого та четвертого (парних) рядків нашої таблиці.

До речі, ми можемо зробити запис трохи коротшим:

TR:nth-child(2n) { text-align: center }

Раніше ми розглядали приклади із розбиттям дочірніх елементів на групи. Але ми можемо скасувати розбиття, вказавши нульову кількість елементів групи.

У цьому випадку Web-браузер знайде b-й дочірній елемент і застосує стиль до нього.

Так, якщо ми створимо такий стиль:

TR:nth-child(0n+2) { text-align: center }

то Web-браузер застосує його до другого рядка таблиці.

Ми можемо записати цей стиль і так:

TR:nth-child(2) { text-align: center }

і Web-браузер правильно його обробить.

Замість вказування формули у дужках ми можемо поставити там зумовлені значення odd та even. Перше прив'язує стиль до непарних дочірніх елементів, друге - до парних:

TR:nth-child(odd) { background-color: grey }

TR:nth-child(even) { background-color: yellow }

Перший стиль буде застосований до непарних рядків таблиці, другий – до парних.

Псевдоклас :nth-last-child аналогічний розглянутому нами псевдокласу :nth-child тим винятком, що відлік дочірніх елементів ведеться з початку, і з кінця батьківського елемента.

Приклад:

TR:nth-last-child(1) { text-align: center }

Цей стиль буде застосовано до останнього рядка таблиці.

Приклад:

#cmain P:nth-last-child(2) { font-weight: bold }

А цей стиль буде застосовано до передостаннього (другого з кінця) абзацу в контейнері cmain.

Ще один структурний псевдоклас: empty. Він прив'язує стиль до елементів, які не мають дочірніх елементів.

Приклад:

P:empty { display: none }

Цей стиль буде прив'язаний до порожніх абзаців, що не мають вмісту.

Псевдокласи :not і *

Спеціальний псевдоклас :not дозволяє прив'язати стиль до будь-якого елементу Web-сторінки, що не відповідає заданим умовам. Такою умовою може бути будь-який селектор:

<основний селектор>:not(<селектор вибору>) { <стиль> }

Стиль буде прив'язаний до елемента Web-сторінки, що відповідає основному селектору і селектору вибору, що не задовольняє.

Приклад:

DIV:not(#cmain) { background-color: yellow }

Тут ми вказали як основний селектор стиль перевизначення тегу <DIV>, а як селектор вибору - іменований стиль cmain. В результаті цей стиль буде застосований до всіх блокових контейнерів, за винятком того, який має ідентифікатор “cmain”.

А ось стиль, який буде застосований до всіх рядків таблиці, за винятком першого:

TR:not(:nth-child(1)) { background-color: grey }

Псевдоклас * ("зірочка") означає будь-який елемент Web-сторінки.

Приклад:

#cmain > *:first-child { border-bottom: medium solid black }

Цей стиль буде застосований до першого елементу будь-якого типу, який безпосередньо вкладений у контейнер з ідентифікатором “cmain”.

5.8. Правила каскадності та пріоритет стилів

У кожного браузера свій стиль для представлення сторінок. Коли браузер завантажує Web-сторінку, він покриває її власним стилем. Якщо він переглядає сторінку, де є посилання на CSS, на екрані будуть відображені особливості, які задані в каскадній таблиці стилів. Основна ідея в тому, що звичайна сторінка відображається так, як задано у браузері, а та, що використовує CSS – так, як хоче її автор.

Браузер, обираючи із запропонованих стилів, користується наступними правилами, які ще називають правилами каскадності:

1. Зовнішня таблиця стилів, посилання, на яку (тег <link>) знаходиться на сторінці пізніше, має пріоритет перед тою, посилання на яке йде раніше.
2. Вбудовані в тег стилі (атрибут тегу style) мають пріоритет перед будь -якими стилями, зазначеними в таблицях стилів. Тобто, якщо є конфлікт між встановленим у ньому стилем і авторськими параметрами, то перевага надається авторським параметрам. Так було в нашому випадку, коли ми встановили стиль для тегу <A>. Якщо конфліктують два стилі і один застосовується тільки в цій ситуації, а інший у всіх випадках, перевага надається першому. Наприклад, задано такий стиль: <STYLE TYPE=”text/css”> P {color: black;} </STYLE>, а в тексті сторінки є абзац з власним стилем, який теж встановлює колір переднього плану: <P STYLE=”color: green;”>Абзац</P>, то перевага буде надана власному стилю цього абзацу, тобто колір переднього плану буде зеленим.
3. Більш конкретні стилі мають пріоритет від менш конкретних. Це означає, наприклад, що клас стилів має пріоритет перед стилем тегу, оскільки клас стилів приєднаний до

конкретних тегів.

Приклад:

.redtext { color: #FF0000; font-weight: normal }

EM { color: #00FF00; font-weight: bold }

Далі ми розмістили такий абзац на веб -сторінці:

<p> <em class = "redtext"> Цей курсив. </p>

Правила каскадності свідчать, що текст цього абзацу буде відображатися звичайним шрифтом червоного кольору.

Але припустимо, що нам потрібно щоб весь текст, виділений тегом <em>, у будь -якому випадку відображався жирним шрифтом.

Стандарт CSS надає нам чудову можливість перетворити індивідуальні властивості стилю у важливі. Параметри, встановлені важливими властивостями стилю, матимуть пріоритет дад всіма аналогічними властивостями стилю, встановлених в інших селекторах, які навіть ще більш конкретні. Властивості стилю, які не оголошені важливими, все ще будуть дотримуватися правил каскадності. Щоб зробити властивість стилю важливою, достатньо вказати !important після її значення:

EM { color: #00FF00; font-weight: bold !important }

Тепер текст, виділений тегом <em>, завжди відображатиметься жирним шрифтом, навіть якщо цей параметр перевизначений у більш конкретному стилі.

Але не слід зловживати таким підходом, оскільки завжди можна знайти варіант опису більш конкретного селектора стилю.

Оскільки в багатьох випадках часто виникає конфлікт стилів, була створена пріоритетна система: в кінцевому підсумку використовується стиль, який виходить із джерела з більш високим пріоритетом. Специфічність - це вага, що надається конкретному правилу CSS.

Таблиця 5.1 допоможе зрозуміти пріорите, в ній вказується вага (значення) кожного селектора. Чим більша вага, тим вище пріоритет.

Коли селектор складається з кількох інших селекторів, необхідно обчислити їх загальну вагу. Ось як обчислюється пріоритет: для кожного селектора до відповідної комірки додається 1. Решта комірок - нулі. Щоб отримати загальну вагу, необхідно «склеїти» всі числа в комірка (табл. 5.2).

Універсальний селектор (*), комбінатори (+,>, ~, '``') і заперечення псевдо -class (: not()) не впливають на специфічність. (Однак, селектори оголошені всередині: not() впливають).

Якщо так сталося, що два селектори мають однакову вагу, то пріоритет надається стилю, який об’явлено пізніше у коді css. Якщо для одного елементу стиль встановлений у зовнішніх та внутрішніх таблицях, то пріоритет надається стилю в таблиці, який нижчий у коді.

Таблиця 5.1

Вага селекторів стилів

Стиль	**Вага**
Селектори тегів (h1), псевдоелементи (::before)	1
Селектори класів (.example), селектори атрибуту ([type=”radio”]) і псевдокласів (:hover)	10
Селектор ID (#example)	100
Inline-стиль (атрибут style)	1000

Таблиця 5.2

Приклад обчислення ваги

Селектор	**ID**	**Клас**	**Тег**	**Загальна вага**
P	0	0	1	1
.your_class	0	1	0	10
p.your_class	0	1	1	11
#your_id	1	0	0	100
#your_id p	1	0	1	101
#your_id .your_class	1	1	0	110
p a	0	0	2	2
#your_id #my_id .your_class p a	2	1	2	212

Отже, для кожного правила браузер обчислює специфічність селектора, і якщо елемент має конфліктні оголошення властивостей, до уваги береться правило, що має найбільшу специфічність. Значення специфічності складається з чотирьох частин: 0, 0, 0, 0. Специфічність селектора визначається так:

для id додається 0, 1, 0, 0;

для class додається 0, 0, 1, 0;

для кожного елемента та псевдоелемента додається 0, 0, 0, 1;

для вбудованого стилю, доданого безпосередньо до елемента – 1, 0, 0, 0;

Універсальний селектор не має специфічності.

Приклади:

h1 {color: lightblue;} /*специфічність 0, 0, 0, 1*/

em {color: silver;} /*специфічність 0, 0, 0, 1*/

h1 em {color: gold;} /*специфічність: 0, 0, 0, 1 + 0, 0, 0, 1 = 0, 0, 0, 2*/

div#main p.about {color: blue;} /*специфічність: 0, 0, 0, 1 + 0, 1, 0, 0 + 0, 0, 0, 1 + 0, 0, 1, 0 = 0, 1, 1, 2*/

.sidebar {color: grey;} /*специфічність 0, 0, 1, 0*/

#sidebar {color: orange;} /*специфічність 0, 1, 0, 0*/

li#sidebar {color: aqua;} /*специфічність: 0, 0, 0, 1 + 0, 1, 0, 0 = 0, 1, 0, 1*/

У результаті до елементу застосовуються ті правила, специфічність яких більше. Наприклад, якщо на елемент діють дві специфічності зі значеннями 0, 0, 0, 2 і 0, 1, 0, 1, то виграє друге правило.

5.9. Коментарі в CSS

Мова CSS дозволяє створювати коментарі. Більше того, за допомогою його допомоги ми можемо формувати коментарі двох типів.

Коментар, що складається з одного рядка, створюється за допомогою символу “/”.

На самому початку рядка, який стане коментарем:

/ Це коментар

P {color: #0000ff}

Один коментар починається з символу / і закінчується кінцем рядка.

Коментар, що складається з довільної кількості рядків, створюється за допомогою послідовностей символів / * та * /. Між ними розміщені рядки, які стануть коментарем:

/*

Це коментар,

Що складається з

кількох рядків.

*/

P {color: #0000ff}

5.10. Параметри шрифта

Властивість стилю font-family визначає ім'я шрифту, яким буде виведений текст:

font-family: <список імен шрифтів, розділених комами>|inherit

Імена шрифтів задаються як їхні назви, наприклад, Arial або Times New Roman.

Якщо ім'я шрифту містить пробіли, його потрібно взяти в лапки:

P { font-family: Arial }

H1 (font-family: "Times New Roman" }

Якщо цей атрибут стилю є у вбудованому стилі, лапки замінюють апострофами:

<P STYLE="font-family: 'Times New Roman'">

Якщо вказаний нами шрифт присутній на комп'ютері відвідувача, браузер його використовує. Якщо такого шрифту немає, то текст виводиться шрифтом, заданим у налаштуваннях за замовчуванням.

Можна вказати кілька найменувань шрифтів через кому:

P { font-family: Verdana, Arial }

Тоді Web-браузер спочатку шукатиме перший із зазначених шрифтів, у разі невдалого пошуку – другий, потім третій тощо.

Замість імені конкретного шрифту можна задати ім'я одного із сімейств шрифтів, які представляють цілі набори аналогічних шрифтів. Таких родин п'ять: serif (шрифти із засічками), sans-serif (шрифти без засічок), cursive (шрифти, імітують рукописний текст), fantasy (декоративні шрифти) та monospace (моноширинні шрифти):

H2 { font-family: Verdana, Arial, sans-serif }

Особливе значення inherit вказує на те, що текст даного елемента Web-сторінки має бути оформлений тим самим шрифтом, як і текст батьківського елемента. Говорять, що в даному випадку елемент Web-сторінки "успадковує" шрифт від батьківського елемента. Це, до речі, значення атрибуту стилю font-family за умовчанням.

Властивість стилю font-size визначає розмір шрифту:

font-size: <розмір>|xx-small|x-small|small|medium|large|x-large|xx-large|larger|smaller|inherit

Розмір шрифту можна задати в абсолютних та відносних величинах. Для цього використовується одна з одиниць вимірювання, що підтримується CSS (табл. 5.3).

Позначення обраної одиниці виміру вказують після самого значення:

P { font-size: 10pt }

STRONG { font-size: 1cm }

EM { font-size: 150% }

Крім числових, властивість font-size може набувати і символьні значення. Так, значення від xx-small до xx-large задають сім обумовлених розмірів шрифту, від найменшого до найбільшого. А значення larger і smaller представляють наступний розмір шрифту, відповідно, за зростанням та зменшенням.

Наприклад, якщо для батьківського елемента визначено шрифт розміру medium, то значення larger встановить для поточного елемента розмір шрифту large.

Значення inherit вказує, що цей елемент Web-сторінки повинен мати той ж розмір шрифту, як і батьківський елемент. Це значення властивості стилю fontsize за умовчанням.

Таблиця 5.3

Одиниці вимірювання стандарту CSS

Назва	**CSS**
Піксели	px
Пункти	pt
Дюйми	In
Сантиметри	Cm
Міліметри	mm
Піки	pc
Розмір букви “m” поточного шрифта	em
Розмір букви “x” поточного шрифта	ex
Відсотки від розміру шрифта батьківського елементу	%

Властивість стилю color визначає колір тексту:

color: <колір>|inherit

Колір можна встановити так званим RGB-кодом (Red, Green, Blue – червоний, зелений, синій). Він записується у форматі

#<частка червоного кольору><частка зеленого кольору><частка синього кольору>,

де частки всіх кольорів вказані у вигляді шістнадцяткових чисел від 00 до FF.

Задамо для тексту червоний колір:

H1 { color: #FF0000 }

А тепер сірий колір:

ADDRESS { color: #CCCCCC }

Крім того, CSS дозволяє задавати кольори за іменами. Так, значення black відповідає чорному кольору, white – білому, red – червоному, green – зеленому, а blue – синьому.

Приклад:

H1 { color: red }

Значення inherit вказує, що цей елемент Web-сторінки повинен мати той ж колір шрифту, як і батьківський елемент.

Опис способів задавання кольору:

1. По назві — використовуються англомовні назви кольорів (Black, Navy, Gray, Blue,

Silver, Aqua, White, Green, Red, Lime, Fuchsia, Teal, Maroon, Yellow, Purple, Olive). Приклад:

p {color: lime}

Ключове слово transparent це найпростіший спосіб задати прозорий колір:

div { background-color: transparent; }

2. RGB в шістнадцятковій системі — спосіб, який часто використовується в HTML. Перед номером кольору ставиться значок дієз (пр.: #517af3, або скорочена форма #02a, що рівноцінно #0022aa). Приклад:

p {color: #0f0}

p {color: #00ff00}

3. RGB в десятковій системі — рівень кожної складової кольору підбирається в діапазоні від 0 до 255 і записується в форматі: “rgb(x, y, z)”, де x, y, z – рівень відповідно червоного, зеленого та голубого кольору. Приклад:

p {color: rgb(0,255,0)}

4. RGBA (із прозорістю) в десятковій системі — рівень кожної складової кольору підбирається в діапазоні від 0 до 255 і записується в форматі: “rgba(x, y, z, a)”, де x, y, z – рівень відповідно червоного, зеленого та стнього кольору, a – прозорість (від 0 до 1). Приклад:

div {color: rgba(255,255,255,0.5);}

5. RGB у відсотках — схожий формат на попередній, але рівень кожної складової кольору підбирається в діапазоні від 0% до 100% (пр.: rgb(10%, 90%, 20%)). Приклад:

p {color: rgb(0%, 100%, 0%)}

6. HSL є альтернативою до RGB-запису. Його ввели в CSS з тією метою, щоб бути більш інтуїтивним ніж RGB. Базовий синтаксис:

hsl(відтінок, насиченість, яскравість)

Відтінок вказує на один з кольорів, які ми можемо бачити. Діапазон кольорів, видимих людському оку може бути зображено у вигляді кола. Таким чином, відтінок може мати значення між 0 і 360.

Насиченість визначає наскільки "кольоровим" буде відтінок. Низьке значення насиченості призводить до більш приглушених відтінків, і навпаки. Насиченість може бути в межах від 0% до 100%.

Яскравість впливає на те, наскільки світлим чи темним буде відтінок. Чим вище значення, тим світлішим буде колір, і навпаки. Яскравість може мати значення між 0% і 100%.

Приклад:

div {   background-color: hsl(120, 100%, 45%); }

7. HSLA це те ж саме, що й HSL, але з додатковим параметром, який дозволяє вказати рівень прозорості кольору. Синтаксис такий:

hsla(відтінок, насиченість, яскравість, прозорість)

де рівень прозорості може мати значення від 0 до 1.

Приклад:

div { color: hsla(0, 100%, 90%, 0.5); }

Властивість стилю opacity дозволяє вказати рівень напівпрозорості елемента Web-сторінки:

opacity: <числове значення>|inherit

Значення напівпрозорості є числом від 0 до 1. При цьому 0 позначає повну прозорість елемента (тобто елемент фактично не видно), а 1 - повну непрозорість (це звичайна поведінка).

Ось приклад завдання половинної прозорості (значення 0,5) для тексту фіксованого форматування:

PRE { opacity: 0.5 }

Зазначимо, як ми вказали дробове число – замість символу комою тут використовується крапка.

Властивість стилю text-decoration задає підкреслення або закреслення, яка буде застосоване до тексту:

text-decoration: none|underline|overline|line-through|blink|inherit

Тут доступні п'ять значень (за винятком inherit):

- none прибирає всі прикраси, задані для шрифту батьківського елемента;
- underline підкреслює текст;
- overline "накреслює" текст, тобто проводить лінію над рядками;
- line-through закреслює текст;
- blink робить шрифт мерехтливим.

Властивість стилю font-variant дозволяє вказати, як виглядатимуть малі літери шрифту:

font-variant: normal|small-caps|inherit

Значення small-caps задає таку поведінку шрифту, коли його малі літери виглядають так само, як великі, просто мають менший розмір. Значення normal задає для шрифту звичайні великі літери.

Властивість стилю text-transform дозволяє змінити регістр символів тексту:

text-transform: capitalize|uppercase|lowercase|none|inherit

Ми можемо перетворити текст до верхнього (значення uppercase) або нижньому (lowercase) регістру, перетворити до верхнього регістру першу букву кожного слова (capitalize) або залишити у первісному вигляді (none).

Властивість стилю line-height визначає висоту рядка тексту:

line-height: normal|<відстань>|inherit

Тут можна встановити абсолютну і відносну величину відстані, вказавши відповідну одиницю вимірювання CSS. За її відсутності задане нами значення спочатку множиться на висоту поточного шрифту і потім використовується. Таким чином, щоб збільшити висоту рядка вдвічі порівняно зі звичайною, ми можемо написати:

P { line-height: 2 }

Значення normal цього атрибута повертає керування висотою рядка Web-браузеру.

Властивість стилю letter-spacing дозволяє задати додаткову відстань між символами тексту:

letter-spacing: normal|<відстань>

Зазначимо, що це додаткова відстань; вона буде додана до початкового, встановленого самим Web-браузером.

Тут також можна задати абсолютну та відносну відстань, вказавши відповідну одиницю виміру CSS. Відстань може бути позитивною та негативною; в останньому випадку символи шрифту будуть розташовуватися один до одного ближче, ніж звичайно. Значення normal встановлює додаткову відстань за умовчанням, що дорівнює нулю.

Властивість стилю letter-spacing не підтримує значення inherit.

Ось приклад завдання додаткової відстані між символами, що дорівнює п'яти пікселам:

H1 { letter-spacing: 5px }

Текст, набраний такими символами, виглядатиме розрідженим. А тут ми задали негативну додаткову відстань між символами рівною двом пікселам:

H6 { letter-spacing: -2px }

Ці два пікселі будуть віднімаються з початкової відстані, в результаті символи зблизяться, а текст виглядатиме стислим. Можливо, символи навіть налізуть один на одного.

Аналогічна властивість стилю word-spacing задає додаткову відстань між окремими словами тексту:

word-spacing: normal|<відстань>

Приклад:

H1 {word-spacing: 5mm}

І насамкінець розглянемо атрибут стилю font, що дозволяє задати одночасно відразу кілька параметрів шрифту:

font: [<накреслення>] [<вид малих літер>] [<"жирність">]

[<розмір>[/<висота рядка тексту>]] <ім'я шрифту>

Як бачимо, обов'язковим є лише ім'я шрифту - інші параметри можуть бути відсутніми.

Задаємо для тексту абзаців шрифт Times New Roman розміром 10 пунктів:

P { font: 10pt "Times New Roman" }

А для заголовків шостого рівня — шрифт Arial розміром 12 пунктів та курсивного зображення:

H6 { font: italic 12pt Verdana }

За замовчуванням Web-браузер розбиває текст на рядки так, щоб вмістити його у вікно і уникнути горизонтального прокручування. Не завжди при цьому він розриває рядки, як нам треба. CSS пропонує два атрибути стилю, що дозволяють нам вказати, як Web-браузеру слід розбивати текст на рядки.

Властивість стилю white-space визначає правила, якими Web-браузер керується при виведенні тексту.

white-space: normal|pre|nowrap|pre-wrap|pre-line|inherit

Властивість стилю white-space може мати п'ять значень.

- normal — послідовності пропусків перетворюються на одну пропуск, переклади рядків також перетворюються на прогалини. Web-браузер сам розриває текст на рядки, щоб вмістити його у вікно по ширині. Фактично це значення наказує Web-браузеру застосовувати для виведення тексту блокових елементів правила за умовчанням;
- pre — послідовності пропусків та переклади рядків зберігаються; текст виводиться точно в такому вигляді, як він записаний в HTML-коді. Web-браузер сам не розриває текст на рядки. Фактично текст виводиться так, немовби він поміщений у тег <PRE>;
- nowrap — послідовності пропусків перетворюються на один пропуск, переведення рядків також перетворюються на прогалини. Однак Web-браузер сам не розриває текст на рядки;
- pre-wrap — послідовності пропусків та переведення рядків зберігаються. Web-браузер може розірвати занадто довгі рядки, щоб уникнути горизонтального прокручування;
- pre-line — послідовності пропусків перетворюються на один пропуск, переведення рядків зберігаються. Web-браузер може розірвати занадто довгі рядки, щоб уникнути горизонтального прокручування.

Ось стиль, що перевизначає тег <PRE> так, щоб при необхідності його вміст розривався на рядки:

PRE {white-space: pre-wrap}

Властивість стилю word-wrap застосовується нечасто, але в деяких випадках без нього не обійтися. Він дозволяє вказати місця, в яких Web-браузер може виконати розрив тексту:

word-wrap: normal|break-word|inherit

Тут є два значення: normal — вказує Web-браузеру, що він може розривати текст на рядки тільки за пробілами (це звичайна поведінка веб-браузера); break-word — дозволяє Web-браузеру виконувати розрив тексту на рядки усередині слів (це може стати в нагоді, якщо текст

містить багато дуже довгих слів, які по ширині не поміщаються у батьківський елемент).

Приклад:

P {word-wrap: break-word}

Тут ми дозволили Web-браузеру виконувати розрив тексту на рядки всередині слів у вмісті тегів (тобто в абзацах).

Іноді виникає ситуація, коли потрібно змістити фрагмент по вертикалі щодо тексту, який його оточує, тобто встановити вертикальне вирівнювання тексту.

Властивість стилю vertical-align задає вертикальне вирівнювання тексту:

vertical-align: baseline|sub|super|top|text-top|middle|bottom|text-bottom|<проміжок між базовими лініями>|inherit

Ця властивість стилю приймає вісім визначених значень:

- baseline — задає вирівнювання базової лінії фрагмента тексту за базовою лінією тексту батьківського елемента (це стандартна поведінка). Базовою називається уявна лінія, де розташовується текст;
- sub - вирівнює базову лінію фрагмента тексту по базовій лінії нижнього індексу батьківського елементу;
- super — вирівнює базову лінію фрагмента тексту базової лінії верхнього індексу батьківського елемента;
- top – вирівнює верхній край фрагмента тексту по верхньому краю батьківського елемента;
- text-top — вирівнює верхній край текстового фрагмента по верхньому краю тексту батьківського елемента;
- middle – вирівнює центр фрагмента тексту по центру батьківського елемента;
- bottom – вирівнює нижній край фрагмента тексту по нижньому краю батьківського елемента;
- text-bottom – вирівнює нижній край фрагмента тексту по нижньому краю тексту батьківського елемента.

Ця властивість стилю придатна для вирівнювання графічних зображень, які є частиною абзацу:

<P>Це зображення: <IMG STYLE="vertical-align: text-bottom" SRC="picture.png">.</P>

Цей HTML-код створює абзац із графічним зображенням. Зображення буде вирівняно по нижньому краю тексту абзацу. Інакше кажучи, зображення буде ніби височіти над текстом.

Параметри тіні визначає властивість стилю text-shadow:

text-shadow: none | <колір> <горизонтальне зміщення> <вертикальне зміщення> [<радіус розмиття>]

Значення none (за замовчуванням) прибирає тінь у тексту.

Колір тіні задається як RGB-коду чи іменованого значення.

Горизонтальне зсунення тіні задається в будь-якій одиниці вимірювання, що підтримується CSS. Якщо встановлено позитивне зміщення, тінь буде розташована правіше тексту, якщо негативне — лівіше. Можна також задати і нульове зміщення; тоді тінь розташовуватиметься прямо під текстом. Нульове зміщення має сенс у тому разі, якщо тіні задано розмиття.

Вертикальне зміщення тіні також задається в будь-якій одиниці вимірювання, що підтримується CSS. Якщо задано позитивне усунення, тінь буде розташована ниже тексту, якщо негативне – вище. Можна також задати і нульове усунення; тоді тінь розташовуватиметься прямо під текстом.

Радіус розмиття тіні також задається в будь-якій одиниці вимірювання, що підтримується CSS. Якщо радіус розмиття не вказано, його значення передбачається рівним нулю; у такому разі тінь не матиме ефекту розмиття.

Приклад:

H1 { text-shadow: black 1mm 1mm 1px }

Тут ми задали для заголовків першого рівня (тега <H1>) тінь, розташовану правіше та нижче тексту на 1 мм і має радіус розмиття 1 піксел.

5.11. Параметри фону

Властивість стилю background-color служить для завдання кольору фону:

background-color: transparent|<колір>|inherit

Колір можна встановити як RGB-код або ім'я. Значення transparent прибирає фон зовсім; тоді елемент Web-сторінки отримає "прозорий" фон. За замовчуванням фон у елементів Web-сторінки відсутній, а фон самої Web-сторінки задає Web-браузер.

Приклад:

BODY { color: white; background-color: black }

Тут ми задали для всієї Web-сторінки чорний фон та білий текст.

Властивість стилю background-image дозволяє призначити як фон графічне зображення (фонове зображення):

background-image: none|url(<інтернет-адреса файлу зображення>);

Звернімо увагу, в якому вигляді задається інтернет-адреса файлу з фоновим зображенням: його укладають у дужки, а перед ними ставлять символи url:

TABLE.bgr { background-image: url("/table_background.png") }

Значення none видаляє графічний фон.

Графічний фон виводиться поверх звичайного фону, заданого нами за допомогою властивості стилю background-color. І, якщо фонове зображення містить "прозорі" фрагменти (таку можливість підтримують формати GIF і PNG), то на місці цих фрагментів і буде виводитись фон зазначеим кольором.

Приклад:

```
TABLE.yellow {
background-color: yellow;background-image: url("/yellow_background.png")
}
```

Тут ми задали для таблиці і звичайний, і графічний фон.

Якщо фонове зображення менше, ніж елемент Web-сторінки (або сама Web-сторінка), для якого воно задано, Web-браузер буде повторювати це зображення, поки не "замостить" ним весь елемент. Параметри цього повторення задає атрибут стилю background-repeat:

background-repeat: no-repeat|repeat|repeat-x|repeat-y|inherit

Тут є чотири значення.

- no-repeat - фонове зображення ніколи не повторюватиметься; в цьому випадку частина фону елемента Web-сторінки залишиться незаповненою ним;
- repeat - фонове зображення повторюватиметься по горизонталі та вертикалі (звичайна поведінка);
- repeat-x - фонове зображення повторюватиметься лише по горизонталі;
- repeat-y - фонове зображення повторюватиметься лише по вертикалі.

За допомогою властивості стилю background-position можна вказати фонову позицію зображення щодо елемента Web-сторінки, для якого воно призначене:

background-position: <горизонтальна позиція> [<вертикальна позиція>] |inherit;

Горизонтальна позиція фонового зображення задається у такому форматі:

<числове значення>|left|center|right

Числове значення вказує розташування фонового зображення в елементі Web-сторінки по горизонталі і може бути задано із застосуванням будь-якої з підтримуваних CSS одиниць вимірювання. Також можна вказати такі значення:

- left - фонове зображення притискається до лівого краю елемента Web-сторінки (це звичайна поведінка);
- center - розташовується по центру;
- right – притискається до правого краю.

Формат завдання вертикальної позиції фонового зображення такий:

<числове значення>|top|center|bottom

Числове значення вказує розташування фонового зображення в елементі

Web-сторінки по вертикалі і може бути задано із застосуванням будь-якої з підтримуваних CSS одиниць вимірювання.

Також можливі такі значення:

- top - фонове зображення притискається до верхнього краю елемента Web-сторінки (це звичайна поведінка);
- center - розташовується по центру;
- bottom – притискається до нижнього краю.

Якщо для будь-якого елемента Web-сторінки вказано лише позицію фонового зображення по горизонталі, його вертикальна позиція приймається рівною center.

Приклад:

TABLE.bgr (background-position: 1cm top }

Цей стиль розміщує фонове зображення на відстані 1 см від лівого краю елемента Web-сторінки і притискає його до нижнього краю елементу.

А ось стиль, що притискає фонове зображення до правого краю елементу Web-сторінки і розміщує його в центрі даного елемента по вертикалі:

TABLE.bgr (background-position: right }

Коли ми прокручуємо вміст Web-сторінки у вікні Web-браузера, разом з нею прокручується і фонове зображення (якщо воно є). Стандарт CSS пропонує цікаву можливість - заборону прокручування графічного фону Web-сторінки та фіксація його на місці. Фіксацією фону керує властивість стилю background-attachment:

background-attachment: scroll|fixed;

Значення scroll змушує Web-браузер прокручувати фон разом із вмістом Web-сторінки (це стандартна поведінка). Значення fixed фіксує фон на місці, і він не прокручуватиметься.

5.12. Параметри абзаців, списків та відображення

Почнемо ми з властивостей стилю, що управляють виведенням тексту в структуруючих текст блокових елементах. Їх дуже мало. І всі вони застосовні лише до блокових елементів.

Властивість стилю text-align задає горизонтальне вирівнювання тексту:

text-align: left|right|center|justify|inherit

Тут доступні значення left (вирівнювання по лівому краю; звичайна поведінка Web-браузера), right (правим краєм), center (по центру) і justify (вирівнювання по ширині).

Приклади:

P {text-align: justify}

H1 {text-align: center}

Властивість стилю text-indent задає відступ для першого рядка блочного елементу:

text-indent: <відступ>

Тут допускаються абсолютні та відносні (щодо ширини абзацу) величини відступу. За замовчуванням відступ дорівнює нулю.

Приклад:

P { text-indent: 5mm }

Параметри списків

Списки серед блокових елементів стоять окремо. В основному, через те, що, по-перше, містять інші блокові елементи (окремі пункти), а по-друге, включають маркери і нумерацію, які розставляє сам Webоглядач. Ось про маркери та нумерацію, а точніше, про атрибути стилю, призначені для завдання їх параметрів, ми зараз і поговоримо.

Властивість стилю list-style-type визначає вид маркерів або нумерації у пунктів списку:

list-style-type: disc|circle|square|decimal|decimal-leading-zero| lower-roman | upper-roman | lower-greek | lower-alpha | lower-latin | upper-alpha|upper-latin|armenian|georgian|none|inherit

Як бачимо, доступних значень цього атрибуту стилю дуже багато. Вони позначають як різні види маркерів, так і різні способи нумерації:

- disc - маркер у вигляді чорного кружка (звичайна поведінка для маркованих списків);
- circle - маркер у вигляді світлого кружка;
- square - маркер у вигляді квадратика. Він може бути світлим або темним, залежно від браузера;
- decimal – нумерація арабськими цифрами (звичайна поведінка для нумерованих списків);
- decimal-leading-zero - нумерація арабськими цифрами від 01 до 99 з початковим нулем;
- lower-roman – нумерація маленькими римськими цифрами;
- upper-roman – нумерація великими римськими цифрами;
- lower-greek – нумерація маленькими грецькими літерами;
- lower-alpha та lower-latin - нумерація маленькими латинськими літерами;
- upper-alpha та upper-latin – нумерація великими латинськими літерами;
- armenian - нумерація традиційними вірменськими цифрами;
- georgian – нумерація традиційними грузинськими цифрами;
- none - маркер і нумерація відсутні (звичайна поведінка для не-списків).

Властивість стилю list-style-type можна задавати як для самих списків, так і окремих пунктів списків. Ось визначення маркованого списку з маркером у вигляді квадратика:

UL { list-style-type: square }

Властивість стилю list-style-image дозволяє задати як маркер пунктів списку будь-яке графічне зображення (створити графічний маркер). У цьому випадку значення атрибуту стилю list-style-type, якщо таке задане, ігнорується:

list-style-image: none|<інтернет-адреса файлу зображення>|inherit

Інтернет-адреса файлу з графічним маркером задається в такому ж форматі, що і інтернет-адреса файлу фонового зображення:

UL { list-style-image: url(/markers/dot.gif) }

Значення none забирає графічний маркер і повертає простий, неграфічний. Це звичайна поведінка.

Властивість стилю list-style-image також можна ставити як для самих списків, так і для окремих пунктів.

Атрибут стилю list-style-position дозволяє вказати розташування маркера або нумерації у пункті списку:

list-style-position: inside|outside|inherit

Доступних значень тут два: inside - маркер або нумерація знаходяться як би всередині пункту списку, є його частиною; outside - маркер або нумерація винесені за межі пункту списку (це нормальна поведінка).

Приклад:

OL { list-style-position: inside }

Параметри відображення

Ще одна група властивостей стилю керує тим, як елемент відображатиметься на Web-сторінці, тобто він буде блоковим або вбудованим, і чи буде відображатися взагалі. Ці атрибути стилю можна застосовувати до будь-яких елементів Web-сторінок.

Атрибут стилю visibility дозволяє вказати, чи елемент відображатиметься на Web-сторінці:

visibility: visible|hidden|collapse|inherit

Він може приймати три значення: visible - елемент відображається на Web-сторінці (це звичайна поведінка); hidden — елемент не відображається на Web-сторінці, проте під нього все ще виділено у ньому місце. Іншими словами, замість елемента на Web-сторінці видно порожнє місце; collapse — застосовуємо тільки до рядків, секцій, стовпців та груп стовпців таблиці. Елемент не відображається на Web-сторінці, і під нього не виділяється місце на ній (тобто ніяких "пропіх"). Однак Web-браузер вважає, що даний елемент Web-сторінки все ще на ній присутній.

Атрибут стилю visibility застосовується досить рідко і тільки для створення спеціальних

ефектів, на кшталт зникаючого елемента Web-сторінки, що з'являється.

Атрибут стилю display дуже примітний. Він дозволяє задати вигляд елемента

Web-сторінки: буде він блоковим, вбудованим чи взагалі пунктом списку.

Приклад:

display: none|inline|block|inline-block|list-item|run-in|table| table-column|table-column-group| table-header-group|table-row-group|table-footer-group|table-row|table-cell|inherit

Доступних значень цього атрибуту стилю дуже багато. До прикладу:

- none — елемент взагалі не відображатиметься на Web-сторінці, наче він і не заданий у її HTML-коді;
- inline – вбудований елемент;
- block - блоковий елемент;
- inline-block — блоковий елемент, який "обтікатиметься" вмістом сусідніх блокових елементів;
- list-item – пункт списку;
- run-in — елемент, що вбудовується. Якщо за таким елементом слідує блоковий елемент, він стає частиною даного блокового елемента (фактично вбудованим у нього елементом), інакше він сам стає блоковим елементом;
- table – таблиця;
- inline-table - таблиця, відформатована як вбудований елемент;
- table-caption – заголовок таблиці;
- table-column — стовпець таблиці;
- table-column-group — група стовпців таблиці;
- table-header-group – секція заголовка таблиці;
- table-row-group – секція тіла таблиці;
- table-footer-group – секція завершення таблиці;
- table-row – рядок таблиці;
- table-cell — комірка таблиці;

За замовчуванням властивість стилю display залежить від конкретного елемента Web-сторінки. Так, для абзацу значенням за замовчуванням буде block, а для графічного зображення - inline.

Ось приклад комбінованого стилю, що дозволяє певним графічним зображенням відображатися як блокові елементи:

IMG.block { display: block }

А ось стиль, після застосування якого пункти списків стануть вбудованими елементами і виводитимуться в один рядок:

LI { display: inline }

Ще один приклад:

.hidden { display: none }

Використання елемента Web-сторінки цього стилю робить елемент невидимим. Більше того, на самій Web-сторінці навіть не залишиться жодної ознаки того, що даний елемент у ній був присутній.

У більшості практичних випадків достатньо значень none, block та inline атрибуту стилю display. Інші значення надто специфічні, щоб часто їх застосовувати.

5.13. Контейнери

Контейнер — елемент Web-сторінки, призначений лише виділення будь-якого її фрагменту. Таким фрагментом може бути частина блокового елемента (абзацу, заголовка, цитати, тексту фіксованого форматування та ін.), блоковий елемент або відразу кілька блокових елементів. Web-браузер не виділяє контейнер на Web-сторінці.

Контейнер служить двом цілям. По-перше, за його допомогою ми можемо прив'язати до певному елементу чи елементів Web-сторінки потрібний стиль; для цього достатньо вкласти цей елемент або елементи в контейнер і прив'язати стиль до нього. По-друге, він може забезпечувати прив'язку поведінки до елемента чи елементів Web-сторінки; виконується це так само, як і у разі стилю.

Контейнери бувають блокові та вбудовані.

Вбудований контейнер є частиною блокового елементу веб-сторінки. Так, блочним контейнером може стати фрагмент абзацу або цитати, графічне зображення, що поміщене в абзац та ін.

Вбудований контейнер створюється за допомогою парного тега <SPAN>. Фрагмент блокового елементу, який потрібно перетворити на вміст вбудованого контейнеру, поміщають у цей тег:

<P><SPAN>Вид</SPAN> створюється за допомогою стилів CSS.</P>

Тут ми помістили у вбудований контейнер фрагмент абзацу.

.bolded { font-weight: bold }

. . .

<P><SPAN CLASS="bolded">Вид</SPAN> створюється за допомогою стилів CSS.</P>

Блоковий контейнер може зберігати тільки блокові елементи: абзаци, заголовки, великі цитати, адресні теги, текст фіксованого форматування, таблиці, аудіо- та відеоролики. Блоковий контейнер може містити як один блоковий елемент, так і кілька.

Блоковий контейнер формується за допомогою парного тегу <DIV>, всередині якого розміщують HTML-код, що формує вміст контейнера

Цей тег використовується для задавання частини сторінки або фрагменту тексту. Все, що знаходиться між <DIV> та </DIV>, сприймається як один об'єкт. Перевага даного тегу в тому, що він не робить ніякого форматування тексту, а тільки помічає його частину.

Одна з переваг даного тегу – можливість задавання спільних параметрів для частини коду сторінки, яка містить різні теги.

Наприклад:

<DIV>

<H3>Це заголовок</H3>

<P>Це перший абзац. </P>

<P>Це другий абзац.</P>

</DIV>

Наступний блоковий контейнер містить таблицю.

<DIV>

<TABLE>

<CAPTION>Це таблиця</CAPTION>

<TR>

<TH>Це перший стовпець</TH>

<TH>Це другий стовпець</TH>

</TR>

. . .

</TABLE>

</DIV>

Блокові контейнери застосовують значно частіше, ніж убудовані. Саме на блочних контейнерах заснований контейнерний Web-дизайн.

Контейнерний Web-дизайн для розміщення окремих фрагментів вмісту Web-сторінок використовує блокові контейнери. Окремі контейнери створюються для заголовку Web-сайту, смуги навігації, основного вмісту та відомостей про авторські права. Якщо основний вміст має складну структуру і складається з багатьох незалежних частин, для кожної з них створюють окремий контейнер.

Для визначення різних параметрів блокових контейнерів передбачені спеціальні властивості стилю CSS. До таких параметрів відносяться розміри (ширина та висота), розташування контейнерів та їх поведінка при переповненні. Також ми можемо задати для контейнерів колір фону, створити відступи та рамки, щоб їх виділити.

Переваги:

- ми можемо розмістити контейнери на Web-сторінці практично як завгодно і помістити в них довільний вміст: текст, таблиці, зображення, аудіо- та відеоролики і навіть інші контейнери. А CSS дозволить нам задати їм практично будь-яке уявлення;
- сучасні Web-браузери дозволяють за допомогою спеціально створеної поведінки завантажити в контейнер Web-сторінку, збережену в окремому файлі, тобто організувати підвантажуваний вміст;
- контейнери та відповідні теги офіційно стандартизовані комітетом W3C та обробляються всіма Web-браузерами однаково;
- HTML-код, що формує контейнери, винятково компактний. Як ми вже знаємо, блоковий контейнер формується лише одним парним тегом <DIV>;
- усі Web-браузери відображають вміст контейнерів прямо в процесі завантаження, так що веб-сторінки візуально завантажуються дуже швидко.

Атрибути стилю першої групи визначають розміри контейнерів. Власне, їх можна застосовувати не тільки в контейнерах, а й у будь-яких інших блокових елементах.

Атрибути стилю width і height дозволяють задати, відповідно, ширину та висоту елемента Web-сторінки:

width: auto|<ширина>|inherit

height: auto|<висота>|inherit

Ми можемо вказати абсолютне значення розміру, скориставшись будь-якою з доступних у CSS одиниць вимірювання. Тоді розмір елемента буде незмінним:

#cnavbar { width: 100px } Можна встановити відносне значення розміру у відсотках від відповідного розмір батьківського елемента. При цьому розмір елемента буде змінюватися при зміні розмірів вікна Web-браузера:

#cheader { height: 10% }

Значення auto віддає управління цим розміром Web-браузером (звичайне поведінка). В останньому випадку Web-браузер встановить такі розміри елемента Web-сторінки, щоб у ньому повністю вмістився його вміст, і не більше.

Якщо ми призначимо для будь-якого елемента Web-сторінки відносну ширину або висоту, нам можуть стати в нагоді атрибути стилю, що вказують мінімальні і максимальні можливі розміри цього елементу. Зрозуміло, що при їх встановленні значення розміру не зможе перевищити максимальне і стати менше мінімального.

Властивості стилю min-width та min-height дозволяють визначити мінімальну ширину та висоту елементу Web-сторінки:

min-width: <ширина>

min-height: <висота>

Значення ширини чи висоти може бути як абсолютним, і відносним.

Приклад:

TABLE {min-width: 500px;min-height: 300px }

Аналогічні атрибути стилю max-width та max-height дозволяють задати максимальну, відповідно, ширину та висоту елемента Web-сторінки:

max-width: <ширина>

max-height: <висота>

І тут значення ширини чи висоти може бути як абсолютним, так і відносним:

TABLE {max-width: 90%;max-height: 60% }

Настройки розміщення. Плаваючі контейнери

Розташування блокових контейнерів (і будь-яких інших блокових елементів) на Web-сторінці визначають два дуже примітні властивості стилю.

Спочатку блокові елементи Web-сторінки розташовуються на ній по вертикалі, суворо один за одним, у тому порядку, в якому вони визначені у HTML-коді. Саме так розташовуються блокові контейнери, абзаци та заголовки на всіх створених нами Web-сторінках. Однак ми можемо встановити для блокового елементу вирівнювання по лівому або правому краю батьківського елементу (блокового контейнеру, в який вкладено, або самої Web-сторінки). При цьому блоковий елемент притискатиметься до відповідного краю батька, а решта вмісту буде обтікати його з протилежного боку.

Властивість стилю float задає, яким краєм батьківського елементу буде вирівнювати цей елемент Web-сторінки:

float: left|right|none|inherit

Можливі три значення:

- left - елемент Web-сторінки вирівнюється по лівому краю батьківського елементу, а решта вмісту обтікає його справа;
- right — елемент Web-сторінки вирівнюється праворуч батьківського елементу, а решта вмісту обтікає його зліва;
- none - звичайна поведінка елемента Web-сторінки, коли він слідує за своїм попередником і розташовується нижче за нього.

Приклад:

TABLE.left-aligned { float: left }

Після застосування цього стилю до таблиці вона буде вирівняна по лівому краю батьківського елемента, а решта вмісту буде обтікати її праворуч.

А якщо ми задамо однакове значення атрибуту стилю float для декількох наступних один за одним блокових елементів? Вони вишикуються по горизонталі один за одним у тому порядку, в якому вони визначені у HTML-коді, і будуть вирівняні за вказаним краєм батьківського елементу.

Цей атрибут стилю зазвичай застосовують для блокових контейнерів, що формують дизайн Web-сторінок. Такі контейнери називають плаваючими.

При створенні контейнерного Web-дизайну, заснованого на плаваючих контейнерах, часто доводиться поміщати будь-які контейнери нижче за ті, що були вирівняні по лівому або правому краю батьківського елементу. Якщо просто прибрати у них атрибут стилю float або задати йому значення none, результат буде непередбачуваним. Тому CSS надає можливість однозначно вказати, що даний блоковий елемент у будь-якому випадку повинен розташовуватися нижче плаваючих контейнерів, що передують йому.

Для цього є атрибут стилю clear:

clear: left | right | both | none | inherit

Як бачимо, доступних значень тут чотири:

- left — елемент Web-сторінки повинен розташовуватися нижче за всі елементи, для яких у атрибуту стилю float встановлено значення left;
- right — елемент Web-сторінки повинен розташовуватися нижче за всі елементи, для яких у атрибуту стилю float встановлено значення right;
- both — елемент Web-сторінки повинен розташовуватися нижче за всі елементи, для яких у атрибуту стилю float встановлено значення left або right;
- none - звичайна поведінка. Якщо контейнеру, для якого вказано цей атрибут стилю, передують плаваючі контейнери, задавати це значення не рекомендується.

Приклад:

#ccopyright { clear: both }

Тут ми поставили для іменованого стилю ccopyright (він застосовується до контейнера, містить відомості про авторські права) розташування нижче всіх контейнерів, вирівняні по лівому або правому краю батьківського елементу.

5.14. Позиціонування елементів

Браузер відображає HTML-документи за один прохід з початку документу до кінця. В результаті він не може переміститись назад і розмістити один елемент поверх іншого. Наприклад, в HTML не можна розмістити заголовок поверх зображення. Після того, як зображення з’явилося, не можна повернутися на крок назад і розмістити текст поверх нього.

Цього обмеження можна уникнути за допомогою стилів, а також тегу <DIV>, який дозволяє розбивати документ на окремі частини, що розміщуються на сторінці.

Насамперед розташуванням елементів Web-сторінок керує сам Web-браузер. У цьому він керується такими правилами:

- елемент виводиться на екран у тому місці, де знаходиться його HTML-код;
- фкщо для елементів встановлено значення none атрибуту стилю float або цей атрибут стилю взагалі відсутній, то елементи вишикуються один за одним по вертикалі;
- при інших значеннях атрибуту стилю float елементи вишикуються по горизонталі.

Довільно керувати розташуванням елементів Web-сторінки у цьому випадку ми не можемо. Тому такі елементи називаються такими, що не позиціонуються.

У мові CSS з'явилася можливість створювати вільно позиціоновані або вільні елементи Web-сторінки. Подібний елемент може розташовуватися будь-де на Web-сторінці, незалежно від місця в HTML-коді, де стоїть його тег.

Розглянемо особливості елементів Web-сторінки, що вільно позиціонуються:

- розташування вільно позиціонованого елемента задається довільно в вигляді горизонтальної та вертикальної координат його верхнього лівого кута. Координати задають відносно лівого верхнього кута батька даного елемента;
- під вільно позиціонований елемент на Web-сторінці місце не виділяється;
- елементи, що вільно позиціонуються, знаходяться "вище" звичайного вмісту Web-сторінки, як би "плавають" над ним і перекривають його;
- елементи, що вільно позиціонуються, можуть перекривати один одного. Звичайний вміст Web-сторінки вільні елементи перекривають у будь-якому разі;
- слово "перекривають" у попередніх двох пунктах позначає, що вміст Web-сторінки, що знаходиться під вільним елементом, не буде видно його сховає вільний елемент.
- вільно позиціоновані елементи можуть мати будь-який вміст, у тому числі і інші елементи, що вільно позиціонуються.

Існуюча реалізація CSS дозволяє зробити вільно позиціонованими тільки блокові контейнери.

Вільні елементи Web-сторінки створюють за допомогою спеціальних властивостей стилю CSS. Найважливішим є position. Він задає спосіб позиціонування елементу Web-сторінки:

position: static|absolute|relative|fixed|inherit

Ця властивість стилю може приймати чотири значення:

- static — контейнер, що не позиціонується (поведінка за умовчанням);
- absolute — елемент Web-сторінки, що вільно позиціонується. Його координати

задаються щодо верхнього лівого кута батька. Місце на Web-сторінці під такий елемент не виділяється. Якщо вміст батька прокручується, вільно позиціонований елемент переміщатиметься разом з ним;

- relative — елемент Web-сторінки, що відносно позиціонується. Його координати відраховуються щодо точки, в якій він був би, якщо був непозиційним. На Web-сторінці виділяється місце під такий елемент;
- fixed — елемент Web-сторінки, який фіксовано позиціонується. Він веде себе як вільний елемент, із двома винятками. По-перше, його координати задаються щодо верхнього лівого кута вікна Web-сторінки. По-друге, якщо вміст батька прокручується, фіксовано елемент, що позиціонується не переміщатиметься разом з ним.

Приклад:

```
#search { position: absolute }
```

Тут ми перетворили контейнер search у вільно позиціонований.

Властивості стилю left і top задають, відповідно, горизонтальну та вертикальну координати верхнього лівого кута вільно, відносно або фіксовано елементу Web-сторінки, що позиціонується:

left|top: <значення>|auto|inherit

Значення координат можна вказувати у будь-яких одиницях вимірювання, які підтримуються стандартом CSS. Значення auto повертає керування відповідною координатою Web-браузеру. Приклад:

```
#search { position: absolute;
left: 200px;
top: 100px;
width: 300px;
height: 200px }
```

Доступно інше значення позиції зване position: sticky, яке дещо новіше ніж інші. По суті, це гібрид відносної і фіксованої позиції, який дозволяє елементу, що позиціонується, вести себе ніби він відносно позиціонований, доки він не буде прокручений до певної порогової точки (наприклад, 10px від вершини вікна перегляду), після чого він стає фіксованим. Це можна використовувати, наприклад, щоб змусити навігаційну панель прокручуватися разом зі сторінкою до певної точки, а потім затримувати у верхній частині сторінки. Приклад:

```
.positioned {
  position: sticky;
  top: 30px;
  left: 30px;
```

}

Цікаве та загальне використання position: sticky полягає у створенні індексних сторінок з прокручуванням, де різні заголовки липнуть до верху сторінки, коли вони досягають його. Розмітка такого прикладу може мати такий вигляд:

```
<h1>Sticky positioning</h1>

<dl>
  <dt>A</dt>
  <dd>Apple</dd>
  <dd>Ant</dd>
  <dd>Altimeter</dd>
  <dd>Airplane</dd>
  <dt>B</dt>
  <dd>Bird</dd>
  <dd>Buzzard</dd>
  <dd>Bee</dd>
  <dd>Banana</dd>
  <dd>Beanstalk</dd>
…
```

У нормальному потоці елементи <dt> прокручувати разом з контентом. Коли ми додаємо position: sticky до елемента <dt>, разом зі значенням top 0, підтримуючі браузери приклеюватимуть заголовки до вершини вікна перегляду, коли вони будуть досягати тієї позиції. кожен наступний заголовок буде замінювати попередній при його прокручуванні вгору до тієї позиції:

```
dt {
  background-color: black;
  color: white;
  padding: 10px;
  position: sticky;
  top: 0;
  left: 0;
  margin: 1em 0;
}
```

Вільні елементи можуть перекривати один одного. При цьому елемент, визначений у HTML-коді пізніше, перекриває елемент, визначений раніше. Однак ми можемо самі

встановити порядок їх перекриття один одним, вказавши так званий z-індекс. Він є цілим числом, що вказує номер у порядку перекриття; при цьому елементи з більшим z-індексом перекривають елементи із меншим z-індексом. Z-індекс задається атрибутом стилю з ім'ям z-index:

z-index: <номер> | auto | inherit

Як мовилося раніше, z-індекс вказується як ціле число. Значення auto повертає управління перекриттям Web-браузеру. Приклад:

```
#search { position: absolute;
left: 200px;
top: 100px;
width: 300px;350 Частина V. Останні штрихи
height: 200px;
z-index: 2}
#main {position: absolute;
left: 100px;
top: 0px;
width: 600px;
height: 500px;
z-index: 0}
```

Контейнер search перекриє контейнер main, оскільки йому заданий більший z-індекс.

Ще одна властивість стилю, яка іноді може бути корисною - clip. Вона визначає координати прямокутної області, що задає видиму область вільного елементу. Фрагмент вмісту елемента, що потрапляє до цієї області (її, ще називають маскою), буде видно на Web-сторінці, решта вмісту буде прихована.

Ось синтаксис запису атрибута clip:

clip: rect(<верхній кордон>, <правий кордон>, <нижній кордон>,<лівий кордон>)|auto|inherit

Тут:

верхня межа — відстань від верхньої межі вільного елементу до верхньої межі маски по вертикалі;

правий кордон — відстань від лівого кордону вільного елементу до правої межі маски по горизонталі;

нижня межа - відстань від верхньої межі вільного елементу до нижньої межі маски по вертикалі;

ліва межа — відстань від лівої межі вільного елементу до лівої межі маски по

горизонталі.

Значення auto атрибуту стилю clip прибирає маску і тим самим робить весь вміст вільного елементу видимим. Це стандартна поведінка.

Приклад:

```
#search { position: absolute;
left: 200px;
top: 100px;
width: 300px;
height: 200px;
z-index: 2;
clip: rect(100px, 200px, 200px, 0px) }
```

5.15. Відступи, рамки та виділення

Стандарт CSS пропонує засоби для створення відступів двох видів.

1. Відступ між уявною межею елемента Web-сторінки та його вмістом – внутрішній відступ. Такий відступ належить даному елементу Web-сторінки, що знаходиться всередині нього.

2. Відступ між уявною межею даного елемента Web-сторінки та уявними межами сусідніх елементів Web-сторінки — зовнішній відступ.

Такий відступ не належить даному елементу Web-сторінки, поза ним.

Щоб краще зрозуміти різницю між внутрішнім та зовнішнім відступами, давайте розглянемо комірку таблиці. Комірка наповнена вмістом, скажімо, текстом, має уявну межу та оточена іншими комірками.

Внутрішній відступ — це відступ між кордоном комірки та її текстом, що міститься в ній.

Зовнішній відступ — це відступ між межами окремих комірок таблиці.

Властивості стилю padding-left, padding-top, padding-right та padding-bottom дозволяють задати величини внутрішніх відступів, відповідно, зліва, зверху, праворуч та знизу елементу:

padding-left|padding-top|padding-right|padding-bottom: <відступ>|auto|inherit

Ми можемо вказати як величину відступу абсолютне або відносне значення. Значення auto задає величину відступу за умовчанням.

У наступному прикладі вказується внутрішній відступ для комірок таблиці, рівний двом пікселам з усіх боків:

```
TD, TH { padding-left: 2px;padding-top: 2px;padding-right: 2px;padding-bottom: 2px }
```

А ось стиль, що створює внутрішні відступи, рівні двом сантиметрам зліва і справа:

.indented { padding-left: 2cm;padding-right: 2cm }

Ми можемо прив'язати такий стиль до абзацу та подивитися, що вийде.

Властивістьт стилю padding дозволяє відразу вказати величини внутрішніх відступів зі всіх сторін елементу Web-сторінки:

padding: <відступ 1> [<відступ 2> [<відступ 3> [<відступ 4>]]]

Якщо вказано одне значення, воно задасть величину відступу з усіх боків елемента Web-сторінки. Якщо вказано два значення, перше встановить величину відступу зверху та знизу, а друге - ліворуч і праворуч. Якщо вказано три значення, перше визначить величину відступу зверху, друге — ліворуч та праворуч, а третє — знизу. Якщо вказано чотири значення, перше задасть величину відступу зверху, друге – праворуч, третє – знизу, а четверте – зліва.

Приклад:

TD, TH { padding: 2px }

.indented { padding: 0cm 2cm 0cm 2cm }

Тут ми просто переписали визначення наведених раніше стилів із використанням властивості стилю padding.

Властивості стилю margin-left, margin-top, margin-right та margin-bottom дозволяють задати величини зовнішніх відступів, відповідно, зліва, зверху, праворуч та знизу:

margin-left|margin-top|margin-right|margin-bottom: <відступ>|auto|inherit

Тут також доступні абсолютні та відносні значення. Значення auto ставить величину відступу за умовчанням, яка, як правило, дорівнює нулю.

Приклад:

H1 { margin-top: 5mm }

Цей стиль створить у всіх заголовків першого рівня відступ зверху 5 мм.

Як значення зовнішніх відступів допустимі негативні величини:

UL { margin-left: -20px }

У цьому випадку Web-браузер створить "негативний" відступ. Такий прийом дозволяє прибрати відступи, створювані Web-браузером за умовчанням, наприклад, відступи зліва у великих цитат та списків.

Зовнішні відступи ми можемо вказати за допомогою властивості стилю margin. Він задає величини відступу одночасно з усіх боків елемента Web-сторінки:

margin: <відступ 1> [<відступ 2> [<відступ 3> [<відступ 4>]]]

Ця властивістьстилю поводиться так само, як padding.

Приклад:

H1 { margin: 5mm 0mm }

Однак ми не можемо використовувати атрибути стилю margin-left, margin-top, marginright, margin-bottom і margin для завдання зовнішніх відступів для комірок таблиць. Замість цього слід застосувати атрибут стилю border-spacing:

border-spacing: <відступ 1> [<відступ 2>]

Відступи можуть бути задані лише у вигляді абсолютних значень.

Якщо вказано одне значення, воно задасть величину відступу з усіх боків комірки таблиці. Якщо вказано два значення, перше задасть величину відступу ліворуч і праворуч, а друге — зверху та знизу.

Властивість стилю застосовується лише до таблиць (тегу <TABLE>):

TABLE { border-spacing: 1px }

Тут ми задали відступи між комірками таблиці, що дорівнює одному пікселу.

Параметри рамки

CSS також дозволяє нам створити суцільну, пунктирну або точкову рамку уявної межі елемента Web-сторінки.

Властивості стилю border-left-width, border-top-width, border-right-width і borderbottom-width задають товщину, відповідно, лівої, верхньої, правої та нижньої сторін рамки:

border-left-width|border-top-width|border-right-width|border-bottom-width: thin|medium|thick|<товщина>|inherit

Ми можемо вказати або абсолютне чи відносне числове значення товщини рамки, або одне з визначених значень: thin (тонка), medium (середня) або thick (товста). В останньому випадку реальна товщина рамки залежить від Web-браузеру. Значення товщини за замовчуванням також залежить від браузеру, тому її завжди краще встановлювати явно.

Приклад:

TD, TH { border-left-width: thin;

border-top-width: thin;

border-right-width: thin;

border-bottom-width: thin }

А ось стиль, що створює у всіх заголовків першого рівня рамку з однієї тільки нижньої сторони завтовшки 5 пікселів:

H1 { border-bottom-width: 5px }

Практично всі заголовки першого рівня виявляться підкресленими.

Властивість стилю border-width дозволяє вказати значення товщини одразу для всіх сторін рамки:

border-width: <товщина 1> [<товщина 2> [<товщина 3> [<товщина 4>]]]

Якщо вказано одне значення, воно встановить товщину всіх сторін рамки. Якщо вказано два значення, перше задасть товщину верхньої та нижньої, а друге — лівої та правої сторін рамки. Якщо вказано три значення, перше задасть товщину верхньої, друге — лівої та правої, а третє – нижньої сторін рамки. Якщо вказано чотири значення, перше задасть товщину верхньої, друге – правої, третє – нижньої, а четверте – лівої сторін рамки.

Приклад:

TD, TH { border-width: thin }

Властивості стилю border-left-color, border-top-color, border-right-color та border-bottom-color задають колір, відповідно, лівої, верхньої, правої та нижньої сторін рамки:

border-left-color|border-top-color|border-right-color| border-bottom-color: transparent|<колір>|inherit

Значення transparent задає "прозорий" колір, крізь який "просвічуватиме" фон батьківського елементу.

Приклад:

H1 { border-bottom-width: 5pxborder-bottom-color: red }

Властивість стилю border-color дозволяє вказати колір одразу для всіх сторін рамки:

border-color: <колір 1> [<колір 2> [<колір 3> [<колір 4>]]]

Вона поводиться так само, як і аналогічна властивість стилю border-width:

TD, TH { border-width: thin; border-color: black }

Властивості стилю border-left-style, border-top-style, border-right-style та border-bottom-style задають стиль ліній, якими буде намальована, відповідно, ліва, верхня, права та нижня сторона рамки:

border-left-style|border-top-style|border-right-style|border-bottom-style: none | hidden | dotted |ridge|inset|outset|inherit

Тут доступні такі значення:

- none та hidden - рамка відсутня (звичайна поведінка);
- dotted - пунктирна лінія;
- dashed - штрихова лінія;
- solid - суцільна лінія;
- double - подвійна лінія;
- groove - "вдавлена" тривимірна лінія;
- ridge - "опукла" тривимірна лінія;
- inset - тривимірна "випуклість";
- outset - тривимірне "поглиблення".

Приклад:

H1 { border-bottom-width: 5px;border-bottom-color: red;border-bottom-style: double }

Властивість стилю border-style дозволяє вказати стиль відразу для всіх сторін рамки:

border-style: <стиль 1> [<стиль 2> [<стиль 3> [<стиль 4>]]]

Вона поводиться так само, як і аналогічні властивості стилю border-width та border-color.

Приклад:

TD, TH { border-width: thin;border-color: black;border-style: dotted }

Властивості стилю border-left, border-top, border-right та border-bottom дозволяють задати всі параметри для, відповідно, лівої, верхньої, правої та нижньої сторони рамки:

border-left|border-top|border-right|border-bottom:<товщина> <стиль> <колір> | inherit

У багатьох випадках ці властивості стилю виявляються кращими:

H1 { border-bottom: 5px double red }

Властивість стилю border дозволяє встановити всі параметри відразу для всіх сторін рамки:

border: <товщина> <стиль> <колір> | inherit

TD, TH { border: thin dotted black }

Параметри виділення

Відомо безліч способів привернути увагу відвідувача до певних елементів Web-сторінок, використовуючи теги HTML або атрибути стилю CSS. Але CSS 3 пропонує нам ще один спосіб зробити це так зване виділенням:

- виділення CSS 3 є рамкою, якою оточується даний елемент Web-сторінки;
- ми можемо задавати параметри виділення: товщину, колір та стиль;
- виділення, на відміну від знайомої нам рамки CSS, не збільшує розміри елемента Web-сторінки. Отже можна спокійно застосовувати виділення, не побоюючись, що воно порушить контейнерний дизайн.

Для завдання параметрів виділення CSS 3 призначено чотири спеціальні властивості стилю.

Властивість стилю outline-width задає товщину рамки виділення:

outline-width: thin|medium|thick|<товщина>|inherit

Тут доступні самі значення, як і для знайомого нам атрибуту стилю border-width.

Приклад:

DFN { outline-width: thin }

Тут ми задали тонку рамку виділення для вмісту тегу <DFN>.

Властивість стилю outline-color задає колір рамки виділення:

outline-color: <колір>|inherit

Приклад: DFN {outline-width: thin;outline-color: black }

Тепер виділення вмісту тега <DFN> буде виведено чорним кольором.

Властивість стилю outline-style визначає стиль ліній, якими буде намальована рамка виділення:

outline-style: none | dotted | dashed | outset | inherit

Значення тут доступні ті самі, як і атрибуту стилю border-style.

Приклад: DFN {outline-width: thin;outline-color: black;outline-style: dotted }

Властивість стилю outline дозволяє встановити відразу всі параметри для рамки виділення:

outline: <товщина> <стиль> <колір> | inherit

DFN { outline: thin dotted #B1BEC6 }

Після цього всі нові терміни (тобто вміст тегів <DFN>) на наших Web-сторінках будуть виділені тонкою точковою рамкою світло-синього кольору.

5.16. Параметри таблиць

Для вирівнювання вмісту комірок таблиці по горизонталі є властивість text-align:

TD, TH {text-align: center}

Ця властивість стилю придатна для вирівнювання тексту в заголовку таблиці (тегу <CAPTION>):

CAPTION { text-align: left }

Вміст комірок по вертикалі ми вирівняємо за допомогою властивості vertical-align:

vertical-align: baseline|sub|super|top|text-top|middle|bottom| text-bottom|<проміжок між базовими лініями>|inherit

Є деяка специфіка для таблиць стосовно цієї властивості на відміну від інших тегів:

- top — вирівнює вміст комірки по її верхньому краю (звичайна поведінка);
- middle - вирівнює вміст комірки по її центру;
- bottom — вирівнює вміст комірки по її нижньому краю;

Приклад: TD, TH { vertical-align: middle }

Для завдання внутрішніх відступів між вмістом комірки та її кордоном можна використати властивості стилю padding-left, padding-top, padding-right, padding-bottom та padding.

Для завдання зовнішніх відступів між межами сусідніх комірок – властивістю border-spacing.

Приклад рамок:

TABLE { align: center; border: medium solid black; border-spacing: 1px }

TD, TH { border: thin dotted black; padding: 2px }

Ми призначили для самої таблиці тонку суцільну чорну рамку і відступ між комірками, що дорівнює одному пікселу, а для комірок цієї таблиці - тонку точкову чорну рамку і відступ між кордоном комірки та її вмістом, що дорівнює двом пікселам.

Якщо ми поставимо рамки навколо комірок таблиці, Web-браузер намалює рамку навколо кожної комірки. Така таблиця буде виглядати як набір прямокутників, укладений у великий прямокутник.

Однак у друкованих виданнях набагато частіше трапляються таблиці іншого виду, де рамки присутні лише між комірками.

Властивість стилю border-collapse вказує Web-браузеру, як малюватимуться рамки комірок у таблиці:

border-collapse: collapse|separate|inherit

separate — кожна комірка таблиці буде в окремій рамці (це звичайна поведінка);

collapse - малюються рамки, що розділяють лише комірки таблиці.

Ця властивість стилю застосовується лише до самих таблиць (тег <TABLE>).

Приклад:

TABLE { border-collapse: collapse }

Параметри розмірів

Для завдання розмірів - ширини та висоти - таблиць та їх комірок підійдуть властивості стилю width і height. Якщо потрібно встановити ширину або висоту всієї таблиці, потрібну властивість стилю вказують саме для неї:

TABLE { width: 100%; height: 300px }

Якщо потрібно встановити ширину стовпця, властивість стилю width вказують для першої комірки, що входить у цей стовпець:

```
<TABLE>
<TR>
<TH>Перший стовпець</TH>
<TH STYLE="width: 40px">Другий стовпець шириною 40 пікселів</TH>
<TH>Третій стовпець</TH>
</TR>
. . .
</TABLE>
```

Якщо потрібно встановити висоту рядка, властивість стилю height вказують для першої комірки цього рядка:

```
<TABLE>
```

```
. . .
<TR>
<TD STYLE="height: 30px">Рядок висотою 30 пікселів</TD>
. . .
</TR>
. . .
</TABLE>
```

Зазвичай усі розміри, які ми поставимо для таблиці та її комірок, - не більш ніж рекомендація для Web-браузера. Якщо вміст таблиці не буде в ній поміщатися, Web-браузер збільшить ширину або висоту таблиці. Найчастіше це може бути неприйнятним, тому стандарт CSS передбачає засоби, що дозволяють змінити таку поведінку Web-браузера.

Властивість стилю table-layout дозволяє вказати, як Web-браузер трактуватиме розміри, задані нами для таблиці та її комірок:

table-layout: auto|fixed|inherit

auto — Web-браузер може змінити розміри таблиці та її осередків, якщо вміст у них не поміщається. Це звичайна поведінка.

fixed — розміри таблиці та її осередків ні в якому разі не змінюватимуться. Якщо вмісту не вистачає, виникне переповнення, параметри якого ми можемо задавати за допомогою властивостей стилю overflow, overflow-x та overflow-y.

Приклад:

```
TABLE { table-layout: fixed;overflow: auto }
```

Інші параметри

Властивість стилю caption-side вказує розташування заголовка таблиці щодо самої таблиці:

caption-side: top | bottom | inherit

top - заголовок розташовується над таблицею (звичайна поведінка).

bottom – заголовок розташовується під таблицею.

Цей атрибут стилю застосовується до таблиці (тегу <TABLE>).

Приклад:

```
TABLE { caption-side: bottom }
```

Властивість стилю empty-cells вказує, як Web-браузер повинен виводити на екран порожні (що не мають вмісту) комірки:

empty-cells: show|hide|inherit

show — порожні комірки виводитимуться на екран. Якщо для них було задано інший фон, то на екран буде виведено фон, а якщо задані рамки, будуть виведені рамки.

hide — порожні комірки не виводитимуться на екран.

Звичайна поведінка залежить від Web-браузера, так що, якщо це критично, краще задати потрібне значення властивості стилю empty-cells.

Властивість стилю empty-cells також застосовується до таблиці (тегу <TABLE>).

Приклад:

TABLE { empty-cells: hide }

5.17. CSS3. Градієнти

Ще однією з можливостей є використання градієнтів. У тому числі у якості фону.

CSS-градієнти представлені типом даних <gradient> (en-US), спеціальним типом <image> (en-US), що складається з послідовного переходу між двома та більше кольорами. Ви можете вибрати один із трьох типів градієнтів: лінійний (створюється за допомогою функції linear-gradient (en-US)), круговий (створюється за допомогою radial-gradient (en-US)) та конічний (створюється за допомогою функції conic-gradient (en-US)). Ви можете також створювати повторювані градієнти за допомогою функцій repeating-linear-gradient (en-US), repeating-radial-gradient (en-US) та repeating-conic-gradient (en-US).

Градієнти можуть бути використані скрізь, де може бути використаний тип <image>, наприклад, як фон. Так як градієнти генеруються динамічно, вони можуть позбавити необхідності використовувати файли растрових зображень, які раніше використовувалися для досягнення схожих ефектів. Крім того, оскільки градієнти генеруються браузером, вони виглядають краще, ніж растрові зображення у разі збільшення масштабу, і їх розмір може бути змінений на льоту.

Лінійний градієнт створює кольорову смугу, що має вигляд прямої лінії.

Лінійний градієнт

Щоб створити найпростіший тип градієнту, все, що вам потрібно, це вказати два кольори. Вони називаються точки зупинки кольору. Їх має бути, як мінімум, дві, але у вас може бути стільки, скільки захочете.

.simple-linear {   background: linear-gradient(blue, pink); }

За замовчуванням лінійні градієнти йдуть зверху донизу. Ви можете змінити кут повороту шляхом завдання напрямку:

.horizontal-gradient {   background: linear-gradient(to right, blue, pink);}

Можна навіть створити градієнт, що проходить діагонально, з кута в кут:

.diagonal-gradient {   background: linear-gradient(to bottom right, blue, pink); }

Якщо ви хочете більше контролювати його напрямок, ви можете задати градієнту певний

кут:

.angled-gradient { background: linear-gradient(70deg, blue, pink);}

При використанні кута 0deg створюється вертикальний градієнт, що йде знизу нагору, 90deg створює горизонтальний градієнт, що йде зліва направо, і так далі за годинниковою стрілкою. Негативні кути йдуть проти годинникової стрілки.

Вам не потрібно обмежуватись двома кольорами – ви можете використовувати стільки, скільки хочете. За замовчуванням кольори рівномірно розподілені за градієнтом.

.auto-spaced-linear-gradient { background: linear-gradient(red, yellow, blue, orange); }

Вам не потрібно залишати ваші точки зупинок кольорів на їхній вихідній позиції. Щоб підправити їх розташування, ви можете не задавати кожному нічого, або задати одну чи дві відсоткові, а для кругових та лінійних градієнтів – абсолютні значення. Якщо ви поставите розташування з відсотками, 0% представлятиме початкову точку, тоді як 100% буде кінцевою точкою; однак, якщо необхідно, ви можете використовувати значення і поза цим діапазоном для досягнення бажаного ефекту. Якщо ви не задаватимете розташування, позиція цієї точки зупинки буде автоматично розрахована за вас так, що перша точка зупинки буде розташована на 0%, а остання – на 100%, а всі інші точки зупинки будуть розташовані на півдорозі між сусідніми зупинками.

.multicolor-linear {   background: linear-gradient(to left, lime 28px, red 77%, cyan); }

Щоб створити різкий перехід між двома кольорами, тобто отримати межу замість поступового переходу, обидві сусідні точки зупинки повинні бути встановлені в одному місці. У цьому прикладі кольору ділять точку зупинки на позначці 50% посередині градієнта:

.striped { background: linear-gradient(to bottom left, cyan 50%, palegoldenrod 50%); }

За умовчанням градієнт іде плавно від одного кольору до іншого. Ви можете додати підказку кольорів, щоб перемістити значення середньої точки переходу в певну точку градієнта. У цьому прикладі ми перемістили середню точку переходу з позначки 50% на 10%.

.color-hint { background: linear-gradient(blue, 10%, pink);}

.simple-linear { background: linear-gradient(blue, pink);}

Щоб додати всередину градієнта суцільну кольорову область без плавного переходу, додайте дві позиції точки зупинки. Точки зупинки можуть бути у двох позиціях, що еквівалентно двом поспіль точкам зупинки з тим самим кольором на різних позиціях. Колір досягне повної насиченості на першій точці, пройде з тією ж насиченістю до другої точки зупинки і перейде до кольору наступної точки зупинки через першу позицію наступної точки зупинки.

.multiposition-stops {

background: linear-gradient(to left,

```
    lime 20%, red 30%, red 45%, cyan 55%, cyan 70%, yellow 80% );
  background: linear-gradient(to left,
    lime 20%, red 30% 45%, cyan 55% 70%, yellow 80% );
}
.multiposition-stop2 {
  background: linear-gradient(to left,
    lime 25%, red 25%, red 50%, cyan 50%, cyan 75%, yellow 75% );
  background: linear-gradient(to left,
    lime 25%, red 25% 50%, cyan 50% 75%, yellow 75% );
}
```

У першому прикладі вище лаймовий колір йде від позначки 0%, далі, як зазначено, до позначки 20%, зробить перехід від лаймового до червоного через 10% ширини градієнта, досягне суцільного червоного на відмітці 30%, і залишиться таким до 45% градієнта, де він потьмяніє до блакитного, залишаючись таким ще 15% градієнта, і таке інше.

У другому прикладі кожна друга точка зупинки для кожного кольору знаходиться на тій же позиції, що перша точка зупинки сусіднього кольору, створюючи смугастий ефект.

За умовчанням градієнт плавно переходить між кольорами двох сусідніх точок зупинки, а середня точка між двома точками зупинки є середнім значенням колірного переходу. Ви можете контролювати інтерполяцію або перехід між двома точками зупинки додаванням його розташування в підказці кольору. У цьому прикладі колір досягає середньої точки переходу від лаймового до блакитного з відривом 20% градієнта замість стандартних 50%. У другому прикладі немає такої підказки, щоб підкреслити відмінність, яка отримується при її використанні:

```
.colorhint-gradient {  background: linear-gradient(to top, black, 20%, cyan);}
.regular-progression {  background: linear-gradient(to top, black, cyan);}
```

Градієнти підтримують прозорість, так що ви можете накладати фони для отримання різних ефектів. Фони накладаються знизу вгору таким чином, що перший оголошений лежатиме поверх інших:

```
.layered-image {
  background: linear-gradient(to right, transparent, mistyrose),
    url("critters.png");
}
```

Ви можете навіть нашарувати градієнти один на одного. Якщо верхні градієнти не повністю непрозорі, градієнти, що лежать під ними, також буде видно:

```
.stacked-linear {
```

```
  background:
    linear-gradient(217deg, rgba(255,0,0,.8), rgba(255,0,0,0) 70.71%),
    linear-gradient(127deg, rgba(0,255,0,.8), rgba(0,255,0,0) 70.71%),
    linear-gradient(336deg, rgba(0,0,255,.8), rgba(0,0,255,0) 70.71%);
}
```

Круговий градієнт

Як і у випадку з лінійними градієнтами, все, що вам потрібно, щоб створити круговий градієнт – це два кольори. За умовчанням центр градієнта знаходиться на позначці 50% 50%, градієнт стає овальним з урахуванням співвідношення сторін блоку, що містить його:

```
.simple-radial {  background: radial-gradient(red, blue);}
```

Знову ж, як і в лінійних градієнтів, ви можете розташувати кожну точку зупинки, вказавши значення у вигляді відсоткової або абсолютної довжини:

```
.radial-gradient {  background: radial-gradient(red 10px, yellow 30%, #1e90ff 50%); }
```

Ви можете розташувати центр градієнта за допомогою ключових значень, процентної чи абсолютної довжини. Значення у вигляді числа чи відсотка повторюються у разі, якщо зазначено лише одне з них, інакше порядок повторення визначатиметься порядком розташування, починаючи зліва та зверху:

```
.radial-gradient { background: radial-gradient(at 0% 30%, red 10px, yellow 30%, #1e90ff 50%); }
```

На відміну від лінійних градієнтів, кругового градієнта можна задавати розміри. Можливі значення включають: найближчий кут, найближча сторона, найдальший кут і найдальша сторона, найдальший кут – значення за умовчанням.

У цьому прикладі використовується значення розміру closest-side, що означає, що розмір визначається відстанню від початкової точки (центру) до найближчої сторони блоку:

```
.radial-ellipse-side {
  background: radial-gradient(ellipse closest-side,
    red, yellow 10%, #1e90ff 50%, beige);
}
```

Цей приклад схожий з попереднім, за винятком того, що його розмір вказаний як farthest-corner, що встановлює розмір градієнта значенням відстані від початкової точки до найдальшого кута блоку:

```
.radial-ellipse-far {
  background: radial-gradient(ellipse farthest-corner at 90% 90%,
    red, yellow 10%, #1e90ff 50%, beige);
}
```

Цей приклад використовує closest-side, що задає розмір кола як відстань між початковою точкою (центром) та найближчою стороною. Радіус кола – це відстань між центром градієнта та найближчою стороною. Коло, з урахуванням позиціонування в точці 25% від лівої сторони і 25% від низу, ближче до низу, у випадку коли відстань по висоті менше, ні по ширині:

```
.radial-circle-close {
  background: radial-gradient(circle closest-side at 25% 75%,
      red, yellow 10%, #1e90ff 50%, beige);
}
```

Ви можете накладати кругові градієнти так само, як лінійні. Перший оголошений буде згори, останній – знизу:

```
.stacked-radial {
  background:
      radial-gradient(circle at 50% 0,
        rgba(255,0,0,.5),
        rgba(255,0,0,0) 70.71%),
      radial-gradient(circle at 6.7% 75%,
        rgba(0,0,255,.5),
        rgba(0,0,255,0) 70.71%),
      radial-gradient(circle at 93.3% 75%,
        rgba(0,255,0,.5),
        rgba(0,255,0,0) 70.71%) beige;
  border-radius: 50%;
}
```

Конічний градієнт

CSS-функція conic-gradient() створює зображення, що складається з градієнта з переходом кольору, обгорнутим навколо центральної точки (на відміну від градієнта, що виходить навколо центральної точки). Зразком конічного градієнта можна назвати кругові діаграми та кольори, але він також може бути використаний для створення шахової дошки (клітини) та інших цікавих ефектів.

Синтаксис конічного градієнта схожий з синтаксисом кругового градієнта, але з тією відмінністю, що точки зупинки кольору розташовуються навколо градієнтної дуги, вздовж довжини кола, а не по градієнтній лінії, що йде від центру градієнта. Також точки зупинки кольору задаються тільки у відсотках або градусах, абсолютні величини неприпустимі.

У круговому градієнті кольори переходять від центру кола назовні, у всіх напрямках. У разі конічного градієнта кольори йдуть, як би обертаючись навколо центру кола, починаючи

згори і рухаючись за годинниковою стрілкою. Так само, як і в круговому градієнті, ви можете вказати розташування центру градієнта. Так само, як і в лінійному градієнті, можна змінювати кут градієнта.

Так само, як і у випадку з лінійними та круговими градієнтами, все, що вам потрібно для створення конічного градієнта – це два кольори. За замовчуванням центр градієнта знаходиться в точці 50% 50%, початок градієнта спрямований нагору:

```
.simple-conic {  background: conic-gradient(red, blue);}
```

Ви можете задати кут, в якому спрямовано початок градієнта значенням типу <angle>, з попереднім ключовим словом "from":

```
.conic-gradient {
  background: conic-gradient(from 45deg, red, orange, yellow, green, blue, purple);
}
```

Використання градієнтів, що повторюються

Функції linear-gradient (en-US), radial-gradient (en-US) і conic-gradient (en-US) не підтримують точки зупинки кольору, що автоматично повторюються. Однак, для реалізації цієї функціональності існують функції repeating-linear-gradient (en-US), repeating-radial-gradient (en-US) та repeating-conic-gradient (en-US).

Розмір градієнтної лінії або дуги, що повторюється, - це довжина від значення першої до значення останньої точки зупинки кольору. Якщо перша точка зупинки містить лише колір без вказівки довжини до точки зупинки, то використовується значення за умовчанням, що дорівнює 0. Якщо остання точка зупинки містить лише колір без вказівки довжини до точки встановлення, то використовується значення за умовчанням, що дорівнює 100%. Якщо ні те, ні інше не визначено, то лінія градієнта дорівнюватиме 100%, що означає, що лінійний і конічний градієнт не повторюватиметься, а круговий градієнт повторюватиметься, тільки якщо радіус градієнта менше, ніж відстань між центром градієнта і найдальшим кутом . Якщо перша точка зупинки визначена і має значення більше 0, градієнт буде повторюватися за умови, що розмір лінії або дуги дорівнює різниці між першою та останньою точкою зупинки, якщо ця різниця менше 100% або 360 градусів.

Лінійні градієнти, що повторюються

У цьому прикладі використовується repeating-linear-gradient (en-US) для створення градієнта, що повторюється, що йде по прямій лінії. Колірна послідовність починається наново з кожним повторенням градієнта. В даному випадку градієнт має довжину 10px.

```
.repeating-linear {
  background: repeating-linear-gradient(-45deg, red, red 5px, blue 5px, blue 10px);
}
```

Множинні лінійні градієнти, що повторюються.

Так само, як і у випадку зі звичайними лінійними та круговими градієнтами, ви можете використовувати множинні градієнти, один поверх іншого. Це має сенс тільки якщо градієнти частково прозорі, що дозволяє бачити одні градієнти крізь прозорі частини інших градієнтів, цього ж можна досягти за умови використання різних розмірів фону (background-size), при цьому можливо ще й при різних значеннях властивості background-position для кожного градієнта. Ми використовували прозорість.

В даному випадку градієнтні лінії мають довжину 300px, 230px та 300px:

```
.multi-repeating-linear {
  background:
    repeating-linear-gradient(190deg, rgba(255, 0, 0, 0.5) 40px,
      rgba(255, 153, 0, 0.5) 80px, rgba(255, 255, 0, 0.5) 120px,
      rgba(0, 255, 0, 0.5) 160px, rgba(0, 0, 255, 0.5) 200px,
      rgba(75, 0, 130, 0.5) 240px, rgba(238, 130, 238, 0.5) 280px,
      rgba(255, 0, 0, 0.5) 300px),
    repeating-linear-gradient(-190deg, rgba(255, 0, 0, 0.5) 30px,
      rgba(255, 153, 0, 0.5) 60px, rgba(255, 255, 0, 0.5) 90px,
      rgba(0, 255, 0, 0.5) 120px, rgba(0, 0, 255, 0.5) 150px,
      rgba(75, 0, 130, 0.5) 180px, rgba(238, 130, 238, 0.5) 210px,
      rgba(255, 0, 0, 0.5) 230px),
    repeating-linear-gradient(23deg, red 50px, orange 100px,
      yellow 150px, green 200px, blue 250px,
      indigo 300px, violet 350px, red 370px);
}
```

Клітчастий градієнт

Для створення картатого градієнта ми використовуємо кілька напівпрозорих градієнтів, що перекривають один одного. У першому оголошенні фону ми внесли кожну зупинку кольору окремо. У другому оголошенні властивості background використовується синтаксис багатопозиційних зупинок кольору:

```
.plaid-gradient {
  background:
    repeating-linear-gradient(90deg, transparent, transparent 50px,
      rgba(255, 127, 0, 0.25) 50px, rgba(255, 127, 0, 0.25) 56px,
      transparent 56px, transparent 63px,
      rgba(255, 127, 0, 0.25) 63px, rgba(255, 127, 0, 0.25) 69px,
```

```
      transparent 69px, transparent 116px,
      rgba(255, 206, 0, 0.25) 116px, rgba(255, 206, 0, 0.25) 166px),
    repeating-linear-gradient(0deg, transparent, transparent 50px,
      rgba(255, 127, 0, 0.25) 50px, rgba(255, 127, 0, 0.25) 56px,
      transparent 56px, transparent 63px,
      rgba(255, 127, 0, 0.25) 63px, rgba(255, 127, 0, 0.25) 69px,
      transparent 69px, transparent 116px,
      rgba(255, 206, 0, 0.25) 116px, rgba(255, 206, 0, 0.25) 166px),
    repeating-linear-gradient(-45deg, transparent, transparent 5px,
      rgba(143, 77, 63, 0.25) 5px, rgba(143, 77, 63, 0.25) 10px),
    repeating-linear-gradient(45deg, transparent, transparent 5px,
      rgba(143, 77, 63, 0.25) 5px, rgba(143, 77, 63, 0.25) 10px);

  background:
    repeating-linear-gradient(90deg, transparent 0 50px,
      rgba(255, 127, 0, 0.25) 50px 56px,
      transparent 56px 63px,
      rgba(255, 127, 0, 0.25) 63px 69px,
      transparent 69px 116px,
      rgba(255, 206, 0, 0.25) 116px 166px),
    repeating-linear-gradient(0deg, transparent 0 50px,
      rgba(255, 127, 0, 0.25) 50px 56px,
      transparent 56px 63px,
      rgba(255, 127, 0, 0.25) 63px 69px,
      transparent 69px 116px,
      rgba(255, 206, 0, 0.25) 116px 166px),
    repeating-linear-gradient(-45deg, transparent 0 5px,
      rgba(143, 77, 63, 0.25) 5px 10px),
    repeating-linear-gradient(45deg, transparent 0 5px,
      rgba(143, 77, 63, 0.25) 5px 10px);
}
```

Повторювані кругові градієнти

У цьому прикладі для створення кругового градієнта, що повторюється з центральної точки, використовується repeating-radial-gradient (en-US). Колірна послідовність починаються знову з кожною ітерацією повторення градієнта:

```
.repeating-radial {
  background: repeating-radial-gradient(black, black 5px, white 5px, white 10px);
}
```

Множинні кругові градієнти, що повторюються:

```
.multi-target {
  background:
      repeating-radial-gradient(ellipse at 80% 50%,rgba(0,0,0,0.5),
        rgba(0,0,0,0.5) 15px, rgba(255,255,255,0.5) 15px,
        rgba(255,255,255,0.5) 30px) top left no-repeat,
      repeating-radial-gradient(ellipse at 20% 50%,rgba(0,0,0,0.5),
        rgba(0,0,0,0.5) 10px, rgba(255,255,255,0.5) 10px,
        rgba(255,255,255,0.5) 20px) top left no-repeat yellow;
  background-size: 200px 200px, 150px 150px;
}
```

5.18. CSS3. Рамки та тіні

Властивість border-radius дозволяє закруглити кути малих та блокових елементів. Крива кожного кута визначається з допомогою одніого чи двох радіусів, визначає його форму - кола чи еліпса. Радіус розповсюджується на весь фон, навіть якщо елемент не має меж, точне січення визначається за допомогою властивості background-clip.

Властивість border-radius дозволяє закруглити всі кути одночасно, а за допомогою властивостей border-top-left-radius, border-top-right-radius, border-bottom-right-radius, border-bottom-left-radius можна закруглити кожен кут окремо. . Синтаксис:

border-radius: <px|em>|< %>|auto|initial|inherit

Якщо задати два значення властивості border-radius, то перше значення закруглить верхній лівий і нижній правий кут, а друге — верхній правий і нижній лівий.

Значення, задані через /, визначають горизонтальні та вертикальні радіуси. Властивість не успадковується.

Приклад:

```
div {width: 100px; height: 100px; border: 5px solid;}
.r1 {border-radius: 0 0 20px 20px;}
.r2 {border-radius: 0 10px 20px;}
.r3 {border-radius: 10px 20px;}
.r4 {border-radius: 10px/20px;}
```

.r5 {border-radius: 5px 10px 15px 30px/30px 15px 10px 5px;}

.r6 {border-radius: 10px 20px 30px 40px/30px;}

.r7 {border-radius: 50%;}

.r8 {border-top: none; border-bottom: none; border-radius: 30px/90px;}

.r9 {border-bottom-left-radius: 100px;}

.r10 {border-radius: 0 100%;}

.r11 {border-radius: 0 50% 50% 50%;}

.r12 {border-top-left-radius: 100% 20px; border-bottom-right-radius: 100% 20px;}

Властивість border-image дозволяє встановлювати зображення як рамку елементу. Основна вимога до зображення - воно має бути симетричним. border-radius не впливає border-image. Синтаксис:

border-image: <border-image-source, border-image-slice, border-image-width, border-image-outset,border-image-repeat>|initial|inherit

Приклад:

/* border-image-source и border-image-slice */

border: 10px solid transparent;

border-image: url(border_round.png) 30;

/* border-image-source и border-image-slice и border-image-repeat */

border: 10px solid transparent;

border-image: url(border_round.png) 30 space;

/* border-image-source и border-image-slice / border-image-width */

border-image: repeating-linear-gradient(45deg, #A7CECC, #A7CECC 10px, transparent 10px, transparent 20px, #F8463F 20px, #F8463F 30px,transparent 30px, transparent 40px) 20 / 20px;

/* border-image-source и border-image-slice / border-image-width / border-image-outset и border-image-repeat */

border-image: linear-gradient(to right, salmon 0%, purple 100%) 10 / 10px / 20px space;

Властивість border-image-size задає ширину зображення межі елементу. Якщо ширина не задано, то за умовчанням вона дорівнює 1. Синтаксис:

border-image-width: <px|em>|< %>|auto|initial|inherit

Приклад:

div {border-image-width: 30px;}

Ресурс рамки-зображення border-image-source задає шлях до зображення, яке використовуватиметься для оформлення меж блоку. Синтаксис:

border-image-source: none|url(url)|initial|inherit

Приклад:

div {border-image-source: url(border.png);}

border-image-slice визначає розмір частин зображення, що використовуються для оформлення меж елемента та ділить зображення на дев'ять частин: чотири кути, чотири краї між кутами та центральну частину. Синтаксис:

border-image-slice: <число>|%|fill| initial|inherit

Розмір частин рамки можна задавати за допомогою одного, двох, трьох або чотирьох значень. Одне значення встановлює межі однакового розміру кожної сторони елементу. Два значення: перше визначає розмір верхньої та нижньої межі, друге - правої та лівої. Три значення: перше визначає розмір верхньої межі, друге - правої та лівої, а третє - нижньої межі. Чотири значення: визначає розміри верхньої, правої, нижньої та лівої межі. Числове значення становить кількість px. % Розміри меж розраховуються щодо розміру зображення. Горизонтальні щодо ширини, вертикальні щодо висоти. Значення fill вказується разом із числом чи відсотковим значенням. Якщо воно задано, зображення не обрізається внутрішнім краєм рамки, а заповнює область всередині рамки.

Приклад:

div {border-image-slice: 50 20;}

border-image-repeat керує заповненням фонового зображення простору між кутами рамки. Можна задавати як за допомогою одного значення, так і за допомогою пари значень. Синтаксис:

border-image-repeat: stretch|repeat|round|space|initial|inherit

stretch - розтягує зображення на весь простір. Значення за замовчуванням;

repeat - повторює заповнюючу частину зображення, при цьому видно місця стиків з кутовою частиною, і якщо довжини зразка не вистачає, він розтягується;

round - найбільш точно заповнює проміжок між кутами рамки, дублюючи частину зображення, що заповнює, при цьому може утворити стики по середині сторони рамки;

space - діє аналогічно до repeat.

Приклад:

div {border-image-repeat: repeat;}

border-image-outset дозволяє перемістити зображення-рамку за межі меж елемента на вказану довжину. Може задаватися як за допомогою одного, так і чотирьох значень. Приклад:

div {border-image-outset: 10px;}

Властивість box-shadow прикріплює одну або кілька тіней до блоку. Властивість приймає або значення none, яке вказує на відсутність тіней, або список тіней через кому, упорядкований від початку до кінця.

Кожна тінь є окремою тінню, представленою від 2 до 4-х значень довжини, необов'язковим кольором та необов'язковим ключовим словом inset. Допустимі довжини 0; опущені кольори за промовчанням дорівнюють значенню якості color.

box-shadow: <x-offset y-offset blur розтягування color inset>|none|initial|inherit

x-offset - задає горизонтальне зсунення тіні. Позитивне значення малює тінь, зміщену вправо від тексту, негативна довжина – вліво;

y-offset - задає вертикальне зсунення тіні. Позитивне значення зміщує тінь донизу, негативне – вгору;

blur - задає радіус розмиття. Негативні значення не допускаються. Якщо значення розмиття дорівнює нулю, край тіні чіткий. В іншому випадку чим більше значення, тим більше розмитий край тіні;

розтягування - задає відстань, на яку збільшується тінь. Позитивні значення змушують тінь розширюватись у всіх напрямках на вказаний радіус. Негативні значення змушують тінь стискатися. Для внутрішніх тіней розширення тіні означає стиск форми периметра тіні;

color - задає колір тіні. Якщо колір відсутній, колір, що використовується, береться з властивості color. Для Safari колір тіні обов'язково вказувати;

inset - змінює тінь блоку, що відкидається, із зовнішньої тіні на внутрішню.

Приклад:

```
.example-shadow-1 span {
  margin: 50px;
  height: 100px;
  width: 200px;
  display: inline-block;
  box-shadow: inset 2px 2px 5px rgba(154, 147, 140, 0.5), 1px 1px 5px rgba(255, 255, 255, 1);
}
```

5.19. CSS3-переходи та фільтри

CSS3-переходи дозволяють анімувати вихідне значення CSS-властивості на нове значення з часом, керуючи швидкістю зміни значень властивостей. Більшість властивостей змінюють свої значення за 16 мілісекунд, тому час стандартного переходу, що рекомендується, — 200ms.

Зміна властивостей відбувається при настанні певної події, що описується відповідним псевдокласом. Найчастіше використовується псевдоклас: hover. Цей псевдоклас не працює на мобільних пристроях, таких як iPhone або Android. Універсальним рішенням, що працює в настільних та мобільних браузерах, буде обробка подій за допомогою JavaScript (наприклад, перемикання класів під час кліку).

Переходи застосовуються до всіх елементів, а також до псевдоелементів before і after. Завдання всіх властивостей переходу зазвичай використовують короткий запис властивості transition.

transition-property містить назву CSS-властивостей, до яких буде використано ефект переходу. Значення властивості може містити як одну властивість, так і список властивостей через кому. Під час створення переходу можна використовувати як початковий, так і кінцевий стан елемента. Властивість не успадковується.

transition-property: none|all|<властивісь1,властивість2,…,властивістьn>|initial|inherit

all - застосовує перехід до усіх властивостей.

Приклад:

div { width: 100px; transition-property: width; }

div:hover { width: 300px; }

transition-duration задає проміжок часу, протягом якого має здійснюватися перехід. Якщо різні властивості мають різні значення для переходу, вони вказуються через кому. Якщо тривалість переходу не зазначена, то анімація при зміні значень властивостей не відбуватиметься. Властивість не успадковується. Синтаксис:

transition-duration: <time(s|ms)>|initial|inherit;

Приклад:

div { transition-duration: .5s; }

transition-timing-function

Властивість задає тимчасову функцію, яка описує швидкість переходу об'єкта від одного значення до іншого. Якщо ви визначаєте більше одного переходу на елемент, наприклад, колір фону елемента та його положення, ви можете використовувати різні функції для кожної властивості. Властивість не успадковується. Синтаксис:

transition-timing-function: linear|ease|ease-in|ease-out|ease-in-out|step-start|step-end|steps(int,start|end)|cubic-bezier(n,n,n,n)|initial|inherit

ease - функція за умовчанням, перехід починається повільно, швидко розганяється і сповільнюється в кінці. Відповідає cubic-bezier(0.25,0.1,0.25,1);

linear - перехід відбувається рівномірно протягом усього часу, без вагань у швидкості. Відповідає cubic-bezier(0,0,1,1);

ease-in - перехід починається повільно, потім плавно прискорюється в кінці. Відповідає cubic-bezier(0.42,0,1,1);

ease-out - аерехід починається швидко і повільно сповільнюється в кінці. Відповідає cubic-bezier(0,0,0.58,1);

ease-in-out - аерехід повільно починається та повільно закінчується. Відповідає cubic-bezier(0.42,0,0.58,1);

cubic-bezier(x1, y1, x2, y2) - дозволяє вручну встановити значення від 0 до 1 для кривої прискорення.

Приклад:

div { transition-timing-function: linear; }

transition-delay - необов'язкова властивість дозволяє зробити так, щоб зміна властивості відбувалася не моментально, а з деякою затримкою. Синтаксис:

transition-delay: <time(s|ms)>|initial|inherit;

Приклад:

div {transition-delay: .5s;}

Короткий запис переходу - всі властивості, що відповідають за зміну зовнішнього вигляду елемента, можна поєднати в одну властивість transition

transition: transition-property transition-duration transition-timing-function transition-delay;

Приклад:

div {transition: background 0.3s ease, color 0.2s linear;}

або

div {

transition-property: height, width, background-color;

transition-duration: 3s;

transition-timing-function: ease-in, ease, linear;

}

CSS3-фільтри відтворюють візуальні ефекти в браузері, схожі на фільтри Photoshop. Фільтри можна додавати не лише до зображень, але й до будь-яких непустих елементів.

Спочатку фільтри було створено як частину специфікації SVG. Їхнім завданням було застосування ефектів, заснованих на піксельній сітці до векторної графіки. За допомогою SVG браузерами з'явилася можливість використовувати ці ефекти безпосередньо в браузерах.

Браузери обробляють сторінку попіксельно застосовуючи задані ефекти та малюють результат поверх оригіналу. Таким чином, застосовуючи кілька фільтрів можна досягати різних ефектів, вони накладаються один на одного. Що більше фільтрів, то більше часу потрібно браузеру, щоб відобразити сторінку.

Можна використовувати кілька фільтрів одночасно. Класичний спосіб застосування таких ефектів при наведенні на елемент :hover.

Значення властивосты filter:

1. blur() - значення визначається в одиницях довжини, наприклад px, em. Застосовує розмиття по Гауссу до вихідного зображення. Чим більше значення радіусу, тим більше розмиття. Якщо значення радіусу не встановлено, за замовчуванням береться 0.

2. brightness() - значення визначається в % або в десяткових дробах. Змінює яскравість зображення. Чим більше значення, тим яскравіше зображення. Значення за промовчанням 1.

3. contrast() - значення визначається в % або в десяткових дробах. Регулює контрастність зображення, тобто. різницю між найтемнішими і найсвітлішими ділянками зображення/фону. Значення за промовчанням 100%. Нульове значення приховає вихідне зображення під темно-сірим фоном. Значення, що збільшуються від 0 до 100% або від 0 до 1, поступово відкриватимуть вихідне зображення до оригінального відображення, а значення збільшуватимуть контраст між світлими і темними ділянками.

4. drop-shadow() - фільтр діє подібно до властивостей box-shadow і text-shadow. Використовує наступні значення: зсув по осі X зміщення по осі Y розмитість розтягування колір тіні. Відмінна риса фільтра у тому, що тінь додається до елементів та її вмісту з урахуванням їх прозорості, тобто. якщо елемент містить текст усередині, то фільтр додасть тінь одночасно для тексту та видимих меж блоку. На відміну від інших фільтрів, для цього фільтра обов'язкове завдання параметрів (мінімальне величина зміщення).

5. grayscale() - витягує всі кольори з картинки, роблячи на виході чорно-біле зображення. Значення визначається в % або десяткових дробах. Чим більше значення, тим сильніший ефект.

6. hue-rotate() - змінює кольори зображення в залежності від заданого кута повороту в колі кольору. Значення визначається в градусах від 0deg до 360deg. 0deg — значення за промовчанням означає відсутність ефекту.

7. invert() - фільтр робить негатив зображення. Значення задається у %. 0% не застосовує фільтр, 100% повністю перетворює кольори.

8. opacity() - фільтр працює аналогічно з властивістю opacity, додаючи прозорість елементу. Відмінна риса - браузери забезпечують апаратне прискорення для фільтра, що дозволяє підвищити продуктивність. Додатковий бонус – фільтр можна одночасно поєднувати з іншими фільтрами, створюючи при цьому цікаві ефекти. Значення задається лише у %, 0% робить елемент повністю прозорим, а 100% не має жодного ефекту.

9. saturate() - керує насиченістю кольорів, працюючи за принципом контрастного фільтра. Значення 0% прибирає кольоровість, а 100% не має жодного ефекту. Значення від 0%

до 100% зменшують насиченість кольору, вище 100% збільшують насиченість кольору. Значення може задаватися як %, так і цілим числом, 1 еквівалентно 100%.

10. sepia() - ефект, що імітує старовину та «ретро». Значення 0% не змінює вигляд елемента, а 100% повністю відтворює ефект сепії.

11. url() - функція приймає розташування зовнішнього XML-файлу з svg-фільтром, або якір до фільтра, що знаходиться у поточному документі.

12. none Значення за замовчуванням. Це означає відсутність ефекту.

Приклад:

```
ilter: blur(3px);
filter: brightness(50%);
filter: brightness(.5);
filter: contrast(20%);
filter: contrast(.2);
filter: drop-shadow(2px 3px 5px black);
filter: grayscale(.5);
filter: grayscale(50%);
filter: hue-rotate(180deg);
filter: invert(100%);
filter: opacity(30%);
filter: saturate(300%);
filter: sepia(150%);
```

5.20. CSS3-трасформації

Модель візуального форматування CSS визначає систему координат всередині кожного позиціонованого елемента. Система координат є точкою відліку властивостей зсуву. Положення та розміри в цьому координатному просторі можна розглядати як задані в пікселях, щодо точки відліку, з позитивними значеннями, що йдуть вправо та вниз. Цей координатний простір можна змінити за допомогою transform.

CSS3-трансформації дозволяють зрушувати, повертати та масштабувати елементи. Трансформації перетворюють елемент, не торкаючись інших елементів веб-сторінки, тобто. інші елементи не зрушуються щодо нього.

У HTML трансформований елемент: елемент з display: block; і display: inline-block;, а також елементи, значення властивості display яких обчислюється як table-row, table-row-group, table-header-group, table-footer-group, table-cell або table-caption.

Трансформованим вважається елемент із будь-яким встановленим значенням властивості transform, відмінним від none.

Існують два види CSS3-трансформацій – 2D та 3D. 2D-трансформації перетворюють елементи у двовимірному просторі за допомогою 2D-матриці перетворень. Ця матриця застосовується для обчислення нових координат об'єкта на основі значень властивостей transform і transform-origin. Перетворення впливають лише на візуальний рендеринг. Щодо макету сторінки вони можуть позначитися на переповненні вмісту блоку. За умовчанням точка трансформації знаходиться у центрі елементу.

Властивість transform задає вид перетворення елемента. Властивість описується за допомогою функцій трансформації, які зміщують елемент щодо його поточного положення на сторінці або змінюють початкові розміри і форму. Властивість не успадковується. Синтаксис:

transform: none|transform-functions|initial|inherit;

Функції трансформації:

matrix(a, c, b, d, x, y) - зміщує елементи та задає спосіб їх трансформації, дозволяючи об'єднати кілька функцій 2D-трансформацій в одній. Як трансформація допустимі поворот, масштабування, нахил і зміна положення. Значення a змінює масштаб по горизонталі. Значення від 0 до 1 зменшує елемент, більше 1 збільшує. Значення c деформує (зсуває) сторони елемента по осі Y, позитивне значення – вгору, негативне – вниз. Значення b деформує (зсуває) сторони елемента по осі X, позитивне значення – вліво, негативне – вправо. Значення d змінює масштаб за вертикаллю. Значення менше 1 зменшує елемент, більше 1 збільшує. Значення x зміщує елемент по осі X, позитивне вправо, негативне вліво. Значення y зміщує елемент по осі Y, позитивне значення вниз, негативне вгору;

translate(x,y) - зміщує елемент на нове місце, переміщуючи відносно звичайного положення вправо та вниз, використовуючи координати X та Y, не торкаючись при цьому сусідніх елементів. Якщо потрібно зрушити елемент вліво або вгору, потрібно використовувати негативні значення;

translateX(n) - зсуває елемент щодо його звичайного положення по осі X;

translateY(n) - зсуває елемент щодо його звичайного положення по осі Y.

scale(x,y) - масштабує елементи, роблячи їх більшими або меншими. Значення від 0 до 1 зменшують елемент. Перше значення масштабує елемент за шириною, друге - за висотою. Негативні значення відображають елемент дзеркально;

scaleX(n) - функція масштабує елемент по ширині, роблячи його ширшим чи вужче. Якщо значення більше одиниці, елемент стає ширшим, якщо значення знаходиться між одиницею та нулем, елемент стає вужчим. Негативні значення відображають елемент дзеркально горизонтально;

scaleY(n) - функція масштабує елемент за висотою, роблячи його вищим або нижчим. Якщо значення більше одиниці, елемент стає вищим, якщо значення знаходиться між одиницею і нулем — нижче. Негативні значення відображають дзеркальний елемент по вертикалі;

rotate(кут) - повертає елементи на задану кількість градусів, негативні значення від -1deg до -360deg повертають елемент проти годинникової стрілки, позитивні - за годинниковою стрілкою. Значення rotate(720deg) повертає елемент на два повні обороти;

skew(x-кут,y-кут) - використовується для деформування (перекручування) сторін елемента щодо координатних осей. Якщо вказано одне значення, друге буде визначено браузером автоматично;

skewX(кут) - деформує сторони елемента щодо осі X;

skewY(кут) - деформує сторони елемента щодо осі Y.

Приклади:

transform: matrix(1.0, 2.0, 3.0, 4.0, 5.0, 6.0);

transform: rotate(45deg);

transform: translate(12px, 50%);

transform: translateX(2em);

transform: translateY(3in);

transform: scale(2, 0.5);

transform: scaleX(2);

transform: scaleY(0.5);

transform: skew(30deg, 20deg);

transform: skewX(30deg);

transform: skewY(1.07rad);

transform: translateX(10px) rotate(10deg) translateY(5px);

Властивість transform-origin дозволяє змістити центр трансформації, щодо якого відбувається зміна положення/розміру/форми елемента. Значення за замовчуванням є центр, або 50% 50%. Використовується лише для трансформованих елементів.

transform-origin: x-axis y-axis z-axis|initial|inherit;

x-axis y-axis z-axis - одиниць довжини чи відсотків визначає, щодо якої частини елементу відбуватиметься трансформація. Значення більше 100% збільшують область трансформації елементу.

Приклад:

transform-origin: 2px;

transform-origin: bottom;

```
transform-origin: 3cm 2px;
transform-origin: left 2px;
transform-origin: right top;
```

5.21. CSS3-анімація

CSS3-анімація надає сайтам динамічності. Вона пожвавлює веб-сторінки, покращуючи взаємодію з користувачем. На відміну від CSS3-переходів, створення анімації базується на ключових кадрах, які дозволяють автоматично відтворювати та повторювати ефекти протягом заданого часу, а також зупиняти анімацію всередині циклу.

CSS3-анімація може застосовуватися практично для всіх html-елементів, а також для псевдоелементів: before і after.

Ключові кадри використовуються для визначення значень властивостей анімації в різних точках анімації. Ключові кадри визначають поведінку одного циклу анімації; анімація може повторюватися нуль або більше разів.

Ключові кадри вказуються за допомогою правила @keyframes, яке визначається таким чином:

```
@keyframes ім'я анімації { список правил }
```

Створення анімації починається з встановлення ключових кадрів правила @keyframes. Кадри визначають, які характеристики на якому етапі будуть анімовані. Кожен кадр може включати один або більше блоків об'яв з однієї або більше пар властивостей і значень. Правило @keyframes містить ім'я анімації елемента, яке пов'язує правило та блок оголошення елемента.

```
@keyframes shadow {
from {text-shadow: 0 0 3px black;}
50% {text-shadow: 0 0 30px black;}
to {text-shadow: 0 0 3px black;}
}
```

Ключові кадри створюються за допомогою ключових слів from і to (еквівалентні значенням 0% і 100%) або за допомогою процентних пунктів, які можна задавати скільки завгодно. Також можна комбінувати ключові слова та процентні пункти. Якщо кадри мають однакові властивості та значення, їх можна поєднати в одне оголошення:

```
@keyframes move {
від,
to {
top: 0;
```

```
left: 0;
}
25%,
75% {top: 100%;}
50% {top: 50%;}
}
```

Якщо 0% або 100% кадри не вказані, то браузер користувача створює їх, використовуючи значення, що обчислюються (спочатку задані) анімованої властивості.

Якщо декілька правил @keyframes визначено з тим самим ім'ям, спрацює останнє в порядку документу, а всі попередні проігноруються.

Після оголошення правила @keyframes, ми можемо посилатися на нього як animation:

```
h1 {
font-size: 3.5em;
color: darkmagenta;
animation: shadow 2s infinite ease-in-out;
}
```

Не рекомендується анімувати нечислові значення (за рідкісним винятком), оскільки результат у браузері може бути непередбачуваним. Також не слід створювати ключові кадри для значень властивостей, що не мають середньої точки, наприклад, для значень властивості color: pink та color: #ffffff, width: auto та width: 100px або border-radius: 0 та border-radius: 50% (у цьому випадку правильно буде вказати border-radius: 0%).

Правило стилю ключового кадру також може оголошувати тимчасову функцію, яка повинна використовуватися для переміщення анімації до наступного ключового кадру.

Приклад

```
@keyframes bounce {
  from {
    top: 100px;
    animation-timing-function: ease-out;
  }
  25% {
    top: 50px;
    animation-timing-function: ease-in;
  }
  50% {
```

```
    top: 100px;
    animation-timing-function: ease-out;
  }
  75% {
    top: 75px;
    animation-timing-function: ease-in;
  }
  to {
    top: 100px;
  }
}
```

П'ять ключових кадрів наведено для анімації з ім'ям "bounce". Між першим та другим ключовим кадром (тобто між 0% та 25%) використовується функція уповільнення. Між другим та третім (тобто між 25% та 50%) — функція плавного прискорення. І так далі. Елемент буде переміщатися вгору по сторінці на 50px, уповільнюючись у міру того, як він досягне своєї найвищої точки, а потім прискорюючись, коли він впаде до 100px. Друга половина анімації поводиться аналогічно, але тільки переміщає елемент на 25px вгору по сторінці.

Властивість animation-name визначає список анімацій, що застосовуються до елементу. Кожне ім'я використовується для вибору ключового кадру у правилі, яке надає значення властивостей анімації. Якщо ім'я не відповідає жодному ключовому кадру у правилі, немає властивостей для анімації, відсутнє ім'я анімації, анімація не виконуватиметься.

Якщо кілька анімацій намагаються змінити ту саму властивість, то виконається анімація, найближча до кінця списку імен.

Ім'я анімації є чутливим до регістру, не допускається використання ключового слова none. Рекомендується використовувати назву, що відображає суть анімації, при цьому можна використовувати одне або кілька слів, перерахованих через дефіс або символ нижнього підкреслення _. Властивість не успадковується. Синтаксис:

```
animation-name: keyframename|none|initial|inherit;
```

Приклади:

```
animation-name: none;
animation-name: test-01;
animation-name: -sliding;
animation-name: moving-vertically;
animation-name: test2;
animation-name: test3, move4;
```

Властивість animation-duration визначає тривалість одного циклу анімації. Визначається в секундах s або мілісекундах ms. Якщо елемент задано більше однієї анімації, можна встановити різний час кожної, перерахувавши значення через кому. Ситнтаксис:

animation-duration: time|initial|inherit;

Приклади:

animation-duration: .5s;

animation-duration: 200ms;

animation-duration: 2s, 10s;

animation-duration: 15s, 30s, 200ms;

Властивість animation-timing-function описує, як розвиватиметься анімація між кожною парою ключових кадрів. Під час затримки анімації тимчасові функції не використовуються. Синтаксис:

animation-timing-function: linear|ease|ease-in|ease-out|ease-in-out|step-start|step-end|steps(int,start|end)|cubic-bezier(n,n,n,n)|initial|inherit;

linear - лінійна функція, анімація відбувається рівномірно протягом усього часу, без вагань у швидкості;

ease - функція за замовчуванням, анімація починається повільно, швидко розганяється і сповільнюється в кінці. Відповідає cubic-bezier(0.25,0.1,0.25,1);

ease-in - фнімація починається повільно, а потім плавно прискорюється наприкінці. Відповідає cubic-bezier(0.42,0,1,1);

ease-out - анімація починається швидко і плавно сповільнюється наприкінці. Відповідає cubic-bezier(0,0,0.58,1);

ease-in-out - анімація повільно починається і повільно закінчується. Відповідає cubic-bezier(0.42,0,0.58,1);

cubic-bezier(x1, y1, x2, y2) - дозволяє встановити значення вручну;

step-start - задає покрокову анімацію, розбиваючи анімацію на відрізки, зміни відбуваються на початку кожного кроку. Обчислюється в steps(1, start);

step-end - покрокова анімація, зміни відбуваються наприкінці кожного кроку. Обчислюється у steps(1, end);

steps(кількість кроків, положення кроку) - ступінчаста тимчасов функція, яка приймає два параметри. Перший параметр вказує кількість інтервалів функції. Це має бути позитивне ціле число більше 0, якщо другим параметром не є jump-none - у цьому випадку воно має бути позитивним цілим числом більше 1. Другий параметр, який є необов'язковим, вказує позицію кроку - момент, в якому починається анімація, використовуючи одне з наступних значень:

jump-start - перший крок відбувається при значенні 0;

jump-end — останній крок відбувається за значенням 1;

jump-none - всі кроки відбуваються в межах діапазону (0, 1);

jump-both - перший крок відбувається при значенні 0, останній - при значенні 1;

start - веде себе як jump-start;

end - веде себе як jump-end;

Зі значенням start анімація починається на початку кожного кроку, зі значенням end — наприкінці кожного кроку із затримкою. Затримка обчислюється як наслідок розподілу часу анімації кількість кроків. Якщо другий параметр не вказано, використовується стандартне значення end.

Приклади:

animation-timing-function: ease;

animation-timing-function: ease-in;

animation-timing-function: ease-out;

animation-timing-function: ease-in-out;

animation-timing-function: linear;

animation-timing-function: step-start;

animation-timing-function: step-end;

animation-timing-function: cubic-bezier(0.1, 0.7, 1.0, 0.1);

animation-timing-function: steps(4, end);

animation-timing-function: ease, step-start, cubic-bezier(0.1, 0.7, 1.0, 0.1);

Властивість animation-iteration-count показує, скільки разів програється цикл анімації. Початкове значення 1 означає, що анімація відтворюватиметься від початку до кінця один раз. Ця властивість часто використовується у поєднанні зі значенням alternate властивості animation-direction, що змушує анімацію відтворюватися у порядку альтернативних циклах. Синтаксис:

animation-iteration-count: number|infinite|initial|inherit;

infinite - анімація програється нескінченно;

число - анімація буде повторюватися вказану кількість разів. Якщо число не є цілим числом, анімація закінчиться у середині останнього циклу. Негативні числа недійсні. Значення 0 викликає миттєве спрацювання анімації.

Приклад:

animation-iteration-count: infinite;

animation-iteration-count: 3;

animation-iteration-count: 2.5;

animation-iteration-count: 2, 0, infinite;

Властивість animation-direction визначає, чи має анімація відтворюватися у зворотньому порядку у деяких чи у всіх циклах. Коли анімація відтворюється у зворотному порядку, тимчасові функції також змінюються місцями. Наприклад, при відтворенні у зворотному порядку функція ease-in поводитиметься як ease-out. Властивість не успадковується. Синтаксис:

animation-direction: normal|reverse|alternate|alternate-reverse|initial|inherit;

normal - усі повтори анімації відтворюються так, як зазначено. Значення за замовчуванням;

reverse - усі повтори анімації відтворюються у зворотному напрямку від того, як вони були визначені;

alternate - кожен непарний повтор циклу анімації відтворюються у нормальному напрямку, кожен парний повтор відтворюється у зворотному напрямку;

alternate-reverse - кожен непарний повтор циклу анімації відтворюються у зворотному напрямку, кожен парний повтор відтворюється у нормальному напрямку.

Приклад:

animation-direction: normal;

animation-direction: reverse;

animation-direction: alternate;

animation-direction: alternate-reverse;

animation-direction: normal, reverse;

animation-direction: alternate, reverse, normal;

Властивість animation-play-state визначає, чи буде анімацію запущено або призупинено. Зупинка анімації всередині циклу можлива через використання цієї властивості у скрипті JavaScript. Також можна зупиняти анімацію при наведенні курсору миші на об'єкт - стан :hover. Властивість не успадковується. Синтаксис:

animation-play-state: paused|running|initial|inherit;

Властивість animation-delay визначає коли анімація почнеться. Визначається в секундах s або мілісекундах ms. Синтаксис:

animation-delay: time|initial|inherit;

Приклад: animation-delay: 5s; animation-delay: 3s, 10ms;

Властивість animation-fill-mode визначає, які значення застосовують анімацію поза часом її виконання. Після завершення анімації елемент повертається до своїх вихідних стилів. За умовчанням анімація не впливає на значення властивостей animation-name та animation-delay, коли анімація застосовується до елементу. Крім того, за умовчанням анімація не впливає на значення властивостей animation-duration та animation-iteration-count після її завершення.

Властивість animation-fill-mode може перевизначити цю поведінку. Властивість не успадковується. Синтаксис:

animation-fill-mode: none|forwards|backwards|both|initial|inherit;

forwards - після того, як анімація закінчується (як визначено значенням animation-iteration-count), анімація буде застосовувати значення властивостей до моменту закінчення анімації. Якщо animation-iteration-count більше за нуль, застосовуються значення для кінця останньої завершеної ітерації анімації (а не значення для початку ітерації, яке буде наступним). Якщо значення animation-iteration-count дорівнює нулю, значення, що застосовуються, будуть ті, які почнуть першу ітерацію (так само, як і в режимі animation-fill-mode: backwards;);

backwards - протягом періоду, визначеного за допомогою animation-delay, анімація буде застосовувати значення властивостей, визначені в ключовому кадрі, які почнуть першу ітерацію анімації. Це значення ключового кадру from (коли animation-direction: normal або animation-direction: alternate), або значення ключового кадру to (коли animation-direction: reverse або animation-direction: alternate).

both - дозволяє залишати елемент у першому ключовому кадрі до початку анімації (ігноруючи позитивне значення затримки) та затримувати на останньому кадрі до кінця останньої анімації.

Усі параметри відтворення анімації можна поєднати в одній властивості - animation, перерахувавши їх через пробіл:

animation: animation-name animation-duration animation-timing-function animation-delay animation-iteration-count animation-direction;

Для відтворення анімації достатньо вказати лише дві властивості - animation-name і animation-duration, решта властивостей прийме значення за замовчуванням. Порядок перерахування властивостей не має значення, єдиний час виконання анімації animation-duration обов'язково повинен стояти перед затримкою animation-delay.

Для одного елемента можна задавати кілька анімацій, перерахувавши їх назви через кому:

div {animation: shadow 1s ease-in-out 0.5s alternate, move 5s linear 2s;}

5.22. CSS3 -медіазапити

У 2001 році в HTML4 і CSS2 було введено підтримку апаратно-залежних таблиць стилів, що дозволило створювати стилі та таблиці стилів для певних типів пристроїв. Як медіа-типів було визначено такі: aural, braille, handheld, print, projection, screen, tty, tv. Таким чином, браузер застосовував таблицю стилів лише у випадку, коли активізувався цей тип пристрою.

Крім того, було введено ключове слово all, яке використовувалося, щоб вказати, що таблиця стилів застосовується до всіх типів носіїв.

У HTML4 медіа-запит записувався таким чином:

```
<link rel="stylesheet" type="text/css" media="screen" href="sans-serif.css">
<link rel="stylesheet" type="text/css" media="print" href="serif.css">
```

Всередині таблиці стилів також можна було оголосити, що блоки оголошень повинні застосовуватися до певних типів носіїв:

```
@media screen {
  * {font-family: sans-serif;}
}
```

Передбачаючи можливість введення нових значень та значень із параметрами в майбутньому, для браузерів була реалізована підтримка значень атрибуту медіа-носія, вказаних таким чином:

```
<link rel="stylesheet" media="screen, 3d-glasses, print and resolution > 90dpi" href="...">
```

Поточний синтаксис HTML5 та CSS3 прямо посилається на першу специфікацію Media Queries, оновлюючи правила для HTML. Також було розширено перелік характеристик медіа-носіїв.

Загалом медіа-запит складається з ключового слова, що описує тип пристрою (необов'язковий параметр) і виразу, що перевіряє характеристики даного пристрою. З усіх характеристик найчастіше перевіряється ширина пристрою width. Медіа-запит є логічним виразом, який повертає істину чи брехню.

Медіа-запити можуть бути додані такими способами:

1. За допомогою HTML:

```
<link rel="stylesheet" media="screen and (color)" href="example.css">
```

2. За допомогою правила @import всередині елемента <style> або зовнішньої таблиці стилів:

```
@import url(color.css) screen and (color);
```

3. Безпосередньо у коді сторінки:

```
<style>
@media (max-width: 600px) {
  #sidebar {display: none;}
}
</style>
```

4. Усередині таблиці стилів style.css:

```
@media (max-width: 600px) {
```

```
#sidebar {display: none;}
}
```

Таблиця стилів, прикріплена через тег <link>, завантажуватиметься разом із документом, навіть якщо її медіа-запит поверне брехню.

За допомогою логічних операторів можна створювати комбіновані медіазапити, в яких перевірятиметься відповідність кільком умовам.

Оператор and

Оператор and пов'язує один з одним різні умови:

```
@media screen and (max-width: 600px) {
/* CSS-стилі */;
}
```

Стилі цього запиту будуть застосовуватися лише для екранних пристроїв із шириною області перегляду не більше 600px.

```
@media (min-width: 600px) and (max-width: 800px) {
/* CSS-стилі */;
}
```

Стилі цього запиту будуть застосовуватися для всіх пристроїв при ширині перегляду від 600px до 800px включно.

Правило @media all and (max-width: 600px) {...} рівнозначне правилу @media (max-width: 600px) {...}.

Оператор кома

Оператор кома працює за аналогією з логічним оператором or.

```
@media screen, projection {
/* CSS-стилі */;
}
```

У цьому випадку CSS-стилі, укладені у фігурні дужки, спрацюють лише для екранних або проекційних пристроїв.

Оператор not

Оператор не дозволяє спрацювати медіазапиту у протилежному випадку. Ключове слово not додається на початку медіазапиту і застосовується до всього запиту цілком, тобто. запит

```
@media not all and (monochrome) {...}
@media not (all and (monochrome)) {...}
```

Якщо медіазапит складено з використанням оператора кома, то заперечення буде поширюватися тільки на ту частину, яка йде до коми, тобто. запит

```
@media not screen and (color), print and (color)
```

буде еквівалентний запиту

@media (не (screen and (color))), print and (color)

Оператор only

Оператор only використовується, щоб приховати стилі від старих браузерів (підтримують синтаксис медіа-запитів CSS2).

media="only screen and (min-width: 401px) and (max-width: 600px)"

Ці браузери чекають на список медіа-типів, розділених комами. І, відповідно до специфікації, вони мають відсікати кожне значення безпосередньо перед першим символом, який не є дефісом. Таким чином, старий браузер повинен інтерпретувати попередній приклад як media="only". Оскільки цього типу медіа немає, то й таблиці стилів ігноруватимуться.

Тип носія є типом пристрою, наприклад, принтери, екрани:

- all - підходить для всіх типів пристроїв;
- print - призначений для сторінок та документів, що відображаються на екрані в режимі попереднього перегляду;
- screen - призначений насамперед для екранів кольорових комп'ютерних моніторів;
- speech - призначений для синтезаторів мовлення.

До характеристик медіаносія відносяться параметри пристрою, що перевіряються. Значення, що використовуються під час завдання характеристик, є контрольними точками.

- width - перевіряє ширину перегляду. Значення задаються в одиницях довжини px, em і т.д., наприклад, (width: 800px). Зазвичай для перевірки використовуються мінімальні та максимальні значення ширини;
- min-width - застосовує правило якщо ширина області перегляду більша за значення, вказане у запиті;
- max-width - ширина області перегляду менше значення, зазначеного у запиті;
- height - перевіряє висоту області перегляду. Значення задаються в одиницях довжини, px, em тощо, наприклад, (height: 500px). Зазвичай для перевірки використовуються мінімальні та максимальні значення висоти;
- min-height - застосовує правило якщо висота області перегляду більша за значення, вказане у запиті;
- max-height - висота області перегляду якого менше значення, зазначеного у запиті;
- aspect-ratio - перевіряє співвідношення ширини до висот області перегляду. Широкоекранний дисплей із співвідношенням сторін 16:9 може бути позначений як (aspect-ratio: 16/9);
- min-aspect-ratio - перевіряє мінімальне співвідношення;
- max-aspect-ratio - максимальне співвідношення ширини до висот області перегляду;

- orientation - перевірка орієнтації області перегляду. Приймає два значення: (orientation: portrait) та (orientation: landscape).
- resolution - перевірка роздільної здатності екрана (кількість пікселів). Значення можуть також перевіряти кількість точок на дюйм (dpi) або кількість точок на сантиметр (dpcm), наприклад, (resolution: 300dpi);
- min-resolution - перевіряє мінімальну роздільну здатність екрану;
- max-resolution - максимальну;
- color - перевіряє кількість біт на кожен з компонентів кольору пристрою виведення. Наприклад (min-color: 4) означає, що екран конкретного пристрою повинен мати 4-бітну глибину кольору;
- min-color - перевіряє мінімальну кількість біт;
- max-color – максимальну кількість біт;
- color-index - перевіряє кількість записів у таблиці встановлення кольорів. Як значення вказується позитивне число, наприклад (color-index: 256);
- min-color-index - перевіряє мінімальну кількість записів;
- max-color-index - максимальну кількість записів;
- monochrome - перевіряє кількість бітів на піксель монохромного пристрою. Значення задається цілим позитивним числом, наприклад (min-monochrome: 8);
- min-monochrome - перевіряє мінімальну кількість бітів, max-monochrome - максимальну кількість бітів.

Для управління розміткою у мобільних браузерах використовується метатег viewport. Спочатку цей тег був представлений розробниками Apple для браузера Safari на iOS. Мобільні браузери відображають сторінки у віртуальному вікні перегляду, яке зазвичай ширше, ніж екран пристрою. За допомогою метатегу viewport можна контролювати розмір вікна перегляду та масштаб. Сторінки, адаптовані для перегляду різних типів пристроїв, повинні містити в розділі <head> метатег viewport:

<meta name="viewport" content="width=device-width, initial-scale=1.0">

Властивість width визначає віртуальну ширину вікна перегляду, значення device-width – фізична ширина пристрою. Іншими словами, width відображає значення document.documentElement.clientWidth, а device-width - screen.width.

При першому завантаженні сторінки властивість initial-scale керує початковим рівнем масштабування, initial-scale=1 означає, що 1 піксель вікна перегляду = 1 піксель CSS.

При складанні медіазапитів необхідно орієнтуватися так звані переломні (контрольні) точки дизайну, тобто. такі значення ширини області перегляду, в яких дизайн сайту суттєво змінюється, наприклад, з'являється горизонтальна смуга прокручування. Щоб визначити ці

точки, потрібно відкрити сайт у браузері та поступово зменшувати область перегляду:

```
/* Smartphones (вертикальна та горизонтальна орієнтація) ----------- */
@media only screen and (min-width : 320px) and (max-width : 480px) {
/* стилі */
}
/* Smartphones (горизонтальна) ----------- */
@media only screen and (min-width: 321px) {
/* стилі */
}
/* Smartphones (вертикальна) ----------- */
@media only screen and (max-width: 320px) {
/* стилі */
}
/* iPads (вертикальна та горизонтальна) ----------- */
@media only screen and (min-width: 768px) and (max-width: 1024px) {
/* стилі */
}
/* iPads (горизонтальна) ----------- */
@media only screen and (min-width: 768px) and (max-width: 1024px) and (orientation: landscape) {
/* стилі */
}
/* iPads (вертикальна) ----------- */
@media only screen and (min-width: 768px) and (max-width: 1024px) and (orientation: portrait) {
/* стилі */
}
/* iPad 3**********/
@media only screen and (min-width: 768px) and (max-width: 1024px) and (orientation: landscape) and (-webkit-min-device-pixel-ratio: 2) {
/* стилі */
}
@media only screen and (min-width: 768px) and (max-width: 1024px) and (orientation: portrait) and (-webkit-min-device-pixel-ratio: 2) {
/* стилі */
```

```
}
/* Настільні комп'ютери та ноутбуки ----------- */
@media only screen and (min-width: 1224px) {
/* стилі */
}
/* Великі екрани ----------- */
@media only screen and (min-width: 1824px) {
/* стилі */
}
/* iPhone 6+ ----------- */
@media only screen and (min-width: 414px) and (max-height: 736px) and (orientation: landscape) and (-webkit-device-pixel-ratio: 2){
/* стилі */
}
@media only screen and (min-width: 414px) and (max-height: 736px) and (orientation: portrait) and (-webkit-device-pixel-ratio: 2){
/* стилі */
}
/* Samsung Galaxy S5 ----------- */
@media only screen and (min-width: 360px) and (max-height: 640px) and (orientation: landscape) and (-webkit-device-pixel-ratio: 3){
/* стилі */
}
@media only screen and (min-width: 360px) and (max-height: 640px) and (orientation: portrait) and (-webkit-device-pixel-ratio: 3){
/* стилі */
}
```

Для створення дизайну, що дозволяє найкраще відображати сайт на різних пристроях, використовують загальні стратегії медіа-запитів:

1. Зменшення кількості колонок (стовпців) та поступове скасування обтікання для мобільних пристроїв.
2. Використання властивості max-width замість width для завдання ширини блоку-контейнера.
3. Зменшення полів та відступів на мобільних пристроях (наприклад, нижніх відступів між заголовком та текстом, лівого відступу для списків тощо).

4. Зменшення розмірів шрифтів для мобільних пристроїв.

5. Перетворення лінійних навігаційних меню на ті, що розкриваються.

6. Приховування другорядного вмісту на мобільних пристроях за допомогою display: none.

7. Підключення фонових зображень зменшених розмірів.

5.23. CSS3 Flexbox

CSS flexbox (Flexible Box Layout Module) - модуль макету гнучкого контейнера - є способом компонування елементів, в основі лежить ідея осі.

Flexbox складається з гнучкого контейнера (flex container) та гнучких елементів (flex items). Гнучкі елементи можуть вишиковуватися в рядок або стовпчик, а вільний простір, що залишився, розподіляється між ними різними способами.

Модуль flexbox дозволяє вирішувати такі завдання:

1. Розташовувати елементи в одному з чотирьох напрямків: ліворуч праворуч, праворуч ліворуч, зверху вниз або знизу вгору.

2. Перевизначати порядок відображення елементів.

3. Автоматично визначати розміри елементів так, щоб вони вписувалися в доступний простір.

4. Вирішувати проблему з горизонтальним та вертикальним центруванням.

5. Переносити елементи всередині контейнера, не допускаючи його переповнення.

6. Створювати стовпчики однакової висоти.

7. Створювати притиснутий донизу сторінки підвал сайту.

Flexbox вирішує специфічні завдання - створення одновимірних макетів, наприклад, навігаційної панелі, оскільки flex-елементи можна розміщувати лише по одній з осей (рис. 5.2).

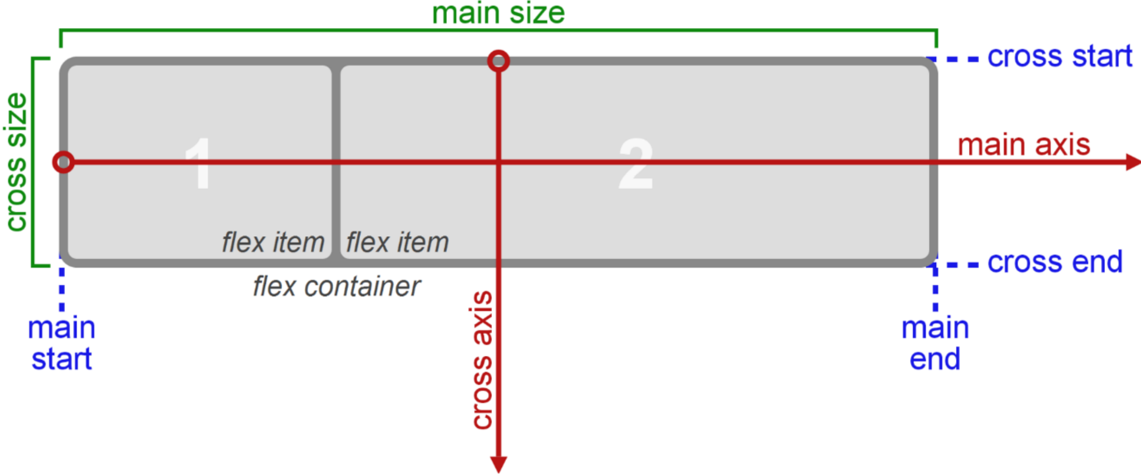


Рис. 5.2. Модель FlexBox

Для опису модуля Flexbox використовується певний набір термінів. Значення flex-flow та режим запису визначають відповідність цих термінів фізичним напрямкам: верх/право/низ/ліво, осям: вертикальна/горизонтальна та розмірам: ширина/висота.

Головна вісь (main axis) - вісь, вздовж якої викладаються flex-елементи. Вона простягається в основному вимірі.

Main start і main end — лінії, які визначають початкову та кінцеву сторони flex-контейнера, щодо яких викладаються flex-елементи (починаючи з main start до main end).

Основний розмір (main size) - ширина або висота flex-контейнера або flex-елементів, залежно від того, що з них знаходиться в основному вимірі, визначають основний розмір flex-контейнера або flex-елемента.

Поперечна вісь (cross axis) - вісь перпендикулярна головній осі. Вона простягається у поперечному вимірі.

Cross start та cross end — лінії, які визначають початкову та кінцеву сторони поперечної осі, щодо яких викладаються flex-елементи.

Поперечний розмір (cross size) - ширина або висота flex-контейнера або flex-елементів, залежно від того, що знаходиться в поперечному вимірі, є поперечним їх розміром.

Flex-контейнер встановлює новий гнучкий контекст форматування для його вмісту. Flex-контейнер не є блоковим контейнером, тому для дочірніх елементів не працюють такі CSS-властивості, як float, clear, vertical-align. Також, на flex-контейнер не впливають властивості column-*, що створюють колонки в тексті та псевдоелементи :: first-line і :: first-letter.

Модель flexbox-розмітки пов'язана з певним значенням CSS-властивості display батьківського html-елемента, що містить у собі дочірні блоки. Для керування елементами за допомогою цієї моделі потрібно встановити властивість display таким чином:

```
.flex-container {
/*генерує flex-контейнер рівня блоку*/
display: -webkit-flex;
display: flex;
}
.flex-container {
/*генерує flex-контейнер рівня рядка*/
  display: -webkit-inline-flex;
  display: inline-flex;
}
```

Після встановлення даних значень властивості кожен дочірній елемент автоматично стає flex-елементом, шикуючись в один ряд (вздовж головної осі). У цьому блокові і малі дочірні

елементи поводяться однаково, тобто. ширина блоків дорівнює ширині їхнього вмісту з урахуванням внутрішніх полів та рамок елемента.

Якщо батьківський блок містить текст або зображення без обгорток, вони стають анонімними flex-елементами. Текст вирівнюється верхнім краєм блоку-контейнеру, а висота зображення стає рівною висоті блоку, тобто. воно деформується.

Flex-елементи - блоки, що являють собою вміст flex-контейнера в потоці. Flex-контейнер встановлює новий контекст форматування для свого вмісту, який зумовлює такі особливості:

1. Для flex-елементів блокується їхнє значення якості display. Значення display: inline-block; та display: table-cell; обчислюється в display: block;
2. Порожній простір між елементами зникає: він не стає своїм власним flex-елементом, навіть якщо міжелементний текст обернутий на анонімний flex-елемент. Для вмісту анонімного flex-елемента неможливо встановити власні стилі, але він успадковуватиме їх (наприклад, параметри шрифту) від flex-контейнера.
3. Абсолютно позиціонований flex-елемент не бере участі в компонуванні гнучкого макету.
4. Поля margin сусідніх flex-елементів не зникають.
5. Відсоткові значення margin і padding обчислюються від внутрішнього розміру блоку, що містить їх.
6. margin: auto; розширюються, поглинаючи додатковий простір у відповідному вимірі. Їх можна використовувати для вирівнювання чи розсунення суміжних flex-елементів.
7. Автоматичний мінімальний розмір flex-елементів за умовчанням є мінімальним розміром його вмісту, тобто min-width:auto; Для контейнерів з прокручуванням автоматичний мінімальний розмір зазвичай дорівнює нулю.

Властивість flex-direction відноситься до flex-контейнера. Керує напрямком головної осі, вздовж якої укладаються flex-елементи, відповідно до поточного режиму запису. Властивість не успадковується.

Значення flex-direction:

- row - значення за замовчуванням, ліворуч праворуч (в rtl зправа наліво). Flex-елементи викладаються у рядок. Початок (main-start) та кінець (main-end) напряму головної осі відповідають початку (inline-start) та кінцю (inline-end) осі рядка (inline-axis);
- row-reverse - напрямок праворуч наліво (в rtl зліва направо). Flex-елементи викладаються в рядок щодо правого краю контейнера (rtl - лівого);
- column - напрямок зверху вниз. Flex-елементи викладаються у колонку;
- column-reverse - колонка з елементами у зворотному порядку, знизу догори;

Приклад:

flex-direction: row;

flex-direction: row-reverse;

flex-direction: column;

flex-direction: column-reverse;

Властивість flex-wrap визначає, буде flex-контейнер однорядковим або багаторядковим, а також задає напрямок поперечної осі, що визначає напрямок укладання нових ліній flex-контейнера. За замовчуванням flex-елементи укладаються в один рядок, уздовж головної осі. При переповненні вони виходитимуть за межі рамки flex-контейнера, що обмежує. Властивість не успадковується.

Значення flex-wrap:

- nowrap - значення за замовчуванням. Flex-елементи не переносяться, а розташовуються в одну лінію ліворуч (в rtl справа наліво).
- wrap - Flex-елементи переносяться, розташовуючись у кілька горизонтальних рядів (якщо не поміщаються в один ряд) у напрямку зліва направо (в rtl справа наліво);
- wrap-reverse - Flex-елементи переносяться на нові лінії, розташовуючись у зворотному порядку ліворуч-праворуч, при цьому перенесення відбувається знизу вгору.

Приклад:

flex-wrap: nowrap;

flex-wrap: wrap;

flex-wrap: wrap-reverse;

Властивість flex-flow дозволяє визначити напрямки головної та поперечної осей, а також можливість перенесення flex-елементів за необхідності на кілька рядків. Є скороченим записом властивостей flex-direction і flex-wrap. Значення за замовчуванням

flex-flow: row nowrap;

Властивість “order” визначає порядок, в якому flex-елементи відображаються та розташовуються усередині flex-контейнера. Спочатку всі flex-елементи мають order: 0;. При вказівці значення -1 для елемента він переміщається на початок, значення 1 - в кінець. Якщо кілька flex-елементів мають однакове значення order, вони відображатимуться відповідно до вихідного порядку.

Властивість flex є скороченим записом властивостей flex-grow, flex-shrink та flex-basis. Значення за замовчуванням: flex: 0 1 auto; Можна вказувати як одне, і всі три значення властивостей. W3C рекомендує використовувати скорочений запис, оскільки він правильно скидає будь-які невказані компоненти, щоб підлаштуватися під типове використання.

Значення flex:

- flex-grow - коефіцієнт збільшення ширини flex-елемента щодо інших flex-елементів;

- flex-shrink - коефіцієнт зменшення ширини flex-елемента щодо інших flex-елементів;
- flex-basis - основна ширина Базова ширина flex-елементу;
- auto - еквівалентно flex: 1 1 auto;.
- none - еквівалентно flex: 0 0 auto;
- initial - встановлює значення властивості значення за замовчуванням. Еквівалентно flex: 0 1 auto;
- inherit - наслідує значення властивості батьківського елемента.

Приклад:

/* Одне значення, число без одиниць: flex-grow */

flex: 2;

/* Одне значення, ширина/висота: flex-basis */

flex: 10em;

flex: 30px;

flex: auto;

flex: content;

/* Два значення: flex-grow | flex-basis */

flex: 1 30px;

/* Два значення: flex-grow | flex-shrink */

flex: 2 2;

/* Три значення: flex-grow | flex-shrink | flex-basis */

flex: 2 2 10%;

Властивість flex-grow визначає коефіцієнт зростання одного flex-елемента щодо інших flex-елементів у flex-контейнері при розподілі позитивного вільного простору. Якщо сума значень flex-grow flex-елементів у рядку менше 1, вони займають менше ніж 100% вільного простору.

Властивість flex-shrink вказує коефіцієнт стиснення flex-елемента щодо інших flex-елементів під час розподілу негативного вільного простору. Збільшується на базовий розмір flex-basis. Негативний простір розподіляється пропорційно до того, наскільки елемент може стиснутися, тому, наприклад, маленький flex-елемент не зменшиться до нуля, поки не буде помітно зменшений flex-елемент більшого розміру.

Властивість flex-basis встановлює початковий основний розмір flex-елемента до розподілу вільного простору відповідно до коефіцієнтів гнучкості. Для всіх значень, крім auto і content, базовий гнучкий розмір визначається так само, як width у горизонтальних режимах запису. Відсоткові значення визначаються щодо розміру flex-контейнера, а якщо розмір не заданий, значенням для flex-basis, що використовується, є розміри його вмісту.

Властивість justify-content вирівнює flex-елементи по головній осі flex-контейнера, розподіляючи вільний простір, незайнятий flex-елементами. Коли елемент перетворюється на flex-контейнер, flex-елементи за замовчуванням згруповані разом (якщо їм не задані поля margin). Проміжки додаються після розрахунку значень margin та flex-grow. Якщо будь-які елементи мають ненульове значення flex-grow або margin: auto;, властивість не впливатиме.

Значення justify-content:

- flex-start - значення за замовчуванням. Flex-елементи викладаються у напрямі, що йде від початку flex-контейнера;
- flex-end - flex-елементи розміщуються наприкінці flex-контейнера;
- center - flex-елементи вирівнюються центром flex-контейнера;
- space-between - flex-елементи рівномірно розподіляються по лінії. Перший flex-елемент міститься нарівні з краєм початкової лінії, останній flex-елемент — нарівні з краєм кінцевої лінії, а решта flex-елементів на лінії розподіляється так, щоб відстань між будь-якими двома сусідніми елементами була однаковою. Якщо вільний простір, що залишився, негативно або в рядку присутній тільки один flex-елемент, це значення ідентичне параметру flex-start.
- space-around - flex-елементи на лінії розподіляються так, щоб відстань між будь-якими двома суміжними flex-елементами була однаковою, а відстань між першим/останнім flex-елементами та краями flex-контейнера становила половину від відстані між flex-елементами.

Приклади:

justify-content: center;

justify-content: flex-start;

justify-content: flex-end;

justify-content: space-between;

justify-content: space-around;

Flex-елементи можна вирівнювати по поперечній осі поточного рядка flex-контейнера. align-items встановлює вирівнювання всіх елементів flex-контейнера, включаючи анонімні flex-елементи. align-self дозволяє перевизначити це вирівнювання окремих flex-елементів. Якщо будь-яке з поперечних margin flex-елемента має значення auto, align-self не має жодного впливу.

Властивість align-items вирівнює flex-елементи, у тому числі і анонімні flex-елементи поперечної осі.

Значення align-items:

- flex-start - верхній край flex-елемента міститься впритул з flex-лінією (або на відстані, з урахуванням заданих полів margin та рамок border елемента), що проходить через

початок поперечної осі;

- flex-end - нижній край flex-елемента міститься впритул з flex-лінією (або на відстані, з урахуванням заданих полів margin та рамок border елемента), що проходить через кінець поперечної осі;
- center - поля flex-елемента центрується по поперечній осі в межах flex-лінії;
- baseline - базові лінії всіх flex-елементів, що беруть участь у вирівнюванні, збігаються;
- stretch - якщо поперечний розмір flex-елементу обчислюється як auto і жодне з поперечних значень margin не дорівнює auto, елемент розтягується. Значення за замовчуванням.

Приклад:

align-items: stretch;

align-items: center;

align-items: flex-start;

align-items: flex-end;

align-items: baseline;

Властивість align-self відповідає за вирівнювання окремо взятого flex-елемента за висотою flex-контейнеру. Перевизначає вирівнювання, задане align-items. Значення аналогічні до align-items.

Властивість align-content вирівнює рядки у flex-контейнері за наявності додаткового простору на поперечній осі, аналогічно до вирівнювання окремих елементів на головній осі за допомогою властивості justify-content. Властивість не впливає на однорядковий flex-контейнер.

Приклад:

align-content: center;

align-content: flex-start;

align-content: flex-end;

align-content: space-between;

align-content: space-around;

align-content: stretch;

Контрольні питання

1. Які способи задавання стилів Ви знаєте?
2. Що таке селектор?

3. Що контекстний селектор?
4. Які спеціальні селектори Ви знаєте?
5. Які властивості стилів для списків Ви знаєте?
6. Які властивості стилів шрифтів Ви знаєте?
7. Які властивості стилів таблиць Ви знаєте?

РОЗДІЛ 6
МОВА JAVASCRIPT

6.1. Створення сценаріїв на сторінці

Основна перевага Web-сторінок – це можливість реагувати на події, що відбуваються із елементами сторінки. Зміна змісту сторінки за допомогою сценаріїв при її появі робить сторінку динамічною. При написанні сценарію використовується мова JavaScript (JScript, JS). Написання сценарію здійснюється за допомогою тегу <script>:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>JavaScript</title>
   <script>
   /* текст на мові */
   //javascript
   alert('Вітаю!');
   </script>
</head>
<body>
</body>
</html>
```

При створенні сторінки тег <SCRIPT> можна розміщувати у будь-якому місці сторінки. Стандарти HTML рекомендують розміщувати тег <SCRIPT> у заголовку сторінки, тобто між тегами <HEAD> та </HEAD>.

Якщо текст сценарію досить великий його можна помістити у окремий файл. Підключення файлу сценарію до web-сторінки слід здійснювати за допомогою атрибуту “src” тегу <SCRIPT>:

```
<script src="ex2.js">
</script>
```

Але в такому випадку не можна поміщати ніякий код між відкриваючим тегом <script> і </script>.

Мова JavaScript чутлива до регістру, тому написання Alert з великої літери в прикладі вище не призведе до виведення модального вікна.

На мові JavaScript кожну інструкцію можна відділяти одну від одної сисмволом «;»:

alert('Вітаю'); alert('програмісте);

у багатьох випадках крапку з комою може замінити перенесення на новий рядок:

alert('Вітаю')

alert('програмісте)

У цьому разі JavaScript інтерпретує перенесення рядка як “неявну” крапку з комою. Це називається автоматичне вставлення крапки з комою.

У деяких випадках новий рядок не означає нової інструкції. Наприклад:

alert(5 +

14)

Однорядкові коментарі починаються з подвійної косої риски “//”

// Це коментар

alert('Вітаю');

alert('програмісте); // Тут теж коментар

Багаторядкові коментарі починаються з косої риски з зірочкою /* і закінчується зірочкою з косою рискою */.

/* Це приклад

Багатострочного коментарію*/

alert('Вітаю');

alert('програмісте);

Багатострочні коментарі не можна вкладати один в одного.

З появою стандарту ECMAScript 5 (ES5) і додаванням нових функцій до мови і змінини деяких існуючих, щоб старий код залишався робочим, більшість таких модифікацій було вимкнено за умовчанням. Щоб увімкнути їх потрібна директива: "use strict":

"use strict";

// цей код працюватиме у сучасному режимі

Ця директива повинна йти на початку скриптів або на початку тіла функції.

Сучасний JavaScript підтримує “класи” і “модулі”, які автоматично вмикають use strict.

6.2. Змінні. Типи даних.

На мові JavaScript змінну можна об’явити за допомогою за допомогою оператору let:

let message;

message = ‘Вітаю’;

alert(message);

Можна присвоїти значеення змінній одразу:

let message = 'Вітаю';

Повторне оголошення змінної, з ім'ям яке існує, призведе до помилки.

JavaScipt дозволяє використовувати в імені змінної тільки букви та цифри, а також символи "$" та "_". Перший символ змінної не може бути числом. У якості букв рекомендується використовувати тільки латиньськи букви.

Щодо стилю: для написання імені змінної, яке містить декілька слів, зазвичай використовують стиль, коли слова йдуть одне за одним, і кожне слово пишуть із великої літери й без пробілів: myVariableWithLongName. Зауважте, що перше слово пишеться з маленької букви.

У старих скриптах можна знайти застарілу конструкцію замість let – var:

var message = 'Вітаю';

let та var дещо відрізняються стосовно області видимості щодо блоку, де вони об'явлені. Так let не існує поза блоком в якому вона об'явлена:

```
…
{
let message = 'Вітаю';
…
}
alert(message); //помилка
```

Область видимості змінної, оголошеної за допомогою var обмежена функцією, де вона була оголошена:

```
function showMessage() {
var message = 'Вітаю';
…
}
alert(message); //помилка
```

var – це застарілий спосіб оголошення змінної.

Щоб оголосити константу використовується ключове слово const. Наприклад:

const COLOR_RED = "#F00";

Змінні, оголошені за допомогою const, називаються "константами". Такі змінні не можна переприсвоювати після їх оголошення.

Значення в JavaScript завжди має певний тип даних. Оскільки JavaScript є мовою вільного використання типів, то змінна отримує тип у процесі присвоєння їй значення. Мови

програмування, які дають змогу таке робити, називаються “динамічно типізованими”.

В мові JavaScript існують такі основні типи даних: логічні (boolean), числові (number), великі числа (BigInt), рядки (string), об’єкти (object), символи (symbol), а також null та undefined.

Число (number)

Тип number представляє і цілі числа, і числа з плаваючою точкою.

```
let num = 14;
alert(typeof num); //number
num = 12.345;
alert(typeof num); //number
```

Окрім звичайних чисел тип number може зберігати спеціальні значення: “Infinity”, “-Infinity”, “NaN”. Наприклад так цей тип зберігає нескінчено велике число:

```
let inf = 1 / 0;
alert(typeof inf); //number
alert(inf); //Infinity
```

Для отримання найбільшого або найменшого доступного значення в межах +/-Infinity можна використовувати константи Number.MAX_VALUE або Number.MIN_VALUE. Починаючи з ECMAScript 2015, ви також можете перевірити, чи знаходиться число в безпечному для цілих чисел діапазоні, використовуючи метод Number.isSafeInteger(), або константи Number.MAX_SAFE_INTEGER і Number.MIN_SAFE_INTEGER. За межами цього діапазону операції з цілими числами будуть небезпечними та повертатимуть наближені значення.

Нуль JavaScript має два уявлення: -0 і +0. ("0" це синонім +0). Насправді це має малопомітний ефект. Наприклад, вираз +0 === -0 є істинним. Однак це може проявитися при діленні на нуль:

```
alert(42 / +0); //Infinity
alert(42 / -0); //-Infinity
```

NaN результат можна отримати, коли в результаті операції не отримується коректне число:

```
let incorrectNum = "abc" / 0;
alert(typeof incorrectNum); //number
alert(incorrectNum); //NaN
```

Ви можете використовувати чотири типи числових літералів: десятковий, двійковий, вісімковий та шістнадцятковий.

Синтаксис двійкових чисел використовує провідний 0, за яким слідує латинська літера

"B" у верхньому або нижньому регістрі (0b or 0B). Якщо цифри після 0b не є 0 або 1, то буде згенеровано SyntaxError з повідомленням: "Missing binary digits after 0b".

```
let FLT_SIGNBIT = 0b10000000000000000000000000000000; // 2147483648
let FLT_EXPONENT = 0b01111111100000000000000000000000; // 2139095040
let FLT_MANTISSA = 0B00000000011111111111111111111111; // 8388607
```

Синтаксис вісімкових чисел використовує нуль на початку. Якщо цифри після 0 не входять до діапазону від 0 до 7, число буде інтерпретовано як десяткове.

```
let n = 0755; // 493
let m = 0644; // 420
```

Синтаксис шістнадцяткових чисел використовує провідний 0 за яким слідує латинська буква "X" у верхньому або нижньому регістрі (0x or 0X). Якщо цифри після 0x не входять до діапазону (0123456789ABCDEF), то буде згенеровано SyntaxError з повідомленням: "Identifier starts immediately after numeric literal".

```
0xFFFFFFFFFFFFFFFFF // 295147905179352830000
0x123456789ABCDEF // 81985529216486900
0XA // 10
```

BigInt

У JavaScript, тип "number" не може містити числа більші за (2^{53}-1) (це 9007199254740991), або менші за -(2^{53}-1) для від'ємних чисел.

Нещодавно в мову був доданий тип BigInt для представлення цілих чисел довільної довжини:

```
const bigInt = 1234567890123456789012345678901234567890n;
alert(typeof bigInt); //bigint
```

Рядок (string)

Рядок у JavaScript має бути узятий у лапки. У JavaScript є три способи об'яви змінних типу string:

- Подвійні лапки: "Привіт".
- Одинарні лапки: 'Привіт'.
- Зворотні лапки: `Привіт`.

Подвійні та одинарні лапки є "звичайними". Тобто немає ніякої різниці, які саме використовувати. Зворотні лапки є розширенням функціональності. Вони дають змогу вбудовувати змінні та вирази в рядок, обрамляючи їх в ${…}, наприклад:

```
let name = "Іван";
// вбудована змінна
alert(`Привіт, ${name}е!`); // Привіт, Іване!
```

// вбудований вираз

alert(`результат: ${1 + 2}`); // результат: 3

Вираз всередині ${…} обчислюється, а результат обчислення стає частиною рядка. Ми можемо вбудувати будь-що: змінну name, або арифметичний вираз 1 + 2, або щось набагато складніше. Будь ласка, зауважте, що вбудовування можна робити тільки зі зворотніми лапками. Інші типи лапок не мають функціональності вбудовування.

Булевий або логічний тип (boolean)

Логічний тип має лише два значення: істина (true) та неправда (false):

let isVisible = true;

Результатом виконання логічних операцій є тип boolean:

let isGreatThen = 7>3;

Значення “null”

Спеціальне значення null свідчить про відсутність об’єктного значення. null є примітивом, і в контексті логічних операцій розглядається як хибне (false).

let refObj = null;

Значення “undefined”

Спеціальне значення undefined представляє власний тип, який означає, що значення не присвоєне. null є певним значенням відсутності об'єкта, тоді як undefined означає невизначеність.

Якщо змінна оголошена, але їй не присвоєне якесь значення, тоді значення такої змінної буде undefined:

let birthday;

alert(typeof birthday);

При перевірці на null або undefined, пам'ятайте про різницю між операторами рівності (==) і ідентичності (===): з першим, виконується перетворення типів.

typeof null // object (не "null" з міркувань зворотної сумісності)

typeof undefined // undefined

null === undefined // false

null == undefined // true

Об’єкти (object) та символи (symbol)

Тип object є особливим типом. Усі інші типи називаються “примітивами”, тому що їхні значення можуть містити тільки один елемент (це може бути рядок, число, або будь-що інше). В об’єктах же зберігаються колекції даних і більш складні структури.

Тип symbol використовується для створення унікальних ідентифікаторів в об’єктах. Ми згадали цей тип для повноти, проте докладніше вивчимо його після об’єктів.

JavaScript дозволяє працювати з примітивами (рядок, число, тощо) так само як з об’єктами. Вони також надають методи для роботи. Ми вивчимо їх найближчим часом, але спочатку подивимось як воно працює, тому що примітиви не є об’єктами (і тут ми зробимо це ще більш зрозумілим).

Примітив є значенням примітивного типу. Існує 7 типів примітивів: string, number, bigint, boolean, symbol, null та undefined.

Об’єкт:

- можна зберігати декілька значень як властивості.

- може бути створений за допомогою {}, наприклад: {name: "Іван", age: 30}. В JavaScript існують й інші об’єкти: функції — це теж об’єкти.

Примітиви повинні бути максимально швидкими та легкими. Примітиви залишаються примітивами. Лише значення, як ви і хотіли. JavaScript дозволяє отримати доступ до методів та властивостей рядків, чисел, булеанів та символів. Для цього створюється спеціальний “об’єкт обгортка” з додатковою функціональністю, який потім знищується.

Для кожного примітиву створюється своя “обгортка”: String, Number, Boolean, Symbol та BigInt. Отже, вони містять різні набори методів.

Наприклад: існує такий метод для рядка, як str.toUpperCase(), який поверне рядок str з великими літерами:

```
let mes = "Вітаю";
alert( mes.toUpperCase() ); //Вітаю
```

Отже, примітиви можуть надавати методи, але залишаються “легкими”.

Оператор typeof

Оператор typeof повертає строку що містить тип переданного аргументу. typeof можна використовувати і як оператор і як функцію. Виклик typeof x повертає рядок із назвою типу:

```
typeof undefined // "undefined"
typeof 0 // "number"
typeof 10n // "bigint"
typeof true // "boolean"
typeof "foo" // "string"
typeof Symbol("id") // "symbol"
typeof Math // "object"  (1)
typeof null // "object"  (2)
typeof alert // "function"  (3)
```

Результатом typeof null є "object". null не є об’єктом. Це особливе значення з власним типом. У цьому разі поведінка typeof некоректна, але залишена для сумісності.

Результатом typeof alert є "function", тому що alert — це функція. Але в JavaScript немає спеціального типу “function”. Функції належать до типу “об’єкт”.

Перетворення типів

Оператори та функції здебільшого неявно перетворюють значення на необхідний їм тип. Наприклад, функція alert автоматично перетворю будь-яке значення на рядок, щоб вивести його у діалоговому вікні.

Перетворення на рядок

Перетворення на рядок відбувається, коли нам потрібне значення у формі рядка.

Можна викликати функцію String(value) для перетворення значення в рядок:

```
let isEnabled = true;
let v = String(isEnabled);
alert(typeof v); //string
```

Перетворення на число

Перетворення на числа відбувається в математичних функціях і виразах автоматично. Наприклад:

```
alert(“14” / “7”); // 2
```

при цьому для виконання операції рядки перетворюються на числа.

Для явного перетворення можна скористатися функцією Number:

```
let s = "19";
let age = Number(s);
alert(age); // 19 як число
alert(typeof age); // number
```

Правила перетворення на числа:

- undefined перетворюється в NaN;
- null в 0;
- true та false в 1 та 0 відповідно;
- string - Пробіли на початку та з кінця видаляються. Якщо після цього рядок містить число, то результатом буде NaN. Інакше – отримауємо число.

Приклади:

```
alert( Number("    789   ") ); // 789
alert( Number("789z") );      // NaN (помилка читання числа на місці символу "z")
alert( Number(true) );        // 1
alert( Number(false) );       // 0
```

Для перетворення в число JavaScript передбачає такі функції:

1. ParseFloat(<строка>) – перетворює вміст строки в число з плаваючою точкою.

```
let  mystr="3.14";
alert(parseFloat(mystr)+1.1); //4.24
alert(parseFloat("1ab2")+1); //2
alert(parseFloat("ab")+1); //NaN
```

Як видно з прикладу, на відміну від функції Number перетворення строки в число відбувається до першого нечислового символу. Якщо строка містить не число, то результатом буде NaN (Not a Number). Перевірити чи може строка бути перетвореною на число можна функцією isNaN(<строка>), яка вертає true або false. Наприклад:

```
alert(isNaN(Number("12.3"))); //false
alert(isNaN(Number("1ab2"))); //true
alert(isNaN(Number("ab2"))); //true
```

2. ParseInt(<строка>, <основа>) – перетворює вміст строки в ціле число (основа – система числення 2-36). Якщо значення параметра string не належить рядковому типу, воно перетворюється на нього. Приклади:

```
alert(parseInt("3.64")); //3
alert(parseInt('0xFF', 16)); //255
```

Прототип Number надає ряд методів для отримання значення числа в різних форматах:

toExponential() Повертає рядок, що представляє число в експоненційному поданні.

toFixed() Повертає рядок, що представляє число із заданою кількістю розрядів після коми.

toPrecision() Повертає рядок, що представляє число із зазначеною точністю.

Вбудований глобальний об'єкт Math містить властивості та методи для математичних констант та функцій.

Цей об’єкт вважається вбудованим і використовується в явному вигляді, як і об’єкти document, window.

Він містить властивості: Math.PI - число та ін., а також методи cos, sin, ceil, floor та ін.

Наприклад, функція округлення до будь скількох знаків після коми з використанням возведення в ступінь:

```
function round_dec(numb, dec){
return Math.round(numb*Math.pow(10,dec))/ Math.pow(10,dec);
}
```

Перетворення на булевий тип

Перетворення на булевий тип відбувається при логічних операціях або явно за допоогою виклику функції Boolean.

Правила перетворення:

- значення, такі як 0, порожній рядок, null, undefined та NaN, стають false;
- інші значення стають true.

Наприклад:

```
alert( Boolean(1) ); // true
alert( Boolean(0) ); // false
alert( Boolean("вітаю") ); // true
alert( Boolean("") ); // false
```

6.3. Оператори.

JavaScript підтримує такі математичні операції, яккі наведені в табл. 6.1.

Операнд – це те, до чого застосовуються оператори. Наприклад, у множенні 5 * 2 є два операнди: лівий операнд 5 і правий операнд 2. Іноді їх називають "аргументами", а не "операндами".

Оператор є унарним, якщо він має один операнд. Наприклад, унарне заперечення - змінює знак числа:

```
let x = 1;
x = -x;
alert( x ); // -1, було застосоване унарне заперечення
```

Оператор є бінарним, якщо він має два операнди. Наприклад, оператор мінус можна використовувати і у бінарній формі:

```
let x = 1, y = 3;
alert( y - x ); // 2, бінарний мінус віднімає значення
```

Таблиця 6.1

Математичні операції

Оператор	Назва	Приклад	Результат
+	додавання	10+4	14
++	інкремент	10++	11
-	віднімання	10-4	6
-	від'ємність	-10	-10
--	декремент	10--	9
*	множення	10*4	40
/	ділення	10/4	2,5
%	залишок від ділення	11%3	2
**	піднесення до ступеню	2**4	16

Зазвичай оператор плюс + додає числа. Але якщо бінарний + застосовується до рядків, він об'єднує їх:

let s = 'мій_' + 'рядок';

alert(s); // мій_рядок

Зверніть увагу, якщо будь-який з операндів є рядком, тоді інший також перетворюється на рядок. Наприклад:

alert('3' + 4); // "34"

alert(3 + '4'); // "34"

Ось ще приклад:

alert(4 + 3 + '2'); // "72", а не "432"

Тут оператори виконуються один за одним зліва направо. Перший + додає два числа, тому він поверне 7; а наступний оператор + вже додасть (об'єднає) попередній результат із рядком 2.

alert('4' + 3 + 2); // "432", а не "45"

У цьому прикладі перший операнд – рядок, тому інші два операнди опрацьовуються як рядки. Операнд 4 приєднується (конкатенується) до '3', тому в результаті буде '4' + 3 = "43", а потім — "43" + 2 = "432".

Лише бінарний + працює з рядками так. Інші арифметичні оператори працюють тільки з числами й завжди перетворюють свої операнди на числа. Ось приклад:

alert(14 - '3'); // 11, '2' перетворюється на число

alert('9' / '3'); // 3, обидва операнди перетворюються на числа

Оператори арифметичного присвоєння приведено у табл. 6.2.

Таблиця 6.2

Оператори арифметичного присвоєння

Оператор	Приклад	Аналогічний вираз
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
^=	x^=y	x=x^y
%=	x%=y	x=x%y

У оператора плюс + є дві форми: бінарна, яку ми використовували вище, та унарна.

Унарний плюс або, іншими словами, оператор плюс +, застосований до одного операнда, нічого не зробить, якщо операнд є числом. Але якщо операнд не є числом, унарний плюс перетворить його на число. Наприклад:

// Нема ніякого впливу на числа

let x = 1;

alert(+x); // 1

```
let y = -2;
alert( +y ); // -2
// Перетворює нечислові значення
alert( +true ); // 1
alert( +"" );   // 0
```

Він насправді працює як і Number(...), але має коротший вигляд.

Необхідність перетворення рядків на числа виникає дуже часто. Наприклад:

```
let apples = "2";
let oranges = "3";
alert( apples + oranges ); // "23", бінарний плюс об’єднує рядки
```

Якщо ми хочемо використовувати їх як числа, нам потрібно конвертувати, а потім підсумувати їх:

```
let apples = "2";
let oranges = "3";
// обидва значення перетворюються на числа перед застосуванням бінарного плюса
alert( +apples + +oranges ); // 5
// довший варіант
// alert( Number(apples) + Number(oranges) ); // 5
```

Це також досить ефективно з огляду на пріоритет операцій.

6.4. Пріоритет операторів

Якщо вираз має більше одного оператора, порядок виконання визначається їхнім пріоритетом, або, іншими словами, типовим порядком першості операторів.

Таблиця 6.3

Пріоритет операторів

Пріоритет	Тип оператора	Асоціативність	Конкретні оператори
19	Группування	Не визначено	(…)
18	Доступ до властивостей	Зліва направо	… . …
	Доступ до властивостей з можливістю обрахування	Зліва направо	… […]
	new зі списком аргументів	Не визначено	new … (…)
	Виклик функції	Зліва направо	… (…)
	Оператор опціональної послідовності ?.	Зліва направо	?.
17	new без списка аргументів	Не визначено	new …
16	Постфіксний інкремент	Не визначено	… ++
	Постфіксний декремент	Не визначено	… --
15	Логічне заперечення (!)	Зправа наліво	! …

Пріоритет	Тип оператора	Асоціативність	Конкретні оператори
	Побітове заперечення (~)	Зправа наліво	~ …
	Унарний плюс	Зправа наліво	+ …
	Унарний мінус	Зправа наліво	- …
	Префіксний інкремент	Зправа наліво	++ …
	Префіксний декремент	Зправа наліво	- …
	typeof	Зправа наліво	typeof …
	void	Зправа наліво	void …
	delete	Зправа наліво	delete …
	await	Зправа наліво	await …
14	Зведення в ступінь (**)	Зправа наліво	… ** …
13	Множення *	Зліва направо	… * …
	Ділення /	Зліва направо	… / …
	Залишок від ділення (%)	Зліва направо	… % …
12	Додавання (+)	Зліва направо	… + …
	Віднімання (-)	Зліва направо	… - …
11	Побітовий зсув вліво (<<)	Зліва направо	… << …
	Побітове зрушення вправо (>>)	Зліва направо	… >> …
	Зсув вправо із заповненням нулів (>>>)	Зліва направо	… >>> …
10	Менше <	Зліва направо	… < …
	Менше або дорівнює <=	Зліва направо	… <= …
	Більше >	Зліва направо	… > …
	Більше або дорівнює >=	Зліва направо	… >= …
	in	Зліва направо	… in …
	instanceof	Зліва направо	… instanceof …
9	Дорівнює ==	Зліва направо	… == …
	Не дорівнює !=	Зліва направо	… != …
	Сувора рівність ===	Зліва направо	… === …
	Сувора нерівність !==	Зліва направо	… !== …
8	Побітове «і» &	Зліва направо	… & …
7	Побітове виключаєче «АБО» ^	Зліва направо	… ^ …
6	Побітове «АБО» |	Зліва направо	… | …
5	Логічне «і» &&	Зліва направо	… && …
4	Логічне «АБО» ||	Зліва направо	… || …
	Оператор нульового злиття ??	Зліва направо	… ?? …
3	Умовний (тернарний) оператор	Зправа наліво	… ? … : …
2	Присвоєння	Зправа наліво	… = … … += … … -= … … **= … … *= … … /= … … %= … … <<= … … >>= … … >>>= … … &= … … ^= … … |= … … &&= …

Пріоритет	Тип оператора	Асоціативність	Конкретні оператори
			… ‖= … … ?? = …
	yield	Зправа наліво	yield …
	yield*	Зправа наліво	yield* …
1	Кома / послідовність	Зліва направо	… , …

У JavaScript є багато операторів. Кожен оператор має відповідний номер пріоритету. Першим виконується той оператор, який має найбільший номер пріоритету. Якщо пріоритет є однаковим, порядок виконання — зліва направо.

Зазначимо, що присвоєння = також є оператором. Воно є у таблиці з пріоритетами й має дуже низький пріоритет 2.

Тому, коли ми присвоюємо значення змінній, наприклад, x = 2 * 2 + 1, спочатку виконуються обчислення, а потім виконується присвоєння = зі збереженням результату в x.

Усі оператори в JavaScript повертають значення. Це очевидно для + та -, але це також правдиво для =. Виклик x = значення записує значення у x, а потім повертає його.

Приклад:

```
let a = 1;
let b = 2;
let c = 3 - (a = b + 1);
alert( a ); // 3
alert( c ); // 0
```

Іншою цікавою особливістю є здатність ланцюгового присвоєння:

```
let a, b, c;
a = b = c = 2 + 2;
alert( a ); // 4
alert( b ); // 4
alert( c ); // 4
```

Ланцюгове присвоєння виконується справа наліво. Спочатку обчислюється найправіший вираз 2 + 2, а потім результат присвоюється змінним ліворуч: c, b та a. Зрештою всі змінні мають спільне значення.

Часто нам потрібно застосувати оператор до змінної й зберегти новий результат у ту ж саму змінну. Наприклад:

```
let n = 2;
n = n + 5;
n = n * 2;
```

Цей запис можна скоротити за допомогою операторів += та *=:

let n = 2;

n += 5; // тепер n = 7 (те ж саме, що n = n + 5)

n *= 2; // тепер n = 14 (те ж саме, що n = n * 2)

alert(n); // 14

Короткі оператори “модифікувати та присвоїти” є для всіх арифметичних та побітових операторів: /=, -= тощо.

Збільшення або зменшення на одиницю є однією з найпоширеніших числових операцій. Тому для цього є спеціальні оператори:

Інкремент ++ збільшує змінну на 1:

let counter = 2;

counter++;   // працює так само, як counter = counter + 1, але запис коротше

alert(counter); // 3

Декремент -- зменшує змінну на 1:

let counter = 2;

counter--;   // працює так само, як counter = counter - 1, але запис коротше

alert(counter); // 1

Оператори ++ та -- можуть розташовуватися до або після змінної.

- Коли оператор йде за змінною, він у “постфіксній формі”: counter++.+;
- “Префіксна форма” – це коли оператор йде попереду змінної: ++counter.

Щоби побачити різницю, наведемо приклад:

let counter = 1;

let a = ++counter; // (*)

alert(a); // 2

let counter = 1;

let a = counter++; // (*) змінили ++counter на counter++

alert(a); // 1

6.5. Оператори порівняння

Багато з операторів порівняння нам відомі з математики. В JavaScript вони записуються ось так:

Більше/менше: a > b, a < b.

Більше/менше або дорівнює: a >= b, a <= b.

Дорівнює: a == b. Зверніть увагу, для порівняння потрібно використовувати два знаки

рівності ==. Один знак рівності a = b означав б присвоєння.

Не дорівнює: в JavaScript записується як a != b.

Результат порівняння має логічний тип

Всі оператори порівняння повертають значення логічного типу:

true – означає “так”, “правильно” або “істина”.

false – означає “ні”, “неправильно” або “хибність”.

Наприклад:

alert(5 > 3); // true (правильно)

alert(7 == 14); // false (неправильно)

alert(8 != 2); // true (правильно)

Результат порівняння можна присвоїти змінній, як і будь-яке інше значення:

let compare = 7 > 1; // присвоїти результат порівняння змінній result

alert(compare); // true

Оператори порівняння наведено в таблиці 6.4.

Таблиця 6.4

Оператори порівняння

Оператор	Назва	Приклад	Результат
==	дорівнює	“10”==10	true
!=	не дорівнює	10!=4	true
>	більше	10>4	true
<	менше	10<4	false
>=	більше або дорівнює	10>=4	true
<=	менше або дорівнює	10<=4	false
===	ідентично або строга рівність	“10”===10	false
!==	не ідентично	“10”!==10	true

Логічні оператори наведені в табл. 6.5.

Таблиця 6.5

Логічні оператори

Оператор	Загальний синтаксис	Результат
&&	Умова1 && Умова2	Якщо обидві умови дорівнюють true, то і результат true, інакше false
||	Умова1 || Умова2	Якщо хоча б одна з умов дорівнює true, то і результат true, інакше false
!	!Умова	Якщо умова true, то результат false та навпаки

Щоб визначити, чи один рядок більший за інший, JavaScript використовує так званий “алфавітний” або “лексикографічний” порядок.

Інакше кажучи, рядки порівнюються посимвольно.

Наприклад:

alert('Я' > 'А'); // true

alert('Соки' > 'Сода'); // true

alert('Комар' > 'Кома'); // true

Коли порівнюються значення різних типів, JavaScript конвертує ці значення в числа.

Наприклад:

alert('2' > 1); // true, рядок '2' стає числом 2

alert('01' == 1); // true, рядок '01' стає числом 1

Логічне значення true стає 1, а false — 0.

Наприклад:

alert(true == 1); // true

alert(false == 0); // true

Оператор строгої рівності === перевіряє рівність без перетворення типів. Якщо a і b мають різні типи, то перевірка a === b негайно поверне результат false без спроби їхнього перетворення. Наприклад:

alert(0 === false); // false, тому що порівнюються різні типи

Оператор “АБО” представлений двома символами вертикальної лінії:

result = a || b;

Якщо операнд не є булевим, він перетворюється на булевий для обчислення.

Дано кілька значень, розділених оператором АБО:

result = value1 || value2 || value3;

Оператор АБО || робить наступне:

1. Обчислює операнди зліва направо.
2. Перетворює значення кожного операнда на булеве. Якщо результат true, зупиняється і повертає початкове значення цього операнда.
3. Якщо всі операнди були обчисленні (тобто усі були false), повертає останній операнд.
4. Значення повертається у первісному вигляді без конвертації.

Іншими словами, ланцюжок з АБО || повертає перше правдиве значення або останнє, якщо правдивого значення не знайдено.

Наприклад:

alert(1 || 0); // 1 (1 є правдивим)

alert(null || 1); // 1 (1 є першим правдивим значенням)

alert(null || 0 || 1); // 1 (перше правдиве значення)

alert(undefined || null || 0); // 0 (усі хибні, повертається останнє значення)

Це призводить до цікавого використання, у порівнянні з “чистим, класичним, виключно-

булевим АБО”:

1. Отримання першого істинного значення зі списку змінних або виразів.

Наприклад, маємо змінні firstName, lastName та nickName, усі необов’язкові (тобто вони можуть бути невизначеними або мати хибні значення).

Використаємо АБО ||, щоб вибрати ту змінну, яка має дані, і виведемо її (або рядок "Анонім", якщо жодна змінна не має даних):

```
let firstName = "";
let lastName = "";
let nickName = "СуперКодер";
alert( firstName || lastName || nickName || "Анонім"); // СуперКодер
```

Якщо всі змінні мали б порожні рядки, тоді показалося слово "Анонім".

2. Обчислення короткого замикання.

Іншою особливістю оператора АБО || є так зване “обчислення короткого замикання”.

Це означає, що оператор || опрацьовує аргументи доти, доки не досягається перше правдиве значення, після чого це значення негайно повертається, без подальшого опрацювання решти аргументів.

Важливість такої особливості стає очевидною, якщо операнд є не просто змінною, а виразом із побічним ефектом, як-от присвоєння змінної або виклик функції.

У наведеному нижче прикладі виведеться лише друге повідомлення:

```
true || alert("не виведеться");
false || alert("виведеться");
```

В першому рядку оператор АБО || зупиняє виконання відразу після того, як “побачить” що лівий вираз є true, тож alert не виконається.

Деколи таку конструкцію використовують, щоб виконувати команди лише при хибності умови ліворуч від оператора.

Оператор І представлений двома амперсандами &&:

```
result = a && b;
```

Дано декілька значень, об’єднаних кількома І:

```
result = value1 && value2 && value3;
```

Оператор І && робить наступне:

1. Обчислює операнди зліва направо.

2. Перетворює кожен операнд на булевий. Якщо результат false, зупиняється і повертає оригінальне значення того операнда.

3. Якщо всі операнди були обчисленні (тобто усі були правдиві), повертає останній операнд.

Іншими словами, І повертає перше хибне значення, або останнє значення, якщо жодного хибного не було знайдено.

Правила, наведені вище, подібні до правил АБО. Різниця полягає в тому, що І повертає перше хибне значення, тоді як АБО повертає перше правдиве.

Приклади:

```
// якщо перший операнд правдивий,
// І повертає другий операнд:
alert( 1 && 0 ); // 0
alert( 1 && 5 ); // 5
// якщо перший операнд хибний,
// І повертає саме його. Другий операнд ігнорується
alert( null && 5 ); // null
alert( 0 && "неважливо" ); // 0
```

Ми також можемо передавати декілька значень поспіль. Подивіться, як повертається перше хибне:

```
alert( 1 && 2 && null && 3 ); // null
```

Коли всі значення є правдивими, повертається останнє значення:

```
alert( 1 && 2 && 3 ); // 3, останнє
```

Булевий оператор НЕ представлений знаком оклику !. Синтаксис:

```
result = !value;
```

Оператор приймає один аргумент і виконує наступне:

Перетворює операнд на булевий тип: true/false.

Повертає зворотне значення.

Наприклад:

```
alert( !true ); // false
alert( !0 ); // true
```

Подвійний НЕ !! іноді використовується для перетворення значення на булевий тип:

```
alert( !!"не пустий рядок" ); // true
alert( !!null ); // false
```

Тобто, перший НЕ перетворює значення на булеве і повертає зворотне, а другий НЕ інвертує його знову. Зрештою ми маємо просте перетворення значень на булевий тип.

Є трохи довший спосіб зробити те ж саме – вбудована функція Boolean:

```
alert( Boolean("не пустий рядок") ); // true
alert( Boolean(null) ); // false
```

Пріоритет НЕ ! є найвищим серед усіх логічних операторів, тому він завжди виконується

першим, перед && або ||.

Результатом a ?? b буде:

a, якщо a визначене,

b, якщо a не визначене.

Інакше кажучи, ?? повертає перший аргумент, якщо він не null/undefined. Інакше, другий.

Оператор об'єднання з null не є абсолютно новим. Це просто хороший синтаксис, щоб отримати перше "визначене" значення з двох.

Ми можемо переписати вираз result = a ?? b, використовуючи оператори, які ми вже знаємо:

result = (a !== null && a !== undefined) ? a : b;

Ось приклад з user, якому присвоєне ім'я:

let user = "Іван";

alert(user ?? "Анонімний"); // Іван (user визначений)

Ми також можемо використовувати послідовність з ??, щоб вибрати перше значення зі списку, яке не є null/undefined.

Скажімо, у нас є дані користувача в змінних firstName, lastName або nickName. Всі вони можуть бути не визначені, якщо користувач вирішив не вводити значення.

Ми хотіли б показати ім'я користувача, використовуючи одну з цих змінних, або показати "Анонімний", якщо всі вони null/undefined.

Використаймо оператор ?? для цього:

let firstName = null;

let lastName = null;

let nickName = "Суперкодер";

// показує перше визначене значення:

alert(firstName ?? lastName ?? nickName ?? "Анонімний"); // Суперкодер

Важлива різниця між || та ?? ними полягає в тому, що:

|| повертає перше істинне значення.

?? повертає перше визначене значення.

Інакше кажучи, оператор || не розрізняє, чи значення false, 0, порожній рядок "" чи null/undefined. Всі вони однакові – хибні значення. Якщо будь-яке з них є першим аргументом ||, тоді ми отримаємо другий аргумент як результат. Наприклад:

let height = 0;

alert(height || 100); // 100

alert(height ?? 100); // 0

З міркувань безпеки, JavaScript забороняє використання ?? разом з операторами && та ||,

якщо пріоритет явно не вказаний дужками.

Код нижче викликає синтаксичну помилку:

```
let x = 1 && 2 ?? 3; // Синтаксична помилка
```

6.6. Перевірка умов

Для перевірки умови в мовах сценарію існує конструкція if. Синтаксис:

if (*умова1*) {

дії у випадку виконання умови 1

}

else if (*умова2*){

дії у випадку виконання умови 2

}

else{

дії у випадку невиконання жодної з умов

}

Наприклад, необхідно визначити суму процентів банку за n місяців при вкладенні s гривен, якщо на суму, меншу за 500 нараховується 1%, на суму від 500 до 1000 - 2%, а на суму більше 1000 - 5%:

```
let s=parseFloat(prompt("Сума: ",100));
let d=parseInt(prompt("Термін: ",1));
if (isNaN(s) || isNaN(d) || s<=0 || d<=0){
   alert("Некоректно введені дані");
}else{
   let st=0.05;
   if (s<500){
      st=0.01;
   }
   else if (s<1000)
   {
      st=0.02;
   }
   alert(`Початкова сума: ${s} термін: ${d}\nВідсотки: ${s*d*st}`);
}
```

Інструкція if (…) обчислює вираз у дужках і перетворює результат у логічний тип.

Так званий “умовний” оператор або оператор “знак питання” дає нам зробити це в більш короткій і простій формі. Оператор представлений знаком питання ?. Іноді його називають “тернарним”, оскільки оператор має три операнди. Синтаксис:

let result = умова ? значення1 : значення2;

Спочатку обчислюється умова: якщо вона є правдивою, тоді повертається значення1, інакше – значення2.

Наприклад:

let accessAllowed = (age > 18) ? true : false;

Послідовність операторів знака питання ? може повернути значення, яке залежить від більш ніж однієї умови.

Наприклад:

```
let age = prompt('Вік?', 18);
let message = (age < 3) ? 'Привіт, крихітко!' :
  (age < 18) ? 'Вітаю!' :
  (age < 100) ? 'Моє шанування!' :
  'Який незвичайний вік!';
alert( message );
```

Для перевірки умов існують також конструкція switch.

switch (*вираз*){

case *знач1* :

дії у випадку, якщо вираз прийняв знач1

break

case *знач2* :

дії у випадку, якщо вираз прийняв знач2

break

default:

дії, якщо вираз не прийняв ні одного із зазначених значень

}

Наступний приклад ілюструє виведення кількості днів згідно з введеним номером місяця та роком.

```
let m=parseInt(prompt("Введіть номер місяця: ",1));
let y=parseInt(prompt("Введіть рік: ",2022));
if (isNaN(m) || isNaN(y) || m<1 || m>12 || y<=0){
    alert("Некоректно введені дані");
}else{
```

```
    let day=31;
    switch (m){
        case 4:
        case 6:
        case 9:
        case 11:
            day=30;
            break;
        case 2:
            day=(y%4==0)?29:28;
    }
    alert(day);
}
```

Необхідно наголосити, що перевірка відповідності є завжди строгою. Значення повинні бути однакового типу аби вони збігалися. Наприклад:

```
let arg = prompt("Введіть значення?");

switch (arg) {
  case '0':
  case '1':
    alert( 'Один або нуль' );
    break;

  case '2':
    alert( 'Два' );
    break;

  case 3:
    alert( 'Ніколи не буде виконано!' );
    break;
  default:
    alert( 'Невідоме значення' );
}
```

6.7. Цикли

Для організації циклів у мові JavaScript існує конструкція for:

for (*ініціалізація до циклу*; *умова*; *вираз, що виконується у кінці кожної ітерації*)

{

[*оператори*]

[*break*]

[*оператори*]

[*continue*]

[*оператори*]

}

JavaScript виконує оператори циклу for доки виконується умова або не зустрінеться оператор виходу break, а continue означає перейти на перевірку умови циклу.

Приклад:

```
let s=parseFloat(prompt("Введіть суму: ",100));
let t=parseFloat(prompt("Введіть термін: ",1));
let p=parseFloat(prompt("Введіть процент: ",1));

if (isNaN(s) || isNaN(t) || isNaN(p)){
   alert("Некоректно введені дані");
}else{
   let sp=0;
   for (let i=1;i<=t;i++)
      sp+=(s+sp)*p/100;
   alert(s+sp);
}
```

Перевагою конструкції while є те, що вона може використовуватись якщо кількість кроків розрахунку наперед невідома. Оператори циклу виконуються доки умова дорівнює true:

while (*умова*)

{

[*оператори*]

[*break*]

[*оператори*]

[*continue*]

[*оператори*]

```
}
```

Приклад:

```
let counter=0, total=0, value;
while ((value=prompt("Введіть значення: ",0))!=null)
{
   if (isNaN(value)){
      alert("Некоректне значення.\nВведіть ще раз.");
      continue;
   }
   total+=parseFloat(value);
   counter++;
}
if (counter) alert("Середнє: "+total/counter);
```

Існує також конструкція циклу, в якій перша ітерація обов'язково виконується, оскільки умова перевіряється не спочатку ітерації циклу, а в кінці:

do{
  [*оператори*]
  [*break*]
  [*оператори*]
  [*continue*]
  [*оператори*]
} **while** (*умова*)

Приклад:

```
Math.random();
let number=Math.ceil(Math.random()*10);
let text="Відгадайте число від 1 до 10:";
let count=0, num;
do{
   num=prompt(text+" "+number,0);
   if (num==null) break;
   if (isNaN(num)){
      text="Некоректне число.\nПовторіть введення";
      continue;
   }
   num=parseInt(num);
```

```
        if (num==number)  {
            alert(count+1);
            break;
        }
        if (num<number)
            text="Замаленьке. Спробуйте ще:";
        else
            text="Завелике. Спробуйте ще:";
        count++;
    }while (count<5);
```

Зазвичай, цикл завершується, коли умова стає false. Але ми можемо в будь-який момент вийти з циклу, використавши спеціальну директиву break.

Директива continue не зупиняє весь цикл. Натомість, вона зупиняє поточну ітерацію і починає виконання циклу спочатку з наступної ітерації (якщо умова циклу досі вірна).

Деколи нам потрібно вийти з кількох вкладених циклів. Для цього корисно застосовувати мітки:

```
labelName: for (...) {
  ...
}
```

Вираз break <labelName> в циклі нижче шукає найближчий цикл з заданою міткою і переходить в його кінець:

```
outer: for (let i = 0; i < 3; i++) {
  for (let j = 0; j < 3; j++) {
    let input = prompt(`Значення в координатах (${i},${j})`, '');
    // якщо порожній рядок або Скасувати, тоді вихід з обох циклів
    if (!input) break outer; // (*)
    // зробити щось із значенням...
  }
}
alert('Готово!');
```

6.8. Створення функцій

Якщо в програмі декілька разів повторюється певний код, тобто у різних місцях сценарію повторюються одні й ті ж дії, то доцільним є винесення цих операторів у певну функцію:

function *ім'я функції*([*аргумент1* **[, ...** *аргументN*]])

{

тіло

[return значення;]

}

Наприклад:

```
alert(podatok(3000))
function podatok(sum){
   if (sum < 1342) return 0;
   if (sum <= 3760) sum-=1342;
   return sum*0.13;
}
```

Змінна, яка оголошена в функції доступна лише в тілі цієї функції.

Наприклад:

```
function show() {
 let m = "Вітаю!"; // локальна змінна
 alert( m );
}
show();
alert(m); // <-- Помилка! Змінна недоступна поза функцією
```

Функція має повний доступ до зовнішньої змінної. Вона теж може її змінювати.

Наприклад:

```
let user = 'Іван';
function show() {
 user = "Богдан"; // (1) змінено зовнішню змінну
 let m = Вітаю, ' + user;
 alert(m);
}
alert( user ); // Іван перед викликом функції showMessage
show();
alert( user ); // Богдан, значення було змінено після виклику функції show
```

Зовнішня змінна використовується тоді, коли немає локальної.

Якщо всередині функції є змінна з таким самим ім'ям, то вона перекриває зовнішню.

```
let user = 'Іван';
function show() {
```

```
  let user = "Богдан"; // (1) змінено зовнішню змінну
  let m = Вітаю, ' + user;
  alert(m);
}
show();
alert( user ); // Іван без змін
```

Ми можемо передати в функцію довільні дані використовуючи параметри.

В наступному прикладі, функція має два параметри: from і text.

```
function show(from, text) { // параметри: from, text
  alert(from + ': ' + text);
}
show('Оля', 'Привіт!'); // Оля: Привіт!
show ('Оля', "Як справи?"); // Оля: Як справи?
```

Під час виклику функції з цими параметрами, відбувається копіювання значень параметрів в локальні змінні from та text.

Якщо викликати функцію без аргументів, тоді відповідні значення стануть undefined. Ми можемо задати значення параметра за умовчанням, яке використовуватиметься, якщо не задати аргумент:

```
function show(from, text = ‘Вітаю’) { // параметри: from, text
  alert(from + ': ' + text);
}
show(‘Оля’);
```

Це може бути складніший вираз, який обчислюється і присвоюється лише якщо параметр відсутній. Отож, такий варіант теж можливий:

```
function show(from, text = anotherFunction()) {
  // anotherFunction() виконується лише якщо `text` не задано
  // результат виконання цієї функції присвоїться змінній `text`
}
```

В JavaScript, типовий параметр обчислюється кожного разу, коли викликається функція без відповідного параметру. В прикладі вище, функція anotherFunction() не викличеться, якщо буде задано параметр text. З іншого боку, вона буде викликатися кожного разу, коли text відсутній.

Під час виконання функції, ми можемо перевірити, чи параметр надано, порівнюючи його з undefined:

```
function show(text) {
```

```
 // ...
 if (text === undefined) { // якщо параметр відсутній
   text = 'порожнє повідомлення';
 }
 alert(text);
}
show(); // порожнє повідомлення
```

або

```
function show(text) {
 // ...
   text = text || 'порожнє повідомлення';
 alert(text);
}
```

У параметрах за замовчуванням можна використовувати значення попередніх (розташованих ліворуч у списку) параметрів:

```
function greet(name, greeting, message = greeting + ' ' + name) {
   return [name, greeting, message];
}

greet('David', 'Hi');  // ["David", "Hi", "Hi David"]
greet('David', 'Hi', 'Happy Birthday!');  // ["David", "Hi", "Happy Birthday!"]
```

В якості результату, функція може повертати назад значення в код, який викликав цю функцію:

```
function add(a, b) {
 return a + b;
}
let sum = add(5, 7);
alert( sum ); // 12
```

Існує й інший синтаксис для створення функції, що називають Функціональним Виразом (Function Expression). Він дозволяє створювати функцію всередині будь-якого виразу. Наприклад:

```
let show = function() {
 alert( "Вітаю" );
};
```

Немає значення, яким чином створено функцію, функція – це завжди значення. В обох

способах вище, функція зберігається в змінній show.

Можна скопіювати функцію в іншу змінну:

```
function show() {   // створюємо
  alert( "Вітаю" );
}
let func = show;    //  копіюємо
func(); // Вітаю     // викликаємо копію
show(); // Вітаю    //теж працює
```

Колбеки (функції зворотного виклику)

Функції можна передавати як аргумет до іншої функції:

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}
function showOk() {
  alert( "Ви погодились." );
}
function showCancel() {
  alert( "Ви скасували виконання." );
}
// використання: функції showOk, showCancel передаються як аргументи для ask
ask("Ви згодні?", showOk, showCancel);
```

Аргументи showOk та showCancel функції ask називаються функціями зворотного виклику або просто колбеками.

Можна використати Функціональний Вираз, щоб записати ту саму функцію коротше:

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}
ask(
  "Ви згодні?",
  function() { alert("Ви погодились."); },
  function() { alert("Ви скасували виконання."); }
);
```

Функціональний Вираз буде створено тільки тоді, коли до нього дійде виконання і тільки

після цього він може бути використаний. Щойно потік виконання досягне правої частини у присвоєнні let sum = function…, функцію буде створено і з цього моменту її можна буде використати (присвоїти змінній, викликати тощо). У випадку з Оголошенням Функції все інакше. Синтаксис Оголошення Функції дозволяє викликати функцію раніше, ніж вона були визначена в коді:

```
sayHi("Іван"); // Привіт, Іван
function sayHi(name) {
  alert( `Привіт, ${name}` );
}
```

З функціональним виразом виникне помилка:

```
sayHi("Іван"); // помилка!
let sayHi = function(name) {
  alert( `Привіт, ${name}` );
};
```

Ще однією особливістю Оголошення Функції є її блокова область видимості.

У суворому режимі, якщо Оголошення Функції знаходиться в блоці {...}, то функція доступна усюди всередині блоку. Але не зовні.

Уявімо, що нам потрібно визначити функцію welcome() залежно від змінної age, яку ми отримаємо під час виконання коду. Далі в скрипті нам буде потрібно викликати цю функцію.

Якщо ми використаємо Оголошення Функції, то це не буде працювати:

```
let age = prompt("Скільки вам років?", 18);
// оголошуємо функцію відповідно до умови
if (age < 18) {
  function welcome() {
    alert("Привіт!");
  }

} else {
  function welcome() {
    alert("Вітання!");
  }
}
// ...спробуємо викликати функцію
```

```
welcome(); // помилка в суворому режимі (ReferenceError: welcome is not defined)
```

Цей код працює як потрібно:

```
let age = prompt("Скільки вам років?", 18);
let welcome;
if (age < 18) {
  welcome = function() {
    alert("Привіт!");
  };
} else {
  welcome = function() {
    alert("Вітання!");
  };

}
welcome(); // тепер все гаразд
```

Існує ще один простий та короткий синтаксис для створення функцій, який часто доцільніше використовувати замість Функціонального Виразу.

Це так звані “стрілкові функції”, а виглядають вони ось так:

```
let func = (arg1, arg2, ..., argN) => expression;
```

Наприклад:

```
let add = (a, b) => a + b;
/* Ця стрілкова функція — це коротша форма для:
let add = function(a, b) {
  return a + b;
};
*/
alert( add(7, 5) ); // 12
```

Є ще один спосіб створити функцію. Він рідко використовується:

var *ім'я функції* **= new Function(** **[***аргумент1*, **[...** *аргументN***,]]** *тіло* **);**

тіло являє собою строку, що містить код функції. Наприклад:

```
var add = new Function("x", "y", "return(x+y)");
alert(add(7,5));//12
```

Функцію можна викликати з будь-якою кількістю аргументів, незалежно від того, як вона визначена:

```
function add(a, b) {
  return a + b;
}
alert(add(1, 2, 3, 4, 5));
```

Решту параметрів можна включити до визначення функції за допомогою трьох крапок ... що передують імені масиву, який їх міститиме. Точки буквально означають "зібрати решту параметрів у масив".

Наприклад, щоб зібрати всі аргументи в масив args:

```
function sumAll(...args) { // args – це ім’я масиву
  let sum = 0;

  for (let arg of args) sum += arg;
  return sum;
}
alert( sumAll(1) ); // 1
alert( sumAll(1, 2) ); // 3
alert( sumAll(1, 2, 3) ); // 6
```

Ми можемо вибрати перші параметри як змінні, а зібрати у масив лише залишки.

У цьому прикладі перші два аргументи переходять у змінні, а решта – в масив titles:

```
function showName(firstName, lastName, ...titles) {
  alert( firstName + ' ' + lastName ); // Юлій Цезар
  // решта параметрів переходять до масиву
  // titles = ["Консул", "Полководець"]
  alert( titles[0] ); // Консул
  alert( titles[1] ); // Полководець
  alert( titles.length ); // 2
}
showName("Юлій", "Цезар", "Консул", "Полководець");
```

Існує також спеціальний об’єкт, подібний масиву arguments який містить усі аргументи за їх індексом.

Наприклад:

```
function showName() {
  alert(arguments.length);
  alert(arguments[0]);
```

```
 alert(arguments[1]);
 // це повторюване
 // for(let arg of arguments) alert(arg);
}
// показує: 2, Julius, Caesar
showName("Julius", "Caesar");
// показує: 1, Ilya, undefined (жодного другого аргументу)
showName("Ilya");
```

В страрих версіях залишкові параметри не існували в мові, єдиний спосіб отримати всі аргументи функції був за допомогою arguments.

Стрілочні функції не мають "arguments".

Синтаксис розширення

Ми тільки що побачили, як отримати масив зі списку параметрів.

Але іноді нам потрібно зробити зворотнє. Наприклад, є вбудована функція Math.max що повертає найбільше число зі списку:

```
alert( Math.max(3, 5, 1) ); // 5
```

Передати “як є” не вийде, бо Math.max очікує список числових аргументів, а не єдиний масив:

```
let arr = [3, 5, 1];
alert( Math.max(arr) ); // NaN
```

Для Math.max:

```
let arr = [3, 5, 1];
alert( Math.max(...arr) ); // 5 (перетворює масив у список аргументів)
```

Таким чином, ми також можемо передати кілька ітерацій:

```
let arr1 = [1, -2, 3, 4];
let arr2 = [8, 3, -8, 1];
alert( Math.max(...arr1, ...arr2) ); // 8
```

Ми навіть можемо поєднати синтаксис розширення з нормальними значеннями:

```
let arr1 = [1, -2, 3, 4];
let arr2 = [8, 3, -8, 1];
alert( Math.max(1, ...arr1, 2, ...arr2, 25) ); // 25
```

6.9. Область видимості змінної, замикання

Якщо змінна оголошена всередині блоку коду {...}, вона буде доступна лише всередині

цього блоку.

Наприклад:

```
{
  // тут виконується певна робота з локальними змінними, яку не слід бачити зовні
  let mes = "Вітаю"; // змінна видима тільки у цьому блоці
  alert(mes); // Вітаю
}
alert(mes); // Помилка: змінну message не було оголошено
```

Ми можемо використовувати це, щоб виділити фрагмент коду, який працює зі змінними, які доступні лише з нього:

```
{
  // показати повідомлення
  let mes = "Вітаю";
  alert(mes);
}
{
  // показати інше повідомлення
  let mes = "На все добре";
  alert(mes);
}
```

Для if, for, while і так далі, змінні, оголошені в {...} також видно тільки всередині:

```
if (true) {
  let s = "Вітаю!";

  alert(s); // Вітаю!
}
alert(s); // Помилка, такої змінної не існує
```

Тут, після завершення if, alert нижче не побачить “s”, отже, помилка.

Це гарна можливість, яка дозволяє нам створювати локально-блокові змінні, специфічні для гілки if.

Те ж саме справедливо і для циклів for та while:

```
for (let i = 0; i < 3; i++) {
  // змінну `i` видно тільки всередині цього циклу for
  alert(i); // 0, потім 1, потім 2
}
```

```
alert(i); // Помилка, такої змінної немає
```

Візуально, let i знаходиться за межами {...}. Але конструкція for особлива: змінна, оголошена всередині неї, вважається частиною блоку.

Функція називається “вкладеною”, коли вона створюється всередині іншої функції:

```
function show(firstName, lastName) {
  // допоміжна вкладена функція для використання нижче
  function fullName() {
    return firstName + " " + lastName;
  }
  alert( "Вітаю, " + fullName() );
  alert( "Бувай, " + fullName() );
}
```

Тут вкладена функція fullName() створена для зручності. Вона має доступ до внутрішніх змінних функції і тому може повернути повне ім’я.

Що ще цікавіше, вкладену функцію можна повернути: як властивість нового об’єкта, або як самостійний результат. Потім її можна використати десь в іншому місці. Незалежно від того, де її викликають, вона завжди буде мати доступ до внутрішніх змінних функцію, в якій її було створено.

Нижче, makeCounter створює функцію “counter”, яка повертає наступний номер при кожному виклику:

```
function makeCounter() {
  let count = 0;
  return function() { return count++;  };
}
let counter = makeCounter();
alert( counter() ); // 0
alert( counter() ); // 1
alert( counter() ); // 2
```

У JavaScript кожна запущена функція, блок коду {...}, і скрипт в цілому мають внутрішній (прихований) асоційований об’єкт, відомий як Лексичне середовище (Lexical Environment).

Об’єкт лексичного середовища складається з двох частин:

1. Запис середовища (Environment Record) – об’єкт, який зберігає всі локальні змінні як властивості (та деяку іншу інформацію, наприклад значення this).
2. Посилання на зовнішнє лексичне середовище, яке пов’язане із зовнішнім кодом.

"Змінна" це лише властивість спеціального внутрішнього об'єкта, Запис середовища (Environment Record). "Отримати або змінити змінну" насправді означає "отримати або змінити властивість цього об'єкта".

Є так зване глобальне лексичне середовище, пов'язане з усім скриптом.

Змінна – це властивість спеціального внутрішнього об'єкта, пов'язана з блоком/функціеї/скриптом що зараз виконується.

Робота зі змінними – це насправді робота з властивостями цього об'єкта.

Функція також є значенням, як і значення у змінних.

Різниця в тому, що функція створена за допомогою Function Declaration, ініціалізується миттєво і повністю.

Коли створюється лексичне середовище, така функція відразу стає готовою до використання (на відміну від значення у змінній let, що непридатна для викорстиння до оголошення).

Ось чому ми можемо використовувати функцію, оголошену з Function Declaration, ще до рядка з оголошенням.

Коли функція виконується, на початку виклику автоматично створюється нове лексичне середовище для зберігання локальних змінних та параметрів виклику.

Під час виклику функції у нас є два лексичні середовища: внутрішнє (для виклику функції) і зовнішнє (глобальне). Внутрішнє лексичне середовище відповідає поточному виконанню функції. Зовнішнє лексичне середовище – це глобальне лексичне середовище. Внутрішнє лексичне середовище має посилання на зовнішнє.

Коли код хоче отримати доступ до змінної – спочатку шукає її у внутрішньому лексичному середовищі, потім у зовнішньому, потім у зовнішньому до попереднього і так далі поки не дійде до глобального.

У програмуванні існує загальний термін "замикання", який розробники зазвичай мають знати. Замикання – це функція, яка запам'ятовує свої зовнішні змінні та може отримати до них доступ. Тобто: функції автоматично запам'ятовують, де вони були створені, використовуючи приховану властивість [[Environment]], а потім їхній код може отримати доступ до зовнішніх змінних.

Функції JavaScript формують так звані замикання. Замикання - це комбінація функції та лексичного оточення, в якому ця функція була оголошена. Це оточення складається з довільної кількості локальних змінних, які були в області дії функції під час створення замикання.

Наприклад:

```
function makeAdder(x) {
  return function(y) {
```

```
  return x + y;
 };
};
var add5 = makeAdder(5);
var add10 = makeAdder(10);
console.log(add5(2));  // 7
console.log(add10(2)); // 12
```

Тут ми визначили функцію makeAdder(x), яка отримує єдиний аргумент x та повертає нову функцію. Ця функція отримує єдиний аргумент y та повертає суму x та y.

По суті, makeAdder — це фабрика функцій: вона створює функції, які можуть додавати певного значення до свого аргументу. У прикладі вище ми використовуємо нашу фабричну функцію для створення двох нових функцій - одна додає 5 до свого аргументу, друга додає 10. add5 і add10 – це приклади замикань. Ці функції ділять одне визначення тіла функції, але вони зберігають різні оточення. В оточенні функції add5 x – це 5, тоді як в оточенні add10 x – це 10.

Замикання корисні тим, що дозволяють пов'язати дані (лексичне оточення) з функцією, яка працює з цими даними. Очевидною є паралель з об'єктно-орієнтованим програмуванням, де об'єкти дозволяють нам пов'язати деякі дані (властивості об'єкта) з одним або декількома методами.

Код нижче ілюструє, як можна використовувати замикання для визначення публічних функцій, які мають доступ до закритих від користувача (private) функцій та змінних:

```
var Counter = (function() {
 var privateCounter = 0;
 function changeBy(val) {
  privateCounter += val;
 }
 return {
  increment: function() {
   changeBy(1);
  },
  decrement: function() {
   changeBy(-1);
  },
  value: function() {
   return privateCounter;
  }
```

```
  };
})();
alert(Counter.value()); /* Alerts 0 */
Counter.increment();
Counter.increment();
alert(Counter.value()); /* Alerts 2 */
Counter.decrement();
alert(Counter.value()); /* Alerts 1 */
```

Це оточення містить два приватні елементи: змінну privateCounter і функцію changeBy(val). Жоден з цих елементів не доступний безпосередньо, за межами цієї анонімної функції. Натомість вони можуть і повинні використовуватися трьома публічними функціями, які повертаються анонімним блоком коду (anonymous wrapper), що виконується в тій самій анонімній функції.

Ці три заагльнодоступні функції є замиканнями, які використовують загальний контекст виконання (оточення). Завдяки механізму lexical scoping в Javascript, всі вони мають доступ до змінної privateCounter і функції changeBy.

Контрольні питання

1. Як включати сценарії на сторінку?
2. Що таке примітиви?
3. Які типи даних JS Ви знаєте?
4. Які види циклів Ви знаєте?
5. Як перевіряти умови?
6. Які способи об’яви функції Ви знаєте?
7. Чи можна створити об’єкт Math?

РОЗДІЛ 7
РОБОТА З ОБ'ЄКТАМИ JAVASCRIPT

7.1. Об'єкти

JavaScript спроектовано на основі простої парадигми. В основі концепції є прості об'єкти. Об'єкт - це набір властивостей, і кожна властивість складається з імені та значення, асоційованого з цим ім'ям. Значення якості може бути функція, яку можна назвати методом об'єкта. На додаток до об'єктів, вбудованих у браузер, ви можете визначити свої власні об'єкти.

Ви можете записати об'єкт синтаксично, і його буде створено інтерпретатором автоматично під час виконання. Ця синтаксична схема наведена нижче:

```
var obj = { property_1: value_1, // property_# may be an identifier...
        2: value_2, // or a number...
        // ...,
        "property n": value_n}; // or a string
```

тут obj це ім'я нового об'єкта, кожне property_i це ідентифікатор (ім'я, число або рядковий літерал), і кожен value_i це значення, призначені property_i. Ім'я obj та посилання об'єкта на нього необов'язкове; якщо далі вам не треба буде посилатися на цей об'єкт, то вам не обов'язково призначати об'єкт змінної.

Наприклад:

```
let myHonda = {
  wheels: 4,
  engine: {
    cylinders: 4,
    size: 2.2
  }
};
```

Властивість має ключ (також відомий як “ім'я” або “ідентифікатор”) перед двокрапкою ":" і значення праворуч від нього.

Нову властивість можна додати у будь-який момент, зверненням до неї:

```
myHonda.color = “blue”;
```

Ім'я властивості може складатися з декількох слів, але тоді воно має бути поміщено в лапки:

```
let myHonda = {
  “cruise control”: true,
```

```
  wheels: 4,
  engine: {
    cylinders: 4,
    size: 2.2
  }
};
```

Щоб видалити властивість ми можемо використати оператор delete:

```
delete myHonda.wheels;
```

Для властивостей, імена яких складаються з декількох слів, доступ до значення «через крапку» не працює. Це помилка:

```
myHonda. cruise control = false;
```

Для таких випадків існує альтернативний спосіб доступу до властивостей через квадратні дужки. Такий спосіб спрацює з будь-яким ім’ям властивості:

```
myHonda[“cruise control”] = false;
myHonda[“wheels”] = 4;
delete myHonda[“wheels”];
```

Ми можемо використовувати квадратні дужки в літеральній нотації для створення обчислюваної властивості.

Наприклад:

```
let fruit = prompt("Які фрукти купити?", "apple");
let bag = {
  [fruit]: 5, // назву властивості взято зі змінної fruit
};
alert( bag.apple ); // 5 якщо fruit="apple"
```

Значення обчислюваної властивості просте: [fruit] означає, що назву властивості слід брати з fruit. По суті, це працює так само, як:

```
let fruit = prompt("Які фрукти купити?", "apple");
let bag = {};
// назву властивості взято зі змінної fruit
bag[fruit] = 5;
```

Ми можемо використати більш складні вирази в квадратних дужках:

```
let fruit = 'apple';
let bag = {
  [fruit + 'Computers']: 5 // bag.appleComputers = 5
};
```

У реальному коді ми часто використовуємо наявні змінні як значення для імен властивостей. Приклад:

```
function makeUser(name, age) {
  return {
    name: name,
    age: age,
    // ...інші властивості
  };
}
let user = makeUser("Іван", 30);
alert(user.name); // Іван
```

наведеному вище прикладі назва властивостей name і age збігаються з назвами змінних, які ми підставляємо в якості значень цих властивостей. Такий підхід настільки поширений, що існують спеціальні короткі властивості для спрощення цього запису.

Замість name: name ми можемо написати просто name:

```
function makeUser(name, age) {
  return {
    name, // те ж саме, що name: name
    age,  // те ж саме, що age: age
    // ...
  };
}
```

Помітною особливістю об’єктів у JavaScript, у порівнянні з багатьма іншими мовами, є можливість доступу до будь-якої властивості. Помилки не буде, якщо властивості не існує.

Читання відсутньої властивості просто повертає undefined. Тому ми можемо легко перевірити, чи існує властивість:

```
let user = {};
alert( user.noSuchProperty === undefined ); // true означає "такої властивості немає"
```

Для цього також є спеціальний оператор "in".

Синтаксис такий:

```
"key" in object
```

Наприклад:

```
let user = { name: "Іван", age: 30 };
alert( "age" in user ); // true, user.age існує
alert( "blabla" in user ); // false, user.blabla не існує
```

Якщо ми опускаємо лапки, це означає, що ми вказуємо змінну, в якій знаходиться ім'я властивості. Наприклад:

```
let user = { age: 30 };
let key = "age";
alert( key in user ); // true, властивість "age" існує
```

Для перебору всіх властивостей об'єкта використовується цикл for..in. Цей цикл відрізняється від вивченого раніше циклу for(;;).

Синтаксис:

```
for (key in object) {
  // тіло циклу виконується для кожної властивості об'єкта
}
```

Приклад:

```
const object = { a: 1, b: 2, c: 3 };
for (const property in object) {
  console.log(`${property}: ${object[property]}`);
}
// Expected output:
// "a: 1"
// "b: 2"
// "c: 3"
```

Властивості з цілочисельними ключами сортуються за зростанням, інші розташовуються в порядку створення. Наприклад:

```
let codes ={
   "49": "Німеччина",
   "41": "Швейцарія",
   "44": "Великобританія",
   // ..,
   "1": "США"
};
for (let code in codes) {
  alert(code); // 1, 41, 44, 49
}
```

Метод - це функція, асоційована з об'єктом або, простіше кажучи, метод - це властивість об'єкта, що є функцією. Методи визначаються так само, як і звичайні функції, за винятком, що вони присвоюються властивості об'єкта. Наприклад, ось так:

```
objectName.methodname = function_name;
var myObj = {
  myMethod: function(params) {
    // ...do something
  }
};
```

Потім ви можете викликати метод у контексті об'єкта таким чином:

```
object.methodname(params);
```

Наприклад:

```
let myHonda = {
  color: “red”,
  wheels: 4,
  engine: {
    cylinders: 4,
    size: 2.2
  },
  show: function() {alert(`${this.wheels} ${this.color}`);}

};
myHonda.show();
```

Існує коротший синтаксис для методів в літералі об’єкта:

```
let myHonda = {
  show() {alert(`${this.wheels} ${this.color}`);}

};
```

Отож якщо користувач не вказав адресу, а ми своєю чергою спробуємо отримати доступ до властивості user.address.street, то отримаємо помилку.

```
let user = {}; // користувач без властивості "address"
alert(user.address.street); // помилка!
```

Найочевиднішим рішенням було б перевірити властивість використавши if або за допомогою умовного оператора ?:

```
let user = {};
alert(user.address ? user.address.street : undefined);
```

Але найкращим способом є використання опціонального ланцюжка “?”. Опціональний ланцюжок ?. припиняє обчислення, якщо значення перед ?. є undefined або null, і повертає

undefined:

let user = {}; // користувач без властивості "address"

alert(user?.address?.street); // undefined (немає помилки)

?. негайно припиняє обчислення, якщо лівої частини не існує. Наприклад:

let user = null;

let x = 0;

user?.sayHi(x++); // немає "user", отже до x++ обчислення не дійде

alert(x); // 0, значення не було збільшено

В прикладі нижче не в усіх користувачів є метод admin:

let userAdmin = {

admin() {

alert("Я адміністратор");

}

};

let userGuest = {};

userAdmin.admin?.(); // Я адміністратор

userGuest.admin?.(); // нічого (немає такого методу)

Також існує синтаксис ?.[], якщо ми хочемо отримати доступ до властивості за допомогою квадратних дужок [], а не через крапку .. Як і в решті випадків, такий спосіб дає змогу безпечно читати властивості об’єкту яких може не існувати.

let key = "firstName";

let user1 = {

firstName: "Іван"

};

let user2 = null;

alert(user1?.[key]); // Іван

alert(user2?.[key]); // undefined

Ми також можемо використовувати ?. з delete:

delete user?.name; // видалити user.name, якщо користувач існує

Опціональний ланцюжок ?. не має сенсу у лівій частині присвоювання.

Наприклад:

let user = null;

user?.name = "Іван"; // Помилка, не спрацює

// це по суті те ж саме що undefined = "John"

Деструктуроване присвоєння працює з об’єктами.

Основний синтаксис такий:

let {var1, var2} = {var1:…, var2:…}

Ми повинні мати існуючий об'єкт праворуч, який ми хочемо розділити на змінні. Ліва частина містить об'єктоподібний "шаблон" для відповідних властивостей. У найпростішому випадку це список імен змінних у {...}.

Наприклад:

```
let options = {
  title: "Меню",
  width: 100,
  height: 200
};
let {title, width, height} = options;
alert(title);  // Меню
alert(width);  // 100
alert(height); // 200
```

Порядок не має значення. Це теж працює:

```
// змінили порядок у let {...}
let {height, width, title} = { title: "Меню", height: 200, width: 100 }
```

Якщо ми хочемо присвоїти властивість змінній з іншим іменем, наприклад, зробити так, щоб options.width переходив до змінної з назвою w, то ми можемо встановити ім'я змінної за допомогою двокрапки:

```
let options = {
  title: "Меню",
  width: 100,
  height: 200
};
// { sourceProperty: targetVariable }
let {width: w, height: h, title} = options;
// width -> w
// height -> h
// title -> title
alert(title);  // Меню
alert(w);      // 100
alert(h);      // 200
```

Для потенційно відсутніх властивостей ми можемо встановити типові значення за

допомогою "=", наприклад:

```
let options = {
  title: "Меню"
};
let {width = 100, height = 200, title} = options;
alert(title);  // Меню
alert(width);  // 100
alert(height); // 200
```

Що робити, якщо об’єкт має більше властивостей, ніж ми маємо змінних? Чи можемо ми взяти частину, а потім призначити кудись «залишок»?

Ми можемо використовувати шаблон залишкового оператору, так само, як ми робили з масивами. Він не підтримується деякими старішими браузерами (IE, використовуйте Babel для поліфілу), але працює в сучасних.

Це виглядає наступним чином:

```
let options = {
  title: "Меню",
  height: 200,
  width: 100
};
// title = властивість з назвою title
// rest = об’єкт з залишковими властивостями
let {title, ...rest} = options;
// тепер title="Меню", rest={height: 200, width: 100}
alert(rest.height);  // 200
alert(rest.width);   // 100
```

Ми можемо передати параметри функції як об’єкт, і функція негайно деструктурує їх на змінні:

```
// ми передаємо об’єкт до функції
let options = {
  title: "My menu",
  items: ["Item1", "Item2"]
};
// ...і вона негайно розгортає його до змінних
function showMenu({title = "Untitled", width = 200, height = 100, items = []}) {
  // title, items – взяті з options,
```

```
  // width, height – використовуються типові значення
  alert( `${title} ${width} ${height}` ); // My Menu 200 100
  alert( items ); // Item1, Item2
}
showMenu(options);
```

7.2. Посилання на об’єкти. Копіювання об’єктів

Однією з принципових відмінностей об’єктів від примітивів є те, що об’єкти зберігаються та копіюються “за посиланням”, тоді як примітивні значення: рядки, числа, логічні значення тощо – завжди копіюються “за значенням”. У JavaScript об'єкти мають тип посилання. Два окремі об'єкти ніколи не будуть рівними, навіть якщо вони мають рівний набір властивостей. Тільки порівняння двох посилань на той самий об'єкт поверне true.

```
// Дві змінні посилаються на два об'єкти з однаковими властивостями
var fruit = {name: 'apple'};
var fruitbear = {name: 'apple'};
fruit == fruitbear; // поверне false
fruit === fruitbear; // поверне false
// Дві змінні посилаються однією загальний об'єкт
var fruit = {name: 'apple'};
var fruitbear = fruit; // надамо змінної fruitbear посилання на об'єкт fruit
// тепер fruitbear і fruit посилаються однією і той самий об'єкт
fruit == fruitbear; // поверне true
fruit === fruitbear; // поверне true
fruit.name = 'grape';
console.log(fruitbear); // поверне { name: "grape"} замість {name: "apple"}
```

Отже, копіювання змінної об’єкта створює ще одне посилання на той самий об’єкт.

Важливим побічним ефектом зберігання об’єктів як посилань є те, що властивості об’єкту, оголошеного як const, може бути змінені.

Якщо нам потрібно створити копію об’єкта, то необхідно перебрати та скопіювати властивості його:

```
let user = {
  name: "Іван",
  age: 30
};
```

let clone = {}; // новий порожній об'єкт

// скопіюймо в нього всі властивості з user

for (let key in user) {

 clone[key] = user[key];

}

// тепер клон - це повністю незалежний об'єкт з однаковим вмістом

clone.name = "Петро"; // змінемо його вміст

alert(user.name); // як і раніше Іван залишився в оригінальному об'єкті

Також ми можемо використати метод Object.assign для цього:

Його синтаксис:

Object.assign(dest, [src1, src2, src3...])

Перший аргумент dest – це цільовий об'єкт, у який ми будемо копіювати.

Наступні аргументи src1, ..., srcN (їх може бути скільки завгодно) – це вихідні об'єкти, з яких ми будемо копіювати.

Він копіює властивості всіх вихідних об'єктів src1, ..., srcN у цільовийdest. Іншими словами, властивості всіх аргументів, починаючи з другого, копіюються в перший об'єкт.

Виклик повертає dest.

const obj = { a: 1 };

const copy = Object.assign({}, obj);

console.log(copy); // { a: 1 }

Якщо приймаючий об'єкт вже має властивість з таким ім'ям, її буде перезаписано:

const target = { a: 1, b: 2 };

const source = { b: 4, c: 5 };

const returnedTarget = Object.assign(target, source);

console.log(target);

// Expected output: Object { a: 1, b: 4, c: 5 }

console.log(returnedTarget === target);

// Expected output: true

Існує також клонування за допомогою методу розширення …

let obj = { a: 1, b: 2, c: 3 };

let objCopy = { ...obj }; // розширить об'єкт у список параметрів

// потім поверне результат у новий об'єкт

// чи однаковий вміст мають об'єкти?

alert(JSON.stringify(obj) === JSON.stringify(objCopy)); // true

// чи однакові об'єкти?

```
alert(obj === objCopy); // false (не однакові посилання)
// зміна нашого початкового об'єкта не змінює копію:
obj.d = 4;
alert(JSON.stringify(obj)); // {"a":1,"b":2,"c":3,"d":4}
alert(JSON.stringify(objCopy)); // {"a":1,"b":2,"c":3}
```

Об'єкти можна також створювати за допомогою методу Object.create. Цей метод дуже зручний, тому що дозволяє вам вказувати об'єкт прототип для нового вашого об'єкта:

```
// Список властивостей і методів для Animal
var Animal = {
  type: 'Invertebrates', // Значення type за умовчанням
  displayType: function() { // Метод, що відображає тип об'єкта Animal
    console.log(this.type);
  }
};

// Створюємо об'єкт Animal
var animal1 = Object.create(Animal);
animal1.displayType(); // Виведе: Invertebrates

// Створюємо об'єкт Animal та присвоюємо йому type = Fishes
var fish = Object.create(Animal);
fish.type = 'Fishes';
fish.displayType(); // Виведе: Fishes
```

Властивості можуть бути посиланнями на інші об’єкти:

```
let user = {
  name: "Іван",
  sizes: {
    height: 182,
    width: 50
  }
};
alert( user.sizes.height ); // 182
```

Тепер при клонуванні недостатньо просто скопіювати clone.sizes = user.sizes, тому що user.sizes є об’єктом, і він буде скопійований за посиланням. Тому clone і user у своїх властивостях sizes будуть посилатися на той самий об’єкт:

Ось так:

```
let user = {
  name: "Іван",
  sizes: {
    height: 182,
    width: 50
  }
};
let clone = Object.assign({}, user);
alert( user.sizes === clone.sizes ); // true, той самий об’єкт
// user і clone мають посилання на єдиний об’єкт у властивості sizes
user.sizes.width++;       // міняємо властивість з одного місця
alert(clone.sizes.width); // 51, бачимо результат в іншому об’єкті
```

Щоб це виправити, слід використовувати цикл клонування, який перевіряє кожне значення user[key], і якщо це об’єкт, то також копіює його структуру. Це називається “глибоким клонуванням”. Наприклад:

```
let cloneObject = function (obj) {
    const clone = {};
    for (let key in obj) {
        if (!obj.hasOwnProperty(key)) continue;
        clone[key] = (typeof obj[key] === 'object') ? cloneObject(obj[key]) : obj[key];
    }
    return clone;
}
let john = {
    firstName: 'John',
    lastName: 'Daniels',
    age: 34,
    address: {
        street: '2nd Avenu',
        appartment: '27',
        city: 'New York'
    },
```

```
        showInfo: function(){
            console.log(`${this.firstName} ${this.lastName} from ${this.address.city}`)
        }
    }
    let cloneJohn = cloneObject(john);
    console.log(cloneJohn === john); //false
    console.log(john, cloneJohn);
    cloneJohn.address.city = 'Chicago';
    console.log(john.address.city);
    console.log(cloneJohn.address.city);
```

7.3. Конструктори та оператор new

Звичайний синтаксис {...} дозволяє створити тільки один об'єкт. Проте часто нам потрібно створити багато однотипних об'єктів, таких як, наприклад, користувачі чи елементи меню тощо. Це можна зробити за допомогою функції-конструктора та оператора "new".

Визначте тип об'єкта, написавши функцію-конструктор. Назва такої функції, як правило, починається з великої літери. Створіть екземпляр об'єкта за допомогою ключового слова new.

```
function Car(brand, model, year) {
  this.brand = brand;
  this.model = model;
  this.year = year;
}
```

Зауважте, що використовується this щоб надати значення (передані як аргументи функції) властивостям об'єкта. Тепер ви можете створити об'єкт, званий mycar, так:

```
let mycar = new Car("Honda", "Accord", 2006);
```

Об'єкт може мати властивість, яка буде іншим об'єктом. Наприклад, далі визначається об'єкт типу Person наступним чином:

```
function Person(name, age, sex) {
  this.name = name;
  this.age = age;
  this.sex = sex;
}
function Car(brand, model, year,owner) {
  this.brand = brand;
```

```
  this.model = model;
  this.year = year;
  this.owner = owner;
}
let rand = new Person("Rand McKinnon", 33, "M");
let mycar = new Car("Honda", "Accord", 2006, rand);
```

Технічно будь-яка функція (окрім стрілкових функцій, оскільки вони не мають власного this) може бути використана як конструктор. Вона може бути запущена через new, і буде виконано наведений вище алгоритм. "Ім'я з великої літери" це загальна домовленість, яка допомагає чітко зрозуміти, що функцію слід запускати з new.

Використовуючи спеціальну властивість new.target всередині функції, ми можемо перевірити чи була ця функція викликана за допомогою оператора new чи без нього.

Якщо функція була викликана за допомогою new, то в new.target буде сама функція, в іншому разі отримаємо undefined:

```
function User() {
  alert(new.target);
}
// виклик без "new":
User(); // undefined
// виклик з "new":
new User(); // function User { ... }
```

Така можливість може бути використана всередині функції для того, щоб дізнатися чи функція була викликана за допомогою оператора new, "у режимі конструктора", чи без нього, "у звичайному режимі".

Ми також можемо зробити щоб обидва виклики, з new та звичайний, робили одне й те саме, таким чином:

```
function User(name) {
  if (!new.target) { // якщо ви викликали без оператора new
    return new User(name); // ...додамо оператор new за вас
  }
  this.name = name;
}
let john = User("Джон"); // перенаправляє виклик до new User
alert(john.name); // Джон
```

Зазвичай конструктори не мають інструкції return. Їх завдання – записати усе необхідне

у this, яке автоматично стане результатом.

Але якщо є інструкція return, то застосовується просте правило:

1. Якщо return викликається з об’єктом, тоді замість this буде повернено цей об’єкт.
2. Якщо return викликається з примітивом, примітив ігнорується.
3. Інакше кажучи, return з об’єктом повертає цей об’єкт, у всіх інших випадках повертається this.

У наступному прикладі return перезаписує this, повертаючи об’єкт:

```
function BigUser() {
  this.name = "Джон";
  return { name: "Ґодзілла" };  // <-- повертає цей об’єкт
}
alert( new BigUser().name );  // Ґодзілла, отримали цей об’єкт
```

А ось приклад з порожнім return (або ми можемо розмістити примітив після нього, не має значення):

```
function SmallUser() {
  this.name = "Джон";
  return; // <-- повертає this
}
alert( new SmallUser().name );  // Джон
```

До речі, ми можемо опустити дужки після new, якщо виклик конструктора відбувається без аргументів:

```
let user = new User; // <-- немає дужок
// те саме, що
let user = new User();
```

Використання конструкторів для створення об’єктів дає велику гнучкість. Конструктор може мати параметри, які визначають, як побудувати об’єкт і що в нього помістити.

Звичайно, ми можемо додати до this не лише властивості, але й методи:

```
function Car(brand, model, year, owner) {
  this.brand = brand;
  this.model = model;
  this.year = year;
  this.owner = owner;
  this.displayCar = function () {
    alert(this.year + " " + this.brand+ " " + this.model);
  };
```

}

JavaScript має спеціальне ключове слово this, яке ви можете використовувати всередині методу, щоб посилатися на поточний об'єкт:

Наприклад:

```
let user = {
  name: "Іван",
  age: 30,
  sayHi() {
    // "this" -- це "поточний об’єкт"
    alert(this.name);
  }
};
user.sayHi(); // Іван
```

Значення this обчислюється під час виконання і залежить від контексту.

Наприклад, тут одна й та ж функція призначена двом різним об’єктам і має різний “this” при викликах:

```
let user = { name: "Іван" };
let admin = { name: "Адмін" };
function sayHi() {
  alert( this.name );
}
// використовуємо одну і ту ж функцію у двох об’єктах
user.f = sayHi;
admin.f = sayHi;
// виклики функцій, приведені нижче, мають різні this
// "this" всередині функції являється посиланням на об’єкт "перед крапкою"
user.f(); // Іван  (this == user)
admin.f(); // Адмін  (this == admin)
admin['f'](); // Адмін (неважливо те, як звертатися до методу об’єкта -- через крапку чи [])
```

7.4. Symbol

Символ (англ. Symbol) — це унікальний і незмінний тип даних, який можна використовувати як ідентифікатор для властивостей об'єктів. Щоб створити новий символьний примітив, достатньо написати Symbol(), вказавши за бажанням рядок як опис

цього символу:

```
let sym1 = Symbol();
let sym2 = Symbol("foo");
let sym3 = Symbol("foo");
```

Символи гарантовано будуть унікальними. Навіть якщо ми створюємо багато символів з однаковим описом, вони мають різні значення. Зауважте, що Symbol("foo") не виконує приведення (англ. coercion) рядка "foo" до символу. Цей вираз створює щоразу новий символ:

```
Symbol("foo") === Symbol("foo"); // false
```

Символи не перераховуються під час ітерації for...in. Завдяки цьому, символи дозволяють нам створювати “приховані” властивості об’єкта, до яких жодна інша частина програми не може випадково отримати доступ або перезаписати їх.

```
var obj = {};
obj[Symbol("a")] = "a";
obj[Symbol.for("b")] = "b";
obj["c"] = "c";
obj.d = "d";
for (var i in obj) {
   console.log(i); // виведе "c" та "d"
}
```

А ось, Object.assign копіює властивості рядка та символу:

```
let id = Symbol("id");
let user = {
  [id]: 123
};
let clone = Object.assign({}, user);
alert( clone[id] ); // 123
```

Існує також глобальний реєстр символів. Ми можемо створити в ньому символи і отримати до них доступ пізніше, і це гарантує, що повторні звернення з тим самим іменем нам повернуть абсолютно однаковий символ. Для того, щоб знайти (створити, якщо його немає) символ у реєстрі, використовуйте Symbol.for(key). Цей виклик перевіряє глобальний реєстр, і якщо є символ з іменем key, тоді повертає його, інакше створює новий символ Symbol(key) і зберігає його в реєстрі за вказаним key.

Наприклад:

```
// шукаємо в глобального реєстрі
let id = Symbol.for("id"); // якщо такого символу немає, він буде створений
```

```
// шукаємо знову, але присвоюємо в іншу змінну (можливо в іншій частині коду)
let idAgain = Symbol.for("id");
// як бачимо, це один і той самий символ
alert( id === idAgain ); // true
```

Символи всередині реєстру називаються глобальні символи. Якщо вам потрібні символи, доступні скрізь у коді – використовуйте глобальні символи.

Для глобальних символів, не тільки Symbol.for(key) повертає символ за іменем, також існує протилежний метод: Symbol.keyFor(sym), який працює навпаки: приймає глобальний символ і повертає його ім’я.

Наприклад:

```
// отримуємо символ за іменем
let sym = Symbol.for("name");
let sym2 = Symbol.for("id");
// отримуємо ім’я за символом
alert( Symbol.keyFor(sym) ); // name
alert( Symbol.keyFor(sym2) ); // id
```

7.5. Перетворення об’єктів в примітиви

Всі об’єкти – це true в булевому контексті.

Числове перетворення відбувається, коли ми віднімаємо об’єкти або застосовуємо математичні функції.

Що стосується перетворення об’єктів на рядки – це зазвичай відбувається, коли ми виводимо об’єкт, наприклад alert(obj) і в подібних контекстах.

Але можна перетворювати об’єкти на рядки та числа, використовуючи спеціальні методи об’єкту.

Є три варіанти перетворення типів, що відбуваються в різних ситуаціях. Вони називаються “підказками” (“hints”), і описані в специфікації:

1. "string". Перетворення об’єкта в рядок відбувається коли ми виконуємо операцію, яка очікує рядок, над об’єктом. Наприклад, alert:

```
// вивід
alert(obj);
// використання об’єкта як ключа властивості об’єкта
anotherObj[obj] = 123;
```

2. "number". Перетворення об’єкта в число, коли ми робимо математичні операції:

// явне перетворення

let num = Number(obj);

// математичні операції (крім бінарного додавання)

let n = +obj; // унарне додавання

let delta = date1 - date2;

// порівняння менше/більше

let greater = user1 > user2;

Більшість вбудованих математичних функцій також включають таке перетворення.

3. "default" Виникає в рідкісних випадках, коли оператор “не впевнений”, який тип очікується.

Наприклад, бінарний плюс + може працювати як з рядками (об’єднувати їх), так і з цифрами (додавати їх), тому обидва випадки – рядки та цифри – будуть працювати. Отже, якщо бінарний плюс отримує об’єкт як аргумент, він використовує підказку "default", щоб перетворити його.

Також, якщо об’єкт порівнюється (==) з рядком, числом чи символом, тоді незрозуміло, яке порівняння використати, тому використовується підказка "default".

// бінарний плюс використовує підказку "default"

let total = obj1 + obj2;

// obj == цифра використовує підказку "default"

if (user == 1) { ... };

Оператори порівняння більше та менше, такі як < >, також можуть працювати як з рядками, так і з цифрами. Проте, вони з історичних причин використовують "number" підказку, а не "default".

Але на практиці все трохи простіше.

Всі вбудовані об’єкти, крім одного випадку (об’єкт Date, ми дізнаємося пізніше) реалізовують "default" перетворення так само, як "number". І ми можемо зробити те ж саме.

Щоб зробити перетворення, JavaScript намагається знайти та викликати три методи об’єкта:

1. Викликати obj[Symbol.toPrimitive](hint) – метод з символьним ключем Symbol.toPrimitive (системний символ), якщо такий метод існує,

2. Інакше, якщо підказка – це "string" спробує obj.toString() або obj.valueOf() – будь-що, що існує.

3. Інакше, якщо підказка – "number" або "default" спробує obj.valueOf() або obj.toString() – будь-що, що існує.

Якщо метод symbol.toPrimitive існує, він використовується для всіх підказок, і не

потрібно більше методів.

Наприклад, тут об’єкт user реалізує його:

```
let user = {
  name: "Іван",
  money: 1000,

  [Symbol.toPrimitive](hint) {
    alert(`hint: ${hint}`);
    return hint == "string" ? `{name: "${this.name}"}` : this.money;
  }
};
// демонстрація перетворення:
alert(user); // hint: string -> {name: "Іван"}
alert(+user); // hint: number -> 1000
alert(user + 500); // hint: default -> 1500
```

Якщо немає Symbol.toPrimitive тоді JavaScript намагається знайти методи toString і valueOf:

1. Для "string" підказка: виклик методу toString, і якщо цей метод не існує, то valueOf (таким чином toString має пріоритет при перетворенні в рядок).
2. Для інших підказок: valueOf, і якщо це не існує, то toString (таким чином valueOf має пріоритет для математики).

Наприклад, тут user робить те ж саме, що й вище, використовуючи комбінацію toString і valueOf замість Symbol.toPrimitive:

```
let user = {
  name: "Іван",
  money: 1000,
  // для hint="string"
  toString() {
    return `{name: "${this.name}"}`;
  },
  // для hint="number" чи "default"
  valueOf() {
    return this.money;
  }
};
```

```
alert(user); // toString -> {name: "Іван"}
alert(+user); // valueOf -> 1000
alert(user + 500); // valueOf -> 1500
```

Важливо знати про всі методи примітивні перетворення те, що вони не обов'язково повертають "підказаний" примітив. Єдина обов'язкова річ: ці методи повинні повертати примітивний тип, а не об'єкт.

7.6. Робота з рядками

Об'єкт String має одну властивість: length, яка містить ціле число, що вказує на кількість символів строки. Наприклад:

```
let str="abc";
alert(str.length); //3
```

Отримати символ рядка, можна за допомогою оператору [] або методу charAt. Ви можете повернути будь-який символ усередині рядка, використовуючи позначення у квадратних дужках. Це означає, що ви додаєте квадратні дужки ([ ]) наприкінці вашого імені змінної. У квадратних дужках ви вказуєте номер символу, починаючи з 0, який хочете повернути.

```
let s = `Вітаю`;
// перший символ
alert( s[0] ); // В
alert( s.charAt(0) ); // В
// останній символ
alert( s[s.length - 1] ); // ю
```

Різниця між ними лише в тому, що якщо символ з такою позицією відсутній, тоді [] поверне undefined, а charAt – порожній рядок.

Ми також можемо перебрати рядок посимвольно, використовуючи for..of:

```
let s = `Вітаю`;
for (let char of s) {
  alert(char);
}
```

В JavaScript рядки не можна змінювати. Змінити символ неможливо. Можна лише створити новий рядок.

Рядкові методи toLowerCase() та toUpperCase() перетворюють усі символи в рядку в нижній або верхній регістр відповідно:

```
alert( 'Вітаю'.toUpperCase() ); // ВІТАЮ
```

alert('Вітаю'.toLowerCase()); // вітаю

Метод – str.indexOf(substr, pos) шукає підрядок substr в рядку str, починаючи з позиції pos, і повертає позицію, де знаходиться збіг, або -1 якщо збігів не було знайдено. Наприклад:

```
if(browserType.indexOf('mozilla') === -1) {
}
```

Щоб знайти усі збіги, нам потрібно запустити indexOf в циклі. Кожен новий виклик здійснюється з позицією після попереднього збігу:

```
let str = "Хитрий, як лисиця, сильний, як Як";
let target = "як";
let pos = -1;
while ((pos = str.indexOf(target, pos + 1)) != -1) {
  alert( pos );
}
```

Також є схожий метод str.lastIndexOf(substr, position), що виконує пошук від кінця рядка до його початку.

Сучасніший метод str.includes(substr, pos) повертає true/false в залежності від того чи є substr в рядку str.

Цей метод доцільно використовувати, коли потрібно перевірити чи є збіг, але не потрібна позиція:

```
alert( "Віджет з ідентифікатором".includes("Віджет") ); // true
alert( "Привіт".includes("Бувай") ); // false
```

Необов'язковий другий аргумент pos – це позиція з якої почнеться пошук:

```
alert( "Віджет".includes("ід") ); // true
alert( "Віджет".includes("ід", 3) ); // false, починаючи з 3-го символа, підрядка "ід" немає
```

Відповідно, методи str.startsWith та str.endsWith перевіряють, чи починається і чи закінчується рядок певним підрядком.

```
alert( "Віджет".startsWith("Від") ); // true, "Віджет" починається з "Від"
alert( "Віджет".endsWith("жет") ); // true, "Віджет" закінчується підрядком "жет"
```

Отримання підрядка можливо за допомогою str.slice(start [, end]). Повертає частину рядка починаючи від start до (але не включно) end

Коли ви знаєте, де підрядок починається всередині рядка, і ви знаєте, на якому символі ви хочете завершити, можна використовувати slice () для вилучення. Наприклад:

```
let browserType = ‘mozilla’
browserType.slice(0,3); //moz
```

Першим параметром є позиція символу, з якого починається вилучення, а другий

параметр - позиція останнього символу, перед яким потрібно відсікнути рядок.

Крім того, якщо ви знаєте, що хочете витягти решту символів у рядку після певного символу, вам не потрібно включати другий параметр. Достатньо включити лише положення символу, з якого ви хочете почати виймання символів, що залишилися в рядку:

browserType.slice(2); //zilla

Також для start/end можна задати від'ємне значення. Це означає, що позиція буде рахуватися з кінця рядка.

str.substring(start [, end]) повертає частину рядка між start та end.

Наприклад:

```
let str = "stringify";
// для substring ці два приклади однакові
alert( str.substring(2, 6) ); // "ring"
alert( str.substring(6, 2) ); // "ring"
// ...але не для slice:
alert( str.slice(2, 6) ); // "ring" (те саме)
alert( str.slice(6, 2) ); // "" (порожній рядок)
```

Від'ємні аргументи (на відміну від slice) не підтримуються, вони інтерпретуються як 0.

str.substr(start [, length]) - повертає частину рядка з позиції start, із заданою довжиною length. На відміну від попередніх методів, цей дозволяє вказати довжину length замість кінцевої позиції:

```
let str = "stringify";
alert( str.substr(2, 4) ); // 'ring', починаючи з позиції 2 отримуємо 4 символа
```

Перший аргумент може бути від'ємним, щоб рахувати з кінця:

```
let str = "stringify";
alert( str.substr(-4, 2) ); // 'gi', починаючи з позиції 4 з кінця отримуєму 2 символа
```

7.7. Робота з масивами

Масиви створюються з квадратних дужок, які містять список елементів, розділених комами:

```
let weekdays = ['Понеділок', 'Вівторок', 'Середа', 'Четвер', 'П'ятниця', 'Субота', 'Неділя'];
```

або

```
let weekdays = new Array('Понеділок', 'Вівторок', 'Середа', 'Четвер', 'П'ятниця', 'Субота', 'Неділя');
```

Ви можете після цього отримувати доступ до окремих елементів масиву,

використовуючи квадратні дужки:

```
alert(weekdays[0]); // Понеділок
alert(weekdays[2]); // Середа
```

Ви також можете змінювати елемент у масиві, просто давши окремому елементу масиву нове значення:

```
weekdays[0] = 'Пн';
```

Ви можете знайти довжину масиву (кількість елементів у ньому) таким самим способом, як ви знаходите довжину рядка (у символах) — використовуючи властивість length:

```
let sequence = [1, 1, 2, 3, 5, 8, 13];
for (let i = 0; i < sequence.length; i++) {
  console.log(sequence[i]);
}
```

У масивах можуть зберігатись елементи будь-якого типу:

```
// різні типи значень
let arr = [ 'Apple', { name: 'John' }, true, function() { alert('hello'); } ];
// отримати елемент з індексом 1 (об’єкт) та вивести його властивість name
alert( arr[1].name ); // John
// отримати елемент з індексом 3 (функція) та виконати її
arr[3](); // hello
```

Можна отримати елемент масиву за індексом з кінця:

```
alert( weekdays.at(-1) ); // Неділя
```

Іншими словами, arr.at(i):

- те ж саме що й arr[i], якщо i >= 0;
- для негативних значень і він шукає елемент відступаючи від кінця масиву.

Метод push() – додає елементи в кінець масиву:

```
myArray.push('Cardiff');
myArray.push('Bradford', 'Brighton');
```

Після завершення виклику методу повертається нова довжина масиву:

```
let newLength = myArray.push('Bristol');
```

Видалення останнього елемента масиву можна здійснити за допомогою виклику методу pop(). Після завершення виклику повертається видалений елемент:

```
let removedItem = myArray.pop();
```

unshift() і shift() працюють точно таким же способом, за винятком того, що вони працюють на початку масиву, а не в кінці:

```
myArray.unshift('Edinburgh'); // додаємо елемент на початку масиву
```

let removedItem = myArray.shift(); // видаляємо елемент з початку масиву

Методи push/pop працюють швидко, на відміну від методів shift/unshift, які працюють повільно.

Масив – це об’єкт, який поводить себе як об’єкт. Це можливо тому, що в основі масивів - об’єкти. Ми можемо додати будь-які властивості до них:

let fruits = []; // створюємо масив

fruits[99999] = 5; // створюємо властивість, індекс якої набагато перевищує довжину масиву

fruits.age = 25; // створюємо властивість з довільним ім’ям

Один з найстаріших методів перебору елементів масиву – це цикл for по індексах:

```
let arr = ["Apple", "Orange", "Pear"];
for (let i = 0; i < arr.length; i++) {
  alert( arr[i] );
}
```

Але для масивів можливий інший варіант циклу, for..of:

```
let fruits = ["Apple", "Orange", "Plum"];
// ітерується по елементам масиву
for (let fruit of fruits) {
  alert( fruit );
}
```

Технічно, так як масив це об’єкт, ми можемо використовувати цикл for..in:

```
let arr = ["Apple", "Orange", "Pear"];
for (let key in arr) {
  alert( arr[key] ); // Apple, Orange, Pear
}
```

Цикл for..in ітерується по всіх властивостях, не тільки по числових.

У браузерах та різних програмних середовищах існують масивоподібні об’єкти, які виглядають як масив. Тобто вони мають властивість length та індекси, проте вони також містять інші нечислові властивості і методи, які нам часто не потрібні. Цикл for..in відобразить і їх. Тому, коли нам необхідно працювати з масивами, ці “екстра” властивості можуть стати проблемою. Цикл for..in оптимізований для довільних об’єктів, не для масивів, і тому значно повільніше.

Метод arr.splice – це універсальний метод для додавання, видалення і заміни елементів.

Його синтаксис:

arr.splice(start[, deleteCount, elem1, ..., elemN])

Він змінює arr починаючи з позиції start: видаляє deleteCount елементів і вставляє elem1, ..., elemN на їх місце. Повертається масив з видалених елементів.

let arr = ["I", "study", "JavaScript"];

arr.splice(1, 1); // з індексу 1 видалимо 1 елемент

alert(arr); // ["I", "JavaScript"]

У наступному прикладі ми видаляємо 3 елементи та замінюємо їх двома іншими:

let arr = ["I", "study", "JavaScript", "right", "now"];

// видалимо 3 перших елементи і замінимо їх іншими

arr.splice(0, 3, "Let's", "dance");

alert(arr) // отримаєм ["Let's", "dance", "right", "now"]

Тут ми бачимо, що splice повертає масив видалених елементів:

let arr = ["I", "study", "JavaScript", "right", "now"];

// видалимо 2 перших елементи

let removed = arr.splice(0, 2);

alert(removed); // "I", "study" <-- масив видалених елементів

Метод splice також може вставляти елементи без будь-яких видалень. Для цього нам потрібно встановити значення 0 для deleteCount:

let arr = ["I", "study", "JavaScript"];

// починаючт з індекса 2

// видалимо 0 елементів

// вaставити "complex" та "language"

arr.splice(2, 0, "complex", "language");

alert(arr); // "I", "study", "complex", "language", "JavaScript"

Метод arr.slice набагато простіший, ніж схожий на нього arr.splice. Синтаксис:

arr.slice([start], [end])

Він повертає новий масив, копіюючи до нього всі елементи від індексу start до end (не включаючи end). І start, і end можуть бути від'ємними. В такому випадку відлік буде здійснюватися з кінця масиву.

Він подібний до рядкового методу str.slice, але замість підрядків створює підмасиви.

Наприклад:

let arr = ["t", "e", "s", "t"];

alert(arr.slice(1, 3)); // e,s (копіює з 1 до 3)

alert(arr.slice(-2)); // s,t (копіює з -2 до кінця)

Метод arr.concat створює новий масив, в який копіює дані з інших масивів та додаткові значення.

Його синтаксис:

```
arr.concat(arg1, arg2...)
```

Він приймає будь-яку кількість аргументів – масивів або значень.

Результатом є новий масив, що містить елементи з arr, потімarg1, arg2 тощо.

Якщо аргумент argN є масивом, то всі його елементи копіюються. В іншому випадку буде скопійовано сам аргумент. Наприклад:

```
let arr = [1, 2];
// створимо масив з: arr і [3,4]
alert( arr.concat([3, 4]) ); // 1,2,3,4
// створимо масив з: arr, [3,4] і [5,6]
alert( arr.concat([3, 4], [5, 6]) ); // 1,2,3,4,5,6
// створимо масив з: arr і [3,4], також добавимо значення 5 і 6
alert( arr.concat([3, 4], 5, 6) ); // 1,2,3,4,5,6
```

Метод arr.forEach дозволяє запускати функцію для кожного елемента масиву:

```
["Bilbo", "Gandalf", "Nazgul"].forEach((item, index, array) => {
  alert(`${item} має позицію ${index} в масиві ${array}`);
});
```

Методи arr.indexOf, arr.lastIndexOf та arr.includes мають однаковий синтаксис і роблять по суті те ж саме, що і їх рядкові аналоги, але працюють з елементами замість символів:

- arr.indexOf(item, from) – шукає item, починаючи з індексу from, і повертає індекс, на якому був знайдений шуканий елемент, в іншому випадку‑1.
- arr.lastIndexOf(item, from) – те ж саме, але шукає справа наліво.
- arr.includes(item, from) – шукає item, починаючи з індексу from, і повертає true, якщо пошук успішний.

Наприклад:

```
let arr = [1, 0, false];
alert( arr.indexOf(0) ); // 1
alert( arr.indexOf(false) ); // 2
alert( arr.indexOf(null) ); // -1
alert( arr.includes(1) ); // true
```

Зверніть увагу, що методи використовують суворе порівняння ===. Таким чином, якщо ми шукаємо false, він знаходить саме false, а не нуль.

Пошук елементів за певною умовою можна здійснити за допомогою методу arr.find(fn).

Його синтаксис такий:

```
let result = arr.find(function(item, index, array) {
  // якщо true - повертається поточний елемент і перебір закінчується
  // якщо всі ітерації виявилися помилковими, повертається undefined
});
```

Функція-колбек викликається по черзі для кожного елемента масиву:

item – черговий елемент масиву.

index – його індекс.

array – сам масив.

Якщо функція повертає true, пошук припиняється, повертається item. Якщо нічого не знайдено, повертається undefined.

Наприклад, у нас є масив користувачів, кожен з яких має поля id та name. Давайте знайдемо той де id == 2:

```
let users = [
  {id: 1, name: "John"},
  {id: 2, name: "Pete"},
  {id: 3, name: "Mary"}
];
let user = users.find(item => item.id == 2);
alert(user.name); // Pete
```

Метод arr.findIndex – по суті, те ж саме, але повертає індекс, на якому був знайдений елемент, а не сам елемент, і -1, якщо нічого не знайдено.

Метод find шукає один (перший) елемент, на якому функція-колбек поверне true.

На той випадок, якщо знайдених елементів може бути багато, передбачений метод arr.filter(fn).

Синтаксис цього методу схожий з find, але filter повертає масив з усіх відфільтрованих елементів:

```
let results = arr.filter(function(item, index, array) {
  // якщо true - елемент додається до результату, і перебір триває
  // повертається порожній масив в разі, якщо нічого не знайдено
});
```

Наприклад:

```
let users = [
  {id: 1, name: "John"},
  {id: 2, name: "Pete"},
  {id: 3, name: "Mary"}
```

```
];
// повертає масив перших двох користувачів
let someUsers = users.filter(item => item.id < 3);
alert(someUsers.length); // 2
```

Метод arr.map є одним з найбільш корисних і часто використовуваних.

Він викликає функцію для кожного елемента масиву і повертає масив результатів виконання цієї функції.

Синтаксис:

```
let result = arr.map(function(item, index, array) {
  // повертається нове значення замість елемента
});
```

Наприклад, тут ми перетворюємо кожен елемент на його довжину:

```
let lengths = ["Bilbo", "Gandalf", "Nazgul"].map(item => item.length);
alert(lengths); // 5,7,6
```

Виклик arr.sort() сортує масив змінюючи в ньому порядок елементів.

Він повертає відсортований масив, але зазвичай повертається значення ігнорується, так як змінюється сам arr.

Наприклад:

```
let arr = [ 1, 2, 15 ];
// метод сортує вміст arr
arr.sort();
alert( arr );  // 1, 15, 2
```

За замовчуванням елементи сортуються як рядки.

Буквально, елементи перетворюються в рядки при порівнянні. Для рядків застосовується лексикографічний порядок, і дійсно виходить, що "2"> "15".

Щоб використовувати наш власний порядок сортування, нам потрібно надати функцію як аргумент arr.sort().

Функція повинна для пари значень повертати:

```
function compare(a, b) {
  if (a > b) return 1; // якщо перше значення більше за друге
  if (a == b) return 0; // якщо значення рівні
  if (a < b) return -1; // якщо перше значення меньше за друге
}
```

Наприклад, для сортування чисел:

```
function compareNumeric(a, b) {
```

```
  if (a > b) return 1;
  if (a == b) return 0;
  if (a < b) return -1;
}
let arr = [ 1, 2, 15 ];
arr.sort(compareNumeric);
alert(arr);  // 1, 2, 15
```

Метод arr.reverse змінює порядок елементів в arr на зворотний.

Наприклад:

```
let arr = [1, 2, 3, 4, 5];
arr.reverse();
alert( arr ); // 5,4,3,2,1
```

Метод str.split(delim) розбиває рядок на масив по заданому роздільнику delim. У прикладі нижче таким роздільником є рядок з коми та пропуску.

```
let names = 'Вася, Петя, Маша';
let arr = names.split(', ');
for (let name of arr) {
  alert( `A message to ${name}.` );
}
```

Виклик arr.join(glue) робить в точності протилежне split. Він створює рядок з елементів arr, вставляючи glue між ними.

Наприклад:

```
let arr = ["Вася", "Петя", "Маша"];
let str = arr.join(';'); // об'єднуємо масив в рядок за допомогою ";"
alert( str ); // Вася;Петя;Маша
```

Методи arr.reduce та arr.reduceRight схожі на методи вище, але вони трохи складніші. Вони використовуються для обчислення якогось одного значення на основі всього масиву.

Синтаксис:

```
let value = arr.reduce(function(accumulator, item, index, array) {
  // ...
}, [initial]);
```

Функція застосовується по черзі до всіх елементів масиву і «переносить» свій результат на наступний виклик.

Аргументи:

accumulator – результат попереднього виклику цієї функції, дорівнює initial при першому

виклику (якщо переданий initial),

item – черговий елемент масиву,

index – його індекс,

array – сам масив.

При виконанні функції результат її виклику на попередньому елементі масиву передається як перший аргумент.

Тут ми отримаємо суму всіх елементів масиву лише одним рядком:

```
let arr = [1, 2, 3, 4, 5];
let result = arr.reduce((sum, current) => sum + current, 0);
alert(result); // 15
```

Ми також можемо опустити початкове значення:

```
let arr = [1, 2, 3, 4, 5];
// прибрано початкове значення (немає 0 в кінці)
let result = arr.reduce((sum, current) => sum + current);
alert( result ); // 15
```

Результат той самий. Це тому, що при відсутності initial в якості першого значення береться перший елемент масиву, а перебір стартує з другого.

Масиви не мають окремого типу в Javascript. Вони засновані на об’єктах. Тому typeof не може відрізнити простий об’єкт від масиву:

```
alert(typeof {}); // об’єкт
alert(typeof []); // також об’єкт
```

Але масиви використовуються настільки часто, що для цього придумали спеціальний метод: Array.isArray(value). Він повертає true, якщо value – це масив, інакше false.

```
alert(Array.isArray({})); // false
alert(Array.isArray([])); // true
```

Майже всі методи масиву, які викликають функції – такі як find, filter, map, за винятком методу sort, приймають необов’язковий параметр thisArg. Ось повний синтаксис цих методів:

```
arr.find(func, thisArg);
arr.filter(func, thisArg);
arr.map(func, thisArg);
// ...
// thisArg - це необов’язковий останній аргумент
```

Значение параметра thisArg становится this для func.

Наприклад, ось тут ми використовуємо метод об’єкта army як фільтр, і thisArg передає йому контекст:

```
let army = {
  minAge: 18,
  maxAge: 27,
  canJoin(user) {
    return user.age >= this.minAge && user.age < this.maxAge;
  }
};
let users = [
  {age: 16},
  {age: 20},
  {age: 23},
  {age: 30}
];
// знайти користувачів, для яких army.canJoin повертає true
let soldiers = users.filter(army.canJoin, army);
alert(soldiers.length); // 2
alert(soldiers[0].age); // 20
alert(soldiers[1].age); // 23
```

Деструктуроване присвоєння – це спеціальний синтаксис, що дозволяє нам “розпаковувати” масиви чи об’єкти в купу змінних, оскільки іноді це зручніше.

Ось приклад того, як масив деструктурується на змінні:

```
// у нас є масив з іменем та прізвищем
let arr = ["Іван", "Петренко"]
// деструктуроване присвоєння
// встановлює firstName = arr[0]
// та surname = arr[1]
let [firstName, surname] = arr;
alert(firstName); // Іван
alert(surname);  // Петренко
```

Це просто коротший спосіб написати:

```
let firstName = arr[0];
let surname = arr[1];
```

Небажані елементи масиву також можна викинути за допомогою додаткової коми:

```
// другий елемент не потрібен
let [firstName, , title] = ["Юлій", "Цезар", "Консул", "Римської республіки"];
```

alert(title); // Консул

Ми можемо використовувати його з будь-якими даними, які перебираються, а не тільки з масивами:

let [a, b, c] = "abc"; // ["a", "b", "c"]

Якщо ми хочемо також зібрати все наступне – ми можемо додати ще один параметр, який отримує "решту", використовуючи три крапки "...":

```
let [name1, name2, ...rest] = ["Юлій", "Цезар", "Консул", "Римської Республіки"];
// rest -- це масив елементів, починаючи з 3-го
alert(rest[0]); // Консул
alert(rest[1]); // Римської Республіки
alert(rest.length); // 2
```

Якщо ми хочемо, щоб "типове" значення замінило б відсутнє, ми можемо надати його за допомогою =:

```
// типове значення
let [name = "Гість", surname = "Анонім"] = ["Юлій"];
alert(name);    // Юлій (з масиву)
alert(surname); // Анонім (використовується типове значення)
```

7.8. Ітератори

Ітеративні об'єкти є узагальненням масивів. Це концепція, яка дозволяє нам зробити будь-який об'єкт придатним для використання в циклі for..of.

Наприклад, у нас є об'єкт, який не є масивом, але виглядає придатним для for..of.

Як, наприклад, об'єкт range, який представляє інтервал чисел:

```
let range = {
  from: 1,
  to: 5
};
```

Щоб зробити об'єкт range ітерабельним (і таким чином дозволити for..of працювати), нам потрібно додати метод до об'єкта з назвою Symbol.iterator (спеціальний вбудований символ саме для цього)^

1. Коли for..of запускається, він викликає цей метод один раз (або викидає помилку, якщо цей метод не знайдено). Метод повинен повернути iterator – об'єкт з методом next.
2. Далі for..of працює лише з поверненим об'єктом.

3. Коли for..of хоче отримати наступне значення, він викликає next() на цьому об'єкті.

4. Результат next() повинен мати вигляд {done: Boolean, value: any}, де done=true означає, що ітерація завершена, інакше value – це наступне значення.

Наприклад:

```
let range = {
  from: 2,
  to: 5
};

// 1. виклик for..of спочатку викликає цю функцію
range[Symbol.iterator] = function() {
  // 2. Далі, for..of працює тільки з цим ітератором, запитуючи у нього наступні значення
  return {
    current: this.from,
    to: this.to,
    next() {
      // 4. він повинен повертати значення як об'єкт {done:.., value :...}
      if (this.current <= this.to) {
        return { done: false, value: this.current++ };
      } else {
        return { done: true };
      }
    }
  };
};
```

*

```
for (let num of range) {
  alert(num); // 2, потім 3, 4, 5
}
```

Зверніть увагу на основну особливість ітеративних об'єктів: розділення проблем.

Сам range не має методу next().

Натомість інший об'єкт, так званий "ітератор", створюється за допомогою виклику range[Symbol.iterator](), а його next() генерує значення для ітерації.

Отже, об’єкт, що ітерує відокремлений від об’єкта, який він ітерує.

Для глибшого розуміння, подивімось, як явно використовувати ітератор:

```
let str = "Привіт";
// робить те ж саме, як
// for (let char of str) alert(char);
let iterator = str[Symbol.iterator]();
while (true) {
  let result = iterator.next();
  if (result.done) break;
  alert(result.value); // виводить символи один за одним
}
```

Існує універсальний метод Array.from, який приймає ітерований об’єкт або псевдомасив і робить з нього “справжній” масив. Тоді ми можемо викликати на ньому методи масиву:

```
// порахуємо квадрат кожного числа
let arr = Array.from(range, num => num * num);
alert(arr); // 1,4,9,16,25
```

7.9. Map та Set

Map це колекція ключ/значення, як і Object. Але основна відмінність полягає в тому, що Map дозволяє мати ключі будь-якого типу.

Методи та властивості:

new Map() – створює колекцію.

map.set(key, value) – зберігає значення value за ключем key.

map.get(key) – повертає значення за ключем; повертає undefined якщо key немає в колекції.

map.has(key) – повертає true якщо key існує, інакше false.

map.delete(key) – видаляє елемент по ключу.

map.clear() – видаляє всі елементи колекції.

map.size – повертає поточну кількість елементів.

Наприклад:

```
let map = new Map();
map.set('1', 'str1');   // рядок як ключ
map.set(1, 'num1');     // цифра як ключ
map.set(true, 'bool1'); // булеве значення як ключ
```

```
// Map зберігає тип ключів, так що в цьому випадку ми отримаємо 2 різних значення:
alert( map.get(1) );   // 'num1'
alert( map.get('1') ); // 'str1'
alert( map.size ); // 3
```

Як ми бачимо, на відміну від об’єктів, ключі не були приведені до рядків. Можна використовувати будь-які типи даних для ключів.

Для перебору колекції Map є 3 метода:

map.keys() – повертає об’єкт-ітератор для ключів,

map.values() – повертає об’єкт-ітератор для значень,

map.entries() – повертає об’єкт-ітератор зі значеннями виду [ключ, значення], цей варіант використовується за умовчанням у for..of.

Наприклад:

```
let recipeMap = new Map([
  ['огірок',   500],
  ['помідори', 350],
  ['цибуля',   50]
]);
// перебираємо ключі (овочі)
for (let vegetable of recipeMap.keys()) {
  alert(vegetable); // огірок, помідори, цибуля
}
// перебираємо значення (кількість)
for (let amount of recipeMap.values()) {
  alert(amount); // 500, 350, 50
}
// перебір елементів у форматі [ключ, значення]
for (let entry of recipeMap) { // те ж саме, що recipeMap.entries()
  alert(entry); // огірок,500 (і так далі)
}
```

При створенні Map ми можемо вказати масив (або інший об’єкт-ітератор) з парами ключ-значення для ініціалізації, як тут:

```
const map = new Map([
  ['firstName', 'Luke'],
  ['lastName', 'Skywalker'],
  ['occupation', 'Jedi Knight'],
```

```
])
```

До речі, цей синтаксис виглядає так само, як результат виклику Object.entries() для об'єкта. Це дає готовий спосіб перетворення об'єкта в карту, як показано в наступному блоці коду:

```
const luke = {
  firstName: 'Luke',
  lastName: 'Skywalker',
  occupation: 'Jedi Knight',
}
const map = new Map(Object.entries(luke))
```

Також ви можете перетворити картку назад на об'єкт або масив за допомогою одного рядка коду. Код нижче перетворює картку на об'єкт:

```
const obj = Object.fromEntries(map)
```

Це те ж саме, оскільки Object.fromEntries чекає аргументом об’єкт-ітератор, не обов’язково масив. А перебір map якраз повертає пари ключ/значення, як і map.entries(). Так що в підсумку ми матимемо звичайний об’єкт з тими ж ключами/значеннями, що і в map.

Для Map доступно дестрктурування:

```
let user = new Map();
user.set("name", "Іван");
user.set("age", "30");
// Map ітерує як пари [key, value], що дуже зручно для деструктурування
for (let [key, value] of user) {
  alert(`${key}:${value}`); // name:Іван, then age:30
}
```

Об’єкт Set – це особливий тип колекції: “множина” значень (без ключів), де кожне значення може з’являтися тільки раз.

Основні методи:

new Set(iterable) – створює Set, якщо аргументом виступає об’єкт-ітератор, тоді значення копіюються в Set.

set.add(value) – додає нове значення до Set, повертає Set.

set.delete(value) – видаляє значення з Set, повертає true, якщо value наявне в множині значень на момент виклику методу, інакше false.

set.has(value) – повертає true, якщо value присутнє в множині Set, інакше false.

set.clear() – видаляє всі значення множини Set.

set.size – повертає кількість елементів в множині.

Родзинкою Set є виклики set.add(value), що повторюються з однаковими значеннями value. Повторні виклики цього методу не змінюють Set. Це причина того, що кожне значення з'являється тільки один раз.

Наприклад, ми очікуємо гостей, і нам необхідно скласти їх список. Але повторні записи не повинні призводити до дублікатів. Кожен гість повинен з'явитися в списку лише один раз.

Множина Set – це саме те, що потрібно для цього:

```
let set = new Set();
let ivan = { name: "Іван" };
let petro = { name: "Петро" };
let maria = { name: "Марія" };
// підраховуємо гостей, деякі приходять кілька разів
set.add(ivan);
set.add(petro);
set.add(maria);
set.add(ivan);
set.add(maria);
// set зберігає тільки 3 унікальних значення
alert( set.size ); // 3
for (let user of set) {
  alert(user.name); // "Іван" (тоді "Петро" і "Марія")
}
```

Ми можемо перебрати вміст об'єкта set як за допомогою методу for..of, так і використовуючи forEach:

```
let set = new Set(["апельсини", "яблука", "банани"]);
for (let value of set) alert(value);
// те ж саме з forEach:
set.forEach((value, valueAgain, set) => {
  alert(value);
});
```

Зауважимо цікаву річ. Функція в forEach у Set має 3 аргументи: значення 'value', потім знову те саме значення 'valueAgain', і тільки потім цільовий об'єкт. Це дійсно так, значення з'являється в списку аргументів двічі.

Це зроблено для сумісності з об'єктом Map, в якому колбек forEach має 3 аргумента. Виглядає трохи дивно, але в деяких випадках може допомогти легко замінити Map на Set і

навпаки.

Set має ті ж вбудовані методи, що і Map:

set.keys() – повертає об'єкт-ітератор для значень,

set.values() – те ж саме, що set.keys(), для сумісності з Map,

set.entries() – повертає об'єкт-ітератор для пар виду [значення, значення], присутній для сумісності з Map.

Об'єкт WeakMap - колекція пар ключ-значення. Як ключі можуть бути використані лише об'єкти, а значення можуть бути довільних типів.

Тепер, якщо ми використовуємо об'єкт як ключ, і немає інших посилань на цей об'єкт – його буде видалено з пам'яті (і з мапи) автоматично.

```
let john = { name: "Іван" };
let weakMap = new WeakMap();
weakMap.set(john, "...");
john = null; // перезапишемо посилання
// john видалено з пам'яті!
```

WeakMap не підтримує ітерацію та методи keys(), values(), entries(), тому немає способу отримати всі ключі або значення від нього.

WeakMap має лише такі методи:

weakMap.get(key)

weakMap.set(key, value)

weakMap.delete(key)

weakMap.has(key)

Основна область застосування для WeakMap – це зберігання додаткових даних.

Якщо ми працюємо з об'єктом, що "належить" до іншого коду, можливо навіть сторонньої бібліотеки, і хотіли б зберегти деякі дані, пов'язані з ним, що повинні існувати лише поки об'єкт живий – тоді WeakMap це саме те, що потрібно.

WeakSet поводитися аналогічно:

Це аналог Set, але ми можемо додати лише об'єкти до WeakSet (не примітиви).

Об'єкт існує в наборі, коли він доступний з де-небудь ще.

Так само як Set, він підтримує add, has і delete, але не підтримує size, keys() та ітерацію.

Будучи "слабким", він також служить зберігання додаткових даних. Але не для довільних даних, а для фактів "так/ні". Приналежність до WeakSet може означати щось про об'єкт.

Наприклад, ми можемо додати користувачів до WeakSet, щоб відстежувати тих, хто відвідав наш сайт:

```
let visitedSet = new WeakSet();
let john = { name: "Іван" };
let pete = { name: "Петро" };
let mary = { name: "Марія" };
visitedSet.add(john); // Іван відвідав нас
visitedSet.add(pete); // Потім Петро
visitedSet.add(john); // Знову Іван
// visitedSet має зараз 2-ох користувачів
// перевірте, чи відвідав Іван?
alert(visitedSet.has(john)); // true
// перевірте, чи відвідала Марія?
alert(visitedSet.has(mary)); // false
john = null;
// visitedSet буде очищено автоматично
```

7.10. Робота з датою та часом

new Date() створює екземпляр об'єкта Date, що є моментом часу. Об'єкт Дата містить число мілісекунд, що пройшли з 1 січня 1970 р. UTC.

```
new Date();
new Date(value);
new Date(dateString);
new Date(year, month[, day[, hour[, minute[, second[, millisecond]]]]]);
```

Приклади:

```
let today = new Date();
let birthday = new Date('December 17, 1995 03:24:00');
let birthday = new Date('1995-12-17T03:24:00');
let birthday = new Date(1995, 11, 17);
let birthday = new Date(1995, 11, 17, 3, 24, 0);
new Date(milliseconds)
```

Створює об'єкт Date з часом, що дорівнює кількості мілісекунд (1/1000 секунди), що минули після 1 січня 1970 UTC+0.

```
// 0 означає 01.01.1970 UTC+0
let Jan01_1970 = new Date(0);
alert( Jan01_1970 );
```

// тепер додамо 24 години, отримаємо 02.01.1970 UTC+0

let Jan02_1970 = new Date(24 * 3600 * 1000);

alert(Jan02_1970);

Ціле число, яке являє собою кількість мілісекунд, що пройшли з початку 1970 року, називається міткою часу (timestamp).

Це легке числове представлення дати. Ми завжди можемо створити дату з timestamp за допомогою new Date(timestamp) і перетворити об’єкт Date, що існує, до timestamp за допомогою методу date.getTime().

Дати до 01.01.1970 р. мають негативний timestamp, наприклад:

// 31 грудня 1969 року

let Dec31_1969 = new Date(-24 * 3600 * 1000);

alert(Dec31_1969);

new Date(datestring)

Якщо є єдиний аргумент, і це рядок, то він автоматично аналізується. Алгоритм той же, що використовує Date.parse.

let date = new Date("2017-01-26");

alert(date);

Якщо значення не задане, використовуються поточні дата та час. Основні функції JavaScript для роботи з датою представлені у таблиці 7.1.

Таблиця 7.1

Методи роботи з датою та часом

Метод	Опис	Значення, що повертається
getDate()	Вертає число місяця від 1 до 31	1-31
getDay()	Вертає день тижня	0-6
getMonth()	Вертає місяць	0-11
getFullYear()	Вєртає рік	повний рік
getHours()	Вертає години	0-23
getMinutes()	Вертає хвилини	0-59
getSeconds()	Вертає секунди	0-59
getTime()	Повертає timestamp для дати – кількість мілісекунд, що пройшли з 1 січня 1970 UTC+0.	
getTimezoneOffset()	Повертає різницю між UTC та місцевим часовим поясом, у хвилинах:	

Наступні методи дозволяють встановити дату/часові компоненти:

setFullYear(year, [month], [date])

setMonth(month, [date])

setDate(date)

setHours(hour, [min], [sec], [ms])

setMinutes(min, [sec], [ms])

setSeconds(sec, [ms])

setMilliseconds(ms)

setTime(milliseconds) (встановлює всю дату в мілісекундах з 01.01.1970 UTC)

Кожен з них, крім secTime() має UTC-аналог, наприклад: setUTCHours().

Як ми бачимо, деякі методи можуть встановити кілька компонентів відразу, наприклад setHours. Компоненти дати/часу, які не згадуються, – не модифікуються.

Наприклад:

let today = new Date();

today.setHours(0);

alert(today); // ще сьогодні, але година змінюється на 0

today.setHours(0, 0, 0, 0);

alert(today); // ще сьогодні, зараз рівно 00:00:00.

Автокорекція – це дуже зручна особливість об'єктів Date. Ми можемо встановити данні за межами діапазону, і вони будуть автоматично налаштувати себе.

Наприклад:

let date = new Date(2013, 0, 32); // 32 січня 2013 ?!?

alert(date); // ...це 1 лютого 2013!

Компоненти поза межами діапазону розподіляються автоматично.

Скажімо, нам потрібно збільшити дату "28 лютого 2016" на 2 дні. Це може бути "2 березня" або "1 березня" у випадку високосного року. Нам не потрібно думати про це. Просто додайте 2 дні. Об'єкт Date зробить все інше:

let date = new Date(2016, 1, 28);

date.setDate(date.getDate() + 2);

alert(date); // 1 березня 2016

Коли Date об'єкт перетворюється на номер, він стає timestamp так само, як date.getTime():

let date = new Date();

alert(+date); // кількість мілісекунд, так само, як date.getTime()

Важливий побічний ефект: дати можуть відніматися, результатом є їх різниця в мілісекундах.

Це можна використовувати для вимірювання часу:

let start = new Date(); // початок вимірювання часу

// виконується робота

```
for (let i = 0; i < 100000; i++) {
  let doSomething = i * i * i;
}
let end = new Date(); // кінець вимірювання часу
alert( `Цикл зайняв ${end - start} мс` );
```

Існує спеціальний метод Date.now(), що повертає поточний timestamp.

Це семантично еквівалентно до new Date().getTime(), але не створює проміжного об’єкта Date. Так що цей підхід працює швидше і не навантажує збирач сміття.

Виклик Date.parse(str) аналізує рядок у заданому форматі та повертає timestamp (кількість мілісекунд з 1 січня 1970 UTC+0). Якщо формат недійсний, повертає NaN.

Наприклад:

```
let ms = Date.parse('2012-01-26T13:51:50.417-07:00');
alert(ms); // 1327611110417  (timestamp)
```

Ми можемо миттєво створити об’єкт за допомогою new Date з timestamp:

```
let date = new Date( Date.parse('2012-01-26T13:51:50.417-07:00') );
alert(date);
```

7.11. Робота з функціями

Існує спеціальний вбудований метод функції func.call(context, …args), що дозволяє викликати функцію явно задаючи їй this.

Синтаксис:

```
func.call(context, arg1, arg2, ...)
```

Вона викликає func, використовуючи перший аргумент як this, а наступний – як аргументи.

Простіше кажучи, ці два виклики майже однакові:

```
func(1, 2, 3);
func.call(obj, 1, 2, 3)
```

Обидва вони викликаюсь func з аргументами 1, 2 і 3. Єдина відмінність полягає в тому, що func.call також встановлює this рівним obj.

Як приклад, у коді нижче ми викликаємо sayHi в контексті різних об’єктів: sayHi.call(user) викликає sayHi, передаючи this=user, а на наступних рядках встановлюється this=admin:

```
function sayHi() {
  alert(this.name);
```

```
}
let user = { name: "Іван" };
let admin = { name: "Адмін" };
// використовуйте call, щоб передати різні об’єкти як "this"
sayHi.call( user ); // Іван
sayHi.call( admin ); // Адмін
```

І тут ми використовуємо call, щоб викликати say з даним контекстом і фразою:

```
function say(phrase) {
  alert(this.name + ': ' + phrase);
}
let user = { name: "Іван" };
// користувач стає this, і "Привіт" стає першим аргументом
say.call( user, "Привіт" ); // Іван: Привіт
```

Замість func.call(this, ...arguments) ми могли б використовувати func.apply(this, arguments).

Синтаксис вбудованого методу func.apply:

```
func.apply(context, args)
```

Він запускає func, встановлюючи this = context і використовує псевдо-масив args як список аргументів.

Єдина різниця синтаксису між call та apply в тому, що call очікує список аргументів, в той час як apply приймає псевдо-масив з ними.

Отже, ці два виклики майже еквівалентні:

```
func.call(context, ...args);
func.apply(context, args);
```

Вони виконують той самий виклик func з даним контекстом та аргументами.

Є тільки тонка різниця щодо args:

Оператор розширення ... дозволяє передати ітерований args як список до call.

apply приймає лише псевдо-масив args.

… і для об’єктів, які є як ітерованими, так і псевдо-масивами, а так само як і справжніми масивами, ми можемо використовувати будь-який з цих методів, але apply, мабуть, буде швидше, тому що більшість рушіїв JavaScript внутрішньо оптимізують його краще.

Передача всіх аргументів разом з контекстом до іншої функції називається переадресація виклику.

Це найпростіша її форма:

```
let wrapper = function() {
  return func.apply(this, arguments);
};
```

Коли зовнішній код викликає такий wrapper, це не відрізняється від виклику оригінальної функції func.

Тепер давайте зробимо ще одне незначне поліпшення функції хешування:

```
function hash(args) {
  return args[0] + ',' + args[1];
}
```

Зараз вона працює лише на двох аргументах. Було б краще, якби вона могла зкріпити будь-яку кількість args.

Звичайним рішенням буде використати arr.join метод:

```
function hash(args) {
  return args.join();
}
```

…На жаль, це не буде працювати. Тому що ми викликаємо hash(arguments), а об’єкт arguments є як ітерованим, так і псевдо-масивом, але не справжнім масивом.

Отже, виклик join на цьому об’єкті буде призводити до помилки, що ми бачимо нижче:

```
function hash() {
  alert( arguments.join() ); // Помилка: arguments.join не є функцією
}
hash(1, 2);
```

Тим не менш, є простий спосіб використи з’єднання масиву:

```
function hash() {
  alert( [].join.call(arguments) ); // 1,2
}
hash(1, 2);
```

Це називається запозичення методу.

Ми беремо (запозичуємо) метод приєднання від звичайного масиву ([].join) і використовуємо [].join.call щоб запустити його в контексті arguments.

Функції надають нам вбудований метод bind, що дозволяє зберегти правильний this.

Основний синтаксис:

```
// більш складний синтаксис буде трохи пізніше
let boundFunc = func.bind(context);
```

Результатом func.bind(context) буде певний об’єкт, який може бути викликаний як

функція та передає виклику func встановлений контекст this=context.

Іншими словами, виклик boundFunc це як виклик func з виправленим this.

Наприклад, тут funcUser передає виклик func з this=user:

```
let user = {
  firstName: "Іван"
};
function func() {
  alert(this.firstName);
}
let funcUser = func.bind(user);
funcUser(); // Іван
```

Тут func.bind(user) як "прив'язаний варіант" функції func, з виправленим this=user.

Всі аргументи передаються початковій функції func "як є", наприклад:

```
let user = {
  firstName: "Іван"
};
function func(phrase) {
  alert(phrase + ', ' + this.firstName);
}
// прив'язка до user
let funcUser = func.bind(user);
funcUser("Привіт"); // Привіт, Іван (переданий аргумент "Привіт" та this=user)
```

Тепер спробуємо з методом об'єкту:

```
let user = {
  firstName: "Іван",
  sayHi() {
    alert(`Привіт, ${this.firstName}!`);
  }
};
let sayHi = user.sayHi.bind(user); // (*)
// можемо викликати без об'єкту
sayHi(); // Привіт, Іван!
setTimeout(sayHi, 1000); // Привіт, Іван!
// навіть якщо значення user зміниться впродовж 1 секунди
// sayHi використовує прив'язане значення, яке посилається на старий об'єкт user
```

```
user = {
  sayHi() { alert("Інший user в setTimeout!"); }
};
```

В строчці (*) ми взяли метод user.sayHi та прив’язали його до user. sayHi є “прив’язаною” функцією, що може бути викликана окремо або передана до setTimeout – не важливо, контекст буде правильний.

В цьому прикладі ми можемо бачити що аргументи передані “як є”, тільки this виправлено за допомогою bind:

```
let user = {
  firstName: "Іван",
  say(phrase) {
    alert(`${phrase}, ${this.firstName}!`);
  }
};
let say = user.say.bind(user);
say("Привіт"); // Привіт, Іван! (Аргумент "Привіт" переданий функції say)
say("Бувай"); // Бувай, Іван! ("Бувай" передане функції say)
```

Ми можемо прив’язати не тільки this, а також аргументи. Це робиться рідко, проте може бути інколи дуже зручним.

Повний синтаксис bind:

```
let bound = func.bind(context, [arg1], [arg2], ...);
```

Це дозволяє прив’язати context як this та початкові аргументи функції.

Наприклад, ми маємо функцію множення mul(a, b):

```
function mul(a, b) {
  return a * b;
}
```

Використаємо bind щоб створити функцію double на її основі:

```
function mul(a, b) {
  return a * b;
}
let double = mul.bind(null, 2);
alert( double(3) ); // = mul(2, 3) = 6
alert( double(4) ); // = mul(2, 4) = 8
alert( double(5) ); // = mul(2, 5) = 10
```

Виклик mul.bind(null, 2) створює нову функцію double що передає виклик mul,

встановлючи null як контекст та 2 як перший аргумент. Подальші аргументи передаються “як є”.

Це називається часткове застосування – ми створюємо нову функцію виправляючи деякі параметри існуючої.

Зверніть увагу, що ми не використовували this в цьому прикладі. Проте bind вимагає цього, тому ми маємо передати щось як заглушку – null.

Перевагою цього є те, що ми можемо створити незалежну функцію з читабельним ім’ям (double). Ми можемо використовувати її та не передавати перший аргумент кожен раз, оскільки це замість нас виконує bind.

7.12. Регулярні вирази

Регулярні вирази - це шаблони, які використовуються для зіставлення послідовностей символів у рядках.

Регулярний вираз можна створити двома способами:

1. Використовуючи літерал регулярного виразу, наприклад:

```
let re = /ab+c/;
```

2. Викликаючи функцію конструктор об'єкта RegExp, наприклад:

```
var re = new RegExp("ab+c");
```

Щоб визначити, чи відповідає регулярне вираз рядку, в об'єкті RegExp визначено метод test(). Цей метод повертає true, якщо рядок відповідає регулярному виразу і false, якщо не відповідає.

```
const initialText = "hello world!";
const exp = /hello/;
const result = exp.test(initialText);
console.log(result); // true
const initialText2 = "beautifull wheather";
const result2 = exp.test(initialText2);
console.log(result2); // false - у рядку "beautifull wheather" немає "hello"
```

Аналогічно працює метод exec - він також перевіряє, чи рядок відповідає регулярному виразу, тільки тепер даний метод повертає ту частину рядка, яка відповідає виразу. Якщо відповідності немає, то повертається значення null.

```
let initialText = "hello world!";
let exp = /hello/;
let result = exp.exec(initialText);
```

```
console.log(result); // hello
initialText = "beautifull wheather";
result = exp.exec(initialText);
console.log(result); // null
```

Регулярний вираз не обов'язково складається зі звичайних рядків, але може включати спеціальні елементи синтаксису регулярних виразів. Один із таких елементів представляють групи символів, укладені у квадратні дужки. Наприклад:

```
let initialText = "обороноспроможність";
let exp = /[абв]/;
let result = exp.test(initialText);
console.log(result); // true
initialText = "місто";
result = exp.test(initialText);
console.log(result); // false
```

Вираз [абв] вказує на те, що рядок повинен мати одну із трьох літер.

Якщо нам треба визначити наявність у рядку літерних символів з певного діапазону, можна відразу задати цей діапазон:

```
let initialText = "обороноспроможність";
let exp = /[а-я]/;
let result = exp.test(initialText);
console.log(result); // true
initialText = "3di0789";
result = exp.test(initialText);
console.log(result); // false
```

У цьому випадку рядок повинен містити хоча б один символ діапазону а-я.

Якщо, навпаки, не треба, щоб рядок мав лише певні символи, то необхідно у квадратних дужках перед перерахуванням символів ставити знак ^:

```
let initialText = "бороноити";
let exp = /[^а-я]/;
let result = exp.test(initialText);
console.log(result); // false
initialText = "3di0789";
exp = /[^0-9]/;
result = exp.test(initialText);
console.log(result); // true
```

У першому випадку рядок не повинен мати лише символи з діапазону а-я, але оскільки рядок "боронити" складається тільки із символів з цього діапазону, то метод test() повертає false, тобто регулярне вираз не відповідає стоку.

У другому випадку ("3di0789") рядок не повинен складатися лише з цифрових символів. Але так як у рядку також є літери, то рядок відповідає регулярному виразу, тому метод test повертає значення true.

За потреби ми можемо збирати комбінації виразів:

```
const initialText = "дома";
const exp = /[дт]про[нм]/;
const result = exp.test(initialText);
console.log(result); // true
```

Вираз [дт]о[нм] вказує ті рядки, які можуть містити підрядки " будинок " , " том " , " дон " , " тон " .

Прапори дозволяють настроїти поведінку регулярних виразів. Кожен прапор є окремим символом, який ставиться в кінці регулярного виразу. У JavaScript використовуються такі прапори:

- прапор global дозволяє знайти всі підрядки, які відповідають регулярному виразу. За замовчуванням при пошуку підрядок регулярне вираз вибирає перший підрядок, що попався, з рядка, який відповідає виразу. Хоча у рядку може бути безліч підрядків, які також відповідають виразу. Для цього застосовується цей прапор у вигляді символу g у виразах;
- прапор ignoreCase дозволяє знайти підстоки, які відповідають регулярному виразу, незалежно від регістру символів у рядку. Для цього у регулярних виразах застосовується символ i;
- прапор multiline дозволяє знайти підрядки, які відповідають регулярному виразу в багаторядковому тексті. Для цього у регулярних виразах застосовується символ m;
- прапор dotAll дозволяє зіставити точку в регулярному виразі з будь-яким символом тексту, у тому числі з роздільником рядка. Для цього в регулярних виразах застосовується символ s.

Розглянемо наступний приклад:

```
const initialText = "Hello World";
const exp = /world/;
const result = exp.test(initialText); // false
```

Тут збігу рядка з виразом немає, оскільки "World" відрізняється від "world" за регістром. У цьому випадку треба змінити регулярний вираз, додавши в нього властивість ignoreCase:

```
const exp = /world/i;
```

Ну і також ми можемо використовувати відразу кілька властивостей:

```
const exp = /world/ig;
```

Прапор s або dotAll дозволяє зіставити символ. (точка) з будь-яким символом, у тому числі з роздільником рядка. Наприклад, візьмемо наступний приклад:

```
const text = "hello\nworld";
const exp = /hello world/;
const result = exp.test (text); // false
console.log(result); // false
```

Тут у рядку "hello\nworld" слова "hello" та "hello" розділені переносом рядка (наприклад, ми маємо справу з багаторядковим текстом). Однак, наприклад, ми хочемо, щоб JavaScript не враховував перенесення рядка і щоб цей текст відповідав регулярному виразу /hello world/. У цьому випадку ми можемо застосувати прапор s:

```
const text = "hello\nworld";
const exp = /hello.world/s;
const result = exp.test (text); // true
console.log(result); // true
```

У виразі /hello.world/s точка означає довільний символ. Однак без прапора s цей вираз не відповідатиме багаторядковому тексту.

Для пошуку всіх відповідностей у рядку застосовується метод match():

```
const initialText = "Він прийшов додому та зробив домашню роботу";
const exp = /до[а-я] * / gi;
const result = initialText.match(exp);
result.forEach(value => console.log(value));
```

Символ зірочки вказує на можливість наявності після рядка "до" невизначеної кількості символів від а до я. У результаті масиві result виявляться такі слова: додому, домашню.

Метод search знаходить індекс першого включення відповідності у рядку:

```
const initialText = "hello world";
const exp = /wor/;
const result = initialText.search(exp);
console.log(result); // 6
```

Метод replace дозволяє замінити всі відповідності регулярному виразу певним рядком:

```
let menu = "Сніданок: каша, чай. Обід: суп, чай. Вечеря: салат, чай.";
const exp=/чай/gi;
menu = menu.replace(exp, "кава");
console.log(menu); //Сніданок: каша, кава. Обід: суп, кава. Вечеря: салат, кава.
```

Регулярні вирази також можуть використовувати метасимволи - символи, які мають певний зміст:

- \d: відповідає будь-якій цифрі від 0 до 9;
- \D: відповідає будь-якому символу, який не є цифрою;
- \w: відповідає будь-якій літері, цифрі або символу підкреслення (діапазони A–Z, a–z, 0–9);
- \W: відповідає будь-якому символу, який не є буквою, цифрою або символом підкреслення (тобто не знаходиться в наступних діапазонах A–Z, a–z, 0–9);
- \s: відповідає пропуску;
- \S: відповідає будь-якому символу, який не є пробілом;
- .: відповідає будь-якому символу.

Тут треба зауважити, що метасимвол \w застосовується тільки для букв латинського алфавіту.

Так, стандартний формат номера телефону +1-234-567-8901 відповідає регулярному виразу \d-d-d-d-d-d. Наприклад, замінимо числа номера нулями:

```
let phoneNumber = "+1-234-567-8901";
let myExp = /d-d-d-d-d-d;
phoneNumber = phoneNumber.replace(myExp, "00000000000");
alert(phoneNumber);
```

Крім вище розглянутих елементів регулярних виразів, є ще одна група комбінацій, яка вказує, як символи в рядку повторюватимуться. Такі комбінації ще називають модифікаторами:

- {n}: відповідає n-й кількості повторень попереднього символу. Наприклад, h{3} відповідає підрядку "hhh";
- {n,}: відповідає n та більше кількості повторень попереднього символу. Наприклад, h{3,} відповідає підрядкам "hhh", "hhhh", "hhhhh" і т.д.;
- {n,m}: відповідає від n до m повторень попереднього символу. Наприклад, h{2, 4} відповідає підрядкам "hh", "hhh", "hhhh";
- ?: відповідає одному входженню попереднього символу в підрядок або його відсутності в підрядку. Наприклад, /h?ome/ відповідає підрядкам "home" і "ome";
- +: відповідає одному і більше повторень попереднього символу;
- *: відповідає будь-якій кількості повторень або відсутності попереднього символу;
- ^: відповідає початку рядка. Наприклад, ^h відповідає рядку "home", але з "ohma", оскільки h повинен представляти початок рядка;
- $: відповідає кінцю рядка. Наприклад, к$ відповідає рядку "будинок", тому що рядок

повинен закінчуватися на букву «к».

Наприклад, візьмемо номер того самого телефону. Йому відповідає регулярне вираз \d-d-d-d-d-d. Однак за допомогою вище розглянутих комбінацій ми його можемо спростити: \d-\d{3}-\d{3}-\d{4}

Також треба зазначити, що оскільки символи ?, +, * мають особливий сенс у регулярних виразах, те щоб їх використовувати у звичайному для них значенні (наприклад, нам треба замінити знак плюс у рядку на мінус), то ці символи треба екранувати за допомогою слєша:

```
let phoneNumber = "+1-234-567-8901";
let myExp = /\+\d-\d{3}-\d{3}-\d{4}/;
phoneNumber = phoneNumber.replace(myExp, "80000000000");
console.log(phoneNumber);
```

Окремо розглянемо застосування комбінації '\b', яка вказує на відповідність у межах слова. Наприклад, ми маємо наступний рядок: "Мови навчання: Java, JavaScript, C++". Згодом ми вирішили, що Java треба замінити на C#. Але звичайна заміна призведе також до заміни рядка "JavaScript" на "C#Script", що неприпустимо. І в цьому випадку ми можемо проводити заміну, якщо регулярний вираз відповідає всьому слову:

```
let initialText = "Мови навчання: Java, JavaScript, C++";
let exp = /\bJava\b/g;
let result = initialText.replace(exp, "C#");
console.log(result); // Мови навчання: C#, JavaScript, C++
```

Але при використанні '\b' треба враховувати, що у JavaScript відсутня повноцінна підтримка юнікоду, тому застосовувати '\b' ми зможемо лише до англомовних слів.

Для пошуку у рядку складніших відповідностей застосовуються групи. У регулярних виразах групи беруть у дужки. Наприклад, ми маємо наступний код html, який містить тег зображення: ''. І припустимо, нам треба отримати з цього коду шляхи до зображень:

```
let initialText = '<img src="picture.png" />';
let exp = /[a-z]+\.(png|jpg)/i;
let result = initialText.match(exp);
result.forEach(function(value, index, array){

    console.log(value);
})
```

Результат:

picture.png

png

Перша частина до дужок ([a-z]+\.) вказує на наявність у рядку від 1 і більше символів діапазону a-z, після яких йде точка. Оскільки точка є спеціальним символом у регулярних виразах, вона екранується слешем. А далі йде: (png | jpg). Ця група вказує, що після точки може використовуватися як "png", і "jpg".

Перевагою використання груп у регулярних виразах є те, що ми можемо отримати значення кожної окремої групи. Наприклад, як відомо, у різних країнах використовуються різні формати дат. Що, якщо ми хочемо отримуємо дату у форматі "рік-місяць-день" і хочемо перетворити її на якийсь інший формат? У цьому випадку для кожного окремого компонента ми можемо визначити свою групу:

```
const exp = /(\d{4})-(\d{2})-(\d{2})/;
const result = exp.exec("2021-09-06");
console.log(result[0]); // 2021-09-06
console.log(result[1]); // 2021
console.log(result[2]); // 09
console.log(result[3]); // 06
console.log(`${result[3]}.${result[2]}.${result[1]}`); // 06.09.2021
```

Тут застосовується регулярне вираз "/(\d{4})-(\d{2})-(\d{2})/", де визначено три групи: перша група (\d{4}) відповідає числу із чотирьох цифр; друга група (\d{2}) відповідає числу з двох цифр; третя група аналогічна до другої.

Отриманий результат представляє масив, де перший елемент (з індексом 0) завжди представляє підрядок, що збігся з регулярним виразом. Усі наступні елементи цього масиву представляють групи. Тобто перша група має індекс 1, друга – індекс 2 і так далі.

JavaScript дозволяє призначити кожній групі у регулярних виразах певне ім'я. З допомогою цього імені можна отримати значення, яке відповідає цій групі. Для встановлення імені групи всередині дужок, які визначають групу, ставиться знак питання, після якого у кутових дужках йде назва групи:

```
(?<назва_групи> ... )
```

Розглянемо наступний приклад:

```
const exp = /(?<year>\d{4})-(?<month>\d{2})-(?<day>\d{2})/u;
const result = exp.exec("2021-09-06");
console.log(result.groups); // {year: "2021", month: "09", day: "06"}
console.log(result.groups.year); // 2021
console.log(result.groups.month); // 09
```

console.log(result.groups.day); // 06

Тут регулярне вираз визначає три групи. Перша група називається "year", друга - "month" та третя "day". Отримавши результат, ми можемо обійтися до кожної групи через властивість groups. Це властивість представляє об'єкт, у якому властивості називаються як і, як і групи, і містять значення кожної групи:

console.log(result.groups); // {year: "2021", month: "09", day: "06"}

Відповідно, за допомогою назви групи ми можемо отримати значення для певної групи.

Твердження або assertiobs дозволяють отримати підрядок, який відповідає регулярному виразу і який передує або, навпаки, не передує певному виразу.

Позитивне твердження (коли підрядок повинен передувати іншим підрядком) визначається за допомогою виразу:

(?<=...)

Після знака дорівнює = йде вираз, яким має передуватися підрядок.

Негативне твердження (коли підрядок не повинен передуватися іншим підрядком) визначається за допомогою виразу:

(?<!...)

Після знака оклику ! йде вираз, яким НЕ повинен передувати підрядок.

Візьмемо просте завдання. Припустимо, у нас є деяка інформація з якоюсь сумою. Але ця сума може бути оределена в доларах, євро, рублях і так далі. Щось на кшталт:

```
const text1 = "All costs: $10.53";
const text2 = "All costs: €10.53";
const exp = /\d+(\.\d*)?/;
const result1 = exp.exec(text1);
console.log(result1[0]); // 10.53
const result2 = exp.exec(text2);
console.log(result2[0]); // 10.53
```

Тут ми бачимо, що і сума в доларах (), і сума в євро відповідає нашому регулярному виразу. Але що якщо ми хочемо отримати тільки суму в доларах. Для цього застосуємо позитивне твердження:

```
const text1 = "All costs: $10.53";
const text2 = "All costs: €10.53";
const exp = /(?<=\$)\d+(\.\d*)?/;
const result1 = exp.exec(text1);
console.log(result1); // ["10.53", ".53", index: 12, input: "All costs: $10.53", groups: undefined]
const result2 = exp.exec(text2);
```

```
console.log(result2); // null
```

Група (?<=\$) вказує, що перед рядком має йти знак долара $. Якщо його немає, то метод exec() не знайде відповідності і поверне null.

7.13. Обробка помилок

У процесі роботи прогами можуть бути різні помилки, які порушують звичний хід програми і навіть змушують її перервати виконання. Мова JavaScript має інструменти для усунення таких ситуацій.

Найпростіша ситуація - виклик функції, якої немає:

```
callSomeFunc();
console.log("Інші інструкції");
```

Тут викликається функція callSomeFunc(), яка ніде не визначена. Відповідно, при виклику цієї функції ми зіткнемося з помилкою:

```
Uncaught ReferenceError: callSomeFunc is not defined
```

Решта інструкцій, які йдуть після рядка, на якому виникла помилка, не виконується. Програма закінчує свою роботу.

Ситуація може здатися штучною, оскільки ми знаємо, що функція callSomeFunc ніде не визначена. Однак коли ми маємо справу з великою програмою, різні шматки якої визначали різні розробники, складніше контолювати код. І таких ситуацій може бути багато. Якісь ми можемо самі відстежити та попередити, а якісь ні.

Для обробки подібних ситуацій JavaScript надає конструкцію try...catch...finally, яка має таке формальне визначення:

```
try {
  інструкції блоку try
}
catch (error) {
  інструкції блоку catch
}
finally {
  інструкції блоку finally
}
```

Після оператора try визначається блок коду. Цей блок містить інструкції, при виконанні яких може виникнути потенційна помилка.

Потім іде оператор catch. Після цього оператора в круглих дужках вказується назва

об'єкта, який міститиме інформацію про помилку. І далі йде блок catch. Цей блок виконується лише при виникненні помилки у блоці try.

Після блоку catch йде оператор finally зі своїм блоком інструкцій. Цей блок виконується в кінці після блоку try і catch незалежно від того, виникла помилка чи ні.

Варто зазначити, що лише блок try є обов'язковим. А один із решти блоків - catch або finally ми можемо опустити. Однак один з цих блоків (не важливо catch або finally) обов'язково має бути присутнім. Тобто ми можемо використовувати такі варіанти цієї конструкції:

try...catch

try...finally

try...catch...finally

Наприклад, обробимо за допомогою цієї конструкції попередню ситуацію з неіснуючою функцією:

```
try{
    callSomeFunc();
    console.log("Кінець блоку try");
}
catch {
    console.log("Виникла помилка!");
}
console.log("Інші інструкції");
```

Отже, спочатку виконується блок try. Однак при виконанні першої інструкції - виклику функції callSomeFunc() виникає помилка. Це призведе до того, що всі наступні інструкції в блоці try не виконуватимуться. А керування перейде до блоку catch. У цьому блоці відображається повідомлення, що виникла помилка. Після виконання блоку catch виконуються інші вказівки програми. Таким чином, програма не перериває свою роботу при виникненні помилки та продовжує свою роботу. І в даному випадку консольне виведення буде наступним:

Виникла помилка!

Інші інструкції

Розглянемо інший приклад:

```
function callSomeFunc(){console.log("Функція callSomeFunc");}
try{
    callSomeFunc();
    console.log("Кінець блоку try");
}
catch (error) {
```

```
    console.log("Виникла помилка!");
}
console.log("Інші інструкції");
```

Тепер функцію callSomeFunc() визначено в прогамі, тому при виклику функції помилки не відбудеться, і блок try допрацює до кінця. А блок catch за відсутності помилки не буде висохнути. І консольне виведення буде наступним:

```
Функція callSomeFunc
Кінець блоку try
Інші інструкції
```

Як параметр в блок catch передається об'єкт з інформацією про помилку:

```
try{
    callSomeFunc();
    console.log("Кінець блоку try");
}
catch (error) {
    console.log("Виникла помилка!");
    console.log(error);
}
```

У цьому випадку ми отримаємо консольне виведення на кшталт наступного:

```
Виникла помилка!
ReferenceError: callSomeFunc is not defined
```

Блок finally

Конструкція try може також містити блок finally. Ми можемо використовувати цей блок разом із блоком catch або замість нього. Блок finally виконується незалежно, відбулася помилка чи ні. Наприклад:

```
try{
    callSomeFunc();
    console.log("Кінець блоку try");
}
catch {
    console.log("Відбулася помилка");
}
finally {
    console.log("Блок finally")
}
```

console.log("Інші інструкції");

Консольне виведення:

Виникла помилка

Блок finally

Інші інструкції

Інтерпретатор JavaScript генерує помилки для низки ситуацій, наприклад, при виклику неіснуючої функції, повторному присвоєння константі значення і т.д. Але при необхідності ми самі можемо генерувати помилки та визначити умови, коли генеруватиметься помилка.

Для створення винятку застосовується оператор throw, після якого вказується інформація про помилку:

throw інформація_про помилку;

Інформація про помилку може надавати будь-який об'єкт.

Так, згенеруємо виняток при передачі в конструктор Person негативного значення властивості age:

```
class Person{

    constructor(name, age) {
        if(age < 0) throw "Вік має бути позитивним";
        this.name = name;
        this.age = age;
    }
    print(){ console.log(`Name: ${this.name} Age: ${this.age}`);}
}
const tom = new Person("Tom", -123); // Uncaught Вік має бути позитивним
tom.print();
```

У результаті при виклику конструктора Person буде згенеровано виняток і програма завершиться помилкою. А на консолі браузера ми побачимо інформацію про помилку, вказану після оператора throw.

Як і в загальному випадку, ми можемо обробити цю помилку за допомогою блоку try...catch:

```
class Person{

    constructor(name, age) {
        if(age < 0) throw "Вік має бути позитивним";
        this.name = name;
```

```
        this.age = age;
    }
    print(){ console.log(`Name: ${this.name} Age: ${this.age}`);}
}

try{
    const tom = new Person("Tom", -123); // Uncaught Вік має бути позитивним
    tom.print();
}
catch (error) {
    console.log("Відбулася помилка");
    console.log(error); // Вік має бути позитивним
}
```

У блоці catch ми можемо отримати інформацію про помилку, яка представляє об'єкт. Усі помилки, які генеруються інтерретатором JavaScript, надають об'єкт типу Error, який має низку властивостей:

message: повідомлення про помилку

name: тип помилки

Варто зазначити, що окремі браузери підтримують ще ряд властивостей, але їхня поведінка в залежності від браузера може відрізнятися:

fileName: назва файлу з кодом JavaScript, де сталася помилка

lineNumber: рядок у файлі, де сталася помилка

columnNumber: стовпець у рядку, де сталася помилка

stack: стек помилки

Отримаємо дані помилки, наприклад, при виклику неіснуючої функції:

```
try{
    callSomeFunc();
}
catch (error) {
    console.log("Тип помилки:", error.name);
    console.log("Помилка:", error.message);
}
```

Консольне виведення:

Тип помилки: ReferenceError

Помилка: callSomeFunc is not defined

Вище ми розглянули, що помилка, що генерується інтерпретатором, представляє тип Error, проте при виклику неіснуючої функції генерується помилка типу ReferenceError. Справа в тому, що тип Error представляє загальний тип помилок. У той же час, є конкретні типи помилок для певних ситуацій:

- EvalError: представляє помилку, що генерується під час виконання глобальної функції eval();
- RangeError: помилка генерується, якщо параметр або змінна, є числом, яке знаходиться поза деяким допустимим діапазоном;
- ReferenceError: помилка генерується при зверненні до неіснуючого посилання;
- SyntaxError: представляє помилку синтаксису;
- TypeError: помилка генерується, якщо значення змінної або параметра представляють некоректний тип або спробу змінити значення, яке не можна змінювати;
- URIError: помилка генерується при передачі функцій encodeURI() та decodeURI() некоректних значень;
- AggregateError: надає помилку, яка об'єднує кілька помилок, що виникли.

Наприклад, при спробі привласнити константі вдруге значення, генерується помилка TypeError:

```
try{
    const num = 9;
    num = 7;
}
catch (error) {
    console.log(error.name); // TypeError
    console.log(error.message); // Assignment to constant variable.
}
```

Так, згенеруємо кілька типів помилок:

```
class Person{

    constructor(pName, pAge) {

        const age = parseInt(pAge);
        if(isNaN(age)) throw new TypeError("Вік повинен представляти число");
        if(age < 0 || age > 120) throw new RangeError("Вік має бути більше 0 і менше 120");
        this.name = pName;
        this.age = age;
```

```
    }
    print(){ console.log(`Name: ${this.name} Age: ${this.age}`);}
}
```

Оскільки для віку можна редагувати не тільки число, а й взагалі будь-яке значення, то спочатку ми намагаємося перетворити це значення на число за допомогою функції parseInt():

```
const age = parseInt(pAge);
if(isNaN(age)) throw new TypeError("Вік повинен представляти число");
```

Далі за допомогою функції isNaN(age) перевіряємо, чи є одержане число числом. Якщо age - НЕ число, то ця функція повертає true. Тому генерується помилка типу TypeError.

Потім перевіряємо, що отримане число входить у допустимий діапазон. Якщо ж не входить, генеруємо помилку типу RangeError:

```
if(age < 0 || age > 120) throw new RangeError("Вік має бути більше 0 і менше 120");
```

Перевіримо генерацію винятків:

```
try{
    const tom = new Person("Tom", -45);
}
catch (error) {
    console.log(error.message); // Вік має бути більше 0 і менше 120
}

try{
    const bob = new Person("Bob", "bla bla");
}
catch (error) {
    console.log(error.message); // Вік повинен бути числом
}

try{
    const sam = New Person ("Sam", 23);
    sam.print(); // Name: Sam Age: 23
}
catch (error) {
    console.log(error.message);
}
```

Консольне виведення:

Вік має бути більше 0 і менше 120

Вік повинен бути числом

Name: Sam Age: 23

При виконанні одного й того ж коду можуть генеруватися помилки різних типів. І іноді буває необхідно розмежувати обробку різних типів. У цьому випадку ми можемо перевіряти тип помилки. Наприклад, приклад із класом Person:

```
class Person{

    constructor(pName, pAge) {

        const age = parseInt(pAge);
        if(isNaN(age)) throw new TypeError("Вік повинен представляти число");
        if(age < 0 || age > 120) throw new RangeError("Вік має бути більше 0 і менше 120");
        this.name = pName;
        this.age = age;
    }
    print(){ console.log(`Name: ${this.name} Age: ${this.age}`);}
}

try{
    const tom = new Person("Tom", -45);
    const bob = new Person("Bob", "bla bla");
}
catch (error) {
    if (error instanceof TypeError) {
        console.log("Некоректний тип даних.");
    } else if (error instanceof RangeError) {
        console.log("Неприпустиме значення");
    }
    console.log(error.message);
}
```

Ми не обмежені лише вбудованими типами помилок і за потреби можемо створювати свої типи помилок, призначені для якихось конкретних ситуацій. Наприклад:

```
class PersonError extends Error {
  constructor(value, ...params) {
```

```
    // Інші параметри передаємо в конструктор базового класу
    super(...params)
    this.name = "PersonError"
    this.argument = value;
  }
}

class Person{

    constructor(pName, pAge) {

        const age = parseInt(pAge);
        if(isNaN(age)) throw new PersonError(pAge, "Вік повинен представляти число");
        if(age < 0 || age > 120) throw new PersonError(pAge, "Вік має бути більше 0 і менше 120");
        this.name = pName;
        this.age = age;
    }
    print(){ console.log(`Name: ${this.name} Age: ${this.age}`);}
}

try{
    // const tom = new Person ("Tom", -45);
    const bob = new Person("Bob", "bla bla");
}
catch (error) {
    if (error instanceof PersonError) {
        console.log("Помилка типу Person. Некоректне значення:", error.argument);
    }
    console.log(error.message);
}
```

Консольне виведення:

Помилка Person. Некоректне значення: bla bla

Вік повинен бути числом

Для представлення помилки класу Person тут визначено тип PersonError, який

успадковується від класу Error:

```
class PersonError extends Error {
  constructor(value, ...params) {
    // Інші параметри передаємо в конструктор базового класу
    super(...params)
    this.name = "PersonError"
    this.argument = value;
  }
}
```

У конструкторі ми визначаємо додаткову властивість – argument. Воно зберігатиме значення, що викликало помилку. За допомогою параметра value конструктора отримуємо це значення. Крім того, перевизначаємо ім'я типу за допомогою властивості this.name.

У класі Person використовуємо цей тип, передаючи конструктор PersonError відповідні значення:

```
if(isNaN(age)) throw new PersonError(pAge, "Вік повинен представляти число");
if(age < 0 || age > 120) throw new PersonError(pAge, "Вік має бути більше 0 і менше 120");
```

Потім при обробці виключення ми можемо перевірити тип, і якщо він представляє клас PersonError, то звернутися до його властивості:

```
catch (error) {
   if (error instanceof PersonError) {
      console.log("Помилка типу Person. Некоректне значення:", error.argument);
   }
   console.log(error.message);
}
```

Контрольні питання

1. Які способи створення об’єктів Ви знаєте?
2. Які функції роботи зі строками Ви знаєте?
3. Як об’явити масив?
4. Які функції роботи з масивами Ви знаєте?
5. Що таке регулярний вираз?
6. Які функції роботи з регулярними виразами Ви знаєте?
7. Яка конструкція для обробки помилок?

ВИСНОВКИ

Інтернет-технології – це автоматизоване середовище отримання, обробки, зберігання, передачі й використання знань у вигляді інформації, що реалізується в мережі Інтернет. Розрізняють логічну і фізичну моделі Інтернету. Під логічної, насамперед, розуміють Всесвітню павутину (World Wide Web), а під фізичною - комп'ютери, сервери і засоби передачі даних між ними. Програмне забезпечення складається із загального (системного) і прикладного (спеціального) програмного забезпечення.

Протокол — домовленості про сигнали, якими обмінюються комп'ютери під час встановлення зв'язку між собою і приймання чи передавання інформації.

Найбільш популярним пакетом програмних засобів є протокол TCP/IP.

Поза сумнівом, зараз найпоширенішим інтерфейсом для Інтернету є Web, заснований на стандартній мові розмітки гіпертексту HTML (Hypertext Markup Language) і протоколі передачі гіпертексту HTTP (Hypertext Transfer Protoco1). Найпростішим компонентом Web є HTML. Це проста мова форматування документів, що відображаються у Web-браузері. Найважливіше завдання браузера – це відображення документів у відповідності із HTML-тегами. Як і будь яка інша мова програмування, HTML передбачає деяку стандартну структуру побудови програми – у даному випадку html-документу.

CSS, на відміну від HTML використовує дещо інший алгоритм опису елементів сторінки. Лише один раз можна вказати властивості кожного елементу в текстовому файлі з розширенням .css (наприклад mystyle.css) та підключити цей файл до html-документу. В такому випадку браузер зчитує значення параметрів (розмір шрифта, колір та ін.) кожного тегу вже зі створеного css-файлу. Більш того, оскільки стилі описуються в окремому файлі, то його можна підключити до будь-якої кількості web-сторінок, що дозволить уникнути необхідності призначати властивості кожному конкретному об'єкту (тегу). Є ще одна перевага такого підходу: якщо необхідно змінити стиль оформлення будь-якого елементу усіх web-сторінок, достатньо виправити один або декілька рядків в одному файлі – файлі зі стилями.

Основна перевага Web-сторінок – це можливість реагувати на події, що відбуваються із елементами сторінки. Зміна змісту сторінки за допомогою сценаріїв при її появі робить сторінку динамічною. Для цього і призначена мова JavaScript.

СПИСОК ЛІТЕРАТУРИ

1. HTML5, CSS3 и JavaScript. Исчерпывающее руководство / Дженнифер Роббинс; [пер. с англ. М. А. Райтман]. — 4-е издание. — М. : Эксмо, 2014. — 528 с.
2. Гоше Х.Д. HTML5. Для профессионалов. - СПб.: Питер, 2013. - 496 с.
3. Джилленуотер З. Сила CSS3. Освой новейший стандарт веб-разработок! - СПб.: Питер, 2012. - 304 с.
4. Дронов В. А. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов. — СПб.: БХВ-Петербург, 2011. — 416 с.
5. Кингсли-Хью Э., Кингсли-Хью К. Java Script 1 .5 : Учебный курс: Пер. с англ.–2002, 272 с.
6. Кириченко А.В., Хрусталев А.А. HTML5+CSS3. Основы современного web-дизайна.-СПб.: "Наука и техника", 2018.-352 с.
7. Клименко Р. А. Веб-мастеринг на 100%. — СПб.: Питер, 2013. — 512 с.
8. Лабберс Питер, Олберс Брайан, Салим Фрэнк. HTML5 для профессионалов: мощные инструменты для разработки современных web-приложений.: Пер. с англ.-М.: И.Д. "Вильямс", 2011.–272 с.
9. Лоусон Б., Шарп Р. Изучаем HTML5. Библиотека специалиста.-СПб.: Питер, 2011.-272 с.
10. Матросов А. В., Сергеев А. О., Чаунин М. П. HTML 4.0.–СПб.: БХВ-Петербург, 2003.–672 с.
11. Мархвида И. В. Создание Web - страниц: HTML, CSS, JavaScript.–СПб.: „Питер”. 2002.–352 с.
12. Пилгрим М. Погружение в HTML5: перев. с англ. — СПб.: БХВ-Петербург, 2011. – 304 с.
13. Полонская Е.Л. Язык HTML. Самоучитель.–М. : Издательский дом "Вильямс", 2003.–320 с.
14. Резиг Джон, Фергюсон Расс, Пакстон Джон. JavaScript для профессионалов, 2-е изд.: Пер. с англ. М.: И.Д. "Вильямс", 2016.–240 с.
15. Фаррелл Б. Веб-компоненты в действии / пер. с англ. Д. А. Беликова. – М.: ДМК Пресс, 2020. – 462 с.
16. Фрэйн Б. HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств. 2-е изд. - СПб.: Питер, 2017. — 272 с.
17. Шмитт К., Симпсон К. HTML5. Рецепты программирования.–СПб.: Питер. 2012.–288 с.