Мова програмування Python

Модуль Matplotlib - наукова графіка

Установка в рамках наукових дистрибутивів

Наукові дистрибутиви мови Python хороші тим, що крім самого Python, встановлюють цілий універсал всіляких наукових утиліт. Наукова графіка далеко не єдиний інструмент, який ви можете використовувати, і matplotlib аж ніяк не єдина бібліотека, яка дозволяє створювати графіки. Практично всі дистрибутиви описані в інструкції по установці NumPy.

Лінійна діаграма

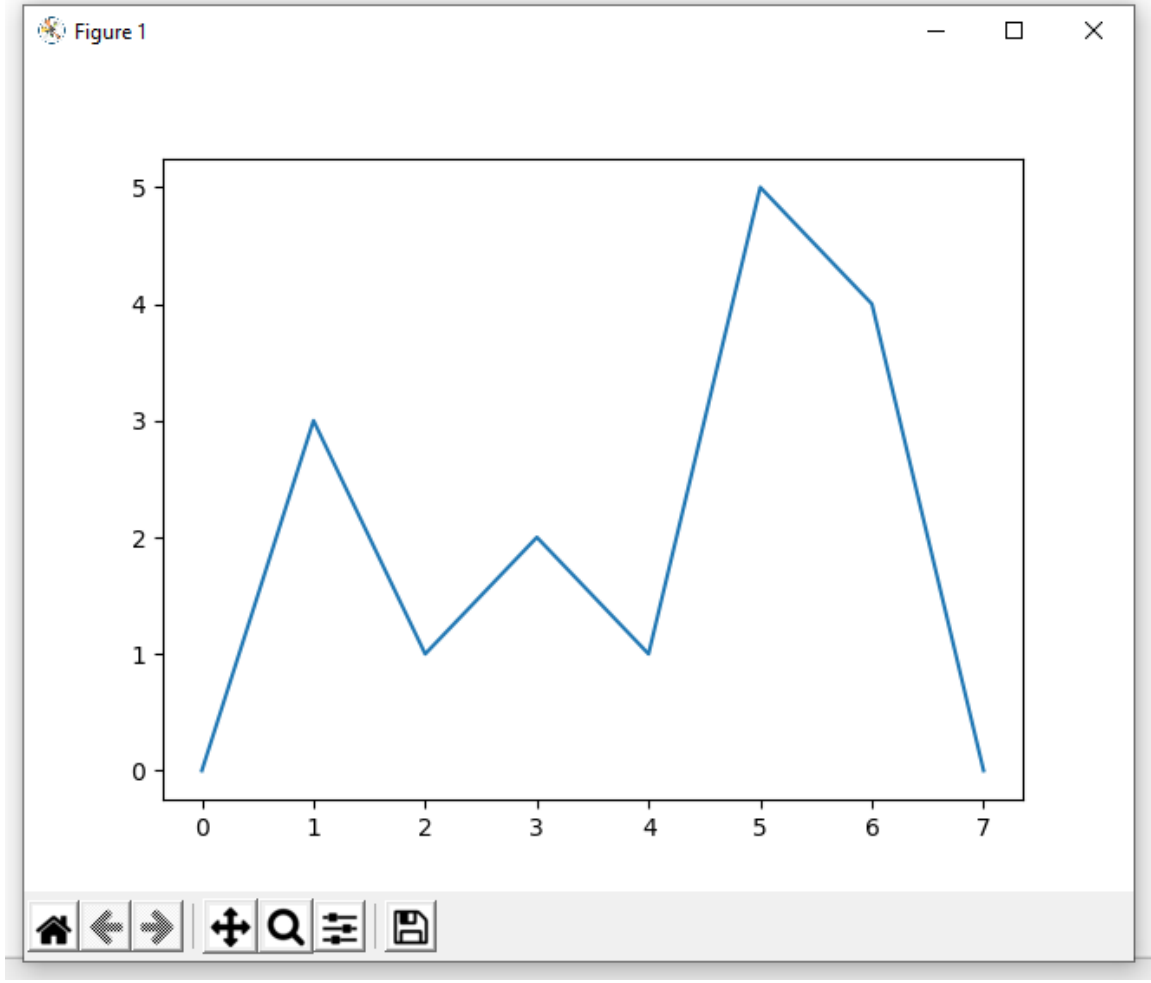
Спосіб малювання лінії дуже простий:
→ існує масив абсцис (x);
→ існує масив ординат (y);
→ елементи з однаковим індексом в цих масивах є координатами точок на площині;
→ послідовні точки з'єднуються лінією.

Масиви — це списки NumPy, кортежи або масиви NumPy. До речі, останні забезпечують набагато більшу зручність, ніж списки і кортежі, тому знання пакета NumPy може значно спростити вам життя.

Як що виконати наступний код:

```
import matplotlib.pyplot as plt
plt.plot((0, 1, 2, 3, 4, 5, 6, 7), (0, 3, 1, 2, 1, 5, 4, 0))
plt.show()
```

Виходить такий простий графік:

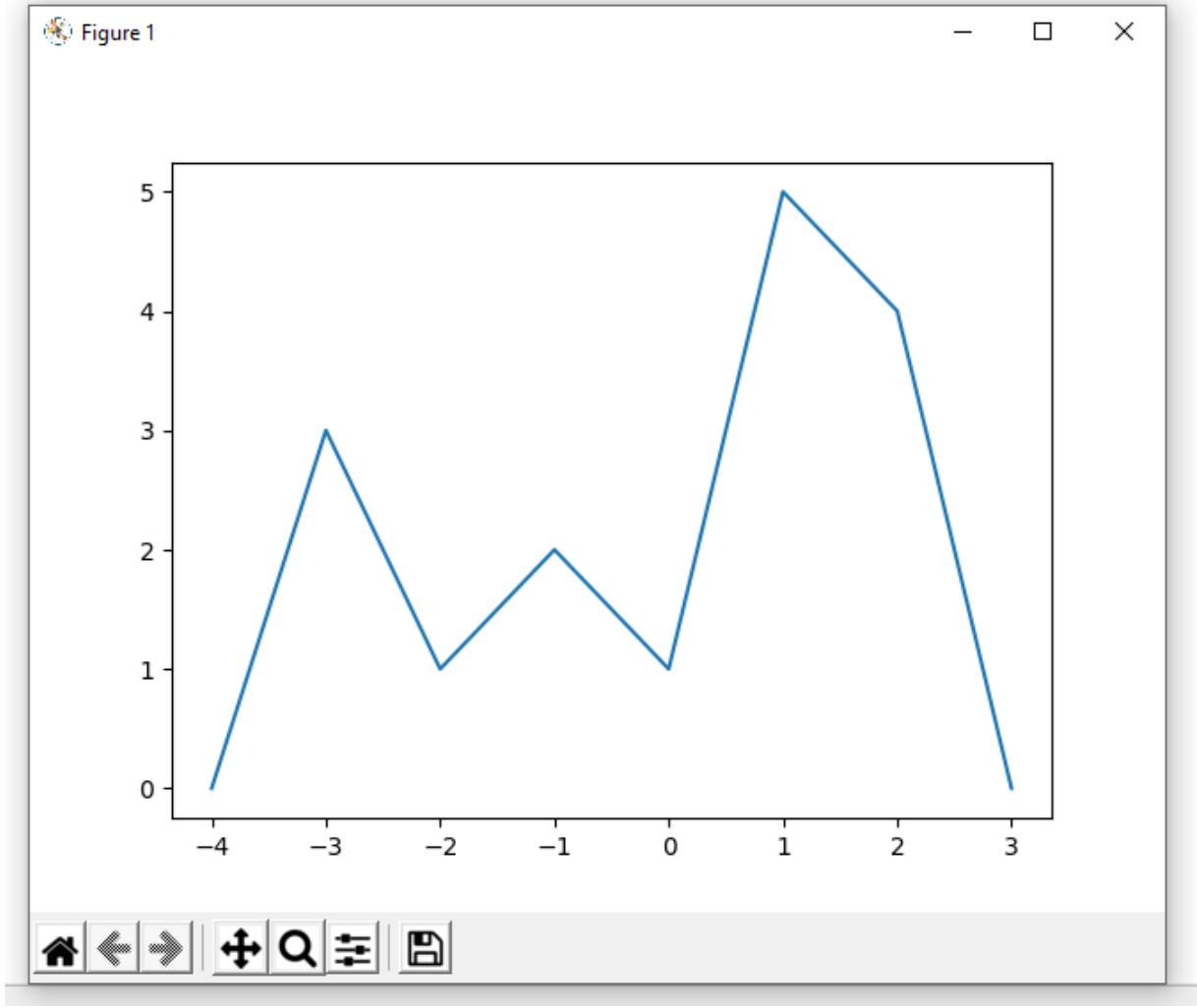


Метод plt.plot() в найпростішому випадку приймає один аргумент - послідовність чисел, яка відповідає осі ординат (y), вісь абсцис (x) будується автоматично від 0 до n, де n - довжина масиву ординат.
Наступний код буде будувати точно такий же графік:

```
import matplotlib.pyplot as plt
plt.plot((0, 3, 1, 2, 1, 5, 4, 0))
plt.show()
```

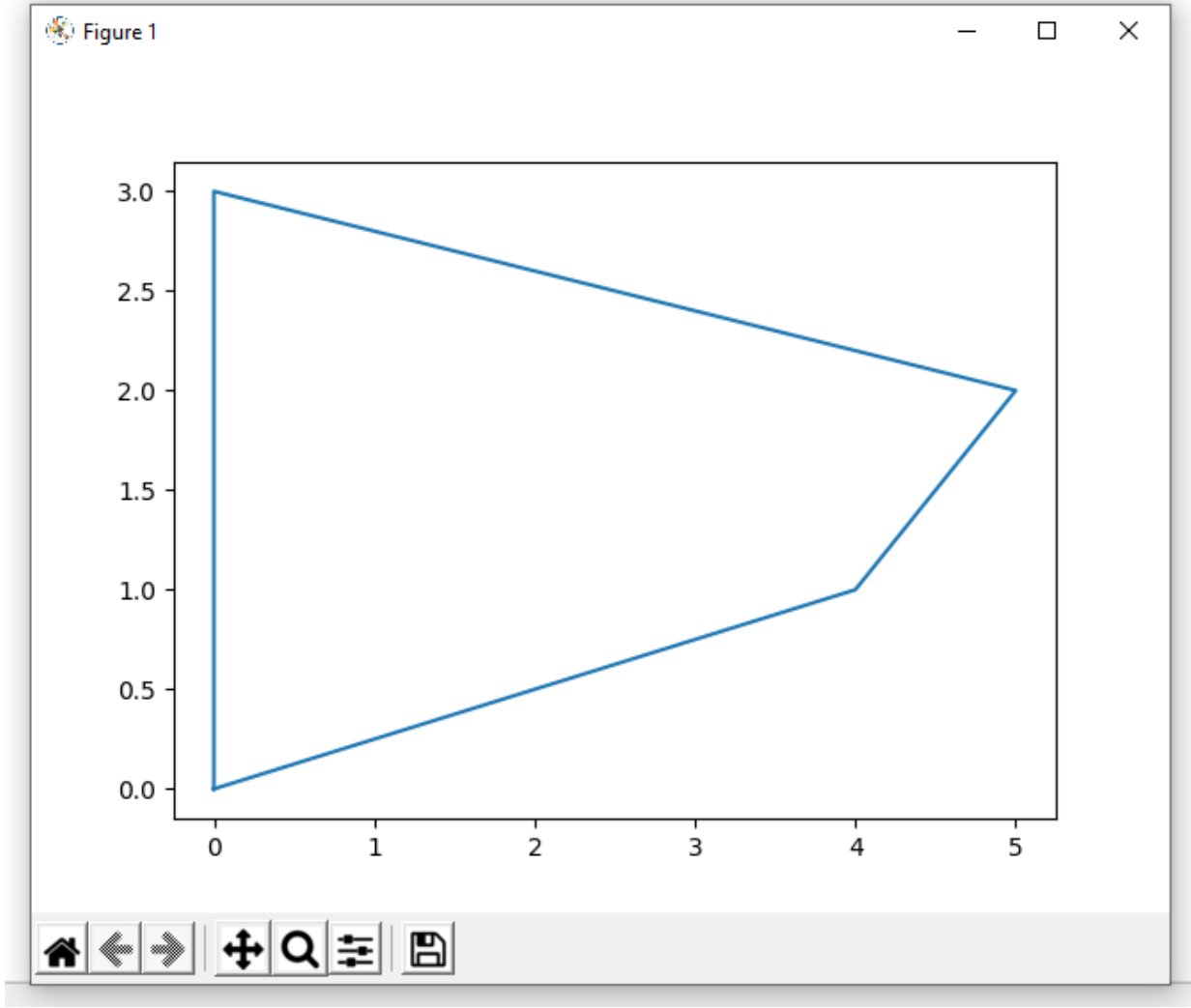
Цей метод може стати в нагоді, якщо діапазон чисел на осі абсцис для вас не важливий. Однак, якщо діапазон або крок важливі, то їх потрібно вказати:

```
import matplotlib.pyplot as plt
plt.plot((-4, -3, -2, -1, 0, 1, 2, 3), (0, 3, 1, 2, 1, 5, 4, 0))
plt.show()
```



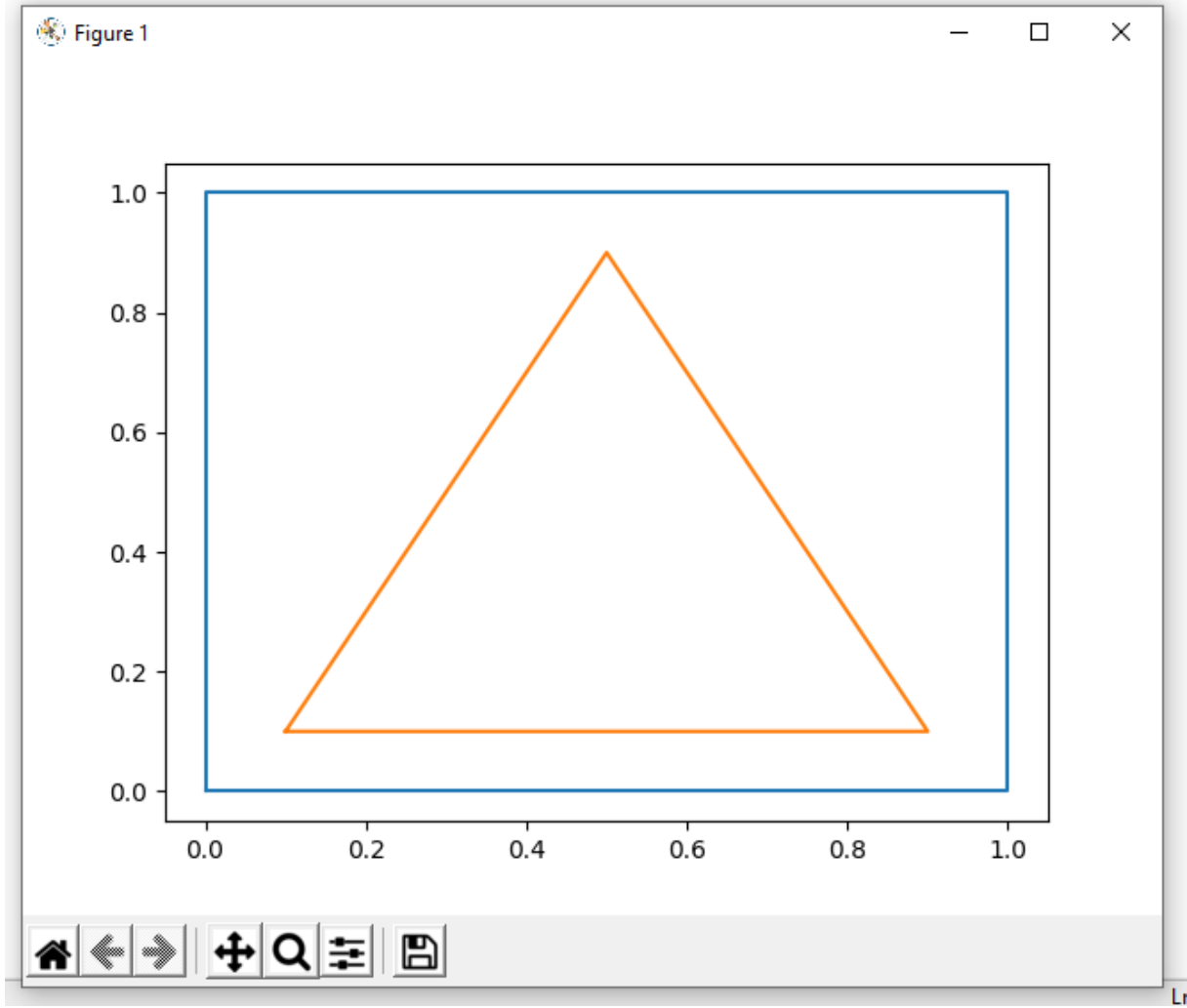
Ще один цікавий момент полягає в тому, що числа в масиві абсцис не обов’язково повинні бути послідовними, тобто можуть бути абсолютно довільними, і тільки послідовні точки будуть з'єднані лінією. Наприклад:

```
import matplotlib.pyplot as plt
plt.plot((0, 0, 5, 4, 0), (0, 3, 2, 1, 0))
plt.show()
```



Така поведінка дуже корисна, коли потрібно побудувати плоскі, закриті криві або геометричні фігури:

```
import matplotlib.pyplot as plt
plt.plot((0, 0, 1, 1, 0), (0, 1, 1, 0, 0))
plt.plot((0.1, 0.5, 0.9, 0.1), (0.1, 0.9, 0.1, 0.1))
plt.show()
```

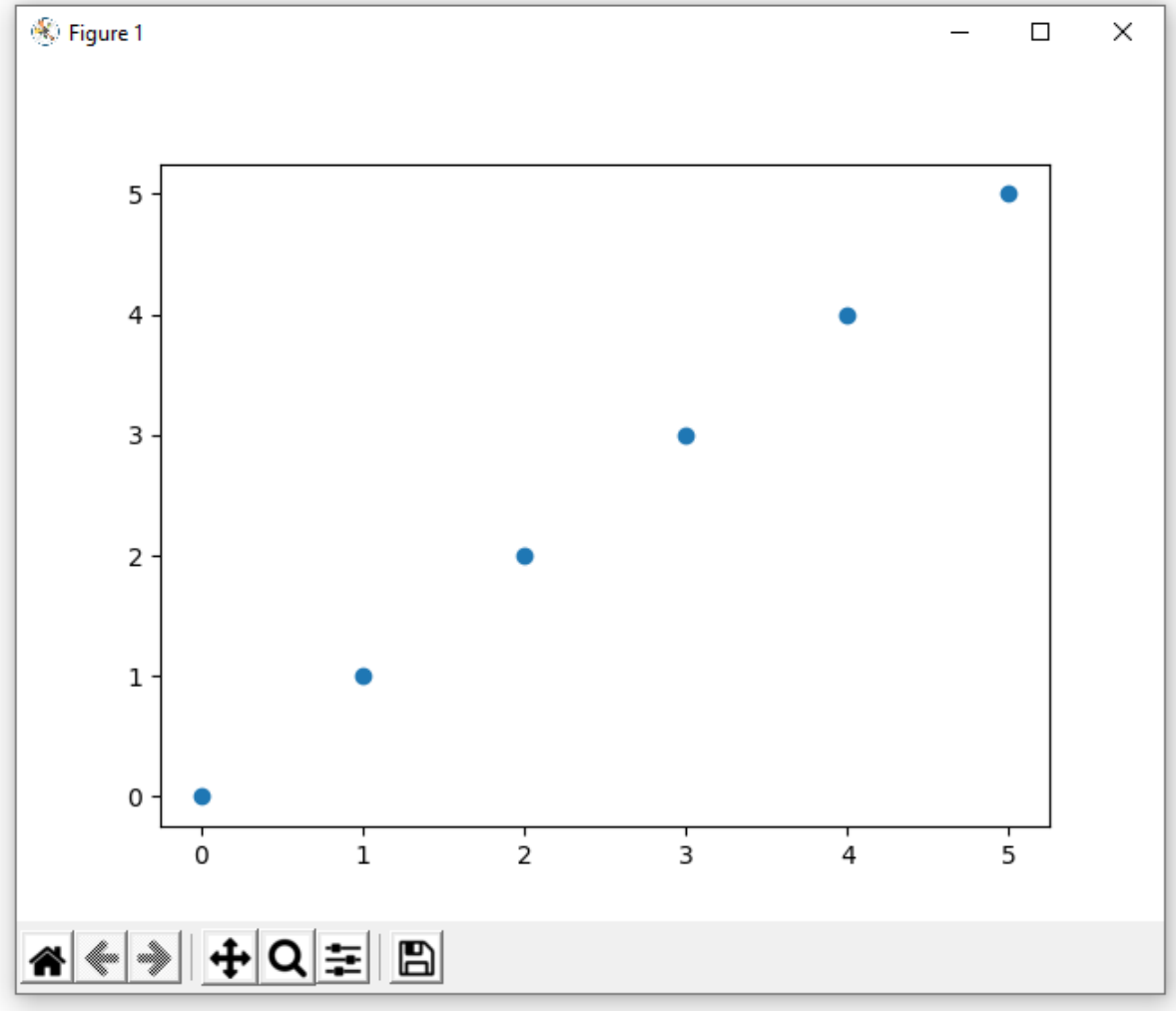


Як ви помітили, ми двічі використовували метод plt.plot(), передаючи різні дані. Можна сказати, що метод plt.plot() займається малюванням наших ліній, а plt.show() відображає саму діаграму. Але ми залишимо всі деталі на потім і рухаємся далі.

Графік множини точок

Єдина відмінність діаграми множини точок від лінійної діаграми полягає в тому, що точки не з'єднані лінією.

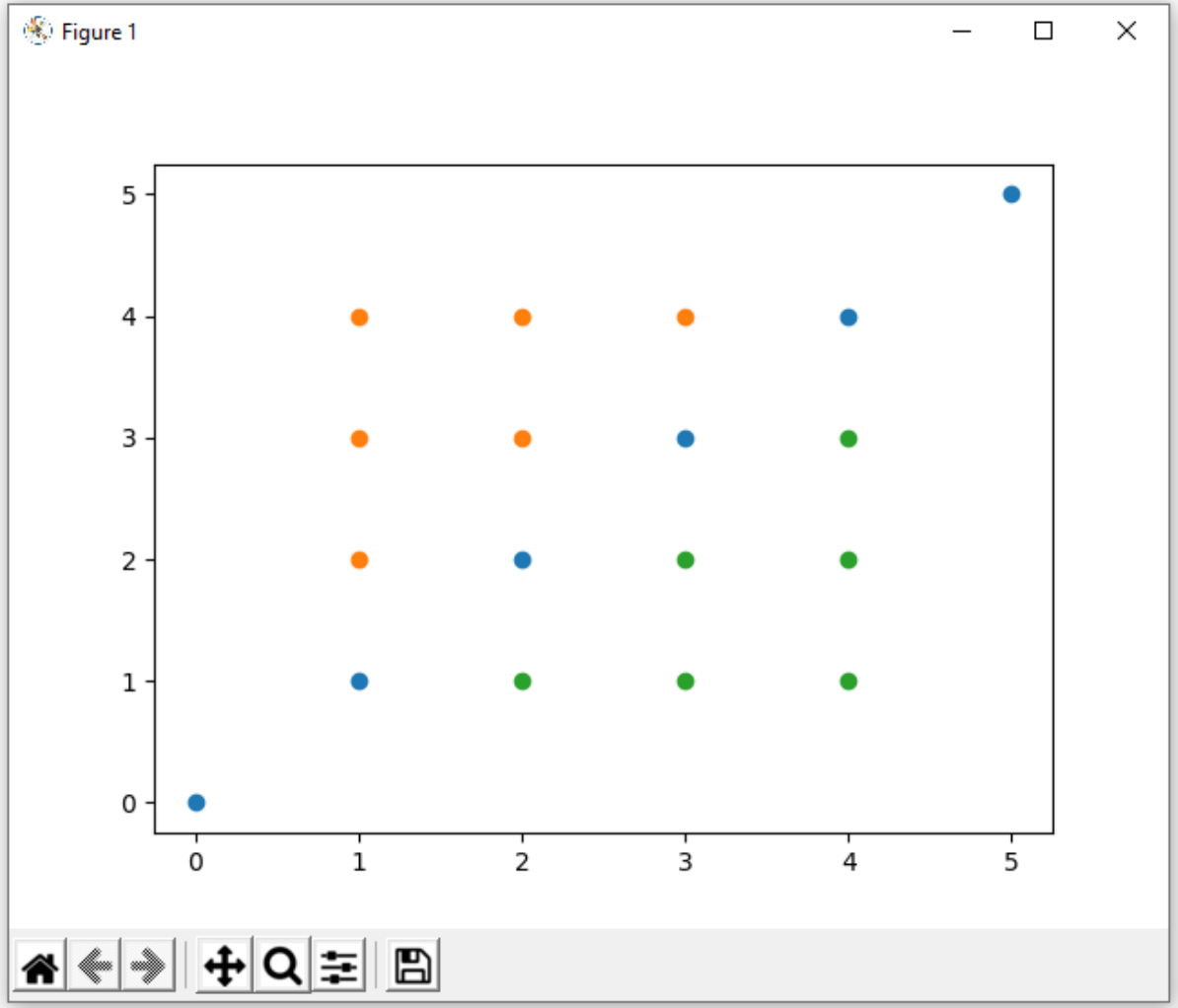
```
import matplotlib.pyplot as plt
plt.scatter([0, 1, 2, 3, 4 , 5], [0, 1, 2, 3, 4 , 5])
plt.show()
```



Як і раніше, двум відповідним значенням з масивів відповідають координати точки.

Якщо у вас кілька множин, то всі вони також можуть бути побудовані на одному графіку:

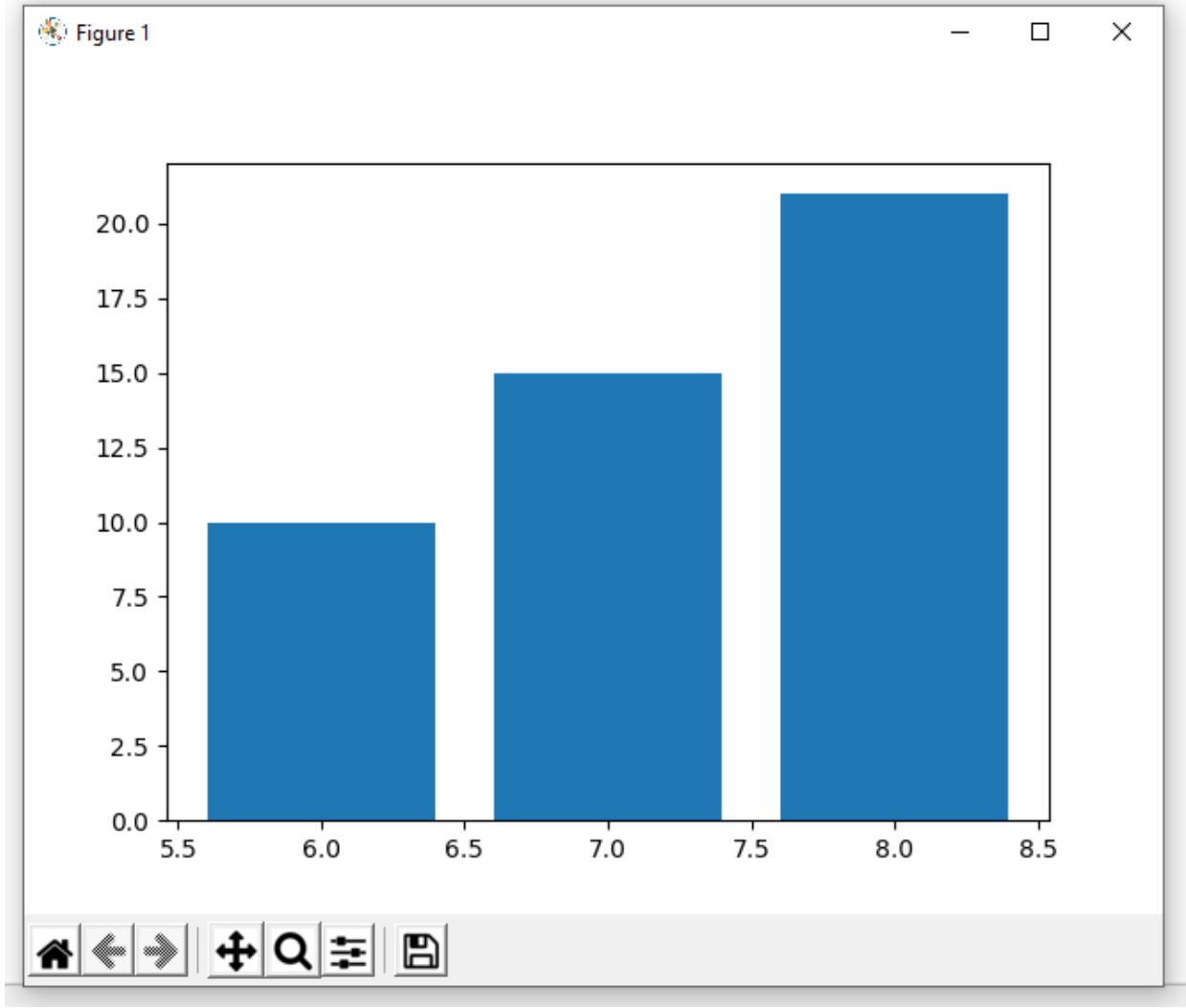
```
import matplotlib.pyplot as plt
plt.scatter([0, 1, 2, 3, 4 , 5], [0, 1, 2, 3, 4 , 5])
plt.scatter([1, 2, 3, 1, 2 , 1], [2, 3, 4, 3, 4 , 4])
plt.scatter([2, 3, 4, 3, 4 , 4], [1, 2, 3, 1, 2 , 1])
plt.show()
```



Гістограма

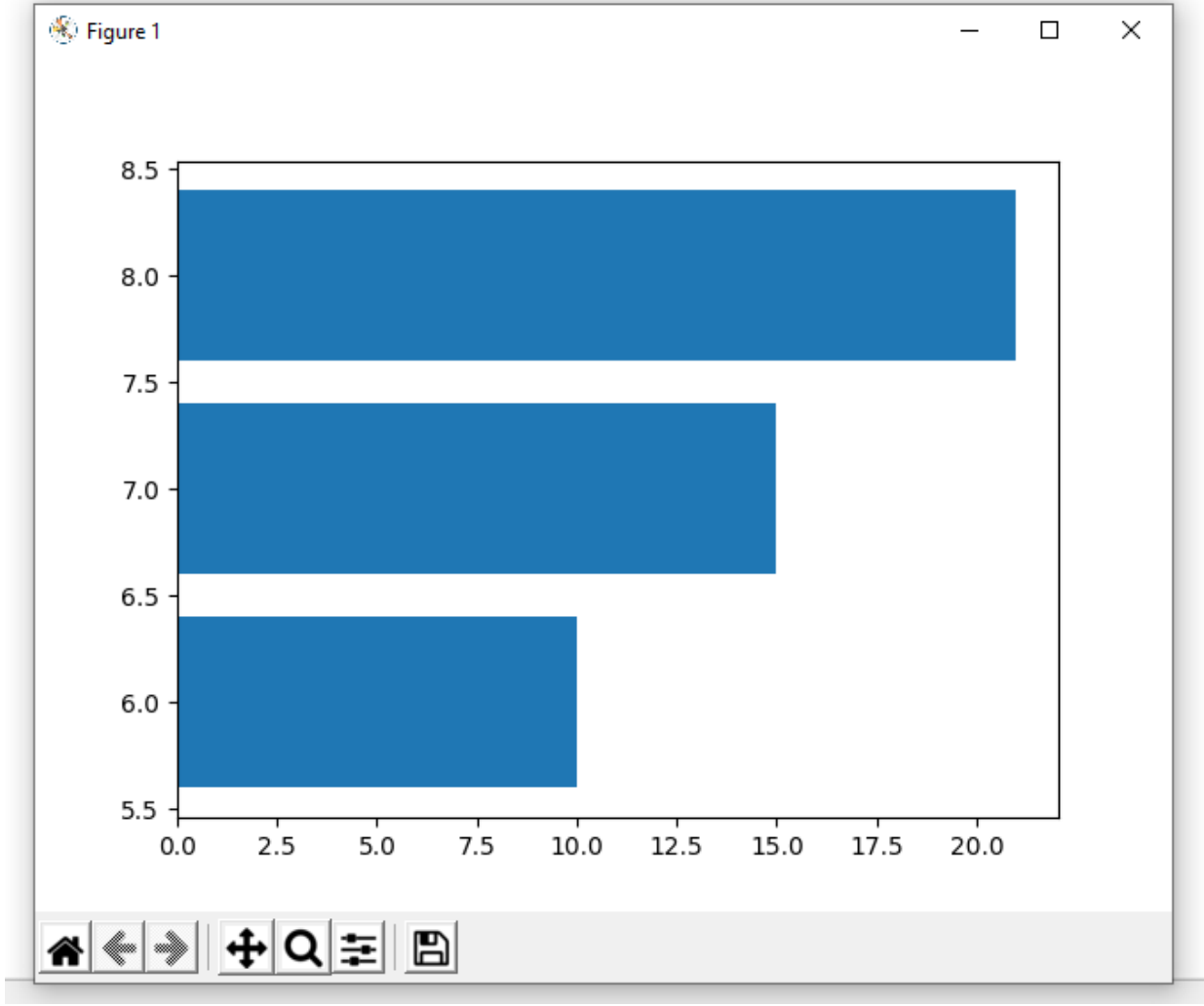
Дуже часто зручно представляти дані у вигляді гістограм. У найпростішому випадку гістограма - це набір прямокутників, площа яких (або висота) пропорційна деякій величині. Наприклад, опади за 3 місяці: в червні випало 10 мм, в липні - 15 мм, в серпні - 21 мм.

```
import matplotlib.pyplot as plt

plt.bar([6, 7, 8], [10, 15, 21])
plt.show()
```



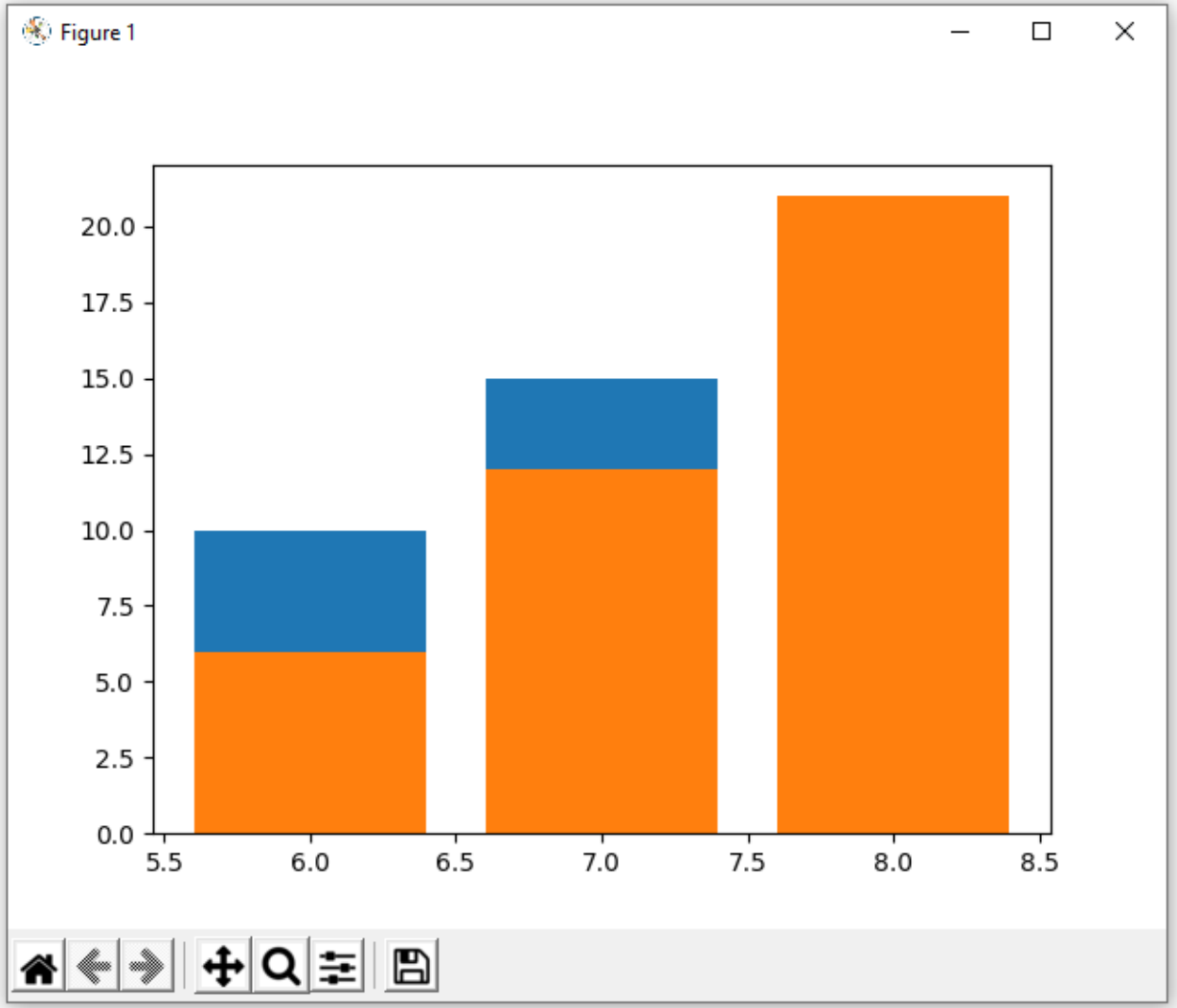
Перший масив містить числа місяців, а другий масив містить значення показників. Ці прямокутники будуються вертикально, але вони також можуть відображатися горизонтально:

```
import matplotlib.pyplot as plt

plt.barh([6, 7, 8], [10, 15, 21])
plt.show()
```



Гістограми можуть відображати кілька наборів даних, що дуже зручно для їх порівняння:

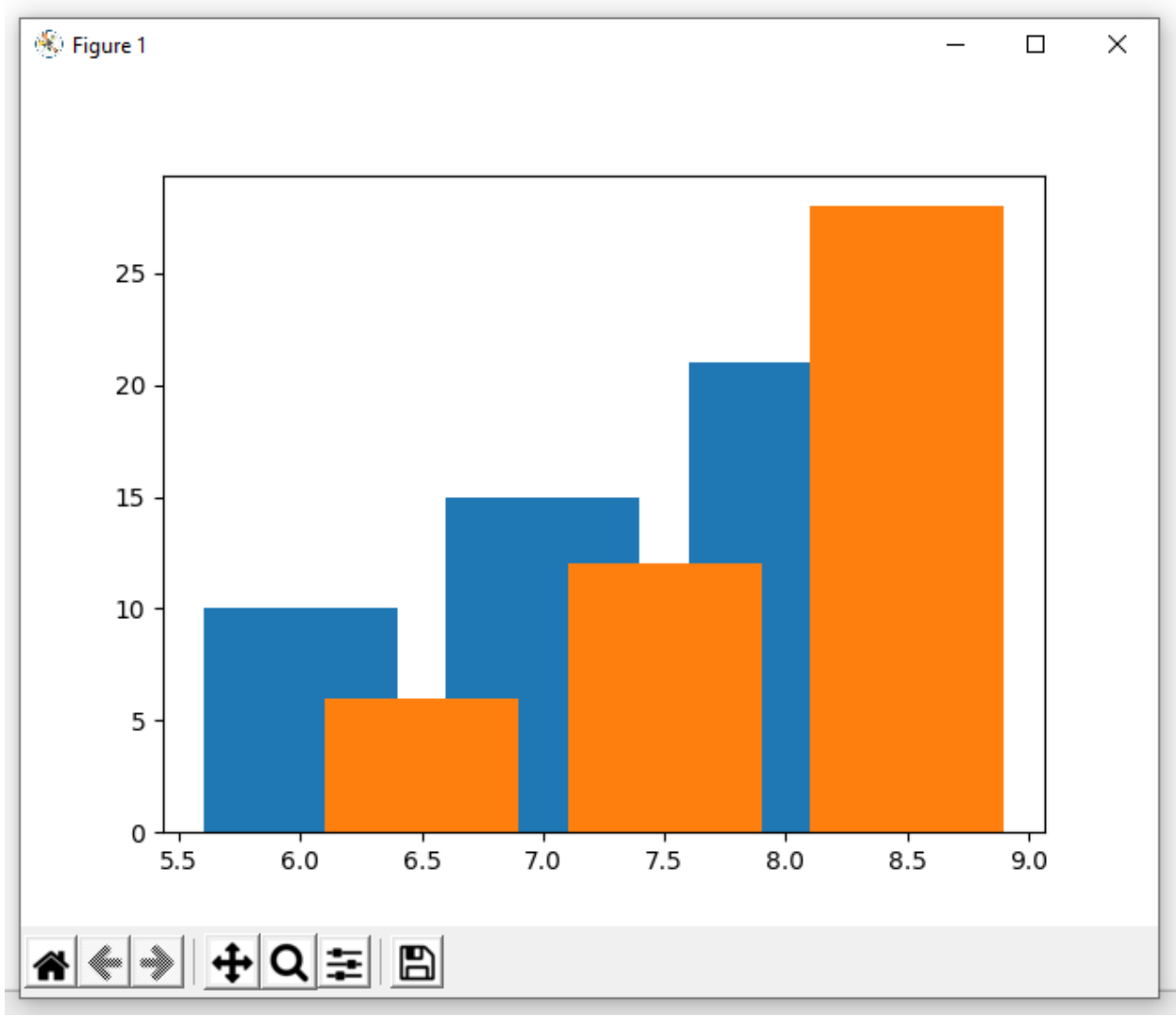
```
import matplotlib.pyplot as plt

plt.bar([6, 7, 8], [10, 15, 21])
plt.bar([6, 7, 8], [6, 12, 21])
plt.show()
```



Такий графік може відображати літні опади протягом двох років. Але ось в чому справа, прямокутники будуються один на одному, і якщо вони рівні, як у випадку з нашими опадами за серпень, то прямокутники будуть перекривати один одного.

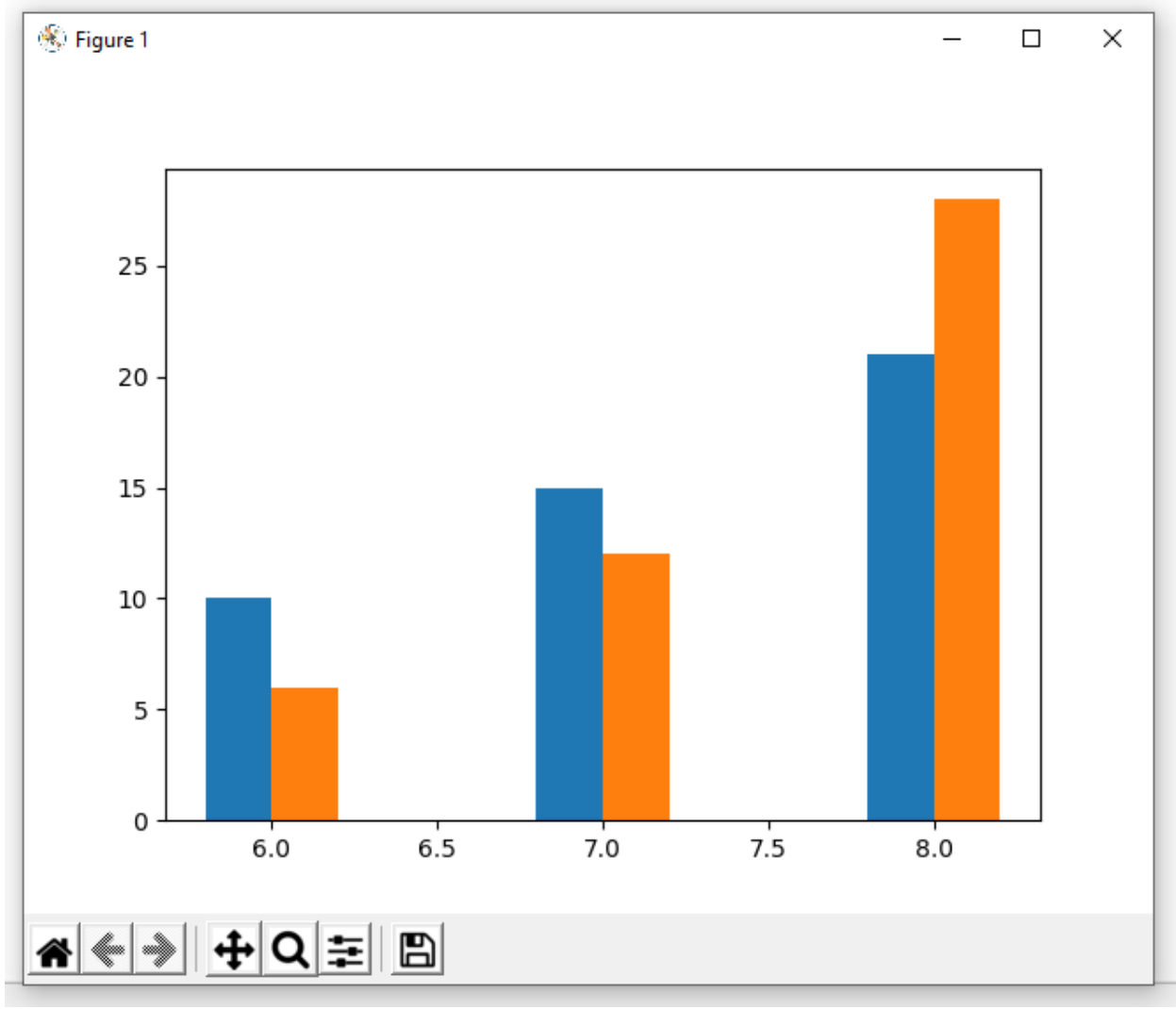
Якщо вказати невеликий зсув по осі x, ситуація не покращиться:

```
import matplotlib.pyplot as plt

plt.bar([6, 7, 8], [10, 15, 21])
plt.bar([6.5, 7.5, 8.5], [6, 12, 28])
plt.show()
```



Графік стане більш привабливим, якщо звузити прямокутники і розташувати їх, не перекривання один з одним:

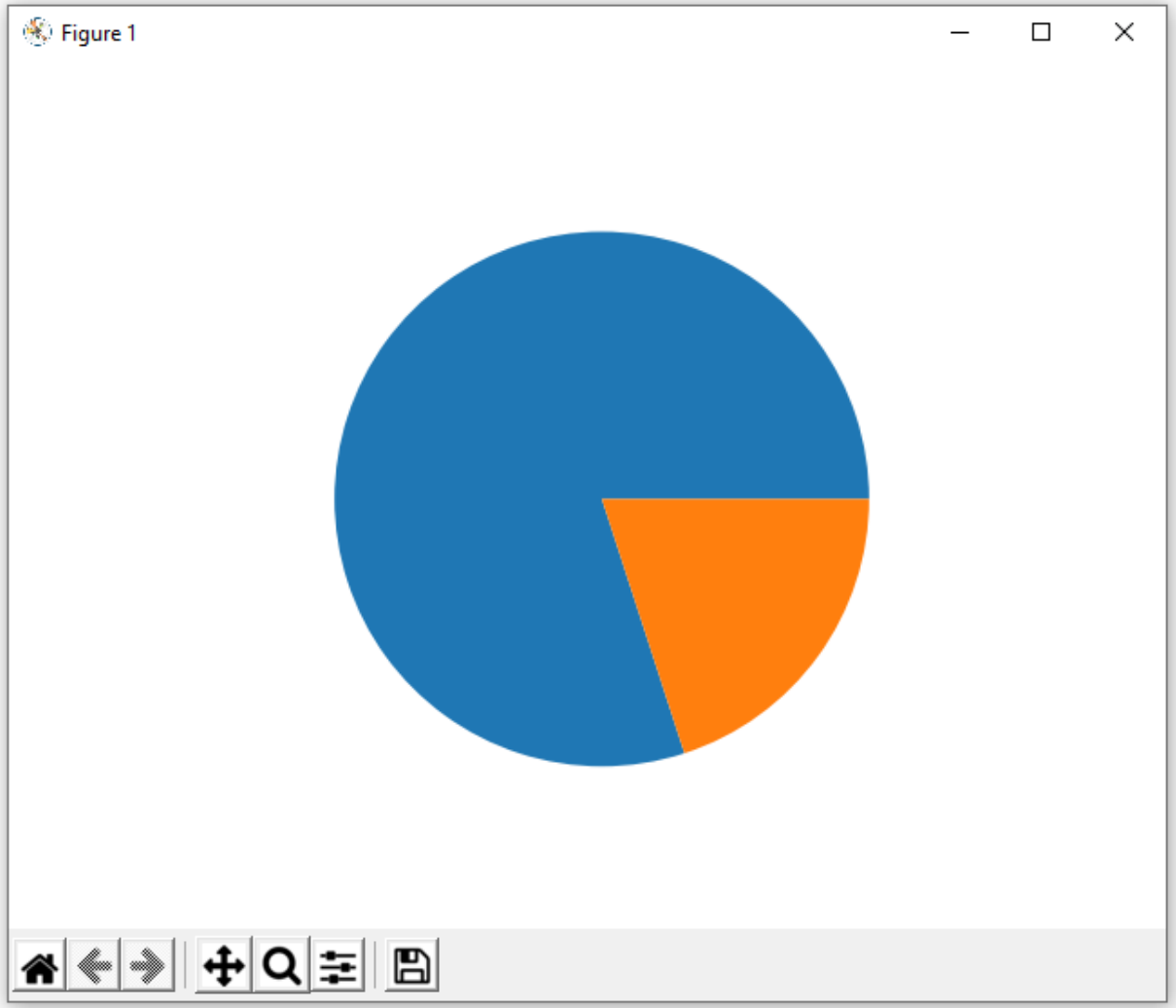
```
import matplotlib.pyplot as plt

plt.bar([5.9, 6.9, 7.9], [10, 15, 21], width = 0.2)
plt.bar([6.1, 7.1, 8.1], [6, 12, 28], width = 0.2)
plt.show()
```



Кругові діаграми

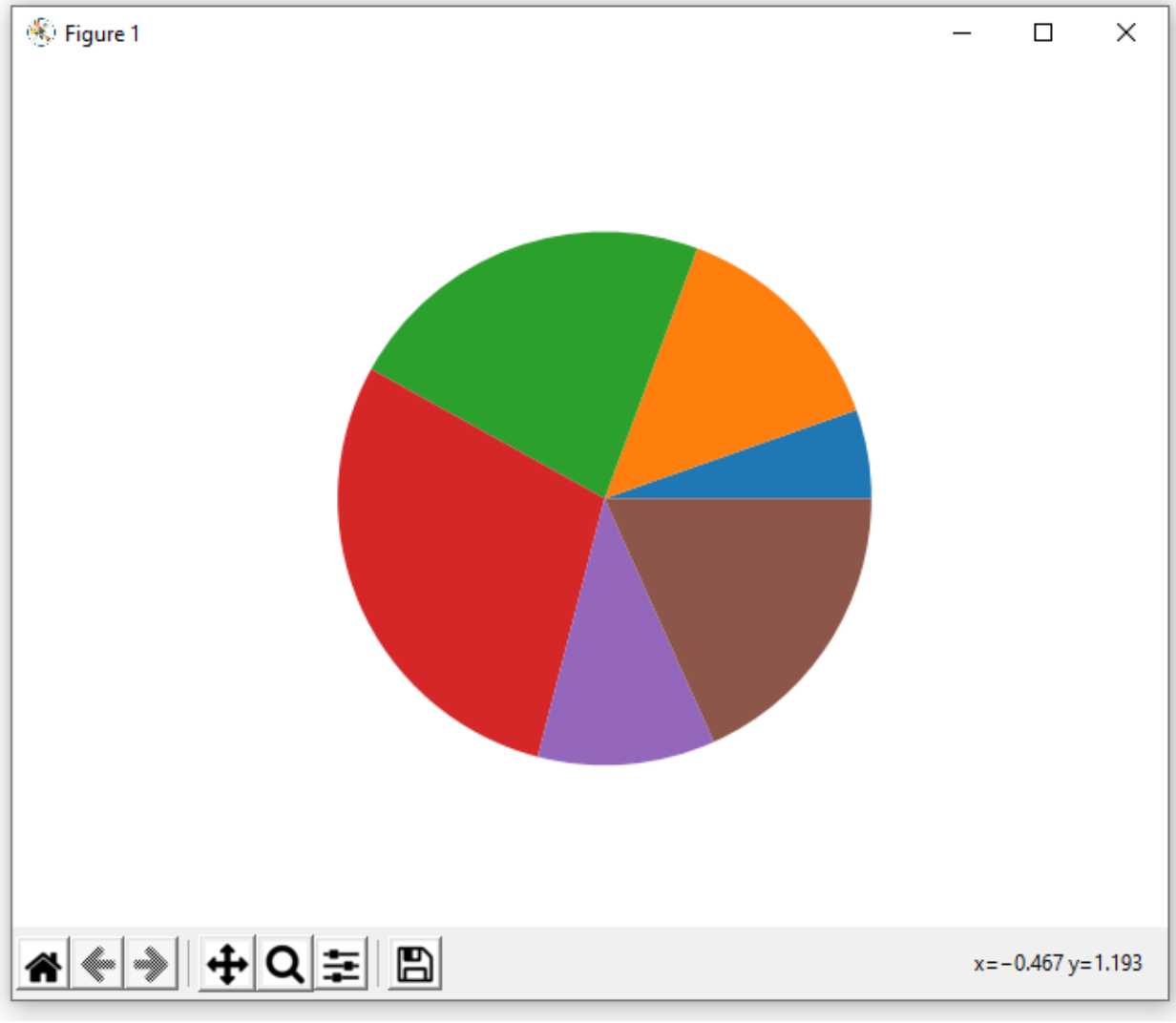
Якщо потрібно візуально відобразити співвідношення частин цілого, то краще використовувати кругову діаграму. Наприклад, в компанії працює 50 осіб, з яких 40 - жінки і 10 - чоловіки:

```
import matplotlib.pyplot as plt

plt.pie([40, 10])
plt.show()
```



Кількість елементів в масиві визначає кількість клинів, а величина значень визначає їх площу:

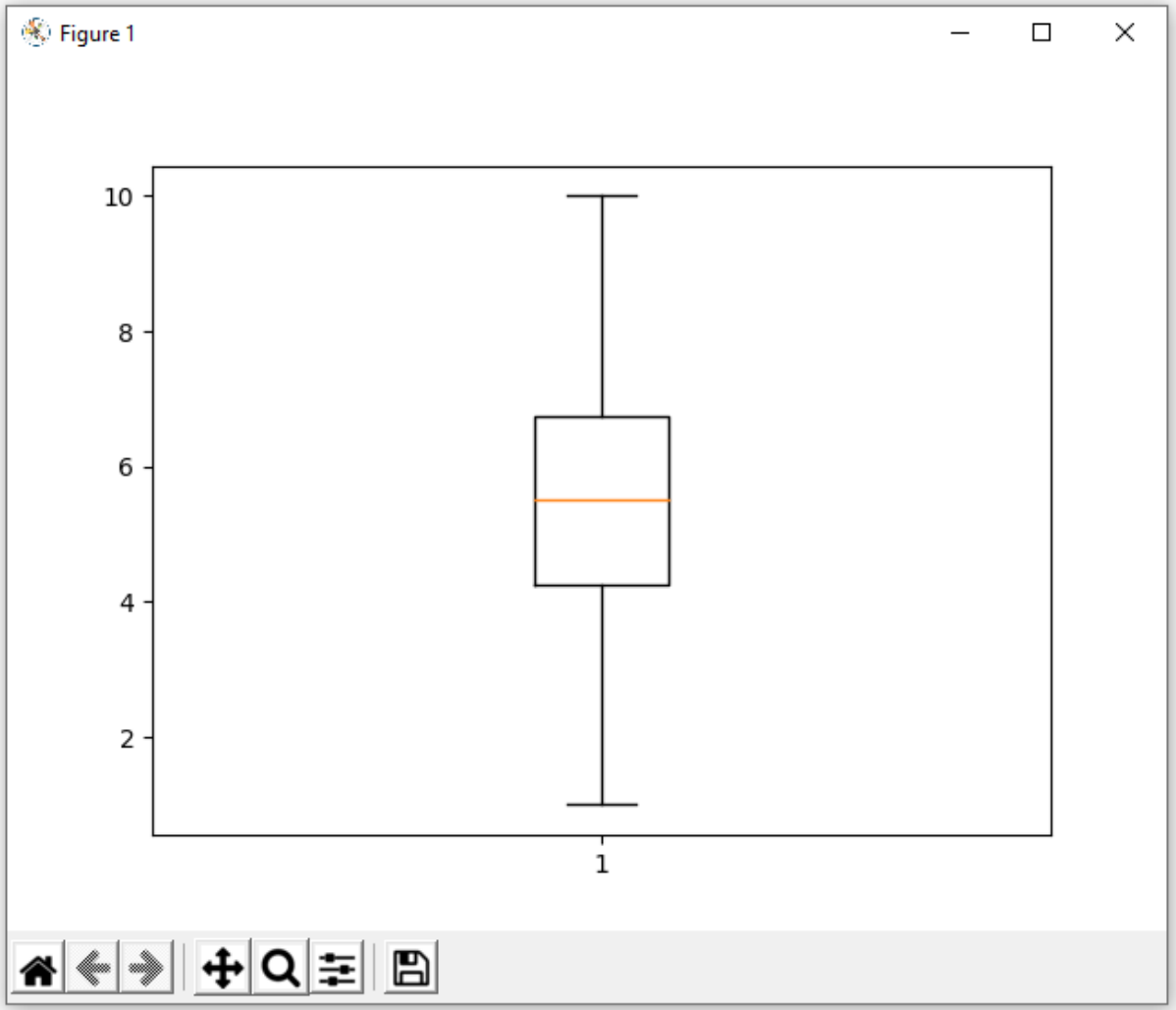
```
import matplotlib.pyplot as plt

plt.pie([5, 13, 21, 27, 10, 17])
plt.show()
```



Коробка з вусами

Цей тип графіка дійсно схожий на коробку з вусами (якщо повернути монітор на 90°):

```
import matplotlib.pyplot as plt

plt.boxplot([1, 5, 7, 4, 6, 10])
plt.show()
```

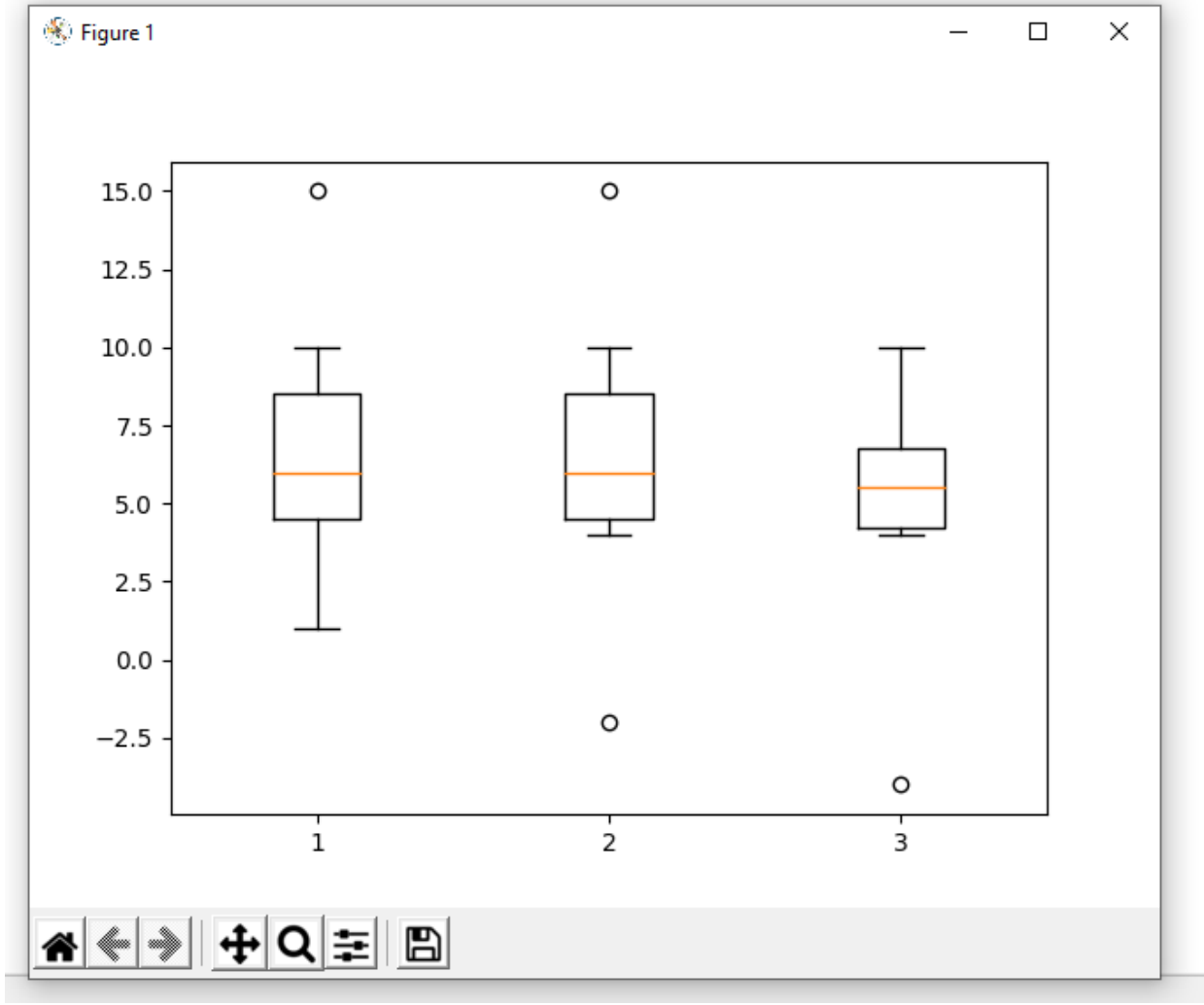


Але насправді ця коробка з вусами є графіком, який служить для відображення випадкової змінної і несе в собі багато інформації. По-перше, всередині коробки помаранчева лінія позначає медіану елементів масиву - це значення, яке розташоване рівно на половині елементів масиву. У нашому випадку це значення становить 5,5 і, як ви можете легко побачити, половина елементів менше, ніж вона, а інша більше. Його межі - 25-й і 75-й процентилі (4,25 і 6,75 для нашого масиву). Ну а вуса, власне (як правило) максимальне і мінімальне значення в наборі даних. Це така складна, але дуже корисна коробка.

Іноді на графіку, поруч з вусами, з'являються одна або дві точки. Такі точки позначають викиди - значення, які дуже далекі від статистично значущої частини даних:

```
import matplotlib.pyplot as plt

plt.boxplot([[1, 5, 7, 4, 6, 10, 15],
             [-2, 5, 7, 4, 6, 10, 15],
             [-4, 5, 7, 4, 6, 10]])
plt.show()
```

Як неважко здогадатися, цей тип графіків є найбільш корисними в області статистики.

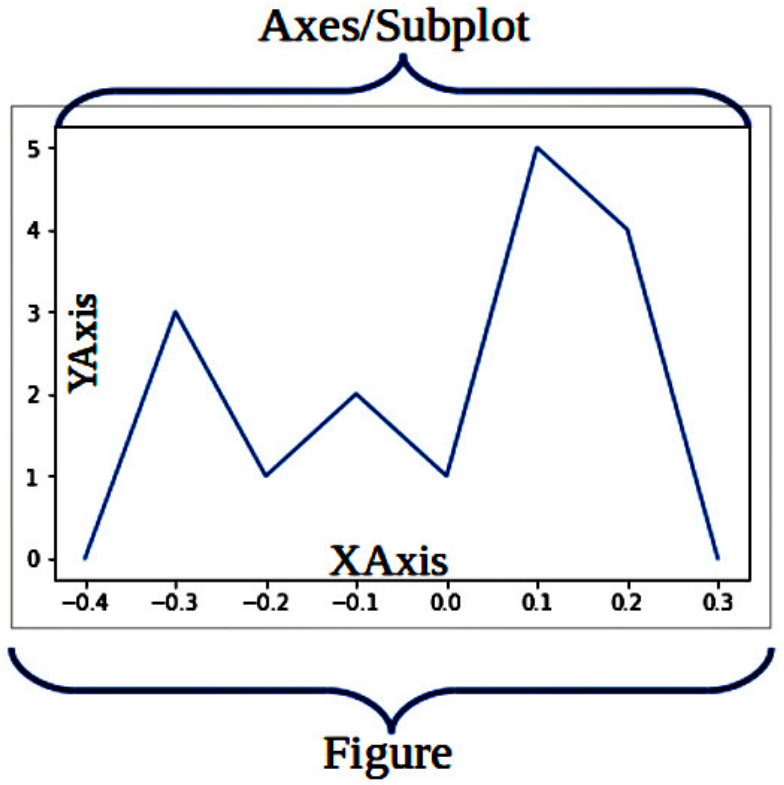


Основні компоненти matplotlib

MPL є дуже великою бібліотекою, але за своєю суттю вона складається з невеликої кількості основних компонентів:

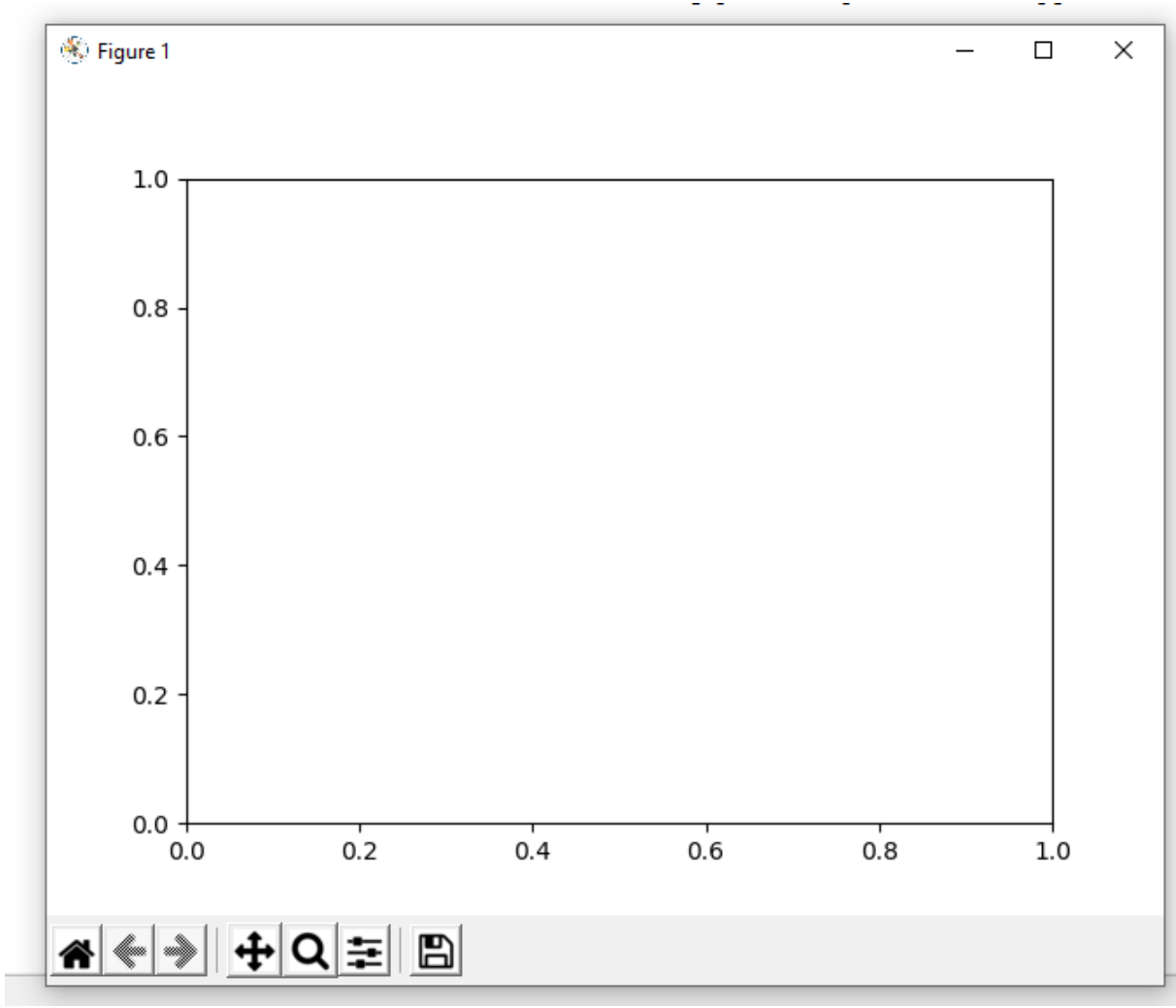
Figure - контейнер верхнього рівня, на якому все намальовано. Таких зон може бути кілька, кожна з яких може містити кілька контейнерів *Axes*.

Axes - це область, де найчастіше відображаються графіки (дані у вигляді графіків), а також всі допоміжні атрибути (лінії сітки, мітки, покажчики і т.д.). Часто установка цієї області супроводжується викликом *subplot*, який розміщує осі на звичайній сітці. Тому часто *Axes* і *Subplot* можна вважати синонімами. Але що це за сітка і як працює це розміщення, розберемося трохи далі.
Кожна область *Axes* містить XAxis і YAxis. Вони містять розділення, мітки та інші допоміжні атрибути.



Виконаємо наступний код:

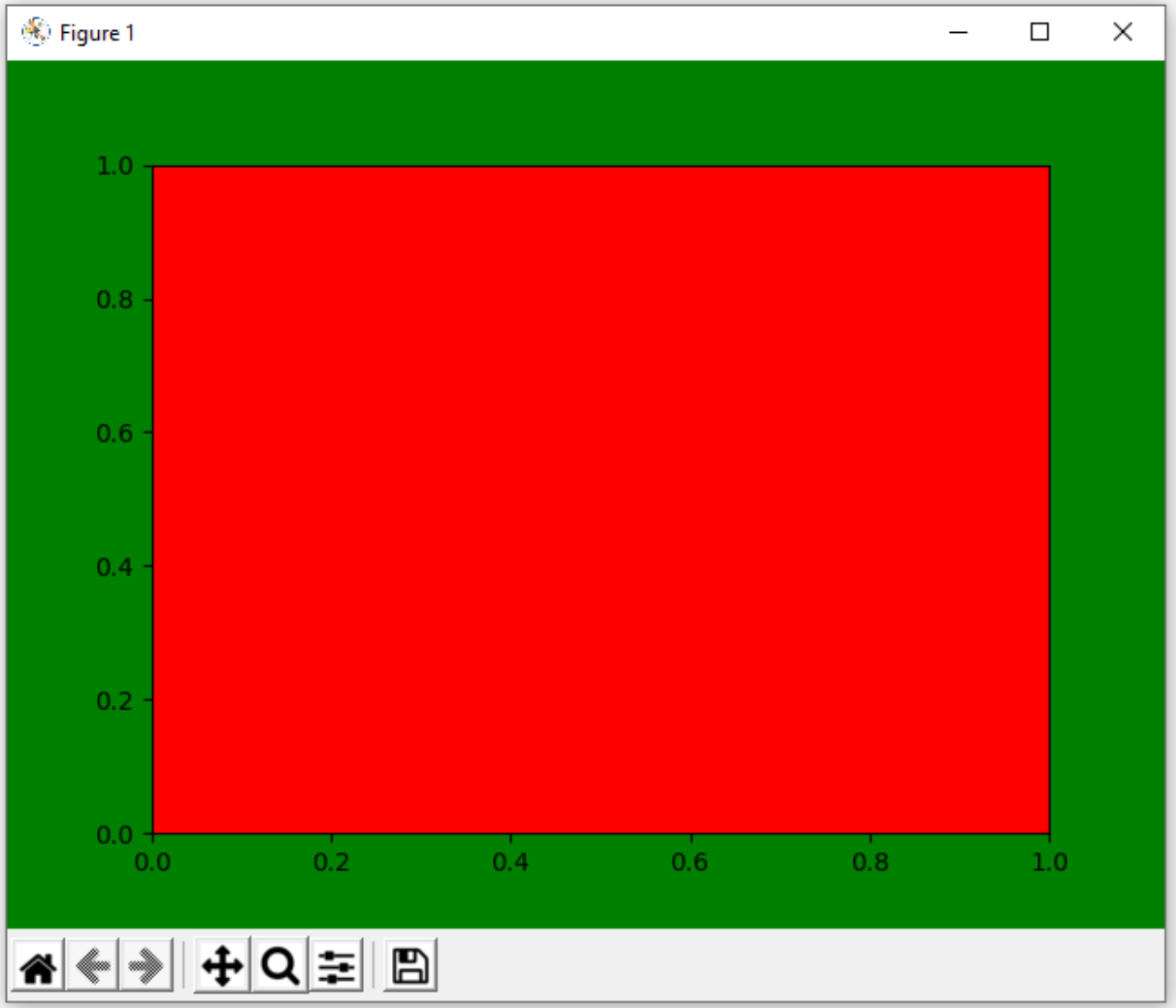
```
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111)

plt.show()
```



В рядку fig = plt.figure() ми створили область Figure (екземпляр класу figure). В рядку ax = fig.add_subplot(111) ми додали к Figure область Axes. В цілому, правильніше було б використовувати fig.add_axes, але в цьому випадку fig.add_subplot(111) набагато зручніше, адже subplot просто розміщує Axes на сітці Figure. Зверніть увагу на параметр, який ми передаємо 111 - це перший рядок, перший стовпець і перша (одиночна) комірка на сітці Figure.

В тому, що *Figure* і *Axes* - це різні області, можна легко перевірити, якщо змінити їх колір:

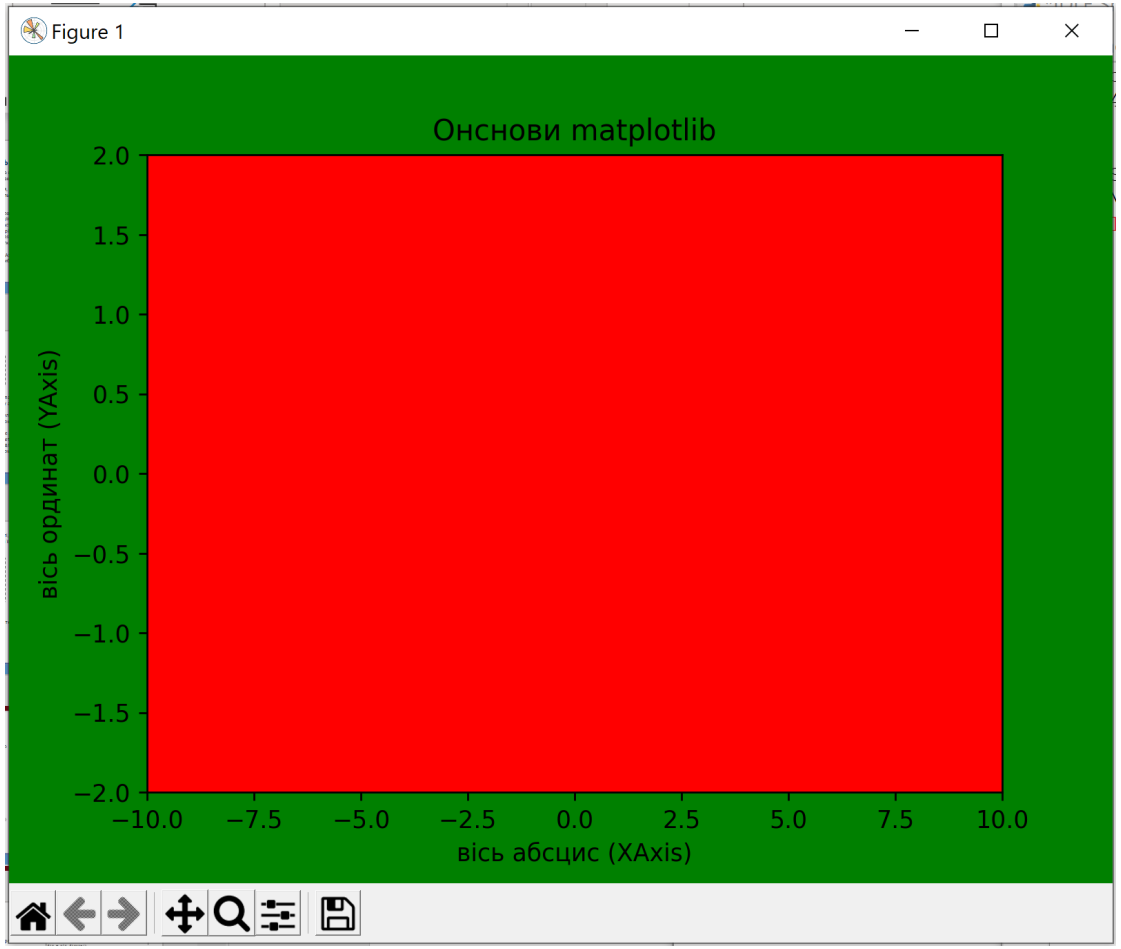
```
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111)

fig.set(facecolor = 'green')
ax.set(facecolor = 'red')

plt.show()
```



До речі, *Axes* повинні належати тільки одній області *Figure*. Як правило, спочатку ви завжди створюєте область Figure, а потім використовуєте add_subplot() у *Figure* розміщується одна або кількох областей *Axes*.

Саме *Axes* доведеться змінювати найчастіше, тому давайте встановимо більше параметрів для цієї області:

```
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111)

fig.set_facecolor('green')

ax.set_facecolor('red')
ax.set_xlim([-10, 10])
ax.set_ylim([-2, 2])
ax.set_title('Онснови matplotlib')
ax.set_xlabel('вісь абсцис (XAxis)')
ax.set_ylabel('вісь ординат (YAxis)')

plt.show()
```



Хоча ми могли б зробити це більш коротшим методом:

Цей метод хороший, якщо у вас є певний досвід і запам'ятовування назв всіх параметрів напам'ять. Для початківців рекомендовано явний спосіб встановлення параметрів: так швидше (завдяки автоматичному додаваю) так і більш очевидно.

```
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111)

fig.set_facecolor('green')

ax.set(facecolor = 'red',
xlim = [-10, 10],
ylim = [-2, 2],
title = 'Онснови matplotlib',
xlabel = 'вісь абсцис (XAxis)',
ylabel = 'вісь ординат (YAxis)')

plt.show()
```

Майже всі об'єкти matplotlib мають метод set. Наприклад, ми раптом захотіли змінити колір і розмір title. Є також два способи зробити це:

```
#  Способ №1:
ax.set_title('Основи matplotlib', color = 'white', size = 20)

#  Способ №2:
ax.set_title('Основи matplotlib')
ax.title.set_color('white')
ax.title.set_size(20)
```

У таких випадках перший спосіб, знову ж таки, здається простішим. Але другий метод дозволяє створити код, який легко зрозуміти і в якому легко орієнтуватися.
І, нарешті, створений нами графік - це просто знущання з сприйняття людиною. Такі графіки можна зробити тільки для прикладу! Створення графіків - це ціла наука (або мистецтво).

Відображення даних на діаграмі

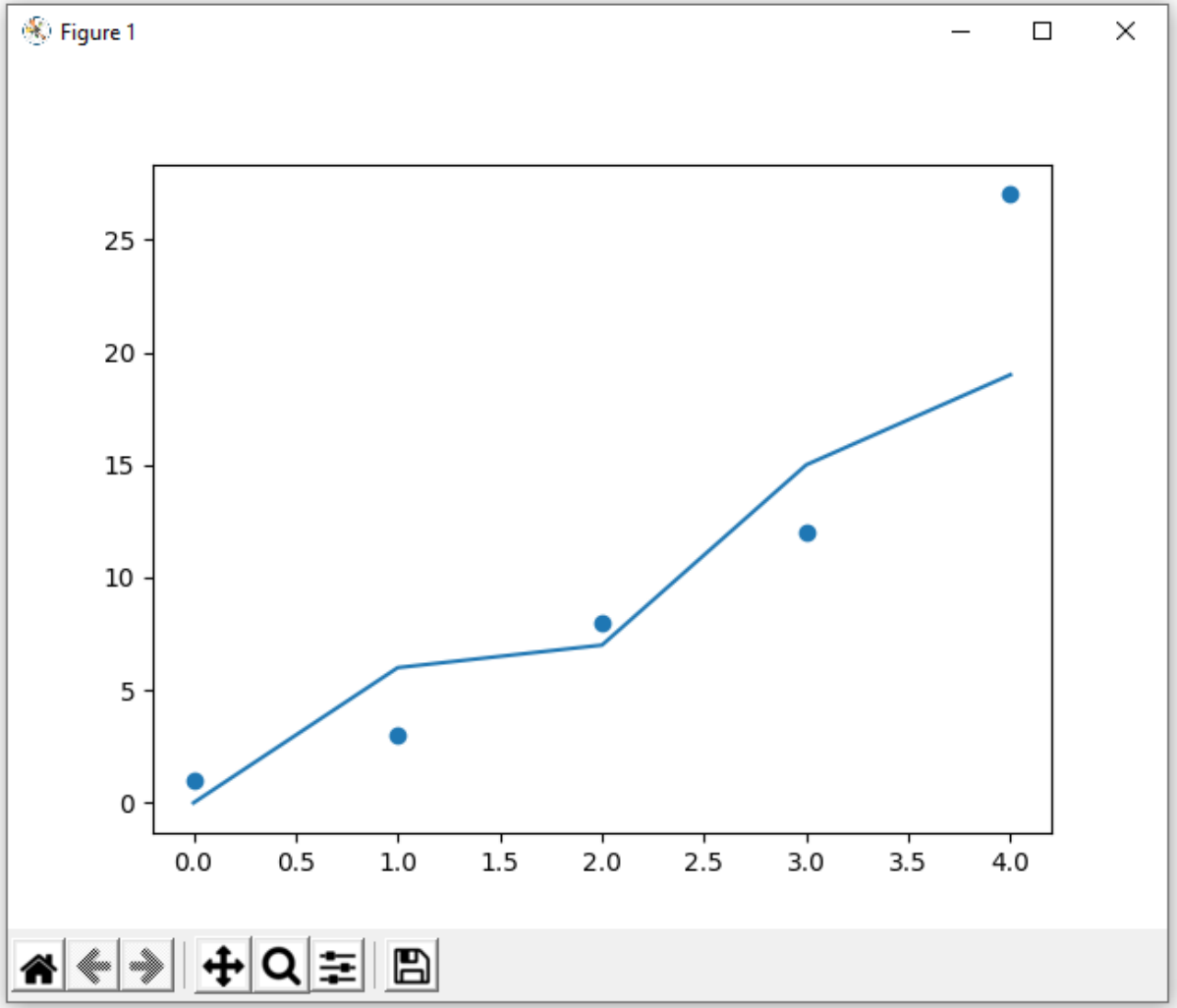
Відображення даних відбуваються на Axes. Тому, щоб малювати на Axes, необхідно використовувати один з його методів. До речі, існує ціла купа цих методів, але ми зосередимося тільки на двох: plot і scatter.

→ plot малює точки, з'єднані лініями;

→ scatter просто малює точки

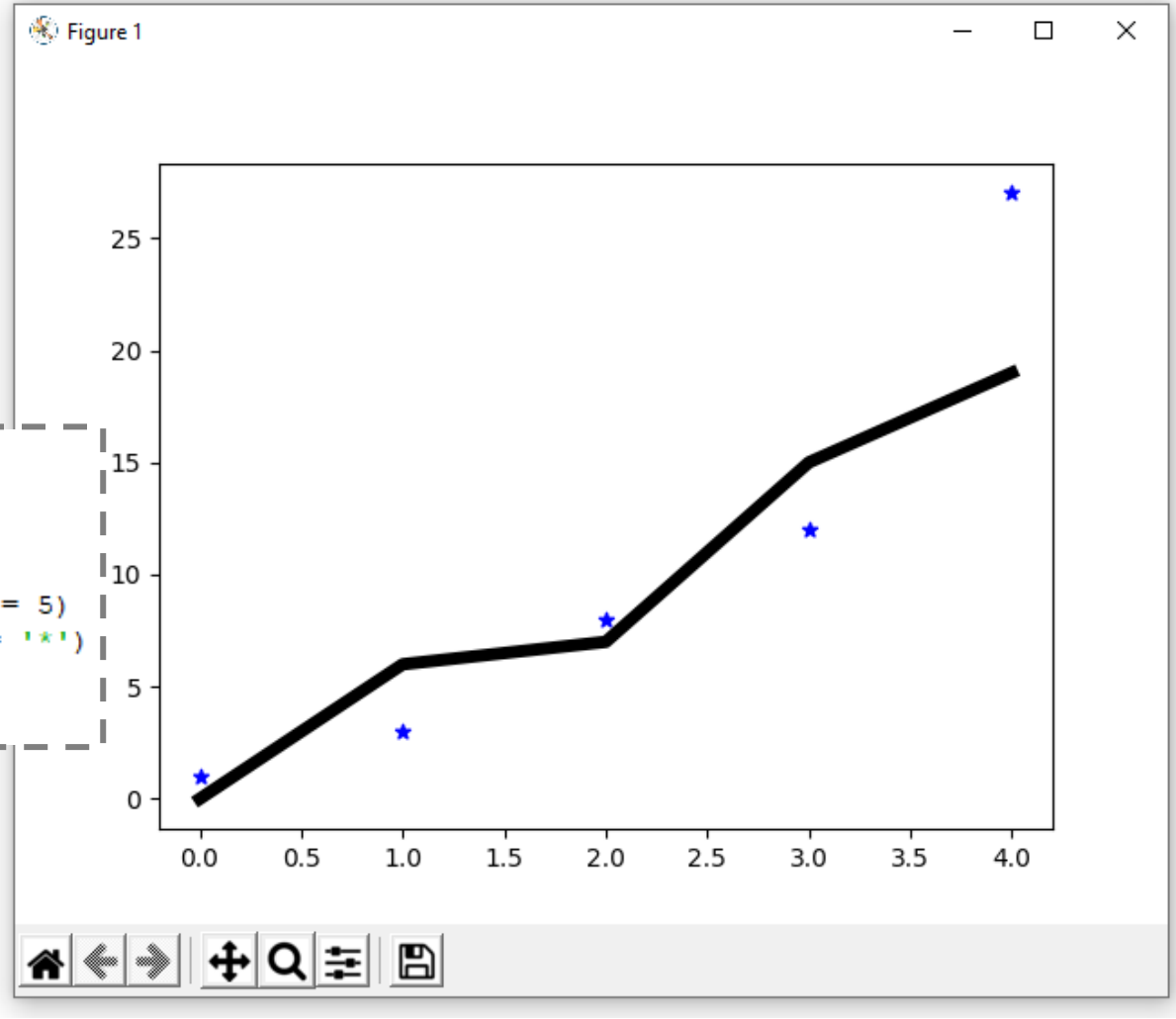
Давайте побудуємо простий графік, на якому буде відображатися деякі дані точками, а інші лініями:

```
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot([0, 1, 2, 3, 4], [0, 6, 7, 15, 19])
ax.scatter([0, 1, 2, 3, 4], [1, 3, 8, 12, 27])

plt.show()
```



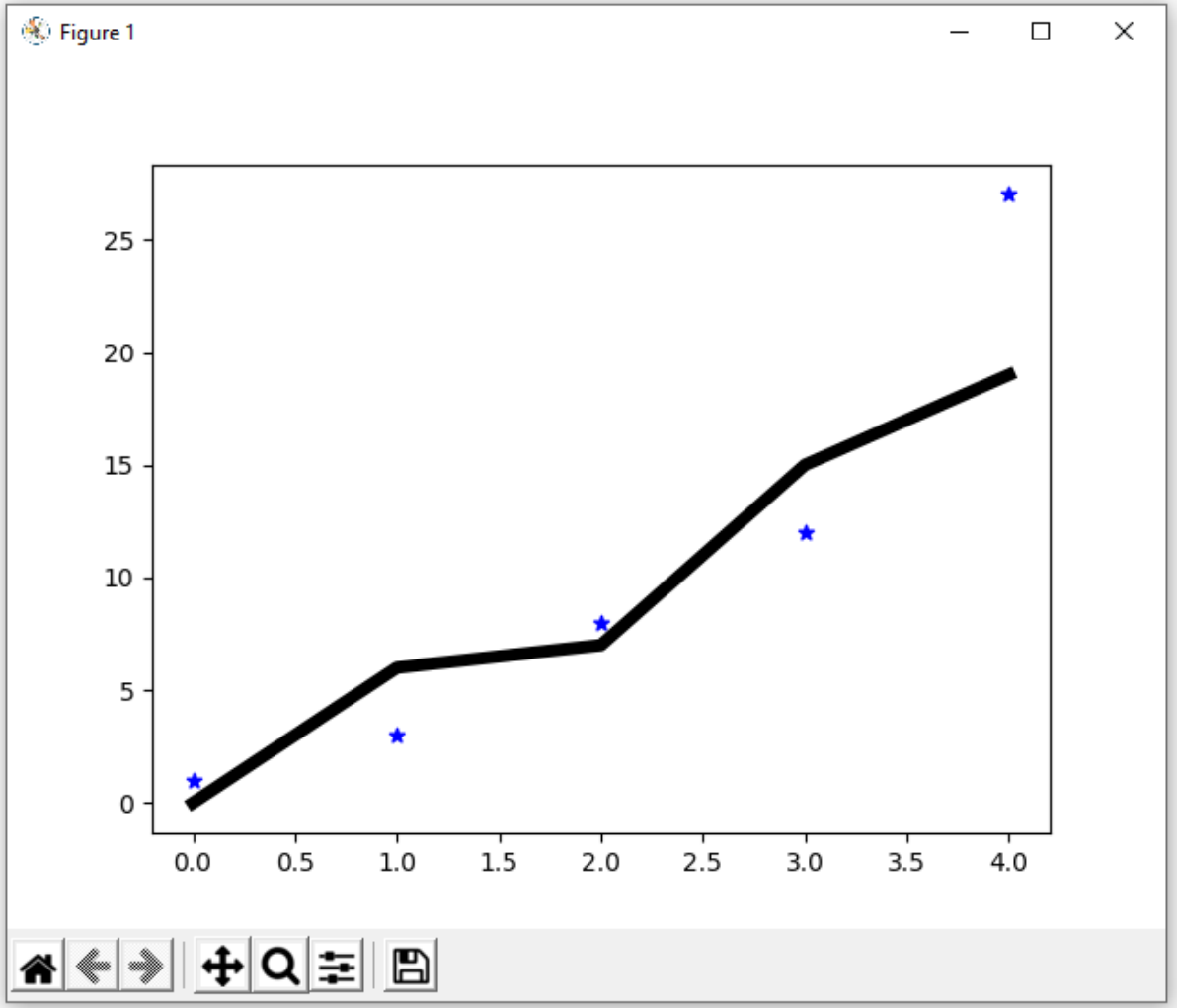
Намальовані дані також підтримують різні параметри зовнішнього вигляду:

```
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot([0, 1, 2, 3, 4], [0, 6, 7, 15, 19], color = 'black', linewidth = 5)
ax.scatter([0, 1, 2, 3, 4], [1, 3, 8, 12, 27], color = 'blue', marker = '*')

plt.show()
```



Вам доведеться вказати параметри зовнішнього вигляду (при необхідності) разом з даними всередині ax.plot() і ax.scatter(). Це суперечить тому, що було сказано вишче, але і в цьому випадку можна зробити код більш читабельним:

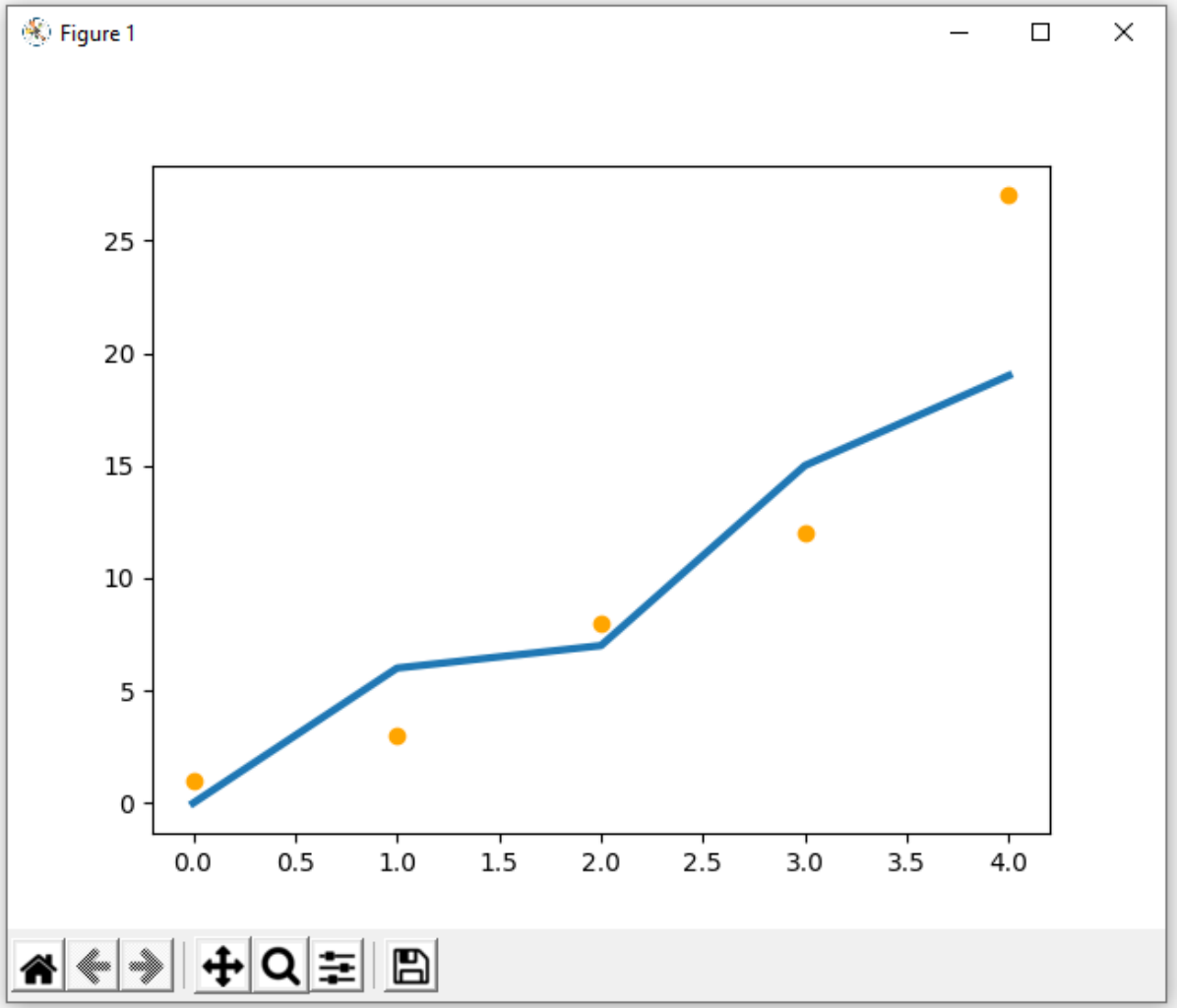
```
import matplotlib.pyplot as plt

x = [0, 1, 2, 3, 4]
y_1 = [0, 6, 7, 15, 19]
y_2 = [1, 3, 8, 12, 27]

fig = plt.figure()
ax = fig.add_subplot(111)

ax.plot(x, y_1,
        color = 'black',
        linewidth = 5)

ax.scatter(x, y_2,
           color = 'blue',
           marker = '*')

plt.show()
```



Методи Axes і pyplot

Тепер ви зрозумієте, як ми створювали діаграми без будь-яких Figure і Axes. Справа в тому, що практично всі методи axes присутні в модулі pyplot. Наприклад, під час виклику plt.title('spam'), модуль pyplot викликає ax.set_title('spam'). Можна сказати, що модуль pyplot автоматично створює Figure і Axes (хоча це не зовсім так). Насправді, ми можемо переписати весь наш приклад наступним чином:

```
import matplotlib.pyplot as plt


plt.plot([0, 1, 2, 3, 4], [0, 6, 7, 15, 19], linewidth = 3)
plt.scatter([0, 1, 2, 3, 4], [1, 3, 8, 12, 27], color = 'orange')

plt.show()
```

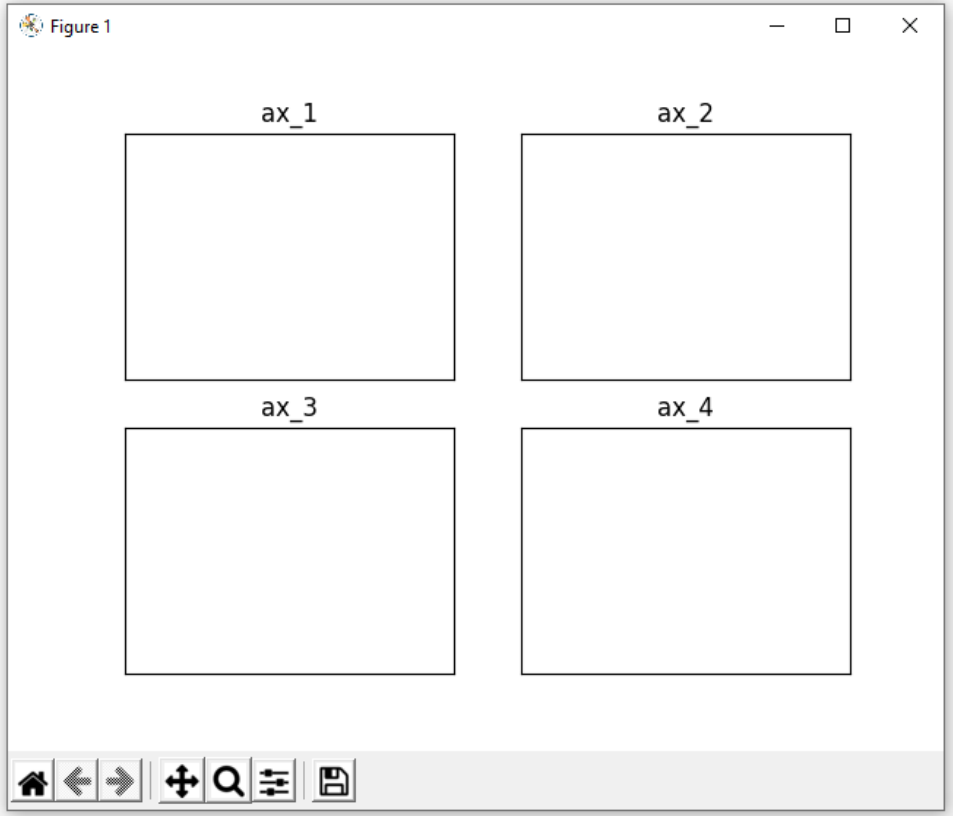


Якщо нам доведеться працювати з декількома областями *Axes* або доведеться створювати великі скрипти для побудови складної графіки, то використання явних визначень Figure і Axes зробить код більш очевидним і зрозумілим, навіть якщо це буде за рахунок збільшення його об’єму.

Кілька Axes на одній Figure

Дуже часто нам потрібно розмістити кілька графіків поруч один з одним. Найпростіше це зробити за допомогою plt.subplots(). Але давайте спочатку розглянемо наступний приклад:

```
import matplotlib.pyplot as plt

fig = plt.figure()

ax_1 = fig.add_subplot(2, 2, 1)
ax_2 = fig.add_subplot(2, 2, 2)
ax_3 = fig.add_subplot(2, 2, 3)
ax_4 = fig.add_subplot(2, 2, 4)

ax_1.set(title = 'ax_1', xticks=[], yticks=[])
ax_2.set(title = 'ax_2', xticks=[], yticks=[])
ax_3.set(title = 'ax_3', xticks=[], yticks=[])
ax_4.set(title = 'ax_4', xticks=[], yticks=[])

plt.show()
```



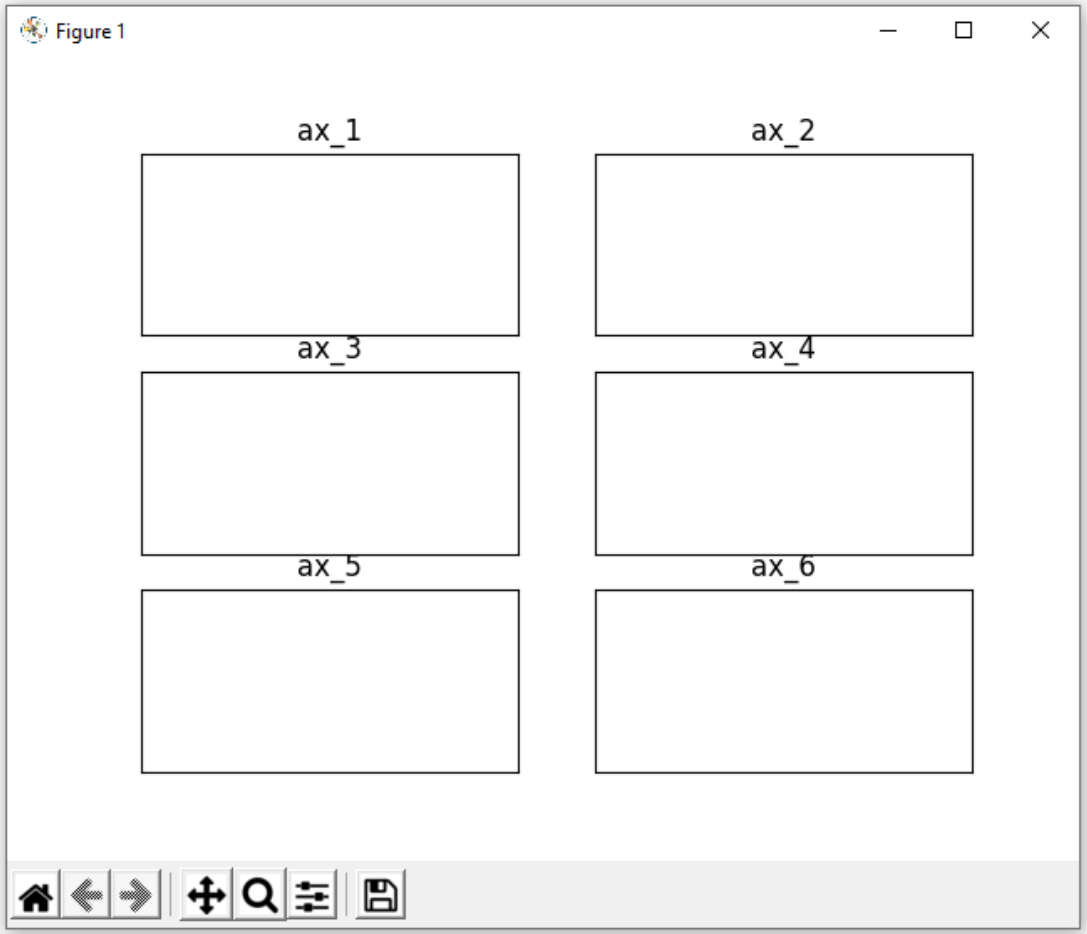
У цьому прикладі, як і раніше, ми спочатку створили область Figure, а потім використали команду fig.add_subplot(), щоб додати, один за одним, область Axes (ax_1, ax_2, ax_3, ax_4). І врахуйте, що кожна область Axes не залежить від інших, тобто на них можна намалювати різні графіки і встановити різні параметри зовнішнього вигляду.

Що робить метод add_subplot(). Він розбиває Figure на задану кількість рядків і стовпців. Після цього розбиття Figure можна представити у вигляді таблиці (або координатної сітки). Потім область Axes розміщується у вказаній клітинці. Для всього цього add_subplot() потрібно всього три числа, які ми передаємо йому як параметри:

→ перше, це кількість рядків;
→ друге, кількість стовпців
→ третє, індекс клітин.

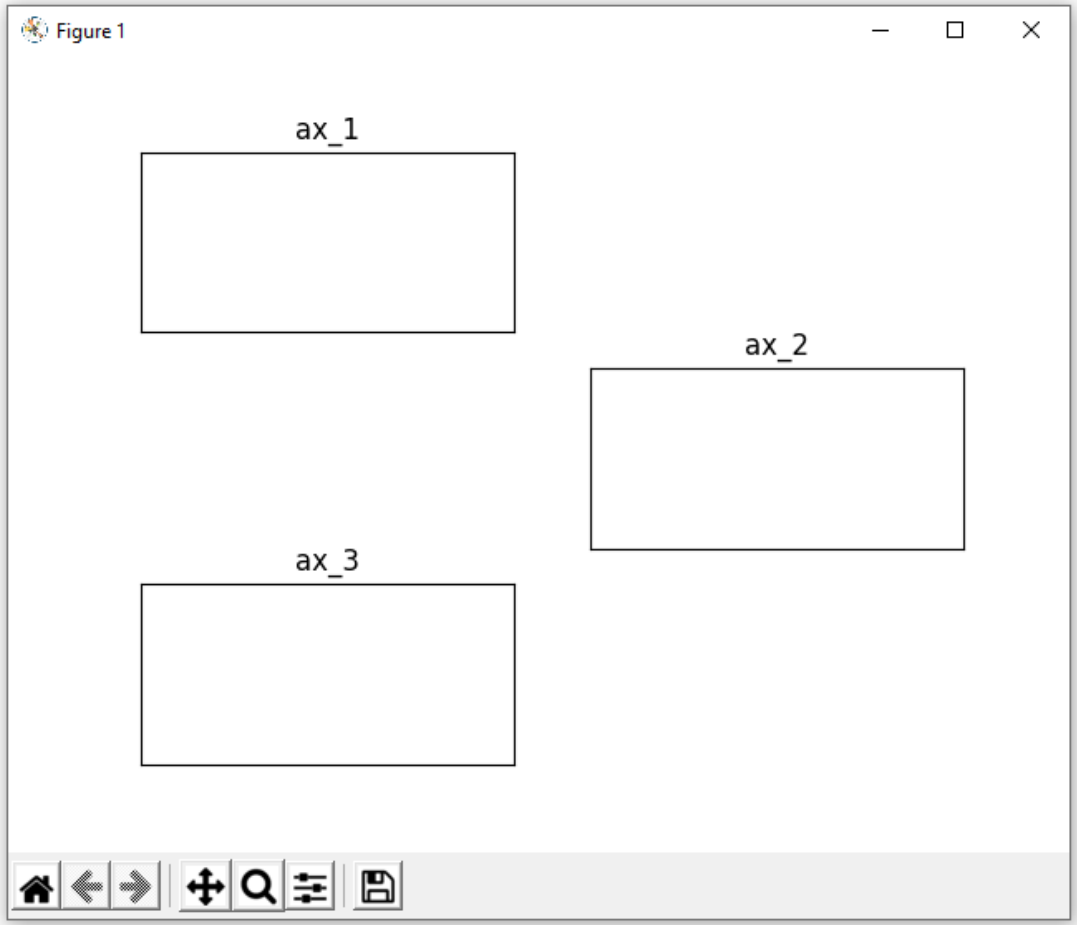
Індексація отриманих комірок починається з верхнього лівого кута, виконується рядок за рядком зліва направо і закінчується в правому нижньому куті:

```
import matplotlib.pyplot as plt

fig = plt.figure()

ax_1 = fig.add_subplot(3, 2, 1)
ax_2 = fig.add_subplot(3, 2, 2)
ax_3 = fig.add_subplot(3, 2, 3)
ax_4 = fig.add_subplot(3, 2, 4)
ax_5 = fig.add_subplot(3, 2, 5)
ax_6 = fig.add_subplot(3, 2, 6)

ax_1.set(title = 'ax_1', xticks=[], yticks=[])
ax_2.set(title = 'ax_2', xticks=[], yticks=[])
ax_3.set(title = 'ax_3', xticks=[], yticks=[])
ax_4.set(title = 'ax_4', xticks=[], yticks=[])
ax_5.set(title = 'ax_5', xticks=[], yticks=[])
ax_6.set(title = 'ax_6', xticks=[], yticks=[])

plt.show()
```



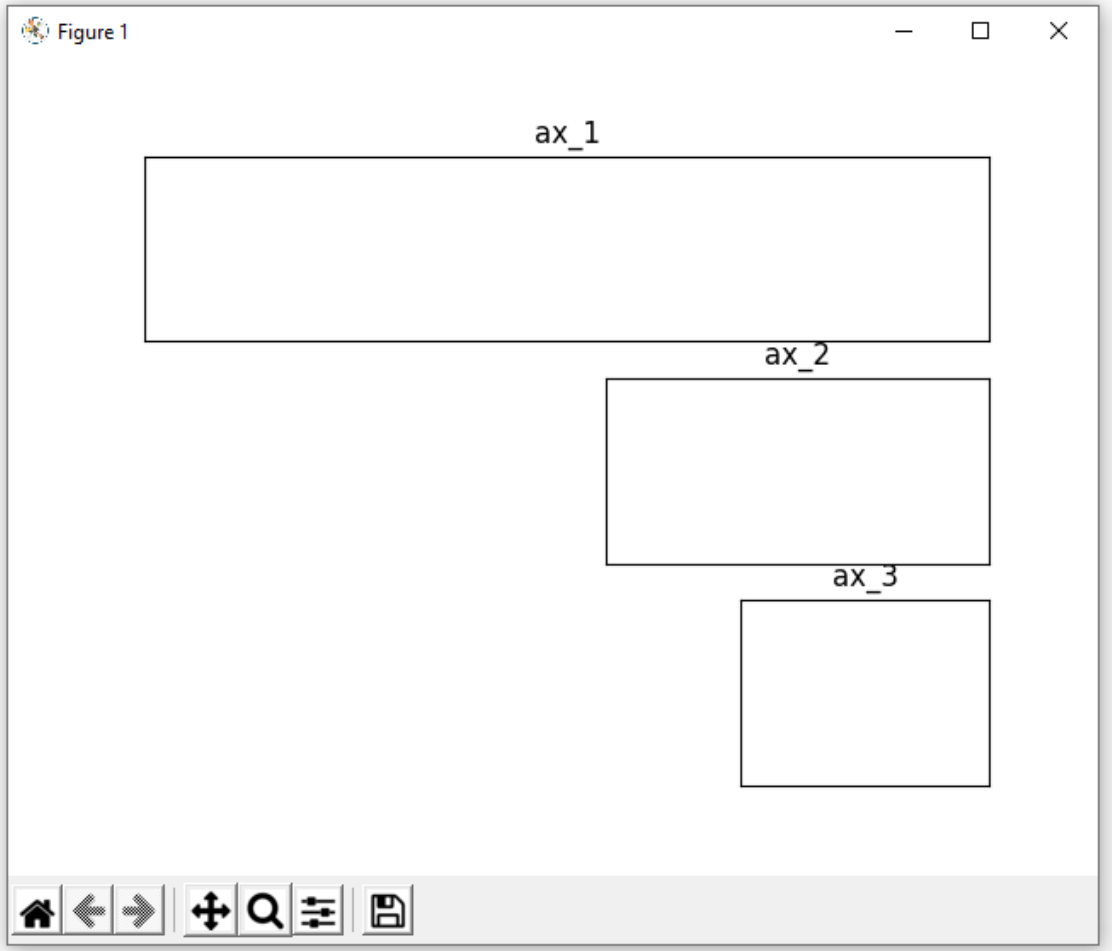
Не обов'язково заповнювати всю площу *Figure* областями *Axes* :

```
import matplotlib.pyplot as plt

fig = plt.figure()

ax_1 = fig.add_subplot(3, 2, 1)
ax_2 = fig.add_subplot(3, 2, 4)
ax_3 = fig.add_subplot(3, 2, 5)

ax_1.set(title = 'ax_1', xticks=[], yticks=[])
ax_2.set(title = 'ax_2', xticks=[], yticks=[])
ax_3.set(title = 'ax_3', xticks=[], yticks=[])

plt.show()
```



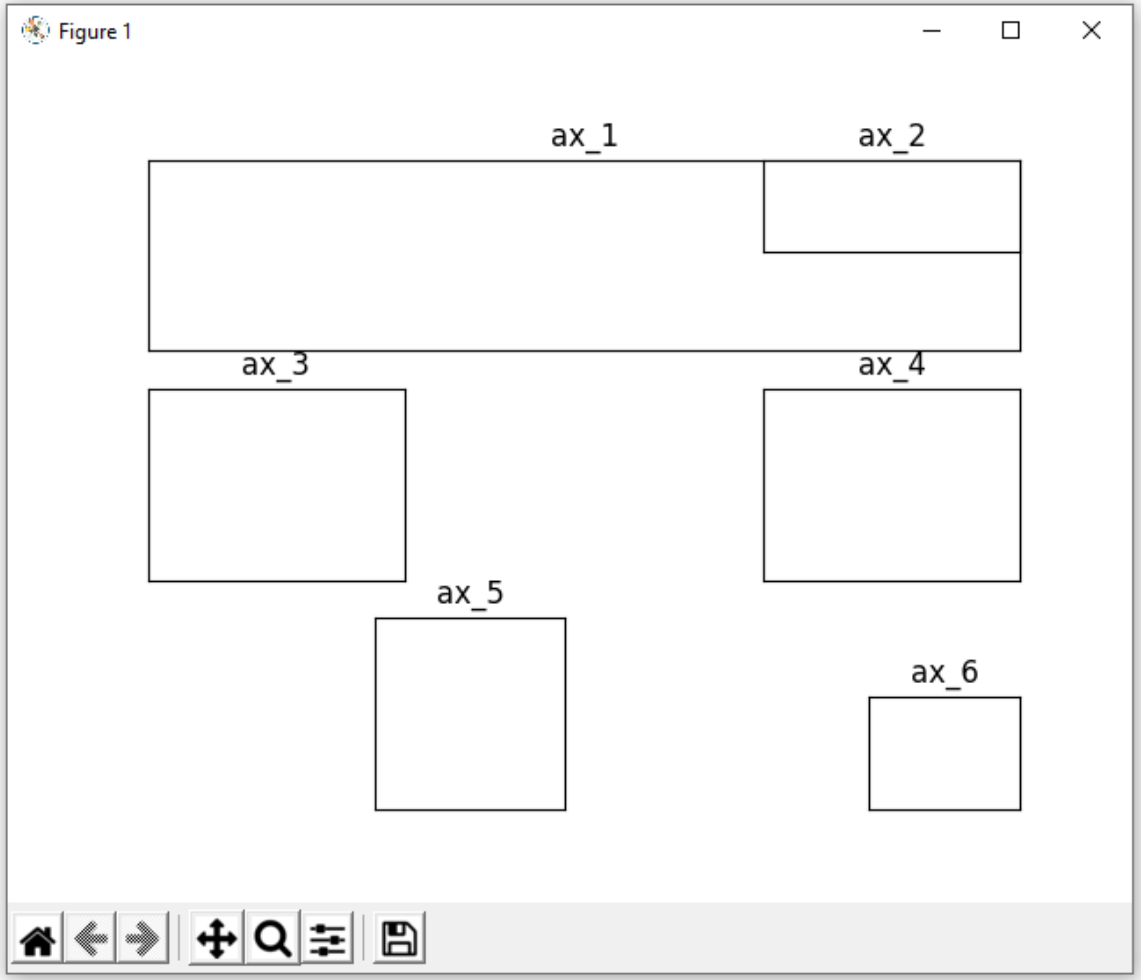
Кожен окремий виклик add_subplot() виконує розбивку Figure, як зазначено в його параметрах і не залежить від попередніх розбивок:

```
import matplotlib.pyplot as plt

fig = plt.figure()

ax_1 = fig.add_subplot(3, 1, 1)
ax_2 = fig.add_subplot(3, 2, 4)
ax_3 = fig.add_subplot(3, 3, 9)

ax_1.set(title = 'ax_1', xticks=[], yticks=[])
ax_2.set(title = 'ax_2', xticks=[], yticks=[])
ax_3.set(title = 'ax_3', xticks=[], yticks=[])

plt.show()
```



Така поведінка методу add_subplot() дозволяє розташувати графіки, як вам потрібно. Ділянки осей можуть перекриватися один з одним, бути різних розмірів або розділені якимось простором, а також розміщуватися в довільних місцях:

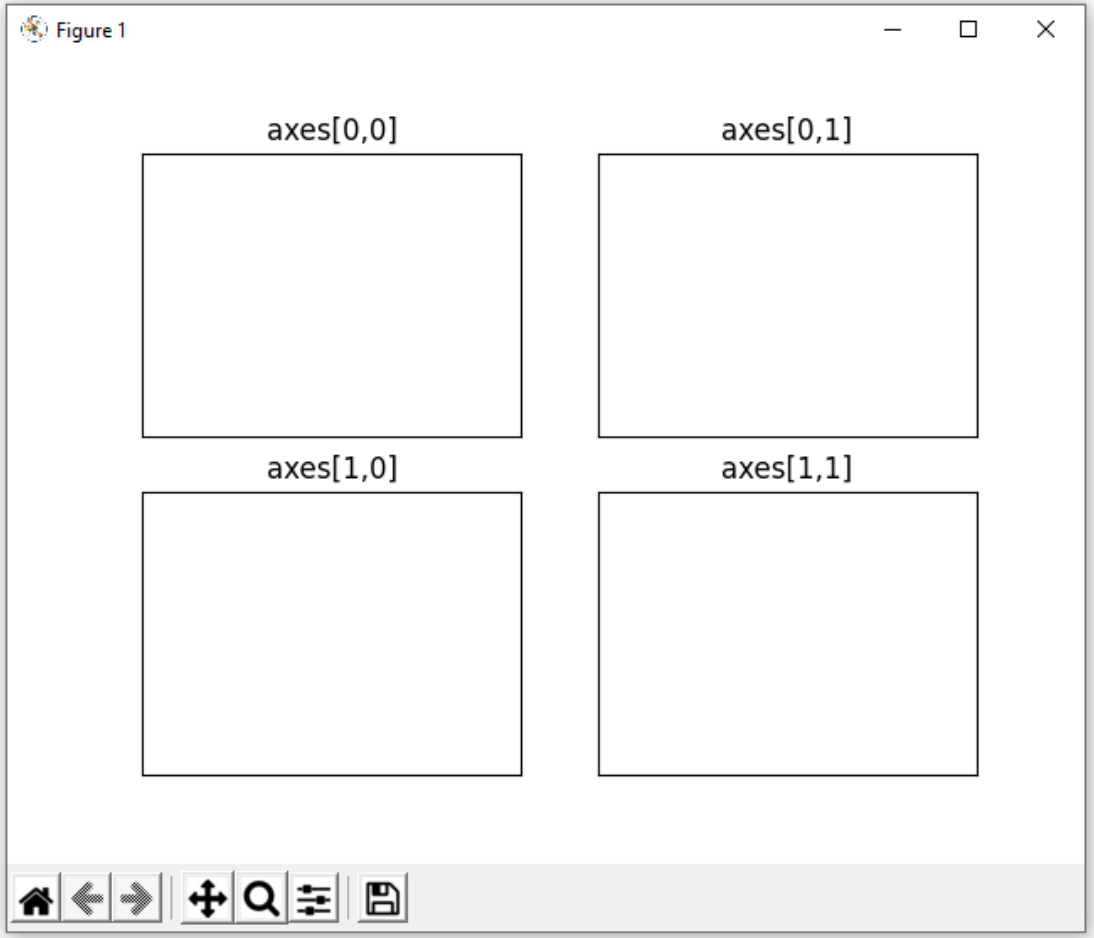
```
import matplotlib.pyplot as plt

fig = plt.figure()

ax_1 = fig.add_subplot(3, 1, 1)
ax_2 = fig.add_subplot(6, 3, 3)
ax_3 = fig.add_subplot(3, 3, 4)
ax_4 = fig.add_subplot(3, 3, 6)
ax_5 = fig.add_subplot(3, 4, 10)
ax_6 = fig.add_subplot(5, 5, 25)

ax_1.set(title = 'ax_1', xticks=[], yticks=[])
ax_2.set(title = 'ax_2', xticks=[], yticks=[])
ax_3.set(title = 'ax_3', xticks=[], yticks=[])
ax_4.set(title = 'ax_4', xticks=[], yticks=[])
ax_5.set(title = 'ax_5', xticks=[], yticks=[])
ax_6.set(title = 'ax_6', xticks=[], yticks=[])

plt.show()
```



subplots()

Звичайно, такий спосіб розміщення певної кількості ділянок Axes на Figure досить гнучкий, але на практиці функція plt.subplots(nrows, ncols) набагато зручніше:

```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(nrows = 2, ncols =2 )

axes[0,0].set(title='axes[0,0]')
axes[0,1].set(title='axes[0,1]')
axes[1,0].set(title='axes[1,0]')
axes[1,1].set(title='axes[1,1]')

for ax in axes.flat:
    ax.set(xticks=[], yticks=[])

plt.show()
```



Дуже часто нам потрібно щоб ділянки Axes бути розташовані на звичайної сітці. Це в основному те, що і робить plt.subplots(nrows, ncols). Однак не варто забувати, що, як правило, все трохи складніше. Виконаймо наступний код:

```
print(plt.subplots(nrows = 2, ncols = 2))
```

```
(<matplotlib.figure.Figure object at 0xaaca234c>, array([[<matplotlib.axes._subplots.AxesSubplot
object at 0xaad7946c>,
        <matplotlib.axes._subplots.AxesSubplot object at 0xab04a6ac>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0xaac8956c>,
        <matplotlib.axes._subplots.AxesSubplot object at 0xaaef0dac>]],
      dtype=object))
```

Якщо подивитися на вивід, то можна побачити, що plt.subplots(nrows, ncols) створює кортеж з двох елементів:

→ Область Figure;
→ Масив об'єктів NumPy, що складається з двох рядків і двох стовпців. Кожен елемент цього масиву представляє окрему область Axes, до яких можна отримати доступ за допомогою його індексу в цьому масиві.

Для подальшої роботи з цими областями нам потрібно розпакувати цей кортеж, тобто:

```
fig, axes = plt.subplots(nrows = 2, ncols = 2)
```

Тепер fig - це Figure, а axes – це масив NumPy, елементи якого є об'єктами Axes. Далі ми вирішили встановити кожній області Axes свій власний заголовок:

```
axes[0, 0].set(title='axes[0, 0]')
axes[0, 1].set(title='axes[0, 1]')
axes[1, 0].set(title='axes[1, 0]')
axes[1, 1].set(title='axes[1, 1]')
```

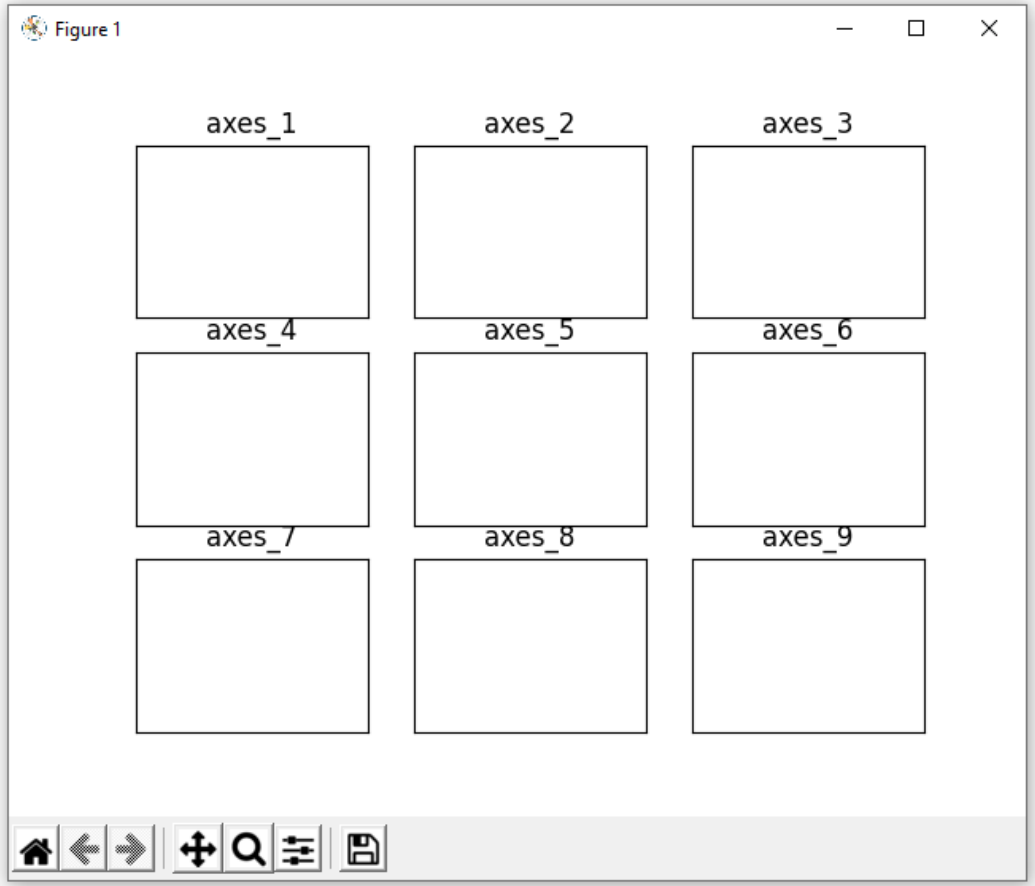
Оскільки кожен заголовок (як би) унікальний, нам довелося вручну викликати кожну область Axes і встановити її параметр title.

Але якщо нам потрібно встановити однакові параметри для кожної області, то це можна зробити в циклі:

```
for ax in axes.flat:
    ax.set(xticks=[], yticks=[])
```

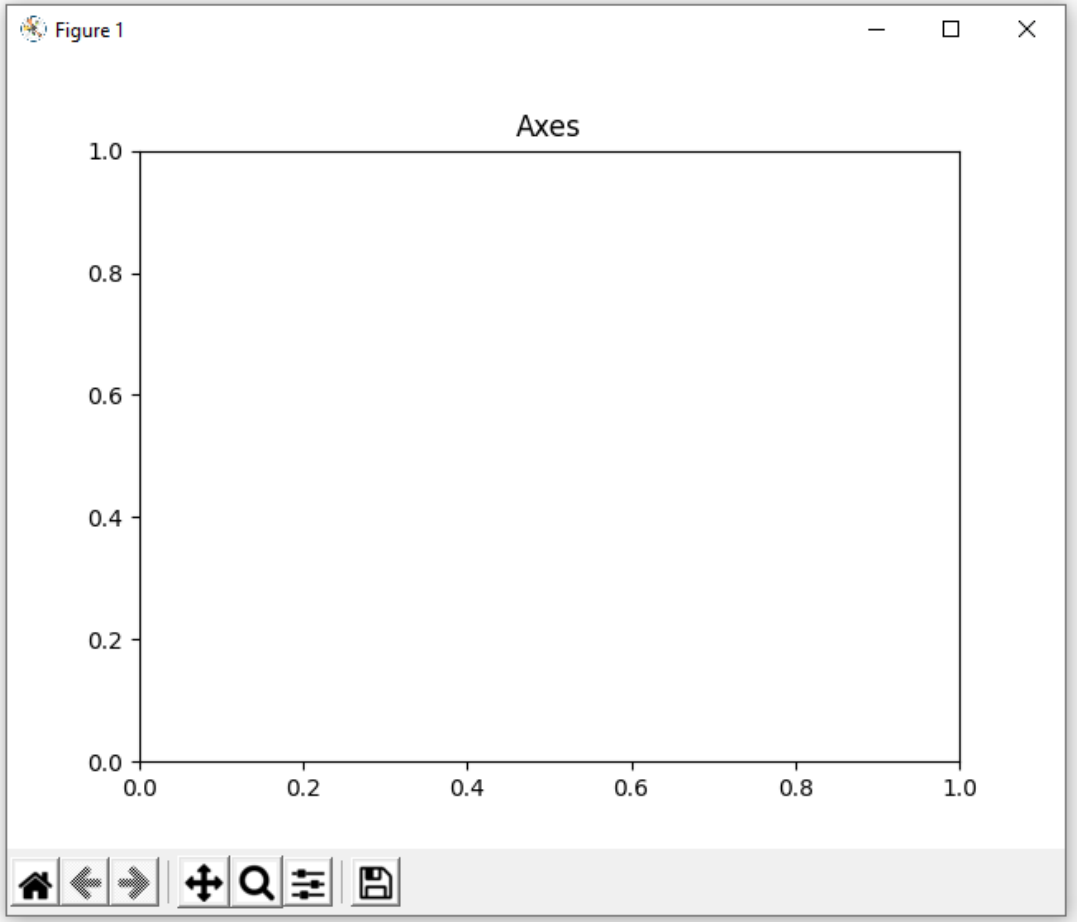
Хоча, часто, навіть заголовки можна встановити в тому ж циклі:

```
import numpy as np
import matplotlib.pyplot as plt

fig, axes = plt.subplots(nrows=3, ncols=3)

n = 1

for ax in axes.flat:
    ax.set(title='axes_' + str(n), xticks=[], yticks=[])
    n += 1
plt.show()
```



За замовчуванням кількість рядків і стовпців в методі subplots становить 1, що корисно для швидкого створення фігури з однією областю осей:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()        #  одна строка замість двох:
                                #  fig = plt.figure()
                                #  ax = fig.add_subplot(111)
ax.set(title='Axes')

plt.show()
```

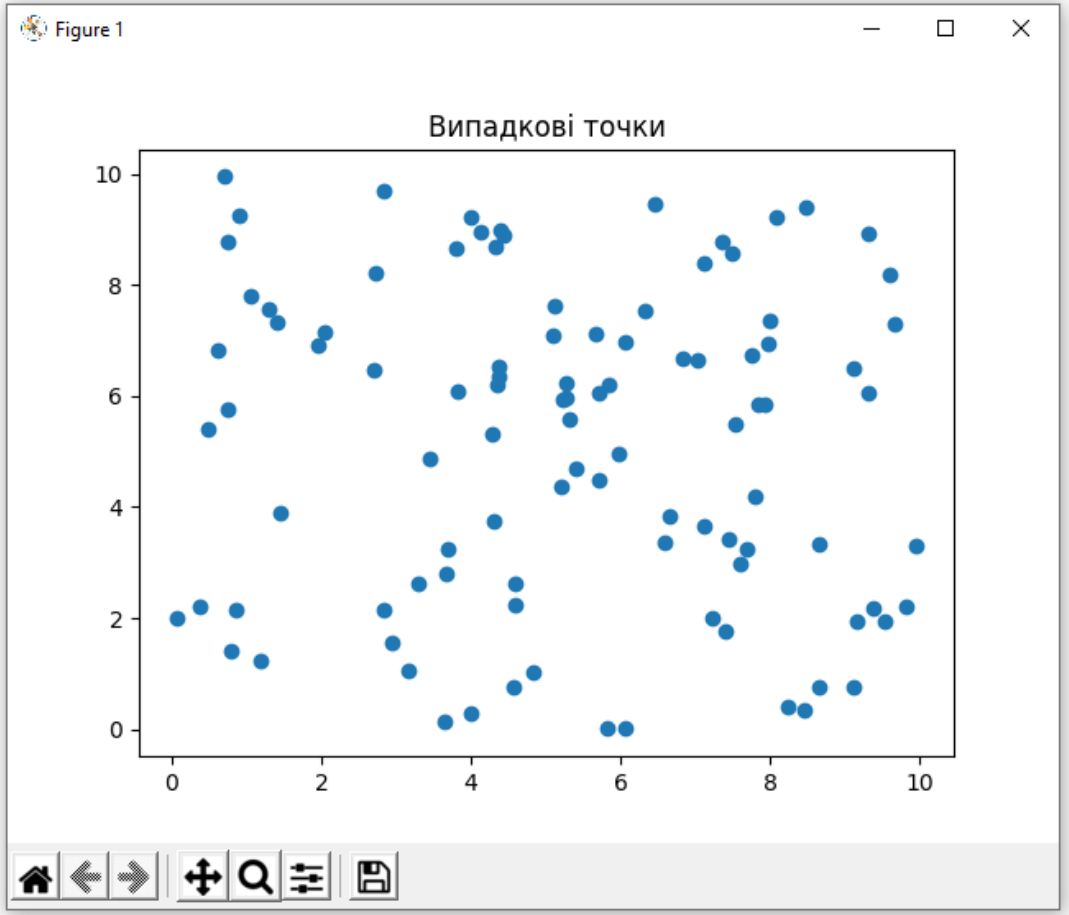


Далі в таких простих випадках ми дуже часто будемо використовувати саме цей рядок fig, ax = plt.subplots() - це скорочує код, але не зменшує його чіткості (так само створюємо Figure і ставимо на неї Axes).

Як будувати діаграми?

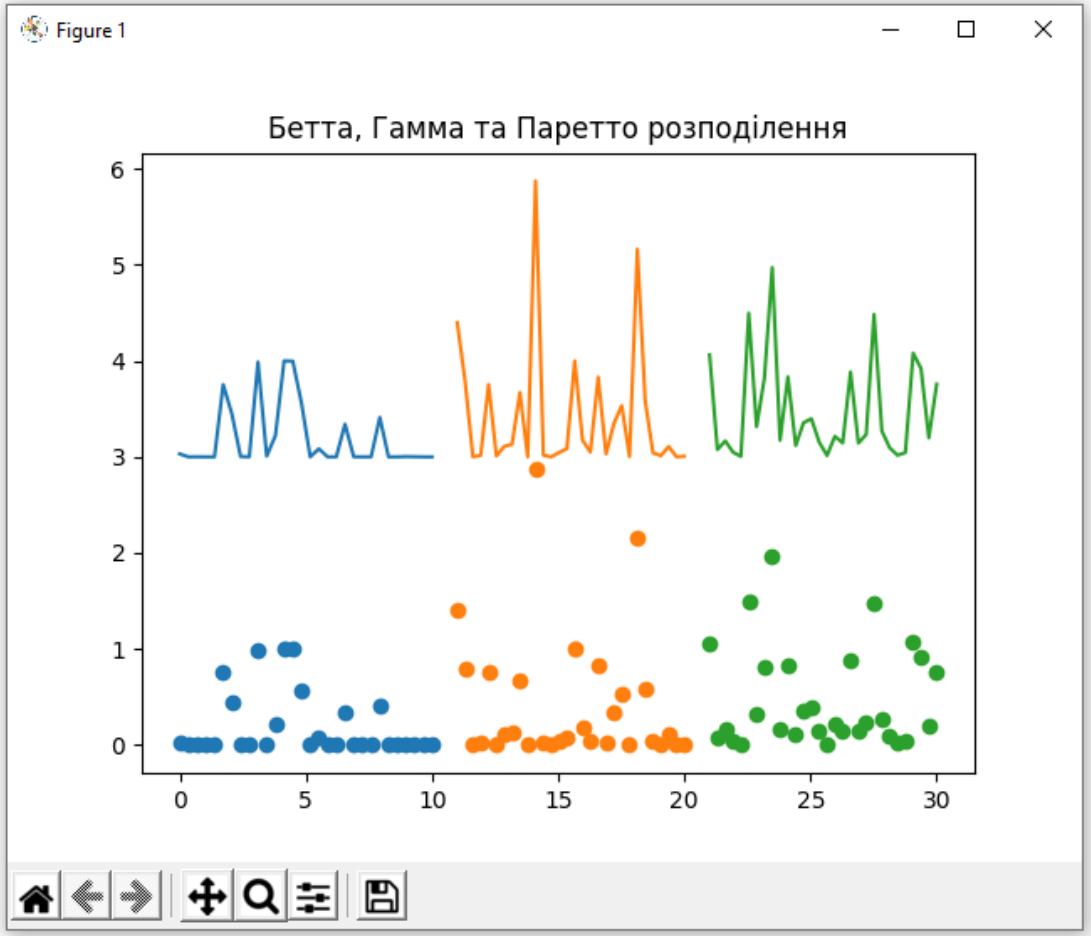
Як розмістити кілька діаграм разом (кілька областей *Axes* на *Figure*) ми розібралися. Після того як створена область *Axes*, яка по суті є об'єктом Python, ми можемо використовувати будь-який з методів цього об'єкта, включаючи ті, які відображають дані на цій області:

```
import numpy as np
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

#  Дані, які відображаємо :
x1 = 10*np.random.rand(100)     #  координати 'x'
y1 = 10*np.random.rand(100)     #  координати 'y'

ax.scatter(x1, y1)     #  метод, що відображаємо дані у вигляді точок
                       #  на площині

ax.set(title='Випадкові точки')     #  метод, що розміщує заголовок
                                    #  над "Axes"

plt.show()
```



У цьому випадку ми можемо відображати кілька наборів даних на одній області *Axes*, як однаковим, так і різними методами:

```
import numpy as np
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

x1 = np.linspace(0, 10, 30)
y1 = np.random.beta(0.1, 0.6, size = 30)
x2 = np.linspace(11, 20, 30)
y2 = np.random.gamma(shape = 0.3, scale = 1.1, size = 30)
x3 = np.linspace(21, 30, 30)
y3 =np.random.pareto(3.5, size = 30)

#  Дані у вигляді точок:
ax.scatter(x1, y1)
ax.scatter(x2, y2)
ax.scatter(x3, y3)

#  Дані у вигляді ліній:
ax.plot(x1, y1 + 3)
ax.plot(x2, y2 + 3)
ax.plot(x3, y3 + 3)

ax.set(title='Бетта, Гамма та Паретто розподілення')

plt.show()
```

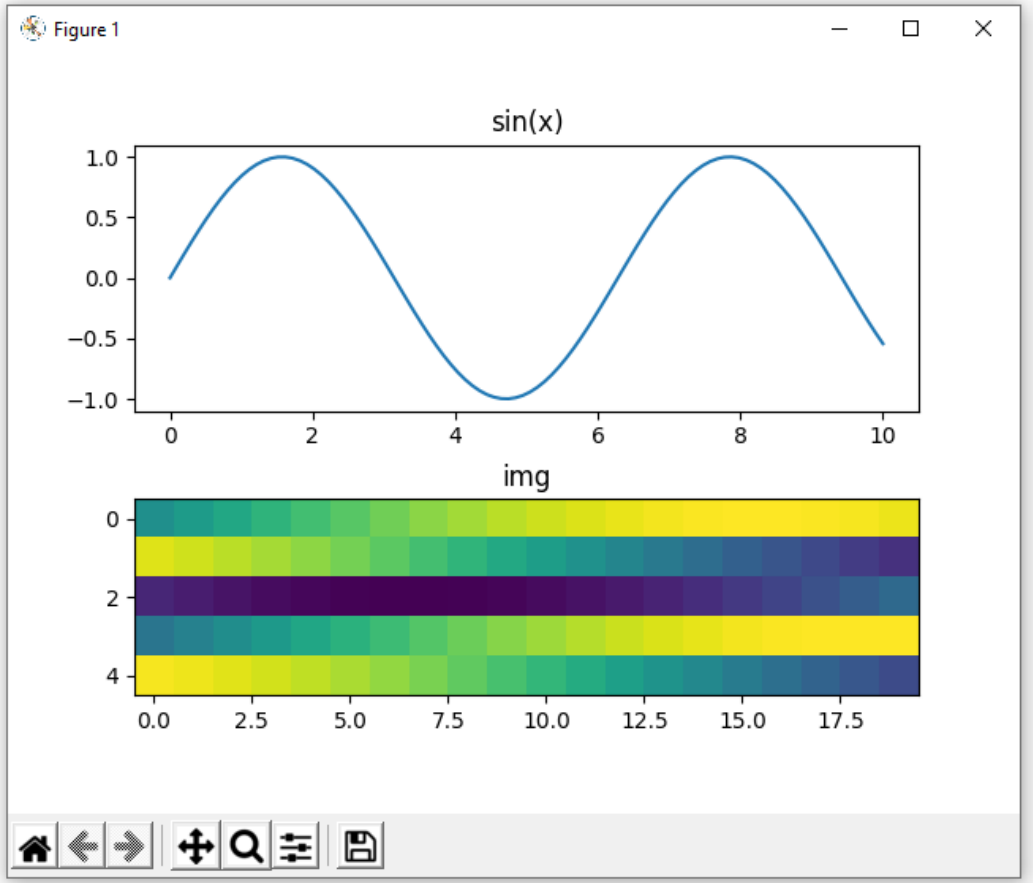


Функція **linspace** утворює лінійний масив еквівалентних вузлів. linspace(a,b,n) — генерує n точок рівномірно розподілених в діапазоні від a до b.

Функція B = **reshape**(A, m, n) повертає масив розміру m x n, утворений з елементів масиву A, послідовно вибираючи їх зі стовпців. Якщо кількість елементів у масиві A не є добуток m * n, відображається повідомлення про помилку.

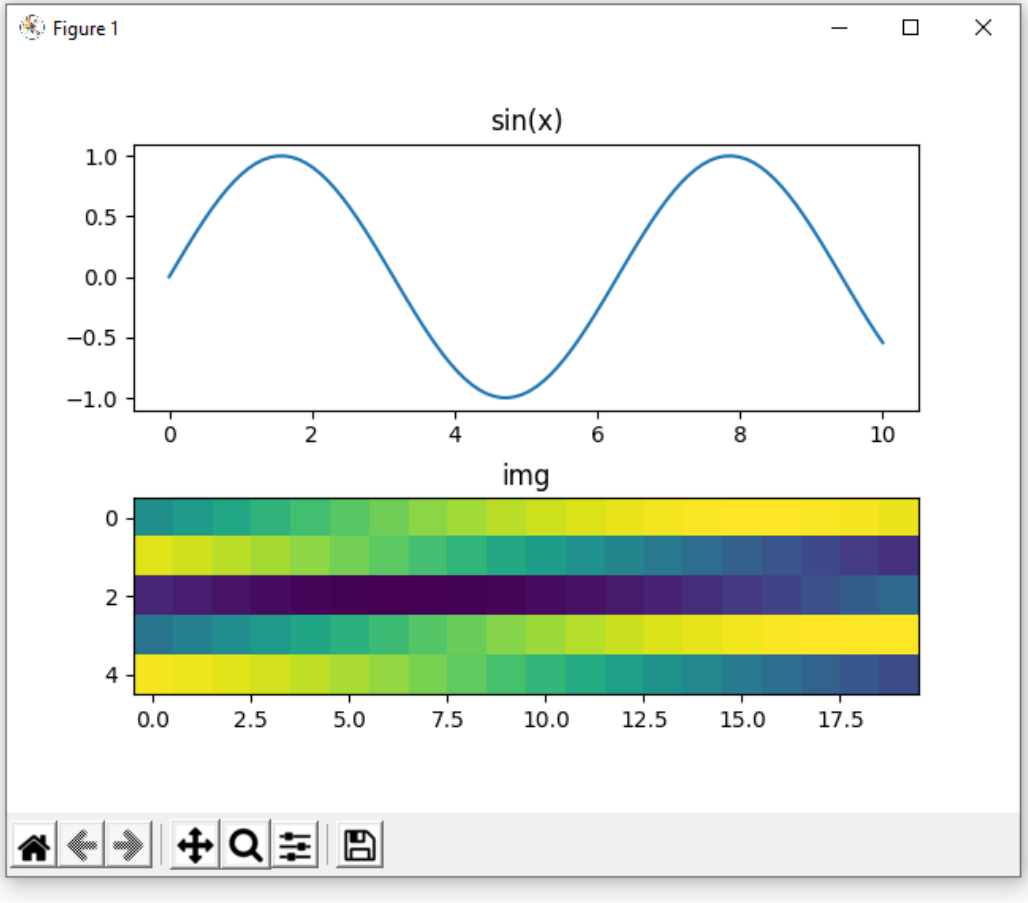
У тому випадку, коли у нас є кілька областей Axes, відображення даних на них нічим не відрізняється від випадку однієї області. Однак, якщо ви пам'ятаєте, області Axes можуть бути додані двома способами: add_subplot() і subplots(). Розглянемо випадок add_subplot():

```
import numpy as np
import matplotlib.pyplot as plt

#  Дані:
x = np.linspace(0, 10, 100)
y = np.sin(x)
img = y.reshape(5, 20)

#  створюємо "Figure" і "Axes":
fig = plt.figure()

ax_1 = fig.add_subplot(2, 1, 1)
ax_2 = fig.add_subplot(2, 1, 2)

#  Методи, що відображають дані:
ax_1.plot(x, y)
ax_2.imshow(img)

#  Додавання заголовків:
ax_1.set(title = 'sin(x)')
ax_2.set(title = 'img')

plt.show()
```



У випадку з subplots() все те ж саме, тільки до областей Axes звертаємося за індексом:

```
import numpy as np
import matplotlib.pyplot as plt

#  Дані:
x = np.linspace(0, 10, 100)
y = np.sin(x)
img = y.reshape(5, 20)

#  створюємо "Figure" і "Axes":
fig, axes = plt.subplots(nrows = 2, ncols = 1)

#  Методи, що відображають дані:
axes[0].plot(x, y)
axes[1].imshow(img)

#  Додавання заголовків:
axes[0].set(title = 'sin(x)')
axes[1].set(title = 'img')

plt.show()
```



І отримуємо той же результат: