

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

*Н.А. Андрущак*

## **МЕТОДИ ТА ЗАСОБИ КОМП'ЮТЕРНОГО НАВЧАННЯ**

### **ЛАБОРАТОРНИЙ ПРАКТИКУМ**

для студентів другого (магістерського) рівня вищої освіти  
спеціальності 122 «Комп'ютерні науки»  
спеціалізація «Системне проектування»

*Затверджено  
на засіданні кафедри «Системи  
автоматизованого проектування»  
Протокол N10 від 10.05.2018р.*

Львів 2018

**Андрущак Н.А. Методи та засоби комп'ютерного навчання:** лабораторний практикум для студентів другого (магістрського) рівня вищої освіти Інституту комп'ютерних наук та інформаційних технологій. – Львів: Видавництво Національного університету «Львівська політехніка», 2018. – 125 с.

В основу лабораторного практикуму покладено засвоєння теоретичних основ методів та засобів, які використовуються в комп'ютерному навчанні, розвинути навички студентів для кращого розумінні основних ідей, що лежать в основі цих методів, навчити студентів роботі з програмним забезпеченням, яке реалізує алгоритми машинного навчання. Розглянуто основні алгоритми та підходів до вирішення задач комп'ютерного навчання: лінійна регресія, метод найближчих сусідів, метод опорних векторів, кластеризація та здатність алгоритмів навчатися. Як основна мова програмування для реалізації алгоритмів комп'ютерного навчання використовується Python. Використання мови програмування Python, яка вважається однією з найкращих та широко вживаних для задач комп'ютерного навчання дозволяє ефективно впроваджувати нові рішення в прикладні сучасні системи. Відповідно, після виконання завдань в лабораторному практикумі студент навчиться виконувати складні алгоритми за допомогою комп'ютера, використовуючи мову програмування Python, а також знати як аналізувати та відображати результати таких обчислень та моделювання.

**Відповідальний за випуск**

Андрущак Н.А., к.т.н., асистент

**Рецензенти**

Щербовських С.В., д.т.н., доцент

Каркульовський В.І., к.т.н., доцент

## ЗМІСТ

<a href="#"><u>ЛАБОРАТОРНА РОБОТА №1</u></a> .....	4
на тему “Вступ в Python”	
<a href="#"><u>ЛАБОРАТОРНА РОБОТА №2</u></a> .....	26
на тему «Використання графічних модулів в Python»	
<a href="#"><u>ЛАБОРАТОРНА РОБОТА №3</u></a> .....	53
на тему «Побудова графіків в Python»	
<a href="#"><u>ЛАБОРАТОРНА РОБОТА №4</u></a> .....	79
на тему «Наукове представлення даних в Python»	
<a href="#"><u>ЛАБОРАТОРНА РОБОТА №5</u></a> .....	109
на тему «Методи класифікації: навчання з вчителем (лінійна регресія, k-найближчих сусідів, метод опорних векторів)»	

# ЛАБОРАТОРНА РОБОТА №1

на тему “Вступ в Python”

## 1. МЕТА РОБОТИ

Розглянути можливості для ініціалізації змінних, виконання арифметичних операцій, ввід та вивід результатів, написання та використання функції, а також використання циклів в програмі.

## 2. ТЕОРЕТИЧНА ЧАСТИНА

### 2.1. Змінні, ввід та вивід, цикли

#### Змінні

Ми використовуємо змінні для опису місця розташування в пам'яті комп'ютера, де ми зберігаємо дані для нашої програми. Для ініціалізації змінних в програмі на Python, вам достатньо просто ввести ім'я змінної та знак рівності, а потім деякий вираз, значення якого дорівнює початковому значенню змінної. Ось деякі приклади:

```
x = 2
mass = 12.3
speed_of_light = 299792458
energy = mass * (speed_of_light **2)
raining = True
cs_2016 = "Principles of Machine Learning"
```

Імена змінних повинні починатися з букви або символу нижнього підкреслення, після чого слідує будь-яка послідовність букв, цифр і символів підкреслення. Змінні чутливі до регістру, тому змінні salary та Salary – різні. Імена змінних не можуть містити пробіли. При необхідності, коли потрібно записати змінну, яка складається із декількох різних слів, потрібно використовувати символ нижнього підкреслення "\_". Приклад такого використання "New\_title".

Наведені вище інструкції називаються *операторами присвоювання*, так як вони присвоюють значення змінній. Всі оператори присвоювання мають одну змінну та знак рівності, який слідує за нею. Після знаку рівності іде значення даних або обчислень (виразів), які містять змінні і/або значення даних. Нижче наведені оператори, які можна використовувати для чисельних обрахунків:

+	додавання
-	віднімання
*	множення
/	ділення
%	залишок від ділення
**	піднесення до степеня

При введенні виразів з більше ніж однією арифметичною операцією, вони оцінюються на основі звичайного «математичного старшинства»: спершу піднесенням до степеня, далі множення, ділення та пошук залишку, і в самому

кінці додавання і віднімання.

Якщо є більше ніж один оператор того ж типу (наприклад, додавання і віднімання), вони обчислюються зліва направо. Якщо ви хочете, щоб нижча за пріоритетом операція виконувалася раніше, тоді потрібно використовувати дужки, як у звичайній математиці. Наприклад, нижче наведено Python твердження, що обчислює загальну вартість отримання кредиту (`total_amount`) з використанням відсотку від суми кредиту, яку позичили (`principal`), з врахуванням процентної ставки за місяць (`rate`) і час кредиту в місяцях (`month`):

```
total_amount = principal * (1 + rate * time)
```

## Ввід та вивід

Якщо ви хочете ввести числові дані у вашу програму з клавіатури, ви можете використовувати функцію введення `input()`:

```
tempF = input("Введіть температуру у Фаренгейтах: ")
```

Функція `input()` виводить рядок, щоб підказати користувачу про те, що потрібно ввести. Користувач вводить чисельне значення, яке потім зберігається у змінній `tempF`.

Для виведення значення змінної або виразу, ви просто використовуєте функцію `print()`. Ось чотири приклади використання цієї функції:

```
print ("The temperature in Fahrenheit: ")
print (tempF)
print ("The temperature in Celsius is ", (5.0/9.0)*(tempF - 32.0))
print (mass, " * ", speed_of_light, " ** 2 = ", energy)
```

У перших двох прикладах, є тільки один елемент, який виводиться в кожному рядку. В останніх двох прикладах, більш ніж один елемент виводиться на тій же лінії.

## Написання програм мовою Python

Звичайна програма на мові Python записується в такому вигляді:

```
def main():
    -----
    Python інструкції записуються в тіло функції
    -----
    <-- порожній рядок

main()
```

Інструкції програми зберігаються у оголошенні функції `def()` з іменем `main`. Остання команда запускає функцію `main()`, яка була визначена попередньо. Ось приклад короткої програми, яка перетворює введену температуру в градусах Фаренгейта у градуси Цельсія:

```
def main():
    tempF = int(input("Input the temperature in Fahrenheit: "))
    tempC = float((5.0/9.0) * (tempF - 32.0))
```

```
    print (tempF, " F = ", tempC, " C")
main()
```

Зверніть увагу, що інструкції в оголошенні функції *main()* все з відступом. Ми пропонуємо вам використовувати один відступ (Tab) для побудови всіх інструкцій в *main*. Крім того, як ми бачимо, змінна *tempF* отримує значення типу *int*, яке вводиться з клавіатури. Оскільки результат ділення 5 на 9 буде не ціле число, то змінна *tempC* буде приймати значення *float* для усіх виконаних операції.

## Цикли

Якщо ви хочете повторювати інструкції певну кількість разів, ви можете використовувати оператор *for*:

```
for i in range(10):
    print (i)
```

Цикл наведених вище значень змінної *i* буде змінюватися в діапазоні від 0 до 9, не включаючи 10. Іншими словами, цикл буде виконаний 10 разів, починаючи від 0. Результат виконання такого циклу показано нижче:

```
0
1
2
3
4
5
6
7
8
9
```

Проте, бувають випадки, коли потрібно вивести значення змінних в один рядок. Ось трохи інша версія використання циклу. Зверніть увагу на додаткову змінну *end=""* в кінці виразу *print*:

```
for i in range(10):
    print(i, end=" ")
```

Оскільки оператор *print* має *end=""*, курсор залишається на тій же лінії, щоб надрукувати наступний пункт. Ось новий вивід:

```
0 1 2 3 4 5 6 7 8 9
```

Зверніть увагу, що в циклі, набір інструкцій, які слідують одна за одною і мають відступ на однакову відстань повторюються в тілі цього циклу. Тобто, цикл може повторити більше однієї інструкції, в якості групи, як показано нижче:

```
for i in range(10):
    print(i, end="\t")
    print(i**2, end="\t")
    print(i**3)
```

Цикл складається з блоку або набору з 3-х інструкцій, які повторюються 10

разів. Цей цикл виводить таблицю квадратів і кубів цілих чисел від 0 до 9. Стрічка "\t" – спеціальний символ для одного натискання клавіші табуляції на клавіатурі. Оскільки в середині `end=" "` міститься символ "\t" в кінці перших двох тверджень, то курсор не переходить на нову стрічку, а залишається на тій же стрічці, поки ми не отримаємо в третій інструкції `print`. Інструкція(ї), які повторюються в циклі називаються *тілом циклу*. Ось вивід такого циклу:

0	0	0
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729

Іноді у вас може виникнути ситуація, коли потрібно проводити повторне обчислення, але ви не знаєте, зокрема, як та скільки разів необхідно повторити обчислення. Для таких випадків, ефективним є використання циклу `while`. На основі умови, яку ви задали, цикл буде повторюватися до тих пір, поки умова виконується. Наприклад, вам необхідно роздрукувати таблицю наведену вище, поки значення кубу не перевищує 500. Ось один із способів, як можна це зробити:

```
i=0
while i**3 < 500:
    print(i, end="\t")
    print(i**2, end="\t")
    print(i**3)
    i = i + 1
```

Зверніть увагу, що ми повинні ініціалізувати значення `i` перед початком циклу, а також повинні додавати 1 до `i`, перш ніж ми почнемо нашу наступну ітерацію циклу. Цикл `for`, який ви бачили вище, робить це додавання автоматично, але цикл `while` має велику гнучкість і може працювати довільне число разів. Ось результат викання цієї програми:

0	0	0
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343

Розглянемо також оператори, які можна використовувати для порівняння

двох значень, що застосовують у циклах:

```
<    менше
>    більше
==   рівне
<=   менше-рівне
>=   більше-рівне
!=    не рівне
```

Представлені вище оператори називаються *операторами порівняння*, так як вони дозволяють визначити відношення однієї величини до іншої.

Як приклад показана програма на Python для обчислення найбільшого спільного дільника (НСД) двох цілих чисел:

```
def main():
    x = int(input("Input the first integer: "))
    y = int(input("Input the second integer: "))
    print ("The GCD of ", x, " and ", y, " is ")
    while y != 0:
        x_prime = y
        y_prime = x % y
        x = x_prime
        y = y_prime
    print (x)
main()
```

Програма запитує значення для  $x$  і  $y$ , та повторює набір із 4 інструкцій циклу `while`, поки  $y$  не рівний 0, і виводить остаточне значення  $x$  в якості відповіді для розрахунку НСД. Символ `#` є коментар, який є просто повідомленням для нас, щоб пояснити мету програми. Текст після `#` в кожному рядку ігнорується інтерпретатором Python. Цікавий нюанс, який може вплинути на правильність виконання програми. Якщо написати коментар українською мовою, то незважаючи на те, що інтерпретатор не буде виконувати цей код, але покаже помилку виконання програми. На це потрібно зважати при написанні коду програми.

## 2.2. Використання оператора умови `if-else`

Розглянемо використання мови програмування Python для виведення календаря на один місяць по заданому  $n$  (значення між 28 і 31 включно, що представляє число днів у місяці) і  $d$  (а значення між 0 і 6 включно, що представляє початковий день місяця, де 0 = неділя, 1 = понеділок, і т.д.). Розпочнемо з цієї простої програми, яка запитує у користувача програми значення  $n$  і  $d$ , а потім виводить значення від 1 до  $n$ , використовуючи цикл:

```
def main():
    n = int(input("Input the number of days in the month (28-31): "))
    d = int(input("Input the starting day (0=Sun, 1=Mon,...):"))
```



```
i = 1
while i <= n:
    print (i)
    i = i + 1
main()
```

Нижче наведений вивід для  $n = 30$  та  $d = 4$ :

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

Як видно із виводу результату, безумовно, у нас присутні всі правильні цифри. Проте, це не подібно на форму представлення календаря. Нам необхідно мати числа, які ідуть горизонтально, тому ми модифікуємо програму, додавши додатковий оператор `end=" "` в кінці виводу, щоб курсор залишається на тій же лінії для наступного номера:

```
def main():
    n = int(input("Input the number of days in the month (28-31): "))
    d = int(input("Input the starting day (0=Sun, 1=Mon,...):"))
    i = 1
    while i <= n:
        print(i,end=" ")
        i = i + 1
main()
```

Нижче наведений вивід для  $n = 30$  та  $d = 4$ :

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

Тепер нам потрібно перемістити курсор на наступний рядок, коли ми дійдемо до кінця тижня. Для цього нам необхідно передбачити обчислення виразу  $(i + d)$  по модулю 7, щоб побачити, чи він дорівнює 0. Якщо це так, то ми переходимо до наступного рядка календаря. Ми можемо зробити це за допомогою оператора умови if:

```
def main():
    n = int(input("Input the number of days in the month (28-31): "))
    d = int(input("Input the starting day (0=Sun, 1=Mon,...):"))
    i = 1
    while i <= n:
        print(i,end=" ")
        if(i+d)%7==0:
            print("")
        i = i + 1
main()
```

Оператор умови if виглядає як умова while, але це не цикл. Він перевіряє дану умову і якщо вона вірна, то виконує інструкції, які наведена нижче оператора if. У цьому випадку, команда з відступом є умовою друку пропуску (""),. Зверніть увагу, що немає оператора end=" " в кінці цього рядка, тому після того, як пропуск надрукується, курсор перейде до наступного рядка. Крім того, в операторі умови if, ми повинні додати круглі дужки до виразу  $i + d$ , так як ми хочемо порахувати їх суму перш ніж ми виконаємо операцію залишку від ділення. Операція залишку від ділення зазвичай має більш високий пріоритет, ніж додавання, так як і множення та ділення. Зверніть увагу, що ми використовуємо подвійний знак рівності ==, не звичайний знак рівності =, так як ми хочемо перевірити на рівність, а не робити операцію присвоювання.

Ось вивід результатів для  $n = 30$  та  $d = 4$ :

```
1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
```

Одна з проблем, яку ми бачимо при виводі, що деякі дати мають одну цифру, а деякі дві. Це призводить до того, що числа виводяться не рівно, тобто не одна під одною. Ми можемо виправити це через додавання додаткового простору, якщо число менше 10:

```
def main():
    n = int(input("Input the number of days in the month (28-31): "))
    d = int(input("Input the starting day (0=Sun, 1=Mon,...):"))
    i = 1
    while i <= n:
        if(i<10):
            print("",i,end=" ")
        else:
            print (i,end=" ")
        if(i+d)%7==0:
            print("")
        i = i + 1
    main()
```

По-перше, ми використали нову версію оператора умови if, яку ми називаємо if-else умовою. Це дозволяє нам визначити, що робити, якщо умова if вірна, і що робити, якщо умова не є вірна (else). У цьому випадку, якщо i менше 10, воно має тільки одну цифру, тому ми повинні друкувати додатковий пробіл перед значенням i. В іншому випадку (else), ми друкуємо значення i, так як ми робили раніше. Особливість Python полягає в тому, що при друці будь якого тексту і залишивши курсор на тій же лінії, додаткові пропуски теж виводяться. Ці пропуски відокремлюють дані, що в тому ж рядку. Якщо ми хочемо додатковий пропуск перед однією цифрою номера, ми можемо вивести порожній рядок "", який нічого не виводить, але викликає додатковий пропуск для виводу, як і інші виводи. Порожній рядок є рядком, який нічого не містить, включаючи пробіли.

Ось вивід результатів виконання програми для n = 30 та d = 4:

```
1  2  3
4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
```

Проте, як видно із попереднього прикладу, перші три цифри не починаються в правильному положенні. Значення d показує нам, скільки стовпців ми повинні пропустити, щоб дістатися до правильної стартової позиції для першого рядка в календарі. Кожен стовпець містить два символи (дві цифри або пробіл і цифру), тому нам потрібно вивести d копії " " перш, ніж ми виведемо перший номер. Ось програма для цього:

```
def main():
    n = int(input("Input the number of days in the month (28-31): "))
    d = int(input("Input the starting day (0=Sun, 1=Mon,...):"))
    for j in range(d):
        print(" ",end=" ")
    i = 1
    while i <= n:
        if(i<10):
            print("",i,end=" ")
        else:
            print (i,end=" ")
        if(i+d)%7==0:
            print("")
        i = i + 1
    main()
```

Нагадаємо, що цикл `for` ініціалізує цикл, який повторюється стільки разів, які визначені у значенні діапазону. Отже, цей цикл повторюється `d` разів, виводячи щоразу два пропуски. (Звичайно, після першого виводу, кожен подальший вивід отримує додатковий пропуск у зв'язку з особливістю Python описаною вище). Якщо користувач вводить 0 як значення `d`, то цикл `for` спрацює 0 раз, тобто він пропускається.

Ось вивід результатів виконання для `n = 30` та `d = 4`:

```

          1  2  3
    4  5  6  7  8  9 10
   11 12 13 14 15 16 17
   18 19 20 21 22 23 24
   25 26 27 28 29 30
```

Підсумовуючи, може стверджувати про те, що у нас є правильний календар на місяць, який має 30 днів (`n = 30`), що починається в четвер (`d = 4`). Спробуйте програму для інших допустимих значень `n` і `d`, щоб побачити, що він працює правильно.

### 2.3. Використання масивів та списків

У мові Python, масив реалізований у вигляді списку даних. Ми розглянемо масиви, використовуючи два алгоритми: знаходження максимуму і бульбашкове сортування.

#### Пошук максимуму

Для ініціалізації масиву достатньо записати вираз `A = []`, де `A` – назва масиву. Щоб створити вектор даних, ми можемо просто визначити змінну (скажімо, `A`), та список значень даних в дужках, що записується наступним чином:

```
A = [4, 7, 3, 9, 8]
```

Щоб відобразити вміст масиву, ми можемо просто вивести масив:

```
print(A)
```

Масив має  $n$  елементів, і ми використовуємо значення  $n$ , щоб визначити, скільки разів виконати цикл чи іншу операцію. Python має функцію під назвою *len()*, що дозволяє нам визначити довжину масиву (тобто скільки елементів він має). Таким чином *len(A)* для масиву *A* поверне значення 5.

Отже, реалізуємо алгоритм знаходження максимального значення в масиві даних з використанням засобів Python:

```
def main():  
    A = [4, 7, 3, 9, 8]  
    i = 0  
    max = A[0]  
    while i < (len(A)-1):  
        i = i + 1  
        if A[i] > max:  
            max = A[i]  
    print ("The maximum positive integer is", max)  
main()
```

Після виконання програми отримаємо :

```
The maximum positive integer is 9
```

Проте, чи можемо ми бути впевнені, що програма працює правильно? Коли кількість елементів в масиві невелика, то можна перевірити самому. Але це є не найкращий спосіб. Щоб отримати краще уявлення про те, що ми робимо, ми можемо додати умову налагодження, що виведе змінні в ключових точках алгоритму:

```
def main():  
    A = [4, 7, 3, 9, 8]  
    print(A)  
    i = 0  
    max = A[0]  
    print(i, " ", max)  
    while i < (len(A)-1):  
        i = i + 1  
        if A[i] > max:  
            max = A[i]  
        print(i, " ", max)  
    print ("The maximum positive integer is", max)  
main()
```

Після виконання програми отримаємо :

```
[4, 7, 3, 9, 8]
```

```
0 4
1 7
2 7
3 9
4 9
```

The maximum positive integer is 9

Це виглядає краще. Для кожного значення ми перевіряємо, чи воно більше за поточне значення максимального елементу. Якщо так, то це значення стає новим максимумом.

### Бульбашкове сортування

Для реалізації цього алгоритму, ми повинні мати справу з двома речами. По-перше, ми повинні виконати цикл певну кількість разів. Пам'ятайте, що цикл `for` може бути використаний для цього. Якщо ми хочемо повторити якусь дію  $n$  разів, ми можемо записати це як:

```
for x in range(n):
```

Це створює змінну  $x$ , яка ітерується від 0 до  $n-1$ , по одному для кожної ітерації. У цьому прикладі, ми не використовуємо  $x$  в наших обчисленнях; це просто лічильник кількості ітерацій. В цілому, ми могли б використовувати змінну  $x$  як частина нашого розрахунку, якщо ми її потребуємо.

Подібну операцію можна використовувати, якщо ми хочемо виконати декілька ітерацій, які відповідають кількості елементів масиву. Для прикладу, виведемо значення масиву  $A$  в стовпчик. Це можна зробити за допомогою такого коду:

```
A = [1, 3, 7, 5, 4]
for x in range(len(A)):
    print(A[x])
    x = x + 1
```

Наступна дія, яку потрібно реалізувати в алгоритмі сортування, це обмін  $A[i]$  та  $A[i+1]$  при необхідності. У Python немає команди такого обміну. Проте, ми можемо зробити це, використовуючи таку послідовність кроків:

```
temp = A[i]
A[i] = A[i+1]
A[i+1] = temp
```

Звідси, можна записати програму, яка буде сортувати елементи масиву від найменшого до найбільшого:

```
def main():
    A = [8, 4, 9, 3, 7, 11, 2]
    print("Before sort: ", A)
    for j in range(len(A)):
        for i in range(len(A)-1):
```

```

        if A[i] > A[i+1]:
            temp = A[i]
            A[i] = A[i+1]
            A[i+1] = temp
        i = i + 1
    print ("After sort: ", A)
main()

```

Зверніть увагу на використання умови виводу, ми можемо відобразити вміст масиву до і після виконання сортування. Ось вивід цієї програми:

```

Before sort: [8, 4, 9, 3, 7, 11, 2]
After sort: [2, 3, 4, 7, 8, 9, 11]

```

## 2.4. Підпрограми та функції

### Функції

Якщо ми будемо продовжувати писати більш складні програми і запишемо всі наші оператори в тілі функції *main()*, то наші програми стануть занадто довгими і складними для читання і редагування. В Python, ми можемо написати додаткові визначення функцій, які будуть виконувати деякі дії для нас. Розглянемо використання функцій на прикладі виведення тексту пісні "З Днем Народження" для Аліси наступним чином:

```

def main():
    print ("Happy Birthday to you!")
    print ("Happy Birthday to you!")
    print ("Happy Birthday, dear Alice!")
    print ("Happy Birthday to you!")
main()

```

Як бачимо, в попередньому коді існує повторення однотипного рядка, проте цей рядок розміщується в різних частинах програми. Ми повторюємо "З Днем Народження тебе" три рази, але не один за іншим, так що ми не можемо просто написати цикл, що повторить вивід 3 рази. Проте, ми можемо написати іншу функцію, яка виконуватиме роль виводу повідомлення "З Днем Народження тебе". Тоді ми можемо просто викликати цю функцію три рази у відповідному місці в *main*:

```

def main():
    printWish()

def printWish():
    happy()
    happy()
    print ("Happy Birthday, dear Alice!")
    happy()

```

```
def happy():  
    print ("Happy Birthday to you!")
```

```
main()
```

В результаті виконання коду програми, ми отримаємо вивід привітання аналогічного попередньому коду. Проте, скажімо, ми хочемо надрукувати текст привітання для Аліни і Тараса. Найлегший спосіб зробити це за допомогою використання функції з параметром. Параметром є значення, яке вставляється в функцію, і яке використовується для виводу результату. В даному випадку параметром функції є ім'я людини.

```
def main():  
    printWish("Alice")  
    print("")  
    printWish("Bob")  
  
def printWish(person):  
    happy()  
    happy()  
    print("Happy Birthday, dear", person + "!")  
    happy()
```

```
def happy():  
    print("Happy Birthday to you!")  
main()
```

У наведеному вище прикладі, у функції *main()* спершу викликається функція *printWish()*, яка має слово "Alice" в якості вхідного параметра. В свою чергу, функція *printWish()* присвоює параметру *person* значення "Alice" і починаю покроково виконувати набір операторів прописаних в функції. Коли ми доходимо до 3-го рядка, *person* визначає Alice і виводить ціле значення рядка. Для того аби додати в кінці до *person* знак оклику, необхідно використати знак +. Наступний раз, коли функція *main()* звертається до функції *printWish()* аби вивести привітання для Bob, то принцип її роботи є аналогічний як для випадку з "Alice". В результаті виконання програми отримаємо ось такий вивід:

```
Happy Birthday to you!  
Happy Birthday to you!  
Happy Birthday, dear Alice!  
Happy Birthday to you
```

```
Happy Birthday to you!  
Happy Birthday to you!  
Happy Birthday, dear Bob!  
Happy Birthday to you!
```



Однією з переваг програмування з використання функцій є те, що, якщо ми хочемо зміни мову виводу тексту на іспанську, нам не потрібно змінювати велику кількість рядків, а лише деякі:

```
def main():
    printWish("Alice")
    print("")
    printWish("Bob")

def printWish(person):
    happy()
    happy()
    print("Feliz cumpleaños a", person + "!")
    happy()

def happy():
    print("Feliz cumpleaños a ti!")
main()
```

В результаті виконання написаної вище програми, буде виведено аналогічне привітання для Alice та Bob, але на іспанській мові.

### Функції, які повертають значення

Нагадаємо, що ми уже написали програму для перетворення температури з Фаренгейт в Цельсії. Скажімо, нам необхідно вивести таблицю з температурами в градусах Фаренгейт від 0 до 100 з кроком в 10 градусів, разом з їх еквівалентами за Цельсієм. По-перше, ми можемо написати функцію, яка перетворює температуру з Фаренгейт в Цельсії:

```
def f_to_c(tempF):
    tempC = (5.0/9.0) * (tempF - 32.0)
    return tempC
```

Зверніть увагу, що вище дана функції має один вхідний параметр tempF. Змінна tempF представляє температуру в градусах Фаренгейт. Описана функція *f\_to\_c()* використовується для обчислення відповідних градусів в Цельсіях. Остання умова в функції повертає цей результат як відповідь функції. Тепер ми можемо написати програму, яка використовує цю функцію, викликаючи її з кожним значенням температури за Фаренгейтом від 0 до 100 з кроком в 10 градусів:

```
def main():
    print ("F", "\t", "C")
    for x in range(11):
```

```
print (x*10, "\t", f_to_c(x*10))
```

```
def f_to_c(tempF):  
    tempC = (5.0/9.0) * (tempF - 32.0)  
    return tempC  
main()
```

Цикл в *main()* виконається 11 разів, присвоюючи  $x = 0, 1, 2, \dots$  і так до 10. Кожен раз ми виводимо  $x*10$  (0, 10, 20, ..., 100) і тоді викликаємо функцію *f\_to\_c()* зі значенням  $x*10$ . Кожен раз як ми викликаємо функцію *f\_to\_c()*, значення  $x*10$  зберігається в *tempF* для його використання у функції. Результат виконання програми буде такий:

F	C
0	-17.77777777777778
10	-12.222222222222223
20	-6.666666666666667
30	-1.1111111111111112
40	4.444444444444445
50	10.0
60	15.555555555555557
70	21.111111111111111
80	26.666666666666668
90	32.22222222222222
100	37.77777777777778

Як ми бачимо, такий вивід температури в градусах Цельсія не дуже зручний. Для того, щоб зробити його більш зручним, ми можемо використати функцію *round()*, яка округлить температуру за Цельсієм до найближчого цілого числа. Зверніть увагу, що *round()* аналогічна функціям, які ми пишемо, за винятком, що вона заздалегідь відома для нас. Наприклад, `print round(42.63)` викличе функцію зі значенням 42.63, і поверне 43, як її результат, який ми потім виведемо. Проте, якщо ми хочемо округлити наше число до 2 знаку після коми (розглянемо на прикладі числа 43.6345), то потрібно викликати функцію у такому вигляді: *round(43.6345, 2)*.

Нижче показана програма для перевodu градусів з використанням функції заокруглення до 2 знаку після коми.

```
def main():  
    print ("F", "\t", "C")  
    for x in range(11):  
        print (x*10, "\t", f_to_c(x*10))  
  
def f_to_c(tempF):  
    tempC = round((5.0/9.0) * (tempF - 32.0),2)
```

```
    return tempC
main()
```

Вивід виконання програми:

F	C
0	-17.78
10	-12.22
20	-6.67
30	-1.11
40	4.44
50	10.0
60	15.56
70	21.11
80	26.67
90	32.22
100	37.78

В попередньому завданні, ми написали програму, що знаходить максимальний елемент масиву цілих чисел. Програма має свої обмеження, оскільки вона працювала тільки з тим масивом, що заданий в програмі. Було б набагато краще, якби ми могли змінити програму так аби знаходити максимум у будь-якому масиві, який ми хочемо. Ми можемо зробити це, написавши функцію, яка пропонує користувачеві ввести значення даних для масиву. Це можна реалізувати за допомогою такого коду:

```
def getData(n):
    numbers = [] # empty array
    for x in range(n):
        print ("Enter value", x, ":", end="")
        value = input()
        numbers.append(value)
    return (numbers)
```

Функція `getData()` у якості вхідного параметра отримує число `n`, яке вказує на кількість елементів в масиві, що будемо сортувати. На початку роботи функції ініціалізується порожній масив `numbers`. Далі в циклі `n` раз користувача просять ввести значення для масиву. Кожне значення, яке вводиться користувачем додається в кінець масиву `numbers` за допомогою функції `append()`. Після того, як цикл виконається `n` разів, ми будемо мати масив `numbers` повністю заповнений вхідними числами.

Запропонована функція `getData()` може бути використана для модифікації нашої програми для пошуку максимуму, як наведено нижче:

```
def main():
    numValues = int(input("Enter the number of data values to analyze: "))
    A = getData(numValues)
```

```

i = 0
max = A[0]
while (i < (len(A)-1)):
    i = i + 1
    if A[i] > max:
        max = A[i]
print ("The maximum value is ", max)

```

```

def getData(n):
    numbers = [] # empty array
    for x in range(n):
        print ("Enter value", x, ":", end="")
        value = input()
        numbers.append(value)
    return (numbers)
main()

```

Спершу, в функції `main` (де починається програма), ми просимо користувача ввести число для кількості елементів масиву `numValues`. Припустимо, що елементи масиву, які будуть вводитися, це додатні числа. Після цього ми викликаємо функцію `getData()`, для якої вхідний параметр є значення `numValues`. Функція `getData()` отримує це значення замість параметра `n`, а потім працює як описано вище. У результаті масив, який створюється повертається до `main`, де `main` зберігає результат у змінній `A`. Після цього інша частина програми обчислює максимум, як ми робили раніше.

Ось приклад виконання програми:

```

Enter the number of data values to analyze: 5
Enter value 0 : 4
Enter value 1 : 8
Enter value 2 : 9
Enter value 3 : 5
Enter value 4 : 7
The maximum value is 9

```

Таким чином, написання коду в такий спосіб робить його більш читабельним, оскільки кожна функція має своє призначення. Функція `getData()` відповідає за ініціалізацію масиву, а функція `main()` відповідає за обчислення максимуму в масиві. Ми написали додаткові функції для того, щоб поділити обчислювальні частини нашої програми на логічні блоки, а також, щоб дозволити нам повторно використовувати код без копіювання знову і знову.

### **3. КОНТРОЛЬНІ ЗАПИТАННЯ**

1. Як відбувається ініціалізація змінних в Python?
2. Що таке оператори присвоювання?
3. Які способи вводу та виводу даних в Python ви знаєте?
4. Які оператори циклу в Python вам відомі?
5. Як працює оператор умови if-else?
6. Що таке масив? Способи ініціалізації масивів.
7. Використання функцій в Python.
8. Для чого потрібний оператор повернення? Приклад використання.

### **4. ЛАБОРАТОРНЕ ЗАВДАННЯ**

1. Ознайомитись з теоретичною частиною лабораторної роботи.
2. Одержати індивідуальне завдання (див. Варіанти індивідуальних завдань). При необхідності уточнити завдання на виконання у викладача.
3. Скласти програму на мові програмування Python відповідно до поставленого завдання. Виконана програма повинна повністю реалізувати розв'язок поставленої задачі.
4. Запустити програму та зберегти результат виконання.
5. Оформити звіт відповідно до встановлених вимог.

### **5. ЗМІСТ ЗВІТУ**

1. Мета роботи.
2. Відповіді на контрольні запитання.
3. Індивідуальне завдання, отримане у викладача, згідно з варіантом.
4. Код програми для реалізації завдання.
5. Аналіз отриманих результатів.
6. Ґрунтовні висновки.
7. Список використаної літератури.

### **6. СПИСОК ЛІТЕРАТУРИ**

1. *Лутц М.*, Изучаем Python. - К.:Символ-Плюс , 2011. - 1280 с.
2. *Бизли Д.*, Python. Подробный справочник, 4-е издание. - К.:Символ-Плюс, 2012. - 864 с.
3. Офіційний сайт Python: <https://www.python.org/doc/>

## ВАРІАНТИ ІНДИВІДУЛЬНИХ ЗАВДАНЬ

### Завдання №1

#### Варіант 1

1. Написати програму на Python, яка просить користувача ввести температуру в градусах Цельсія після чого перетворює і виводить температуру в градусах Фаренгейта. Використовуйте формулу наведену в прикладі вище.
2. Вивести значення парних чисел від 2 до 24. В кінці програми вивести суму парних чисел.

#### Варіант 2

1. Використовуючи мову програмування Python, написати програму для виведення значення температури в Фаренгейтах, якщо значення температури в Цельсіях змінюється від 20°C до 34°C, з кроком 2°C.
2. Вивести значення непарних чисел в діапазоні від 1 до 21. В кінці програми вивести добуток значень цих непарних чисел.

#### Варіант 3

1. Написати програму, яка зчитує значення довжини сторін трикутника, які вводяться користувачем і обчислює площу трикутника за формулою Герона ( $S = \sqrt{p(p-a)(p-b)(p-c)}$ , де  $p=(a+b+c)/2$ ,  $a$ ,  $b$ ,  $c$  – довжини сторін трикутника).
2. З використанням циклу while вивести значення чисел від 2 до 14, з кроком 2, в одній колонці, а в іншій колонці, значення цих чисел піднесених до 4 степені.

#### Варіант 4

1. Написати програму, яка обчислює об'єм кулі ( $V = \frac{4}{3}\pi R^3$ ). Користувач повинен вводити значення діаметру кулі.
2. Написати програму, яка просить користувача ввести значення числа з консолі після чого виводить значення 7 парних чисел, які ідуть після нього. В кінці програми вивести суму парних чисел.

#### Варіант 5

1. Написати програму, яка просить користувача ввести своє ім'я, після чого вона виводить це ім'я 5 разів з нового рядка і 3 рази в одному рядку.
2. Написати програму, яка просить користувача ввести значення додатного числа з консолі, після чого виводить усі значення непарних чисел, які ідуть від 0 і до цього числа. В кінці програми вивести суму цих чисел.

### Завдання №2

#### Варіант 1

1. Модифікуйте програму для виводу календаря так, щоб вивести календар для суспільства, що має 8 днів на тиждень.

2. Написати програму виводу на екран кожного третього числа, що лежить в діапазоні від -1 до -21.

#### *Варіант 2*

1. Змініть програму так, щоб вона вивела 5 календарів, запитуючи користувача значення  $n$  і  $d$  кожного разу перед виводом.

2. Написати програму виводу на екран кожного парного числа, що лежить в діапазоні від -3 до -25.

#### *Варіант 3*

1. Написати і запустити програму на Python, яка питає користувача рік і виводить "Високосний рік" або "Не високосний" залежно від року введення. Як ви знаєте, чи число ділиться на 4? Який залишок ділення високосного року на 4?

2. Напишіть програму, де користувача просять ввести число від 10 до 100. Якщо число менше нуля, то вивести повідомлення "Введіть додатне число", якщо число від 0 до 9, то вивести "Введіть число більше 10", а якщо більше 10, то вивести "Ви ввели число (номер числа введене користувачем)".

#### *Варіант 4*

1. Написати і запустити програму на Python, який запитує позитивне ціле число  $n$ , а потім виводить 50 зірочок,  $n$  в рядок. Останній рядок може не мати всі  $n$  зірочок.

2. Написати програму, яка запитує у користувача два числа. Якщо перше число більше другого, то обчислити їх різницю і вивести результат. Якщо друге число більше першого, то обчислити їх суму. Якщо два числа однакові, то вивести добуток цих чисел. Використовувати тільки 3 змінні.

#### *Варіант 5*

1. Модифікуйте програму для виводу календаря так, щоб вивести календар для суспільства, яке має 6 днів на тиждень.

2. Напишіть програму, де користувача просять ввести будь-яке число. Якщо число менше нуля, то вивести повідомлення "Число менше нуля", якщо рівне нулю, то вивести "Число рівне нулю", а якщо більше нуля, то вивести "Число більше нуля".

### **Завдання №3**

#### *Варіант 1*

1. Написати програму, яка знаходить мінімальний елемент в масиві з 15 елементів. Масив задається випадковими числами.

2. Реалізуйте метод швидкого сортування для масиву 50 випадкових чисел.

#### *Варіант 2*

1. Написати програму, яка би знаходила індекс максимального елемента в масиві з 12 елементів.

2. Реалізуйте метод сортування Шелла для масиву 50 випадкових чисел.

#### *Варіант 3*

1. Написати програму, яка виводить суму елементів масиву з 15 елементів. Масив задається випадковими числами.

2. Реалізуйте метод пірамідального сортування для масиву 50 випадкових чисел.

#### *Варіант 4*

1. Написати програму, яка виводить добуток елементів масиву з 17 елементів. Масив задається випадковими числами.

2. Реалізуйте метод сортування вставками для масиву 50 випадкових чисел.

#### *Варіант 5*

1. Написати програму, яка виводить суму парних елементів масиву з 22 елементів. Масив задається випадковими числами.

2. Реалізуйте метод шейкерного (коктейльного) сортування для масиву 50 випадкових чисел.

### **Завдання №4**

#### *Варіант 1*

1. Написати програму з використанням функції, яка в якості вхідних параметрів отримує *ім'я* та *прізвища*, та виводить 5 раз привітання "Hello *ім'я* та *прізвище*".

2. Написати програму з використанням функції, яка в якості вхідних параметрів отримує коефіцієнти квадратного рівняння, та виводить корені квадратного рівняння. Передбачити можливості виводу результату для випадку дискримінанту більше, менше та рівне нулю.

#### *Варіант 2*

1. Написати програму з використанням функції, яка в якості вхідних параметрів отримує значення *першого* та *другого* числа, а виводить результат у вигляді "Добуток *першого* та *другого* числа рівний ...".

2. Написати програму мовою Python, яка виводить таблицю температур за Цельсієм від 0 до 50 з кроком 5, разом з їх відповідними градусами у Фаренгейтах (як цілі числа). Використайте функцію, яка перетворює температуру з градусів Цельсія в градуси Фаренгейта і повертає їх результат.

#### *Варіант 3*

1. Написати програму з використанням функції, яка в якості вхідних параметрів отримує значення *ширини* та *довжини* прямокутного столу, а виводить результат у вигляді "Площа столу розміром *ширини* та *довжини* рівна ...".

2. Модифікуйте програму, яку розглядалася для бульбашкового сортування так, щоб вона запитувала користувача кількість значень даних в масиві, а потім



зчитувала їх від користувача програми. Для зчитування вхідних даних використовуйте власну функцію.

#### *Варіант 4*

1. Написати програму з використанням функції, яка в якості вхідних параметрів отримує значення *першої*, *другої* та *третьої* сторони трикутника, а виводить результат у вигляді "Периметр *першої*, *другої* та *третьої* сторін трикутника рівний ...".
2. Написати програму, яка містить функцію, яка має один параметр  $n$ , що представляє число більше 0. Функція повинна повертати  $n!$  ( $n$  факторіал). Після цього, написати функцію `main`, яка викликає цю функцію і передає їй значення від 1 до 20. Вивести два стовпчики, де в першому число від 1 до 20, а в другому – факторіал цього числа.

#### *Варіант 5*

1. Написати програму з використанням функції, яка в якості вхідних параметрів отримує значення *діаметра* круга, а виводить результат у вигляді "Площа круга з діаметром значення *діаметра* рівна ..., а радіус круга дорівнює ....".
2. Написати програму, яка містить функцію, що приймає один параметр  $n$ . В залежності від параметра  $n$ , функція друкує напис "Значення  $n$  більше нуля", для всіх додатних чисел, "Значення  $n$  менше нуля", для всіх від'ємних чисел, та "Параметр  $n$  дорівнює нулю", коли параметр дорівнює нулю. Показати дію функції для значень від -10 до 10, з кроком 2.

## ЛАБОРАТОРНА РОБОТА №2

на тему «Використання графічних модулів в Python»

### 1. МЕТА РОБОТИ

Ознайомитися із можливостями мови програмування Python для графічного представлення та побудови фігур на основі модулів комп'ютерної графіки, використання графічних операторів, запис та читання з файлів.

### 2. ТЕОРЕТИЧНА ЧАСТИНА

#### 2.1. Вступ до графіки

*Використання модуля Python*

Код програми, який написаний кимось іншим, може бути використаний в інших програмах та для інших завдань. У Python, це відбувається за допомогою *модулів*. В даній лабораторній роботі буде використовуватися модуль комп'ютерної графіки під назвою *graphics.py* за допомогою якого можна з легкістю малювати картинки в Python.

Завантажити копію цього модуля можна в мережі інтернет або ж отримати у викладача. Після того, як цей файл буде завантажений на ваш комп'ютер, перемістіть його у вашу робочу область Python. Цей модуль Python повинен бути в тій же папці, що і будь-яка програма, яка використовує його код.

*Використання модуля Graphics*

Програма, яка включає комп'ютерну графіку, як правило, виглядає наступним чином:

```
from graphics import *
def main():
    win = GraphWin()
    win.getMouse() #відображати форму поки не натиснути мишкою
    win.close()
main()
```

За допомогою команди `import *` ми включаємо, або іншими словами додаємо в програму всі елементи з графічного модуля `graphics`, які необхідні у цій програмі. Перша інструкція головної функції викликає функцію під назвою `GraphWin()`, що створює графічне вікно, в якому ми можемо малювати. Змінна `win` – це посилання на створене нами вікно. Тому, кожен раз, коли ми будемо малювати щось в цьому вікні, ми використовуємо ім'я цієї змінної. Після закінчення програми, з використанням функції `getMouse()`, яка застосовується до створеної зміни `win` і чекаємо, поки користувач натисне мишкою на відкритому вікні. Після цього вікно

закривається.

### *Координати вікна*

За замовчуванням вікно, яке створюється за допомогою функції GraphWin має розміри 200 пікселів в ширину і 200 пікселів у висоту. Піксель - це "елемент зображення" який є неподільною точкою на екрані. Кожен піксель в вікні має координату  $x$  та  $y$ . У верхньому лівому кутку області відображення вікна  $x$  та  $y$  матимуть координати (0,0). Координата  $x$  збільшується зліва направо. Координата  $y$  збільшується зверху вниз, а не навпаки, як ви звикли з декартовою системою координат. Якщо вікно 200x200, нижній правий кут вікна матиме координати (199,199). Ви можете створювати вікна будь-якого розміру, використовуючи іншу версію функції GraphWin, для потрібно лише три параметра: назва для вікна, його ширина (в пікселях) і його висота (в пікселях). Наприклад, така інструкція створює вікно 250 пікселів в ширину і 400 пікселів у висоту, що має назву "Головна форма":

```
win = GraphWin("Головна форма", 250, 400)
```

### *Малювання об'єктів*

Щоб намалювати об'єкт, ми повинні спочатку створити точки, які далі будуть з'єднуватися між собою у заданій послідовності, яка визначається послідовністю введення команд. Щоб створити точку, ми повинні задати їй координати  $(x, y)$ :

```
a = Point(25,25)
```

```
b = Point(100,175)
```

```
c = Point(100,100)
```

Тепер, з трьома створеними точками, ми можемо визначити об'єкти, такі як прямокутники чи кола. Щоб створити прямокутник, ми створюємо об'єкт Rectangle вказавши дві точки, його початок та кінець:

```
rect = Rectangle(a, b)
```

Щоб створити коло, ми створюємо об'єкт Circle вказавши точку його центру та його радіус (в пікселях):

```
circ = Circle(c, 50)
```

### *Налаштування властивостей графічних об'єктів*

Кожен графічний об'єкт має набір властивостей, які можна задати для об'єкта.

Ось деякі приклади:

```
rect.setFill ("red")
```

```
circ.setFill ("blue")
```

```
rect.setOutline ("green")
```

```
circ.setOutline ("yellow")
rect.setWidth (5)
circ.setWidth (10)
```

Функція `setFill` встановлює колір, використовуваний для заповнення об'єкта. Зверніть увагу, що колір задається в кавичках. Функція `setOutline` встановлює колір, для малювання контуру об'єкта. Функція `setWidth` визначає ширину об'єкт в пікселях.

Крім того можна визначити кольори, використовуючи функцію `color_rgb`, який вимагає три параметри: інтенсивність червоного, зеленого і синього кольору в межах від 0 (немає) до 255 (повне). Таким чином, можна встановити колір заливки кола в пурпуровий `writingcirc.setFill (color_rgb (130,0,130))`.

### *Малювання об'єктів*

Після того, як властивості об'єктів встановлюються, ви можете намалювати їх за допомогою функції `draw`. Функція застосовується над графічним об'єктом і вимагає параметр, який вказує вікно в якому буде намальовано об'єкт. Наприклад, щоб намалювати прямокутник і коло, які ми створили вище, необхідно використати наступні інструкції:

```
rect.draw (win)
circ.draw (win)
```

Ось наша остаточна програма:

```
from graphics import *
def main():
    win = GraphWin()
    a = Point(25, 25)
    b = Point(100, 175)
    c = Point(100, 100)

    rect = Rectangle(a, b)
    circ = Circle (c, 50)

    rect.setFill("red")
    circ.setFill("blue")
    rect.setOutline("green")
    circ.setOutline("yellow")
    rect.setWidth(5)
    circ.setWidth(10)

    rect.draw(win)
```

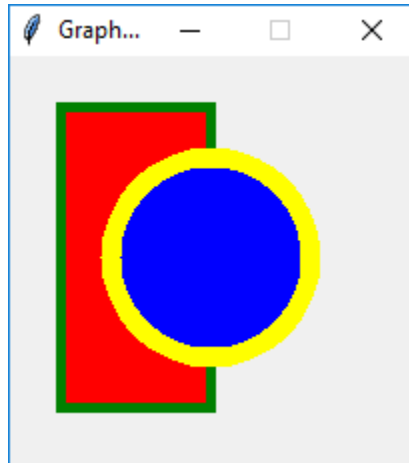
```

    circ.draw(win)

    win.getMouse()
    win.close()
main()

```

Результат виконання матиме наступний вигляд:



Зверніть увагу, що все, що ви малюєте спочатку перекривається тим, що малюєте після того.

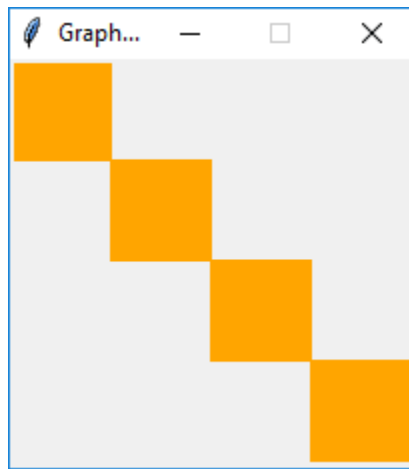
### *Інший приклад*

Розглянемо використання можливостей графічного відображення з іншими операторами python. Як приклад, виведемо п'ять квадратів розміром 50x50, які відображаються по діагоналі. Для цього використаємо цикли та особливості для застосування параметрів функцій графічного відображення.

```

from graphics import *
def main():
    win = GraphWin()
    for i in range(4):
        rect = Rectangle(Point(i*50,i*50),Point((i+1)*50,(i+1)*50))
        rect.setFill("orange")
        rect.setOutline("orange")
        rect.draw(win)
    win.getMouse()
    win.close()
main()

```



## 2.2. Більше графіки і взаємодії з користувачем

### *Додаткові графічні операції*

В попередньому завданні, ми почали використовувати графічний модуль, який включає функції, які можуть створити малюнок на екрані, не знаючи, як фізично відбувається рух пікселів. Більша частина програмного забезпечення написана по такому принципу. Програмісти створюють бібліотеки для обробки конкретних операцій, а потім інші програмісти створюють нові бібліотеки на основі попередніх, для того щоб зробити більш складні програми.

Ось ще декілька графічних об'єктів, які можна використовувати:

**Line(point, point2)** – створює лінію від point1 до point2.

Приклад: `line1 = Line(Point(0,200), Point(200,0))`

**Oval(point1, point2)** – створює овал навколо рамки, який визначається point1 і point2 (протилежні кути рамки)

Приклад: `oval2 = Oval(Point(50,50), Point(150,100))`

**Polygon(point1, point2, point3, ...)** – створює багатокутник із заданих точок, які є кінцевими точками для кожної лінії по порядку. Остання точка з'єднана з першою точкою.

Приклад: `triangle = Polygon(Point(25,100), Point(175, 50), Point(175, 150))`

Варто звернути увагу, що ми можете використовувати попередньо визначені функції `setFill`, `setOutline` and `setWidth` до всіх об'єктів, так само як до кругів і до прямокутників.

**Text(anchorpoint, string)** – створює текстовий об'єкт, який знаходиться в даній точці прив'язки.

Приклад: `message = Text(Point(150,150), "150-150")`

Для текстових об'єктів ви можете використовувати наступні функції, перед тим як намалювати текстовий об'єкт.

`setFace(font)` – встановлює шрифт, де шрифт може бути "helvetica", "courier", "times roman" чи "arial".

Приклад: `message.setFace("courier")`

`setSize(point)` – встановлює розмір шрифту до даної точки (між 5 і 36).

Приклад: `message.setSize(18)`

`setStyle(style)` – встановлює стиль, де стиль може бути "normal", "bold", "italic", або "bold italic"

Приклад: `message.setStyle("italic")`

`setTextColor(color)` – встановлює колір тексту (так само, як і функція `setFill`)

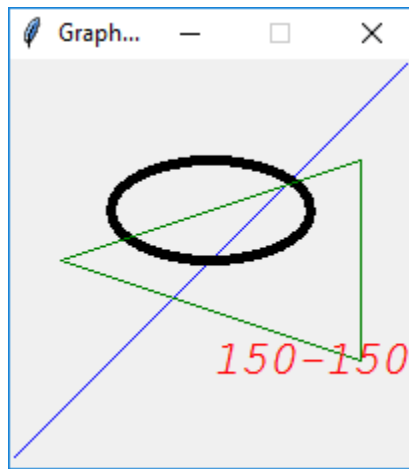
Приклад: `message.setTextColor("red")`

Ось проста програма на Python, яка показує, як використовувати графічні об'єкти і їхні функції:

```
from graphics import *
def main():
    win = GraphWin()
    line1 = Line(Point(0, 200), Point(200, 0))
    line1.setFill("blue")
    oval2 = Oval(Point(50, 50), Point(150, 100))
    oval2.setWidth(5)
    triangle = Polygon(Point(25, 100), Point(175, 50), Point(175, 150))
    triangle.setOutline("green")
    message = Text(Point(150, 150), "150-150")
    message.setFace("courier")
    message.setStyle("italic")
    message.setSize(18)
    message.setTextColor("red")
    line1.draw(win)
    oval2.draw(win)
    triangle.draw(win)
    message.draw(win)

    win.getMouse()
    win.close()
main()
```

Результат виконання програми:



### *Ввід даних користувачем*

Більшість графічних користувацьких інтерфейсів дозволяють користувачам вводити дані в програму, використовуючи ряд методів, у тому числі натиснувши мишкою, введення значень, вибираючи опцію з меню, і т.д. Графічний модуль, який ми використовуємо, має кілька простих способів, які дозволяють користувачам взаємодіяти з графічним вікном: розглянемо принцип дії на прикладі використання роботи мишки.

Нагадаємо, що місце кожного пікселя в графічному вікні задається в  $x$  та  $y$  координатах. Якщо користувач клацає мишкою всередині вікна, розташування показника мишки (який би піксель не був натиснутий) записується для використання. Вікно має функцію під назвою `getMouse`, яка чекає, поки користувач не натисне у вікні, а потім повертає значення точки, який вказує, де користувач натиснув. Потім ми можемо використовувати функції `getX` і `getY`, щоб дізнатися, де користувач натиснув. Ось проста програма, яка відстежує 10 зроблених кліків користувачем:

```
from graphics import *
def main():
    win = GraphWin()

    for i in range(10):
        point = win.getMouse()
        print (point.getX(), point.getY())
main()
```

В результаті виконання програми, після кожного натискання клавішою мишки у відповідному вікні програми, будуть виводитися на друк координати тієї точки.

Розглянемо програму, яка дозволяє користувачеві намалювати трикутник. В результаті того, що користувач клікає на три точки на екрані, а в результаті отримує трикутник на основі цих трьох точок:

```
from graphics import*
```



```
def main():
    print ("Намалювати трикутник. Натисніть 3 рази у вікні.")
    win = GraphWin()
    point1 = win.getMouse()
    point2 = win.getMouse()
    point3 = win.getMouse()
    triangle = Polygon(point1, point2, point3)
    triangle.setFill("orange")
    triangle.draw(win)

    win.getMouse()
    win.close()
main()
```

Тут внесені зміни, які дозволяють користувачеві створити 5 трикутників різних кольорів:

```
from graphics import*
def main():
    colors = ["green", "yellow", "red", "orange", "blue"]
    win = GraphWin()
    for i in range(5):
        print ("Намалювати трикутник. Натисніть 3 рази у вікні.")
        point1 = win.getMouse()
        point2 = win.getMouse()
        point3 = win.getMouse()
        triangle = Polygon(point1, point2, point3)
        triangle.setFill(colors[i])
        triangle.draw(win)
    win.getMouse()
    win.close()
main()
```

## 2.3. Використання графічних модулів для побудови гри “Вимкнути вогники”

### *Читання з файлу*

Зазвичай дані, які нам потрібні для програми будуть зчитуватись з файлу даних, щоб нам не вводити дані знову і знову за допомогою клавіатури.

Давайте припустимо, що є текстовий файл, який містить значення п'яти чисел, по одному в кожному рядку, для того щоб знайти максимум з 5 чисел. Ми може змінити програму, щоб ініціалізувати масив з 5 значень з файлу наступним чином:

```
def main():
```

```

A = [] #створюємо порожній масив
filename = input("Input filename containing data: ")
infile = open(filename, "r")
for j in range(5):
    number = eval(infile.readline())
    A.append(number)
    print(A[j])
i = 0
max = A[0]
while i < len(A)-1:
    i = i + 1
    if A[i] > max:
        max = A[i]
print ("The maximum positive integer is", max)
main()

```

Припустимо, що є текстовий файл на ім'я nums.txt в тій же папці, що і наша програма і містить наступні дані:

```

4
7
3
9
8

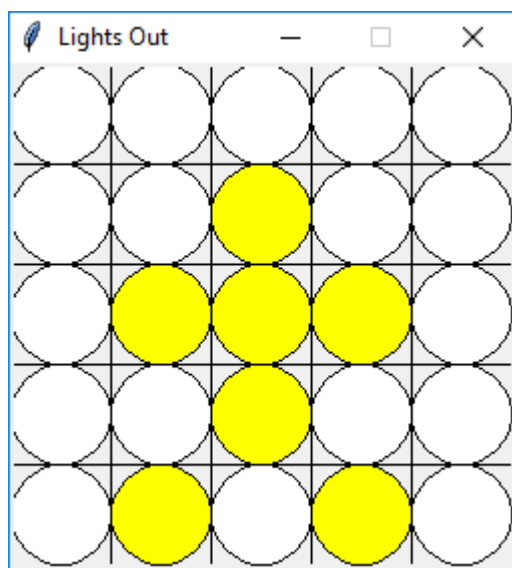
```

Програма починається з порожнього масиву A. Потім програма запитує у користувача ім'я файлу даних за допомогою функції input. Наприклад, користувач може ввести nums.txt. Далі, ми відкриємо файл для читання за допомогою функції open. Параметр "r" означає, що ми хочемо відкрити файл для читання за вказаним ім'ям. Функція повертає посилання на файл, яке ми зберігаємо в змінній infile. Після цього, кожного разу, коли ми хочемо перейти до файлу для даних, ми використовуємо змінну infile.

Програма проходить цикл 5 разів і читає кожен рядок файлу за одну ітерацію, перетворюючи прочитаний рядок до значення (number) за допомогою eval (infile.readline ()), а потім додаючи це значення до масиву. Вираз infile.readline () зчитує наступний рядок з текстового файлу, і функція eval перетворює його в число, оскільки він зчитується як рядок.

### *Гра Вимкнути вогники*

Гра Вимкнути вогники це популярна гра, яка складається із сітки 5x5 вогників. Коли гра починається, деякі з вогнів горять, а деякі не горять, як показано нижче.



Суть гри полягає в тому, щоб вимкнути всі вогники. При натисканні на вогник, він переключається (з вимкненого стану до ввімкнутого, або з ввімкнутого на вимкнений) і так само його чотири сусіди (верхній, нижній, лівий і правий). Не всі вогники мають 4 сусіди. Наприклад, кутові вогники мають тільки 2 сусідів.

Початковий стан вогників буде дано файлом даних, де 0 означає вимкнутий а 1 ввімкнений. Файл матиме 25 чисел, одне для кожного вогника, по одному в рядку, в порядку по рядках зліва направо. Файл puzzle1.txt буде зберігати всю інформацію про стани вогників на початку гри. Розглянемо роботу гри на прикладі такої комбінації: 0 0 0 0 0 0 1 0 0 0 1 1 1 0 0 0 1 0 0 0 1 0 1 0 (у файлі дані мають зберігатися вертикально).

### Алгоритм гри

Гра має такий алгоритм:

1. Малюємо лінії для сітки.
2. Ініціалізуєте вогники сітки.
3. Малюємо всі вогники.
4. Встановлюємо GameOver - false.
5. Поки GameOver - false робимо наступне:
  - а. Отримайте значення рядка і стовпця вогника, на який натиснули.
  - б. Переключіть обраний вогник і його сусідів.
  - в. Перемалюйте всі вогники на сітці.
  - г. Якщо всі вогники на борту вимкнені, встановіть GameOver - true.
6. Виведіть на екран "Ви виграли!"

Для кожного з основних кроків алгоритму буде представлена функція Python. Ось головна функція та інші функції, які зараз порожні:

```
from graphics import *
def main():
```

```

filename = input("Input name of puzzle file: ")
window = GraphWin("Lights Out", 250, 250)

board = initialize_board(filename)
draw_lines(window)
display_lights(window, board)
game_over = False
while game_over == False:
    p = window.getMouse()
    print (p.getX(), " ", p.getY()) # see where player clicked
    column = p.getX()/50 # compute column where player clicked
    row = p.getY()/50 # compute row where player clicked
    update_board(board, row, column)
    display_lights(window, board)
    game_over = check_for_winner(board)
print ("GAME OVER")

window.getMouse()
window.close()
main()

```

Основна функція починається із запитування користувача ім'я файлу, що містить дані для вогники. Далі відкривається вікно 250x250 для гри і ініціалізується масив сітки з даних файлу, використовуючи іншу функцію з ім'ям *initialize\_board*. Після цього малюються горизонтальні і вертикальні лінії у вікні, щоб розділити його на 25 рівних квадратів, а потім малюються кругові вогники, один на кожен квадрат, використовуючи *draw\_lines* і *display\_lights* функції.

### Ініціалізація сітки

Змінна сітки являє собою двовимірний масив (масив масивів). Щоб створити масив 5x5, ми повинні створити порожній масив і потім додати 5 порожніх масивів до нього: [ [], [], [], [], [] ].

Для того аби зчитати 25 значень з нашого файлу даних і додавати їх у масив використаємо функцію *initialize\_board* з відповідним синтаксисом:

```

def initialize_board(filename):
    infile = open(filename, "r")
    board = []
    for i in range(5):
        board.append([])
    for row in range(5):
        for col in range(5):
            columnvalue = eval(infile.readline())

```

```
        board[row].append(columnvalue)
    return board
```

Перші чотири рядки створюють порожній масив масивів. Тоді ми маємо вкладений цикл, який читає 25 значень масиву масивів.

### *Малювання ліній*

Намалювати 8 ліній необхідно розділити вікно:

```
def draw_lines(window):
    line1 = Line(Point(0,50),Point(250,50))
    line1.draw(window)
    line2 = Line(Point(0,100),Point(250,100))
    line2.draw(window)
    line3 = Line(Point(0,150),Point(250,150))
    line3.draw(window)
    line4 = Line(Point(0,200),Point(250,200))
    line4.draw(window)
    line5 = Line(Point(50,0),Point(50,250))
    line5.draw(window)
    line6 = Line(Point(100,0),Point(100,250))
    line6.draw(window)
    line7 = Line(Point(150,0),Point(150,250))
    line7.draw(window)
    line8 = Line(Point(200,0),Point(200,250))
    line8.draw(window)
```

### *Малювання вогників*

Щоб намалювати вогник, нам необхідно намалювати 25 кіл, які заповнені білим (вимкнено) або жовтим (увімкнено) кольорами. Ось рядки і стовпці гри, разом з розташуванням центрів кіл у вікні:

row	column	center for circle in window
0	0	25,25
0	1	75,25
0	2	125,25
0	3	175,25
0	4	225,25
1	0	25,75
1	1	75,75
1	2	125,75
1	3	175,75
1	4	225,75
2	0	25,125

2	1	75,125
2	2	125,125
2	3	175,125
2	4	225,125

etc.

Для кожного рядка і стовпця, коло має центр в  $(\text{column} * 50 + 25, \text{row} * 50 + 25)$ .  
Всі кола мають радіус 25. Ось ця функція, яка виводить вогники сітки у вікні:

```
def display_lights(window, board):
    for row in range(5):
        for column in range(5):
            center = Point(column*50+25,row*50+25)
            circ = Circle(center,25)
            if (board[row][column] == 1):
                circ.setFill("yellow")
            else:
                circ.setFill("white")
            circ.draw(window)
```

В результаті виконання описаної вище програми отримаємо зображення дошки для гри з відповідними кольорами на ній.

Після цього, створимо програму щоб грати в гру Вимкнути вогники, тобто запрограмує нашу програму аби вона реагувала на натискання мишкою, та мала можливість змінювати колір зображення круга при його натисканні.

На початку гри створюється вікно, в якому зображається сітка з вогниками розміром 5x5 - жовті кола з використанням даних, які ми читаємо з файлу. Стан кожного вогника може бути 1 (ввімкнений) або 0 (вимкнений). Стани зберігаються в двовимірному масиві (тобто масив масивів). Сітка зберігається у змінній з ім'ям board так:

На мові Python:

```
[[0,0,0,0,0],[0,0,1,0,0],[0,1,1,1,0],[0,0,1,0,0],[0,1,0,1,0]]
```

Наприклад:

*колонки*

```
0 1 2 3 4
```

*рядки*

```
0    0 0 0 0 0
```

```
1    0 0 1 0 0
```

```
2    0 1 1 1 1
```

```
3    0 0 1 0 0
```

```
4    0 1 0 1 0
```

Кожен з основних кроків алгоритму уже представлений функцією Python. Ми вже створили основну функцію і функції для виконання перших трьох кроків.

Основна функція містить наступний цикл, щоб контролювати гру, коли вона буде створена:

```

game_over = False
while game_over == False:
    p = window.getMouse()
    print (p.getX(), " ", p.getY()) # see where player clicked
    column = p.getX()/50 # compute column where player clicked
    row = p.getY()/50 # compute row where player clicked
    update_board(board, row, column)
    display_lights(window, board)
    game_over = check_for_winner(board)
print ("GAME OVER")

```

Змінна `game_over` це логічна Boolean змінна, яка може зберігати значення `true` чи `false`. Поки ця змінна `false`, гра повторює цикл.

Кожна ітерація чекає щоб користувач натиснув мишею у вікні гри. Функція повертає `getMouse` точку `Point p`, де користувач натиснув. Потім ми можемо використовувати функції `getX` і `getY`, щоб отримати координати `x` і `y` цієї точки. Для визначення рядка і стовпця вогника на який натиснув користувач, ми можемо просто розділити кожну `y` і кожну `x` координати по 50 відповідно, оскільки кожна квадратна область у вікні має розмір 50x50. Наприклад, якщо користувач натискає на вогник в третьому ряду (тобто рядок з індексом 2, оскільки ми рахуємо рядки від 0), `y` координата має бути між 100 і 149. Відзначимо тут, що користувач не повинен натиснути саме на круг, тільки на квадрат, який містить круг.

Тепер як у нас є рядок і стовпець, на який натиснув користувач, нам потрібно щоб вогник та його сусіди перемикалися. Функція `update_board` зробить це (див. нижче). Після того, як дошка оновилася, ми показуємо вогники, знову використовуючи функцію, яку ми написали раніше.

Нарешті, ми повинні перевірити сітку, щоб побачити, чи всі вогники вимкнені. Ми будемо робити це функцією `check_for_winner` (описана нижче). Ця функція буде повертати нове значення для `game_over`. Якщо `game_over` значення `false`, цикл повторюється і ми чекатимемо наступного натиску миші. Якщо повернене значення `true`, то цикл завершується і гра закінчується.

### Оновлення вогників

Коли функція виконується, основна функція передає сітку цієї функції (тому вона знає стан всіх вогнів), рядок і стовпець, вибраний користувачем. Ці три елементи даних є параметри цієї функції. Python налаштований таким чином, що дані не доступні глобально для всіх функцій. Замість цього, якщо одна функція має деякі дані (наприклад, основна), а інша функція її потребує (наприклад, `update_board`), основна функція повинна передати ці дані в функцію за допомогою параметрів.

Для перемикання вогників, ми повинні змінити значення елемента масиву для рядка і колонки від 1 до 0 або від 0 до 1, разом з сусідніми елементами (вище, нижче, ліворуч і праворуч).

Майте на увазі, що деякі вогники можуть не мати всіх 4 сусідів. Ми можемо перемикаєти вогник, який був обраний наступним чином:

```
if board[row][column] == 1:
    board[row][column] = 0
else:
    board[row][column] = 1
```

У наведеному вище коді, в операторі умови `else` не потрібно перевіряти, чи дошка [рядок] [стовпець] є 0, оскільки це єдиний інший варіант у цьому випадку.

Ми повинні визначити чи цей вогник горить. Довжина цього рядка є `row - 1`. Вогник обраного рядка світиться, поки `row - 1 > 0`, або якщо `row > 0` (або ж `row ≥ 1`).

Ось код, який нам потрібен:

```
if row > 0:
    if board[row-1][column] == 1:
        board[row-1][column] = 0
    else:
        board[row-1][column] = 1
```

Коли ми змінюємо масив, переконайтеся, що ви зміните вогник в `row-1`, а не `row`. Також зверніть увагу: другий `if` блок виконується тільки якщо рядок обраний користувачем більше 0. В іншому випадку, блок пропускається.

Ось код, який визначає, чи є сусід нижче, ліворуч і праворуч від обраного рядка і стовпця:

```
Існує сусід знизу, якщо row < 4.
Існує сусід зліва, якщо column > 0.
Існує сусід справа, якщо column < 4.
```

Ось ціла функція:

```
def update_board(board, row, column):
    # toggle chosen light in the given row and column
    if board[row][column] == 1:
        board[row][column] = 0
    else:
        board[row][column] = 1
    if row > 0: # is there a light above the chosen light?
        if board[row-1][column] == 1:
            board[row-1][column] = 0
        else:
            board[row-1][column] = 1
    if row < 4: # is there a light below the chosen light?
        if board[row+1][column] == 1:
            board[row+1][column] = 0
        else:
            board[row+1][column] = 1
```



```

if column > 0:  # is there a light to the left of the chosen light?
    if board[row][column-1] == 1:
        board[row][column-1] = 0
    else:
        board[row][column-1] = 1
if column < 4:  # is there a light to the right of the chosen light?
    if board[row][column+1] == 1:
        board[row][column+1] = 0
    else:
        board[row][column+1] = 1

```

Ви думаєте, що ми змінюємо копію сітки в цій функції, а не оригінальну сітку з головної функції, але ми міняємо той же масив. Коли масив передається від однієї функції до іншої, то передається вказівник на масив, а не копія масиву. Таким чином, функція *update\_board* використовує той же масив з головної функції, незважаючи на ім'я (ця техніка називається *викликом за посиланням* у термінології програмування). З іншого боку, рядок і стовпець це копії вихідних значень даних з *main* функції, так як вони лише значення даних (це називається *виклик за значенням*). Запустіть програму зараз і подивіться, що вогники перемикаються правильно.

### Перевірка на закінчення гри

Після того, як сітка оновилася, як дізнатись, що гравець виграв? Якщо всі вогники вимкнені, то кожний квадрат масиву повинен бути значення 0. Якщо скласти всі значення в масиві, отримаємо загальну суму 0, якщо всі вогники вимкнені. Нам просто потрібна змінна, щоб відстежувати суму, змінна спочатку рівна 0. Потім ми додаємо кожне значення до суми, рядок за рядком. Це вимагає набору вкладених циклів. Далі ми перевіряємо остаточну суму, щоб побачити, чи вона дорівнює 0. Якщо це так, ми повертаємо true (гра закінчена). В іншому випадку, ми повертаємо false (гра ще не закінчена). Це значення зберігається у змінній *game\_over* з основної функції і попадає в цикл управління грою.

Ось код для нашої функції:

```

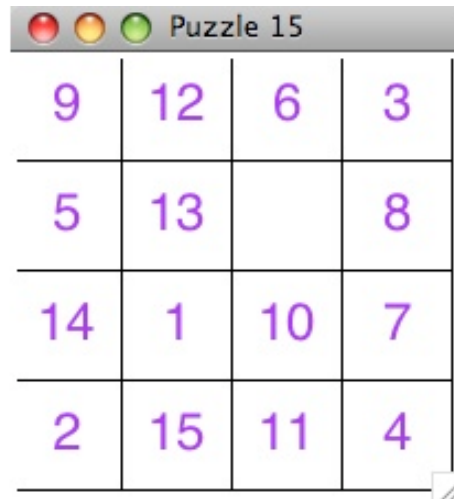
def check_for_winner(board):
    sum = 0
    for row in range(5):
        for column in range(5):
            sum = sum + board[row][column]
    if sum == 0:
        return True
    else:
        return False

```

Тепер зіграйте в гру і подивіться чи зможете ви виграти!

#### 2.4. Головоломка з 15 чисел

Далі розглянемо створення гри-головоломки "Гра в 15". Гра в 15 - популярна гра, що складається з сітки чисел (4 на 4) заповненої числами від 1 до 15 та однією порожньою клітинкою. Цифри не впорядковано в сітці, як показано нижче:



Puzzle 15			
9	12	6	3
5	13		8
14	1	10	7
2	15	11	4

Гравець може "пересовувати" прилеглі до порожньої клітинки цифри в цей порожній простір за допомогою курсора мишки. Ціль цієї гри в тому, щоб розмістити всі числа в порядку від 1 до 15 як показано нижче:



Puzzle 15			
1	2	3	4
5	6	7	8
GAME OVER			
9	10	11	12
13	14	15	

Початковий стан гри повинен бути заданий файлом вхідних даних, що містить рядок всіх чисел від (1 до 15) та 0, що представляє порожню комірку.

#### Основний алгоритм гри

Гра має наступний алгоритм:

1. Створюється вікно гри.
2. Ініціалізується ігрове поле номерами з файлу даних.
3. Малюється ігрове поле у вікні.

4. Поки гравець не переміг у грі, виконуються наступні дії:

а. Визначається рядок і стовпець на ігровому полі, на який, за допомогою миші, натискає гравець.

б. Перемістіть цей номер в порожню клітку, якщо цей номер межує з нею, горизонтально або вертикально.

в. Оновіть ігрове поле у вікні.

г. Перевірте, чи гравець переміг.

5. Виведіть GAME OVER і чекати, щоб гравець натиснув Enter щоб закрити вікно.

Основні кроки алгоритму представлені нижче:

```
from graphics import *
```

```
def main():
```

```
    filename = input("Input name of puzzle file: ")
```

```
    window = GraphWin("Puzzle 15", 200, 200)
```

```
    board = initialize_board(filename)
```

```
    display_numbers(window, board)
```

```
    game_over = False
```

```
    while game_over == False:
```

```
        p = window.getMouse()
```

```
        col = p.getX()/50 # compute column where player clicked
```

```
        row = p.getY()/50 # compute row where player clicked
```

```
        update_board(board, row, col)
```

```
        display_numbers(window, board)
```

```
        game_over = check_for_winner(board)
```

```
    message = Text(Point(100,100), "GAME OVER")
```

```
    message.setSize(24)
```

```
    message.setTextColor("orange")
```

```
    message.draw(window)
```

```
    input("Press <ENTER> to quit.")
```

```
    window.close()
```

```
main()
```

Основна функція починається із запиту користувача імені файлу, що містить номера. Після цього відкриється вікно 200x200 для гри, яке ініціалізується масивом, використовуючи функцію *initialize\_board*. Далі програма повинна відобразити числа в том порядку, в якому задані в масиві за допомогою функції *display\_numbers*.

### Ініціалізація ігрового поля

Ігрове поле являє собою двовимірний масив (масив масивів). Щоб створити його, ми маємо створити масив 4x4, ми повинні створити порожній масив і потім

додати 4 порожні масиви для нього: `[[], [], [], []]`.

Наступний код буде читати 16 числових значень з нашого файлу даних і додати їх до цього масиву:

```
def initialize_board(filename):
    infile = open(filename, "r")
    board = []
    for i in range(4):
        board.append([])
    for row in range(4):
        for col in range(4):
            columnvalue = eval(infile.readline())
            board[row].append(columnvalue)
    return board
```

Перші чотири рядки створюють порожній масив масивів. Тоді ми маємо вкладений цикл, який читає 16 значень для масиву масивів.

### Відображення чисел

Для відображення чисел, ми повинні намалювати 16 білих квадратів, кожен 50 x 50 з чорною окантовкою. Для малювання квадрата, ми використовуємо об'єкт `Rectangle` з верхнім лівим і нижнім правим кутами в точках. Після побудови на кожному квадраті, ми відобразимо число отримане з масиву. Щоб очистити номер квадрату ми повинні намалювати на ньому такий же білий квадрат.

Ось код для цієї функції:

```
def display_numbers(window, board):
    for row in range(4):
        for col in range(4):
            square = Rectangle(Point(col*50, row*50), Point((col + 1)*50,
                (row + 1)*50))
            square.setFill("white")
            square.draw(window)
            if board[row][col] != 0:
                center = Point(col * 50 + 25, row * 50 + 25)
                number = Text(center, board[row][col])
                number.setSize(24)
                number.setTextColor("purple")
                number.draw(window)
```

Запустіть цю програму і подивитися, як і очікувалось ви бачите настільна гра створена, як і очікувалося.

Проте, якщо спробувати перемістити клітинки, то програма видасть помилку, оскільки ще 2 функції не були задекларовані, а саме.

```
def update_board(board, row, col):  
    return          #remove this line when you complete this function
```

```
def check_for_winner(board):  
    return False    #remove this line when you complete this function main()
```

#### Оновлення дошки

Функція *update\_board* має три параметри, які вона отримує від основної функції: *board*, *row* і *col*. Змінна *board* являє собою двовимірний масив (масив масивів). У цьому масиві, ми зберігаємо поточне місцеположення кожного числа від 1 до 15. Порожня клітинка представлена значенням 0. Змінні *row* і *col* представляють рядок і стовпець, де гравець натиснув у вікні гри.

Те, що ми повинні зробити, це визначити чи користувач натиснув на рядок і стовпець, де порожні клітинки (тобто там, де 0 значення в масиві). Якщо так, то число в цій клітинці треба поміняти з 0.

Ми повинні бути обережні. Щоб побачити, чи гравець натиснув на клітинку поруч з клітинкою з 0, ми могли б просто подивитися зверху, знизу, зліва і праворуч від клітинки на яку натиснули чи це значення - 0, але гравець може натиснути на клітинку уздовж краю сітки. Наприклад, якщо гравець натискає вздовж верхнього рядка, і клітинки зверху немає. Таким чином, ми повинні перевірити дві речі: чи є рядок поруч клітинки на яку натиснули, і якщо це так, то чи є там 0? Ось як ми можемо добитися цього на мові Python:

```
def update_board(board, row, col):  
    if row > 0 and board[row-1][col] == 0:  
        board[row-1][col] = board[row][col]  
        board[row][col] = 0  
        return  
    if row < 3 and board[row+1][col] == 0:  
        board[row+1][col] = board[row][col]  
        board[row][col] = 0  
        return  
    if col > 0 and board[row][col-1] == 0:  
        board[row][col-1] = board[row][col]  
        board[row][col] = 0  
        return  
    if col < 3 and board[row][col+1] == 0:  
        board[row][col+1] = board[row][col]  
        board[row][col] = 0  
        return
```

В першому блоці *if* ми спочатку перевіряємо, чи ми не в першому рядку (тобто не в рядку 0), якщо ні та клітинка вище поточного рядка і стовпця (на *board[row-1][col]*) містить 0, то ми копіюємо значення в клітинку вище і змінюємо клітинку на яку натиснули в 0.

У Python, ми повинні перевірити ці умови в зазначеному порядку. Як правило, ви могли б подумати, що A і B це те саме як B і A, але не в коді програми. При обчисленні A і B, спочатку обчислюється A, і якщо вона false, то B ніколи не обчислиться. У першому if блоці, це рятує нас, якщо рядок == 0. Вираз row > 0 є false, тому ми ніколи не намагаємося перевірити чи board[row-1][col] є 0, що добре, так як row-1 буде -1, якщо row є 0 і немає рядків з індексом -1 в масиві.

### *Перевірка переможця*

Щоб визначити чи гравець виграв, ми повинні перевірити чи числа масива ряд за рядом, зліва направо, в порядку від 1 до 15. Таким чином, ми можемо використати наступний алгоритм, показаний на мові Python.

```
def check_for_winner(board):
    num = 1
    row = 0
    col = 0
    while num <= 15:
        if board[row][col] == num:
            num = num + 1
            col = col + 1
            if col > 3:
                col = 0
                row = row + 1
        else:
            return False
    return True
```

Ця функція сканує масив, починаючи з рядка 0 і стовпця 0. Якщо він знаходить 1 в елементі масиву, він встановлює num в 2 і рухається до наступної колонки. Якщо він знаходить 2 в наступну елементі масиву, він встановлює num до 3, і т.д. Якщо він потрапляє на кінець рядка (стовпець більше, ніж 3), він скидає колонку назад в 0, і збільшує рядок на 1, щоб перейти до наступного рядка. У будь-який момент, якщо наступний num не очікуване значення, ми повертаємо false відразу, виходимо з циклу і повертаємось до головної функції. Якщо збігаються всі 15 чисел, елемент за елементом, то цикл завершується і ми повертаємо true.

## **3. КОНТРОЛЬНІ ЗАПИТАННЯ**

1. Для чого використовується графічний модуль graphics?
2. Як підключити графічний модуль graphics?
3. Властивості відображення графічних об'єктів?
4. Яким чином відбувається зчитування інформації з файлу? Навести приклад.

5. Яким чином відбувається зчитування та вивід координат за допомогою мишки? Навести приклад.
6. Яка послідовність виводу графічних об'єктів?
7. Що таке масив в Python? Як відбувається його ініціалізація.
8. Навести приклад запису інформації в файл.

#### **4. ЛАБОРАТОРНЕ ЗАВДАННЯ**

1. Ознайомитись з теоретичною частиною лабораторної роботи.
2. Одержати індивідуальне завдання (див. Варіанти індивідуальних завдань). При необхідності уточнити завдання на виконання у викладача.
3. Скласти програму на мові програмування Python відповідно до поставленого завдання. Виконана програма повинна повністю реалізувати розв'язок поставленої задачі.
4. Запустити програму та зберегти результат виконання.
5. Оформити звіт відповідно до встановлених вимог.

#### **5. ЗМІСТ ЗВІТУ**

1. Мета роботи.
2. Відповіді на контрольні запитання.
4. Індивідуальне завдання, отримане у викладача, згідно з варіантом.
5. Код програми для реалізації завдання.
5. Аналіз отриманих результатів.
6. Ґрунтовні висновки.
7. Список використаної літератури.

#### **6. СПИСОК ЛІТЕРАТУРИ**

1. *Лутц М.*, Изучаем Python. - К.:Символ-Плюс , 2011. - 1280 с.
2. *Бизли Д.*, Python. Подробный справочник, 4-е издание. - К.:Символ-Плюс, 2012. - 864 с.
3. Офіційний сайт Python: <https://www.python.org/doc/>

## ВАРІАНТИ ІНДИВІДУЛЬНИХ ЗАВДАНЬ

### Завдання №1

#### Варіант 1

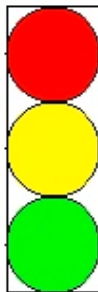
1. Намалуйте мішень (набір концентричних кіл, чергуючи червоний і білий) в графічному вікні розмірами 200 пікселів в ширину і 200 пікселів у висоту.



2. Використовуючи цикл намалувати фігуру, подібну до тої, що зображена вище, де радіус самого великого кола 80 пікселів, а радіус кожного наступного кола менший на 10 пікселів від попереднього. Зерегти чергування червоно-білих кольорів.

#### Варіант 2

1. Намалуйте простий світлофор в графічному вікні розмірами 200 пікселів в ширину і 200 пікселів у висоту. Три вогні повинні мати діаметр 50 пікселів кожен, і світлофор повинні бути в центрі графічного вікна.

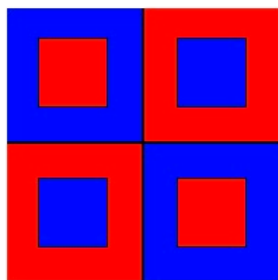


2. Використовуючи цикл, намалувати кола, які дотикаються один до одного зовнішніми частиною кола та розміщені горизонтально. Радіус найбільшого кола 60 пікселів, кожного наступного на 10 менше.

#### Варіант 3

1. Намалуйте зображення, показане нижче.

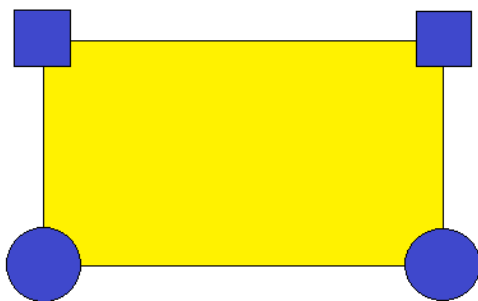




2. Використовуючи цикл, нарисувати 3 квадрати зеленого і 3 квадрати білого коліру, діагональ яких рівна  $8\sqrt{2}$ , які розміщені горизонтально і дотикаються один до одного.

#### Варіант 4

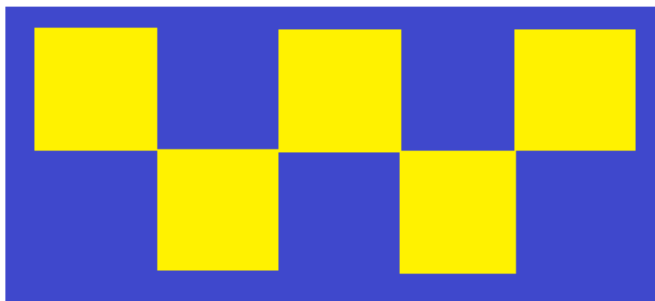
1. Намалювати зображення, яке зображено нижче. Розмір прямокутника 400x200 пікселів. Діагональ квадрата  $17\sqrt{2}$ , діаметр круга 28 пікселів.



2. За допомогою циклу, створення шахову дошку в графічному вікні, розміром 200 пікселів в ширину і 200 пікселів у висоту. Кожен квадрат повинен бути 25x25.

#### Варіант 5

1. Намалюйте зображення, показане нижче. Розмір сторони квадрату 30 пікселів.

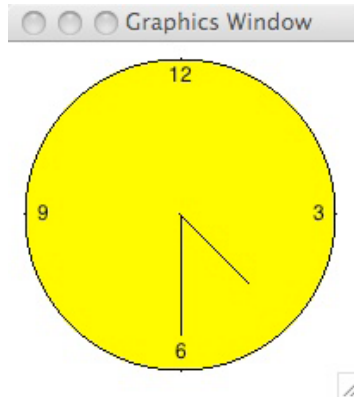


2. Використовуючи цикл, виведіть зображення зображене вище для запропонованого варіанту.

## Завдання №2

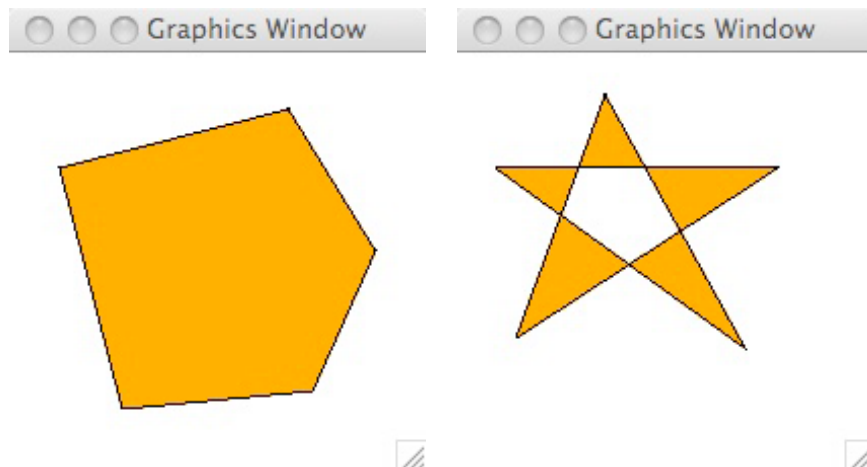
### Варіант 1

Напишіть програму на Python, яка відображатиме простий годинник на якому показано 4:30 година. Виведіть цифри 12, 3, 6 і 9 на табло годинника.



### Варіант 2

Напишіть програму на Python, яка попросить користувача ввести 5 точок і тоді намалює оранжевий многокутник, який з'єднає ці точки. Запустіть програму натиском на 5 різних точок у вікні. Далі запустіть програму натиснувши на вікно, і спробуйте сформувати п'ятикутну зірку. Зверніть увагу як змінюється колір, коли лінії пересікають одна одну.



### Варіант 3

Напишіть програму на Python, яка створить вікно на 200 на 200 пікселів і намалює червоний круг в центрі вікна з радіусом 25. Зачекайте, поки користувач натисне на екран, і круг пересунеться на іншу точку, але щоб його центр опинився там, де користувач клікнув. Повторіть 5 раз.

### Варіант 4

Напишіть програму на Python, яка створює вікно розміром 400 на 400 пікселів, яке

зафарбоване в червоний колір, а посередині цього вікна створюється квадрат білого кольору зі стороною 80 пікселів. Його сторони повинні зменшуватися на 10% кожного разу, як користувач натискає на квадрат і залишається без змін, якщо на іншу частину вікна.

#### *Варіант 5*

Напишіть програму на Python, яка створює форму розміром 320 на 450 пікселів, на якій намальований квадрат зі стороною 200 пікселів. В залежності від того, на яку сторону квадрата натиснути, та частина витирається. Передбачити аналогічний випадок для прямокутника, який утворюється після першого поділу квадрату. Процес ділення зупиняється, коли сторона утвореного прямокутника буде менша 3 пікселів.

### **Завдання №3 – Гра Lights Out**

Для кожного варіанту завдань: Створіть новий файл даних, який містить різну конфігурацію гри і запустіть програму з цим файлом даних, щоб побачити, що він відображається правильно.

#### *Варіант 1*

Подивіться на функції `draw_lines`. Замість того щоб малювати вертикальні лінії 4 окремо і 4 горизонтальні лінії окремо, переписати цю функцію, щоб вона використовувала 2 цикли, один буде малювати горизонтальні лінії, а 2-й цикл буде малювати вертикальні лінії. Не поміщайте цикли один в одного. Подивіться на початковий код і знайдіть шаблон.

#### *Варіант 2*

Спростити функцію `update_board` так щоб вам не потрібні були `if` оператори для перемикання вогників. Для цього потрібно взяти значення вогників (0 або 1), додаючи 1, і виконати модуль 2 для цього значення, щоб отримати значення перемикання. Зверніть увагу, що якщо світло має значення 1, то  $(1 + 1) \% 2 = 0$  і, якщо світло має значення 0, то  $(0 + 1) \% 2 = 1$ . В обох випадках ми отримуємо значення вогника. Ви все ж будете використовувати оператор `if` щоб перевірити рядок і стовпець для сусідів.

#### *Варіант 3*

Альтернативний алгоритм для визначення чи гравець виграв, це перевірка сітки ряд за рядом, по одному стовпцю за раз, як ми робили вище. Але замість обчислення загальної суми, ми розглянемо кожну клітинку, і якщо ми бачимо 1, ми відразу ж знаємо, що всі вогні не можуть бути вимкнені, так що ми можемо повернути `false` відразу. Якщо ми пройдемо через всі клітини без повернення `false`, то ми можемо повернути `true` після цього, так як кожна клітинка є 0. Реалізуйте цей альтернативний алгоритм і перевірте ваше рішення. Який алгоритм є більш ефективним в цілому?

#### *Варіант 4*

Реалізуйте виконання гри за описаним вище алгоритмом для випадку, коли дошка має розміри 6x6.

#### *Варіант 5*

Реалізуйте алгоритм розв'язання завдання гри для випадку комбінації вогників представлених в лабораторній роботі.

### **Завдання №4 – Гра 15**

Для кожного варіанту: створіть новий файл даних, який містить різну конфігурацію гри, і запустіть програму з цим файлом даних, щоб побачити, чи правильно вони відображатимуться.

#### *Варіант 1*

Змініть функцію `display_numbers` так, що всі парні числа друкуються фіолетовим, а всі непарні числа друкуються рожевим. Колір кожної клітинки з парним числом має бути сірий, а де є непарне синій.

#### *Варіант 2*

Змініть цю програму, щоб вона мала ігрову зону 5x5, з номерами від 1 до 24 і однією порожньою клітинкою.

#### *Варіант 3*

Змініть цю програму, щоб вона мала ігрову зону 6x6, з номерами від 1 до 35 і однією порожньою клітинкою.

#### *Варіант 4*

Змініть функцію `display_numbers` так, що всі парні числа друкуються червоним, а всі непарні числа друкуються синім. Колір порожньої клітинки має бути завжди білий, всіх інших клітинок жовтий.

#### *Варіант 5*

Змініть цю програму, щоб вона мала ігрову зону 3x3, з номерами від 1 до 8 і однією порожньою клітинкою.

# ЛАБОРАТОРНА РОБОТА №3

на тему «Побудова графіків в Python»

## 1. МЕТА РОБОТИ

Розглянути можливості побудови графіків з використанням засобів представлення даних Python.

## 2. ТЕОРЕТИЧНА ЧАСТИНА

### *Візуалізація даних*

Графіки є ефективними методами представлення даних. В залежності від якості підготовки графіку, він може передати ідею краще, ніж ціла стаття.

На графіку слід зображати функцію, які будуть зрозумілі вашій цільовій аудиторії. Отже, коли ви будете графік, то пам'ятайте про це. Якщо ваша цільова аудиторія технічні люди, вони можуть потребувати додаткову технічну інформацію. Якщо ваша цільова аудиторія інвестори, необхідно застосовувати інший підхід. Ми не будемо обговорювати, що представляти, а що ні; замість цього ми розглянемо різні методи, як можна представити дані за допомогою Python.

### *Matplotlib пакет*

*Matplotlib* пакет, доступний на сайті <http://matplotlib.org>, є основним інструмент для побудови графіків, який ми будемо використовуватися. Пакет є універсальним і настраюється, підтримуючи декілька графічних інтерфейсів. *Matplotlib*, разом з *NumPy* і *SciPy*, забезпечує MATLAB-подібні графічні можливості.

Переваги використання *matplotlib* в контексті аналізу і візуалізації даних полягає в наступному:

- ✓ Побудовані дані є простими і інтуїтивно зрозумілим.
- ✓ Висока продуктивність; професійний вихід.
- ✓ Інтеграція з *NumPy* і *SciPy* (використовується для обробки сигналів і чисельного аналізу) не викликає труднощів.
- ✓ Пакет настраюється і конфігурується для потреби більшості людей.

Пакет досить великий і дозволяє, наприклад, вкладення ділянки в графічному інтерфейсі. В даний час пакет підтримує декілька графічних інтерфейсів, у тому числі WxPython (<http://www.wxpython.org>) і PyGTK (<http://www.pygtk.org>). Для повного врахування *matplotlib*, перегляньте онлайн документацію, доступну як PDF документ за адресою: <http://matplotlib.sourceforge.net/Matplotlib.pdf>.

Забігаючи наперед, для виконання робіт та прикладів, представлених в лабораторній роботі, ви повинні переконатися, що *matplotlib* встановлена і працює правильно.

### *Інтерактивна графіка та файли зображень*

Є кілька способів як можна використовувати *matplotlib*:

- ✓ Створити динамічний контент, який буде обслуговуватися на сервері. Наприклад, ви могли б генерувати зображення цін на акції в реальному режимі часу або ж відображати дорожню інформацію у верхній частині карти.
- ✓ Вставляти його в графічному інтерфейсі користувача, що дозволяє користувачам взаємодіяти з додатком для візуалізації даних.
- ✓ Автоматично обробляти дані і генерувати результат в різних форматах, в тому числі JPEG, PNG, PDF, PostScript (PS). Ця опція найкраще підходить для пакетної обробки великої кількості файлів.

Розглянемо два можливих варіанти відображення даних: а) генерувати графіки з різних форматів файлів та б) за допомогою *matplotlib* в інтерактивному режимі. Зазвичай використовуються обидва варіанти, в залежності від стадії написаного коду. На ранніх стадіях розвитку, переважно зручніше працювати в інтерактивному режимі з невеликою вибіркою даних: вивід даних, збільшити, змінити параметри графіка, коментувати, та інше. Далі, після того як код буде готовий, настає час для використання повного набору файлів даних.

Імпортувати бібліотеку *PyLab* можна наступним чином:

```
from pylab import *
```

Така команда дозволяє імпортувати *matplotlib*, *NumPy* і *SciPy*. Хоча, взагалі кажучи, ми не повинні імпортувати всі додаткові бібліотеки і повинні зробити виняток у разі *PyLab*: це значно простіше працювати з цілим пакетом, що завантажується в пам'ять; і якщо пам'ять не є обмеженою, зручність від такого використання очевидна. Забігаючи наперед, будемо вважати, що імпортували *PyLab*, як описано вище.

Якщо ви не знаєте, чи конкретна функція є частиною *NumPy* або *matplotlib*, то необхідно використовувати функцію *help()*, якщо ми працюємо через консоль. Наприклад, ось результат виконання функції *help(diff)*:

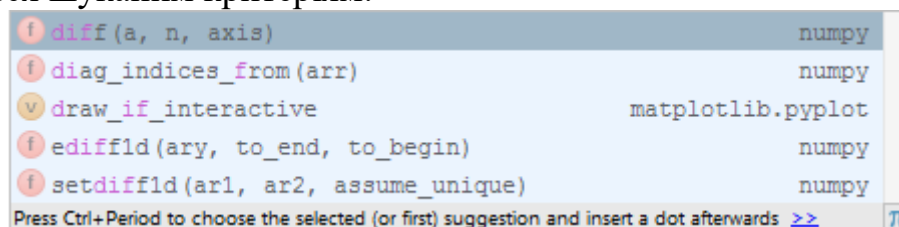
```
help(diff)
```

```
Help on function diff in module numpy.lib.function_base:
```

```
diff(a, n=1, axis=-1)
```

```
Calculate the nth order discrete difference along given axis.
```

Як можна бачити на першій лінії, *diff()* є функцією пакета *NumPy*, або, більш конкретно, *numpy.lib.function\_base*. Якщо ми працюємо в графічних редакторах, таких як *PyCharm*, то для того аби подивитися інформацію про функцію, достатньо надрукувати її ім'я у робочій області файлі і буде запропоновано ряд функцій, які відповідаються шуканим критеріям:



На далі, усі приклади розглянуті в лабораторній роботі будуть показані в PyCharm Community Edition 2016.1. Для інсталювання додаткових бібліотек було використано WinPython-64bit-3.4.3.5, який дозволив встановити `pylab`, `numpy` та `matplotlib`.

Наш наступний крок є побудова графіка. Для прикладу, нам необхідно відобразити список `[0, 1, 2]` і для цього використаємо наступну команду:

```
plot(range(3))
```

Проте, після виконання такої функції немає видимого відображення даних, і причина цього в тому, що ми не вказали, як саме хочемо зробити відображення: *інтерактивний графік* або *файли в друкованому вигляді*.

### Інтерактивні графіки

Інтерактивні графіки, як показано на рис. 3.1, будують графік в окремому вікні за допомогою функції `show()`.

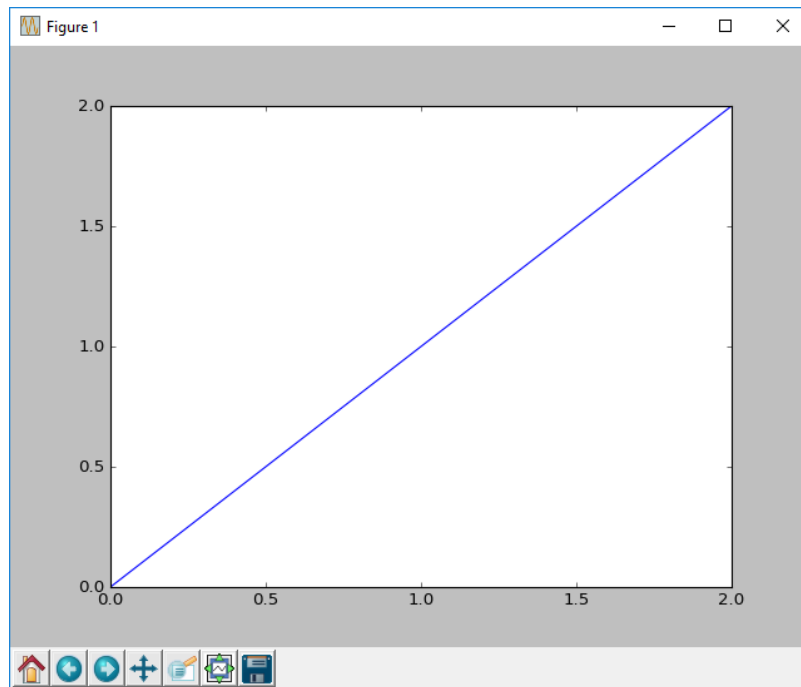


Рис. 3.1. Інтерактивний графік

Функція `show()` відкриває інтерактивне вікно. Нижче наведені декілька зауваження щодо цього вікна:

- ✓ Вікно пронумероване, як можна бачити на рис. 3.1. зверху у підписі "Figure 1". Це корисно, якщо у вас є декілька вікон і ви хотіли би щоб наступні графіки або переписували дані у певному вікні або з'являлися на конкретній цифрі вікна. Для перемикавання між цифрами, використовуйте функцію `figure(n)`, де `n` позначає індекс фігури. Якщо ви хочете нову фігуру, а індекс фігури не має значення,



виконайте команду `figure()`, яка створить порожню фігуру наступного доступного індексу.

- ✓ Осі  $x$  та  $y$  створюються автоматично аби відповідати до даних. У багатьох випадках, `matplotlib` робить відмінну роботу, автоматично вибираючи правильну вісь (як в даному прикладі). Тим не менше, якщо ви хочете інший діапазон значень, які будуть відображатися, це можна зробити за допомогою команди `axis()`, яку розглянемо далі.
- ✓ Місце миші друкується в правому кутку фігури. Це дуже корисно, якщо ви намагаєтеся збільшити дані і знайти конкретну точку даних. Ця функція не доступна, коли ви будувате графік у файл (тобто неінтерактивний режим).
- ✓ У вас є декілька кнопок на лівій нижній частині рис. 3.1, які дозволяють взаємодію з графіком. П'ять найбільш лівіших кнопок використовуються для масштабування графіку. Перша кнопка зліва (із позначкою будинку) використовується для зміни осей в оригінальних осях графіку. Наступні дві кнопки зі стрілками вперед і назад дозволяють повертатися через попередні зміни осей. Четверта кнопка дозволяє змінити початок осей, а п'ята кнопка зліва дозволяє масштабування. Шоста кнопка зліва управляє полями графіку по відношенню до існуючого вікна. Нарешті, сьома кнопка дозволяє зберегти зображення на диск.

### Збереження графіків в файл

Функція `savefig()` дозволяє записувати зображення різних форматів в файл. `Matplotlib` підтримує декілька форматів файлів, у тому числі PDF, PNG і PS. Найпростіший спосіб створити файл, що містить графік, це ввести `savefig(filename)`, де `filename` має розширення, пов'язане з обраним форматом зображення:

```
figure()
plot(range(3))
savefig('line.png')
```

Функцію `savefig()` дозволяє зберігати файл з назвою `line` та розширенням `png` в папку, в якій розміщений файл, що викликає функцію. Інші розширення файлів, яку можуть бути використані для представлення даних є: `eps`, `pdf`, `pgf`, `png`, `ps`, `raw`, `rgba`, `svg`, `svgz`.

Крім того, ми можемо передати аргумент формату зображення у функцію `savefig()` для управління вихідний файл. Можна контролювати інші параметри вихідного зображення, такі як кількість точок на дюйм (`dpi`) і колір рисунка. Більш загальна форма запису функції `savefig()` є `savefig(ім'я_файлу [, параметр1 = значення1] [, параметр2 = значення2], ...)`.

В Таблиця 3.1 перераховані деякі параметри, де під *fn* мається на увазі назва файлу.

З використанням значень в Табл. 3.1 можна об'єднати декілька параметрів, як показано в прикладі:

```
savefig('file', dpi=150, format='png')
```



В результаті виконання такої команди, буде створено файл з ім'ям `file` з розширенням `png` та роздільною здатністю 150. Для того аби подивитися додаткові властивості та параметри функція скористайтеся функцією `help(savefig)`.

Таблиця 3.1. Параметри функції `savefig()`

Параметр	Пояснення	Значення за замовчуванням	Приклад
<code>dpi</code>	Роздільна здатність в крапках на дюйм	Немає	<code>savefig(fn, dpi=150)</code>
<code>facecolor*</code>	Колір картинки	'w' для білого коліру основи	<code>savefig(fn, facecolor='b')</code>
<code>transparent</code>	Коли картинка прозора	False	<code>savefig(fn, transparent=True)</code>
<code>format</code>	Формат файлу	'png'	<code>savefig('image', format='pdf')</code>

#### Побудова графіків: лінії і маркери

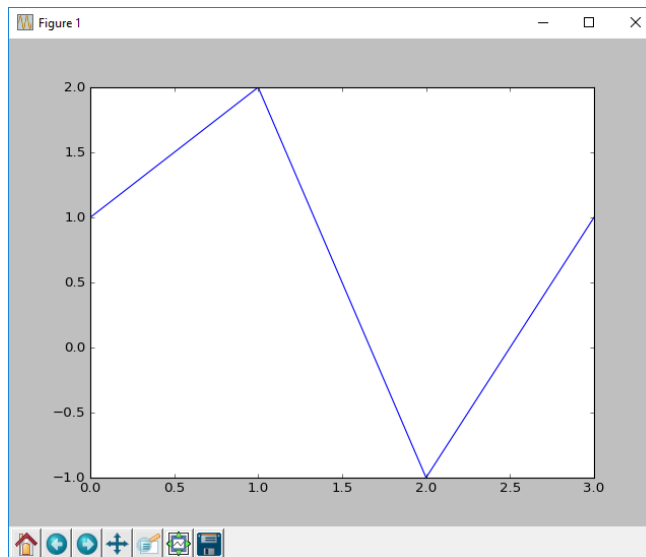
Ми почнемо зі створення вектора для побудови за допомогою пакету NumPy:

```
figure()
y = array([1, 2, -1, 1])
plot(y)
show()
```

Якщо у вас немає графічного інтерфейсу, встановлений з Matplotlib, замініть `show()` на `savefig('filename')` і відкрийте створений файл зображення в програмі перегляду зображень.

На вхід функції `plot()` передається масив `y`. У результаті функція `plot()` намалює графік масиву `y` за допомогою автоінкрементування чисел для осі `x`. Іншими словами, якщо не вказано значення зміни по осі `x`, то функція `plot()` буде автоматично генерувати для вас: `plot(y)` еквівалентна `plot(range(len(y)), y)`.

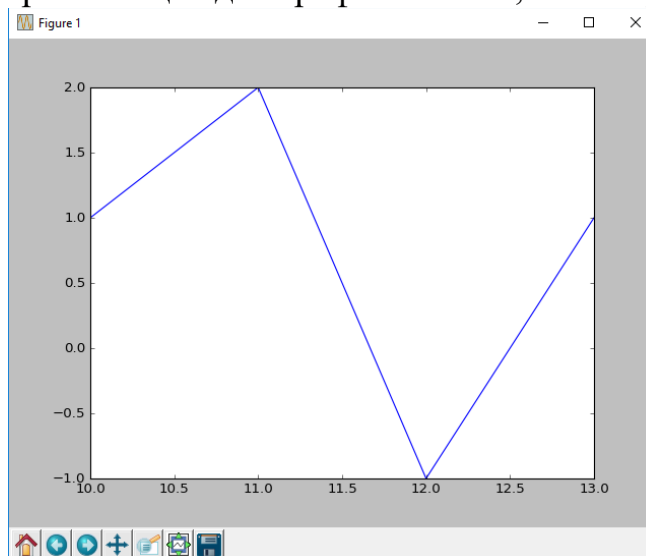
В результаті виконання коду ми отримаємо:



Розглянемо інший випадок, коли значення по осі  $x$  задані:

```
figure()
y = array([1, 2, -1, 1])
x = array([10, 11, 12, 13])
plot(x, y)
show()
```

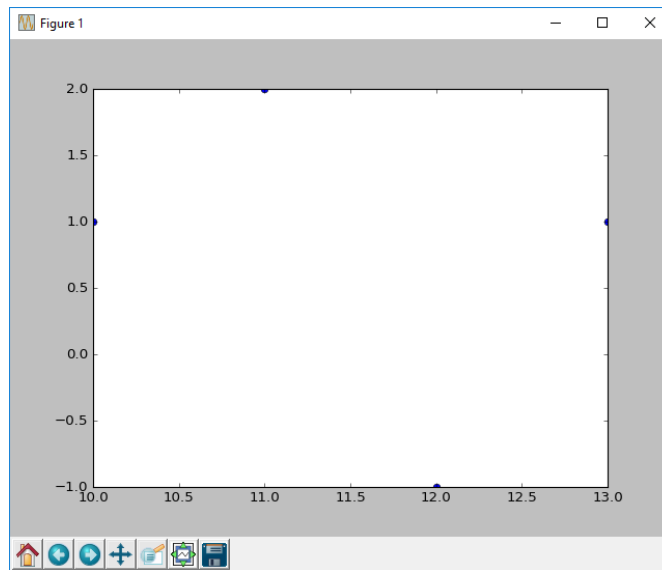
Виклик функції `figure()` створює нову робочу ділянку, якщо використовувати і попередній код для виводу результатів на екран. Тобто після того, як користувався закрити графік, який було виведено в результаті виконання попереднього коду, програмою буде згенеровано ще одне графічне вікно, яке зображено нижче:



Розглянемо ще декілька варіантів. Для прикладу, ми хочемо побудувати залежність  $y$  від  $x$ , але відображати тільки маркери, а не лінії. Це легко зробити:

```
figure()
plot(x, y, 'o')
show()
```

В результаті виконання коду отримаємо:



Для вибору іншого маркера, необхідно замінити символ 'o' з іншим символом маркера. В Таблиці 3.2 перераховані деякі популярні варіанти маркерів.

Таблиця 3.2. Доступні маркери

Характер	Маркер Символ
'o'	Коло
'^'	Трикутник
's'	Квадрат
'+'	Плюс
'x'	Хрест (множення)
'D'	Ромб

Подібно до того, як існують маркери, є також різні стилі ліній, деякі з яких перераховані в Таблиці 3.3.

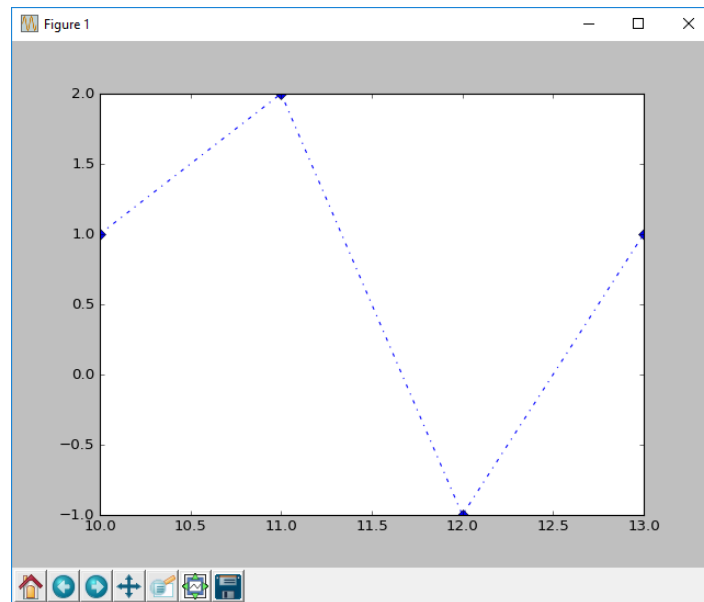
Таблиця 3.3. Доступні стилі ліній

Символ	Стиль лінії
'-'	Суцільна лінія
'--'	Штрихова лінія
'-.'	Штрих-пунктирна лінія
':'	Пунктирна лінія

Якщо необхідно представити на графіку маркери і лінії, потрібно об'єднати символи для стилів ліній і маркерів. Для побудови штрих-пунктичної лінії і ромбоподібних символів в якості маркерів, необхідний такий код:

```
plot(x, y, 'D-')
show()
```

Результат виконання коду програми:



### Побудова декількох графіків на одній фігурі

Ми використовуємо графіки для візуалізації даних і їх порівняння. Що більш важливо, якщо відобразити результати на одному графіку, то можна буде провести більш якісне порівняння даних. Є два способи зробити таке порівняння з використанням бібліотеки Matplotlib. Перший спосіб, щоб додати більше векторів це використання функції `plot()`:

```
plot(x, y, x, 2*y)
```

Інший спосіб, це з використанням маркерів:

```
plot(x, y, '+', x, 2*y, 's-')
```

Другий спосіб полягає у використанні функції `plot()` декілька разів. Іноді у вас може бути необхідність виводити дані тільки частково. Для прикладу, коли ми викликаємо функцію `plot()` з уже існуючим графічним вікном, 2 ситуації можуть трапитися. Перша, коли фігура появляється і новий графік малюється. Інша, коли фігура не витирається, а новий графік добавляється до уже існуючого. Це залежить від стану фіксації малюнка. Контролювати стан утримання можна за допомогою функції `hold()`: викликаючи `hold(True)` забезпечить, що нові графіки не будуть стерати фігуру, в той час як `hold(False)` буде робити протилежне. Викликаючи функцію `hold()` без аргументів буде перемикати статус блокування.

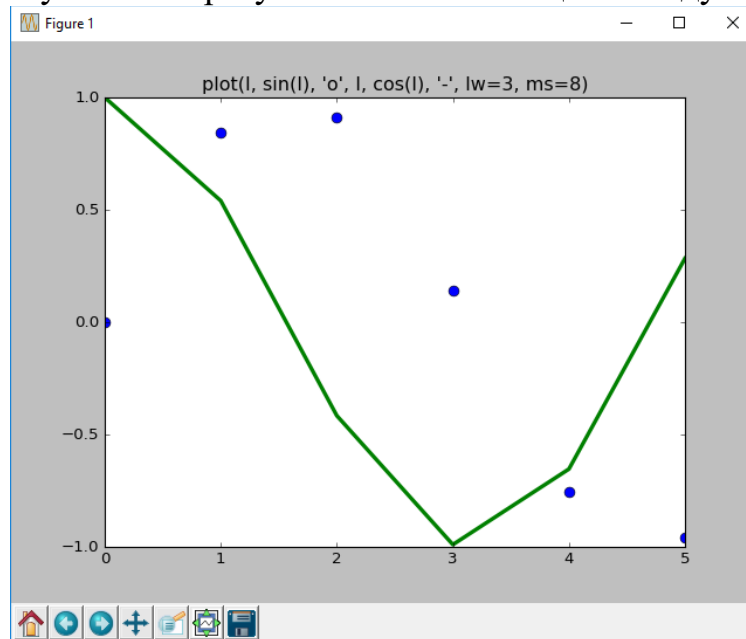
### Ширина лінії і розмір маркерів

Для того аби змінити ширину лінії (або `lw`) і розмір маркерів (або `markersize`), достатньо в функції `plot()` задати їх параметри. Обидва аргументи приймають значення з плаваючою комою; значення за замовчуванням 1.

```
I = arange(6)
plot(I, sin(I), 'o', I, cos(I), '-', lw=3, ms=8)
```

```
title('plot(I, sin(I), 'o', I, cos(I), '-', lw=3, ms=8)')
show()
```

На рисунку знизу показано результат виконання цього коду:



При виведенні декількох графіків за допомогою виклику однієї функції *plot()*, параметри *linewidth* і *markersize* відносяться до усіх ліній, які відображаються за допомогою команди *plot()*. Якщо необхідно мати різні лінії з різними стилями ліній та різних розмірів маркерів на тому ж рисунку, потрібно викликати функцію *plot()* разом з функцією *hold()*, як показано в коді нижче:

```
figure(); hold(True)
I = arange(6)
plot(I, sin(I), lw=4)
plot(I, cos(I), lw=2)
show()
```

### Кольори

Так само, як зі стилем маркера і стилем лінії, ви можете контролювати колір за допомогою однієї букви, відповідно до переліку в Таблиці 3.4.

Таблиця 3.4. Кольори та відповідні їм букви

Символ	Колір
'b'	Синій
'c'	Блакитний
'g'	Зелений
'k'	Чорний
'm'	Пурпуровий
'r'	Червоний
'w'	Білий

## 'y' Жовтий

Як ви помітили, Matplotlib автоматично вибирає інший колір для подальших лінійних ділянок, якщо колір не вказаний. Ви можете вибрати потрібний колір шляхом вибору букви кольору:

```
figure()
I = arange(6)
plot(I, sin(I), 'k+-', I, cos(I), 'm:')
show()
```

В результаті утвориться дві лінії: перша це чорна лінія з маркерів плюс, які з'єднані суцільною лінією, а друга лінія - пурпурний пунктир.

Якщо ви хочете колір, який не показано в таблиці 3.4, ви можете вибрати один з словника об'єктів, `matplotlib.colors.cnames`. Словник містить понад сто значень кольорів. Якщо словника замало, ви можете створити свій власний колір шляхом поєднання червоного-зеленого-синього кольорів (RGB значень). Якщо ви використовуєте словник значення або значення RGB, ви повинні задати колір як аргумент функції `plot()`:

```
plot(randn(5), 'y', lw=5) # 'y' з таблиці кольорів
plot(randn(5), color='yellowgreen', lw=5) # бібліотека matplotlib.colors.cname
plot(randn(5), color='#ffff00', lw=5) # жовтий колір RGB
```

## Керування графіком: осі координат

Функція `axis()` керує поведінкою діапазону осей  $x$  та  $y$ . Якщо ви не передасте жодного параметра в функцію `axis()`, то вона повертається значення кортежу в формі `axis([Xmin, Xmax, Ymin, Ymax])`. Ви можете використовувати `axis()` для того аби встановити новий діапазон осей, вказавши нові значення. Якщо ви хочете встановити або отримати значення по осі  $x$  або значення по осі  $y$ , то це можна зробити за допомогою функції `xlim(xmin, xmax)` або `ylim(ymin, ymax)`, відповідно.

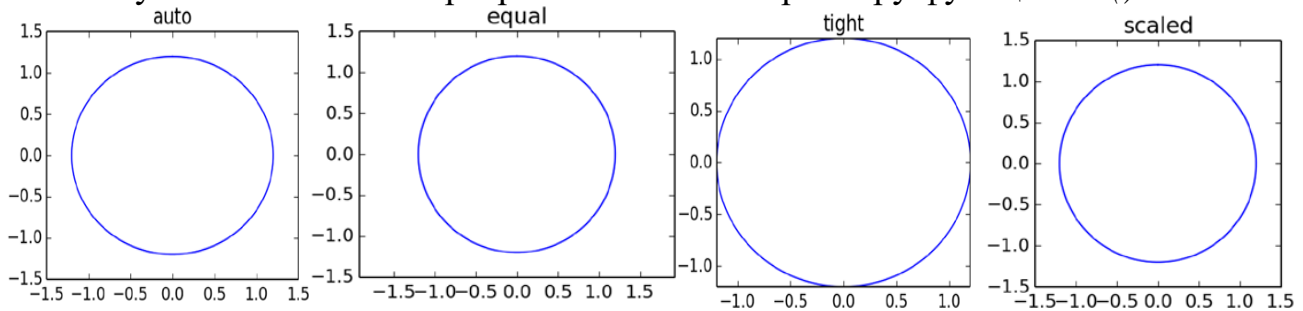
Крім значення діапазону, функція `axis()` може приймати значення: 'auto', 'equal', 'tight', 'scaled' та 'off'. Параметр 'auto', стоїть за замовчуванням і дозволяє приймати рішення, які значення найбільш підходячі. Значення 'equal', змушує кожне значення  $x$  бути такої ж довжини, як і кожне значення  $y$ , що важливо, якщо ви намагаєтеся передати фізичні відстані, наприклад, в GPS відображенні. Значення 'tight' змушує осі змінитися таким чином, щоб максимальне і мінімальне значення  $x$  і  $y$  торкнулися краю графа. Значення 'scaled' змінює діапазон осей  $x$  та  $y$  так, що  $x$  і  $y$  будуть мати однакову довжину (тобто, співвідношення 1). Параметр 'off' видаляє осі і написи.

Проілюструємо використання цих параметрів на прикладі відображення кола за допомогою коду:

```
R = 1.2
I = arange(0, 2*pi, 0.01)
plot(sin(I)*R, cos(I)*R)
```

```
axis('scaled')
show()
```

Результат виконання програми зі зміною параметру функції `axis()`:

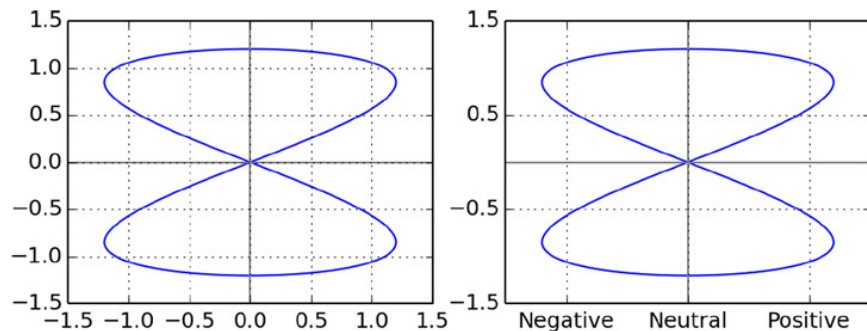


### Сітка і підписи

Функція `grid()` малює сітку у відповідному рисунку. Сітка складається з набору горизонтальних і вертикальних пунктирних ліній, що збігаються з  $x$  і  $y$  позначками. Ви можете перемикаєти сітку, викликавши функцію `grid()` та зробити її видимою `grid(True)` або прихованою `grid(False)`.

Для управління підписами використовуються функції `xticks()` і `yticks()`. Функції отримують вхідний масив в якому записані цифрові значення підписів, а також кортеж із підписами до них. Якщо кортеж міток не записаний, то номери міток використовуються як їх значення.

```
R = 1.2
I = arange(0, 4 * pi, 0.01)
plot(sin(I) * R, cos(0.5 * I) * R)
axhline(color='gray')
axvline(color='gray')
grid()
xticks([-1, 0, 1], ('Negative', 'Neutral', 'Positive'))
yticks(arange(-1.5, 2.0, 1))
show()
```



На рисунку зображено результат виконання коду без звернення до функцій `xticks()` і `yticks()` (лівий графік) та `xticks()` і `yticks()` (правий графік). Зверніть увагу на написи на осі  $x$ .

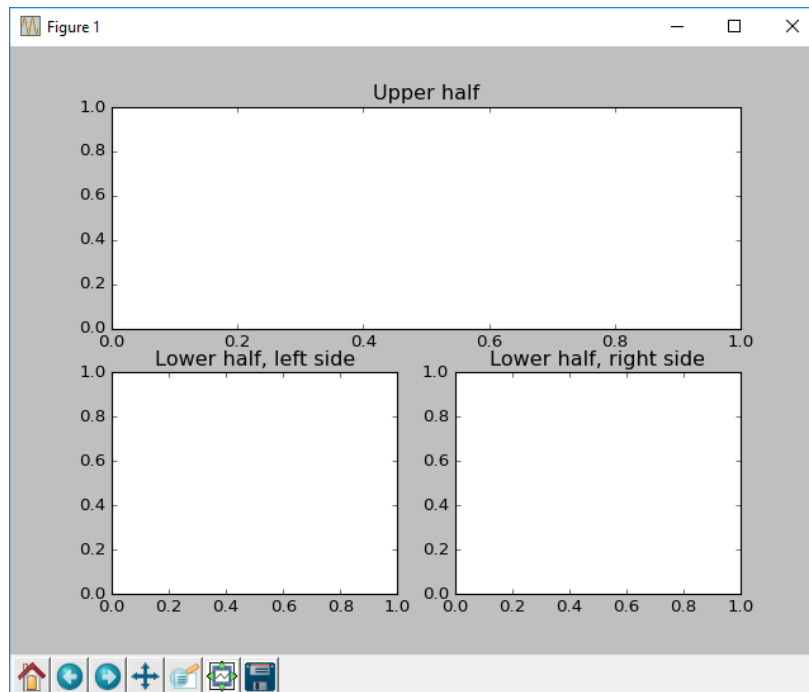
Крім того, було використано функції `axvline()` і `axhline()`, які дозволяють намалювати лінії вздовж осі абсцис і ординат, відповідно. Функції `axhline()` і `axvline()` приймають безліч параметрів, у тому числі колір, ширину лінії і стиль ліній.

### Підграфіки

Для того аби відобразити декілька графіків на одній фігурі необхідно скористатися функцією `subplot()`. Підграфіки нумеруються зліва направо, зверху вниз, тому верхній лівий підграфік має значення 1, а нижній правий відповідає кількості графіків.

Щоб розділити фігуру на 2x2 графіка і вибрати верхній лівий графік для відображення даних, необхідно було би записати команду `subplot(2, 2, 1)`. Іншими словами, перші два числа, які передаються в функцію, відображають умовну матрицю графіків, де третє число показує, який це графік по рахунку, починаючи з лівого верхнього краю. Крім того, можна було би записати цілий рядок '221', який робить те ж саме: `subplot('221')`. Також можна комбінувати підграфіки різних розмірів на одному рисунку. Покажемо це на прикладі коду:

```
figure()
subplot(2, 1, 1)
title('Upper half')
subplot(2, 2, 3)
title('Lower half, left side')
subplot(2, 2, 4)
title('Lower half, right side')
show()
```





### Стирання графік

Функції `cla()` і `clf()` очищають осі і саму фігуру, відповідно. Ці функції корисні, коли ви працюєте з інтерактивним середовищем і хотіли б очистити відповідні осі (тобто встановити значення осей за замовчуванням і очистити намальовані лінії). Також можливо очистити рисунок в цілому, стираючи також осі і підграфіки, використовуючи функцію `clf()`.

Нарешті, ви можете вибрати аби повністю закрити вікно рисунка за допомогою функції `close()`. Якщо надати номер для функції `close()`, то буде закритий рисунок, який відповідає цьому номеру. Тобто, `close(1)` закриє Figure 1, залишивши всі інші фігури відкритими. Якщо ж ви хочете закрити всі фігури, необхідно ввести `close('all')`.

### Додавання тексту

Є кілька варіантів, щоб коментувати ваші графік з текстом. Ви вже бачили, деякі, такі, як використання функцій `xticks()` і `yticks()`. Наступні функції дадуть вам більше контролю над текстом у вигляді графіка.

### Назва

Функція `title(str)` встановлює значення `str` як назва для графіка, яка відображається як підпис над ним. Функція приймає аргументи, перераховані в таблиці 3.5.

Таблиця 3.5. Аргументи

Аргумент	Опис	Значення
<b>fontsize</b>	Контролює розмір шрифту	'large', 'medium', 'small', or фактичний розмір (i.e., 50)
<b>verticalalignment or va</b>	Контролює вирівнювання по вертикалі	'top', 'baseline', 'bottom', 'center'
<b>horizontalalignment or ha</b>	Контролює вирівнювання по горизонталі	'center', 'left', 'right'

Всі вирівнювання базуються на місцезнахоженні за замовчуванням, яке зосереджене над серединою графіка. Таким чином, параметр `ha='left'` друкуватиме назву, починаючи з середини (по горизонталі) і розширюватися вправо. Аналогічним чином, встановивши `ha='right'` друкуватиме заголовок, що закінчується в середині графіка (по горизонталі). Те ж саме відноситься для вертикального вирівнювання. Для прикладу:

`title('Left aligned, large title', fontsize=24, va='baseline', ha = 'left')`

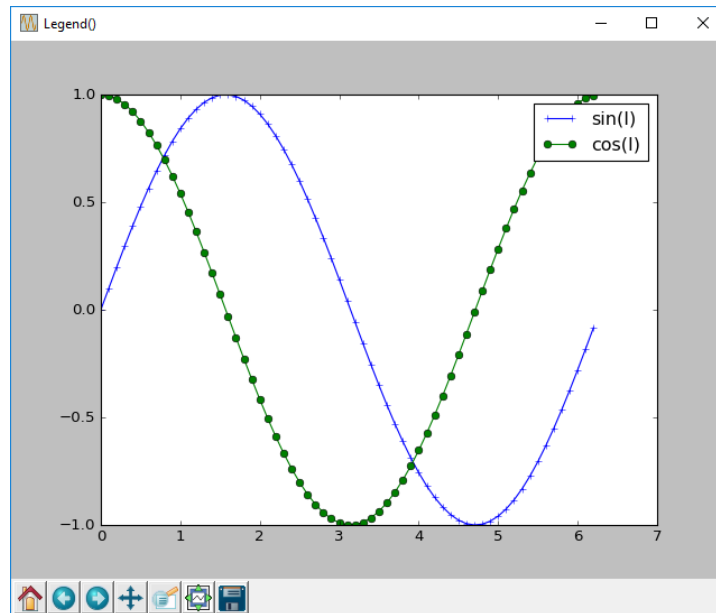
### Підписи осей та легенда

Функції `xlabel()` і `ylabel()` схожі на `title()`, тільки вони використовуються для установки підписів до  $x$  та  $y$  осей. Обидві ці функції приймають текстові аргументи з таблиці 3.5:

```
xlabel('time [seconds]')
```

Наступна в нашому списку текстових функцій `legend()`. Функція `legend()` додає легенду у вікно і показує відповідність між графіком та текстом.

```
I = arange(0, 2 * pi, 0.1)
plot(I, sin(I), '+-', I, cos(I), 'o-')
legend(['sin(I)', 'cos(I)'])
show()
```



Альтернативний підхід аби вивести легенду є визначити аргумент мітки за допомогою виклику функції `plot()`, а потім викликати `legend()` без параметрів:

```
I = arange(0, 2 * pi, 0.1)
plot(I, sin(I), '+-', label='sin(I)')
plot(I, cos(I), 'o-', label='cos(I)')
legend()
show()
```

Крім того, функція `legend()` має додаткові властивості, які дозволяють додати тінь і контролювати відстані між текстом в легенді.

### Вивід тексту

Функція `text(x, y, str)` приймає координати в одиницях графіку  $x$ ,  $y$  та рядок `str`, який необхідно роздрукувати. Вивід тексту відбувається в тій точці, яка задається координатами. Для прикладу, ось код, який виводить текст «origin» в місці з координатою (0, 0):

```
figure()
```

```
plot([-1, 1], [-1, 1])
text(0, 0, 'origin', va='center', ha='center')
show()
```

Функція `text()` має багато інших аргументів, такі як `rotation` (поворот) і `fontsize` (розмір тексту).

### *Математичні символи і вирази*

Синтаксис використання математичних символів, який передбачений в Matplotlib є схожий на TEX. Для того аби вивести математичні вирази, необхідно використовувати рядок і вкласти необхідний математичний вираз за допомогою символу `$`. Для грецьких літер, необхідно починати з косої риски, за якою слідує ім'я літери. Таким чином, щоб надрукувати альфа( $\alpha$ ), ваш рядок повинен бути `r'$\alpha$'`. Дроби можуть бути утворені за допомогою команди `\frac{num}{den}`. Наприклад, `r'$\frac{\pi}{4}$'` в результаті виведе число  $\pi$  поділена на 4. Індокси виводяться за допомогою знаку підкреслення, тобто для того аби вивести  $a_i$  необхідно записати `r'$a_i$'`.

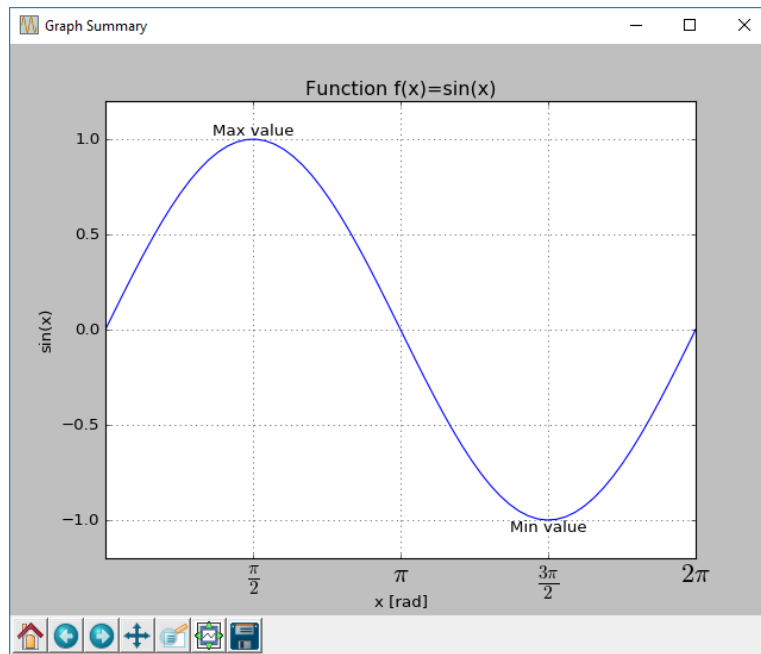
Для отримання додаткової інформації, зверніться до Matplotlib сайту (<http://matplotlib.sourceforge.net/users/mathtext.html>).

Покажемо використання виводу функцій на прикладі:

```
I = arange(0, 2 * pi + 0.1, 0.1)
plot(I, sin(I), label='sin(I)')
title('Function f(x)=sin(x)')
xlabel('x [rad]', va='bottom')
ylabel('sin(x)')
text(pi / 2, 1, 'Max value', ha='center', va='bottom')
text(3 * pi / 2, -1, 'Min value', ha='center', va='top')
xticks(linspace(pi / 2, 2 * pi, 4), (r'$\frac{\pi}{2}$', r'$\pi$', \
                                     r'$\frac{3\pi}{2}$', r'$2 \pi$'), fontsize=20)

xlim([0, 2 * pi])
ylim([-1.2, 1.2])
grid()
show()
```

Результат виконання коду:



### Інші типи графіків

У той час як звичайні лінії і графіки з маркерами є відмінними інструментами візуалізації, вони є не єдині, які активно використовуються для представлення даних.

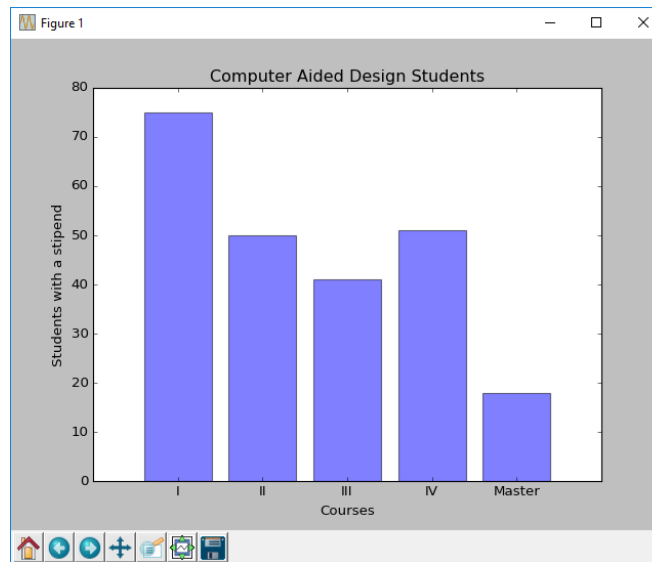
### Стовпцева діаграма (bar chart)

Стовпцева діаграма дозволяє кількісне порівняння декількох значень, що відображаються на графіку у вигляді стовпців, які можуть бути розміщені вертикально або горизонтально. Щоб використовувати стовпцеву діаграму, потрібно викликати функцію `bar(left, height)`, де `left` відповідає координатам  $x$ , а `height` висота стовчика. Функція `bar()` дозволяє мати значні настройки і отримувати більше вхідних параметрів.

Для початку, покажемо використання стовпцевих діаграм на прикладі представлення кількості студентів кафедри САПР, які отримують стипендію, в залежності від курсу.

```
classes = ('I', 'II', 'III', 'IV', 'Master')
y_pos = arange(len(classes))
students = [75, 50, 41, 51, 18]
title('Computer Aided Design Students')
xlabel('Courses')
ylabel('Students with a stipend')
bar(y_pos, students, align='center', alpha=0.5)
xticks(y_pos, classes)
show()
```

Результат виконання представленого вище коду

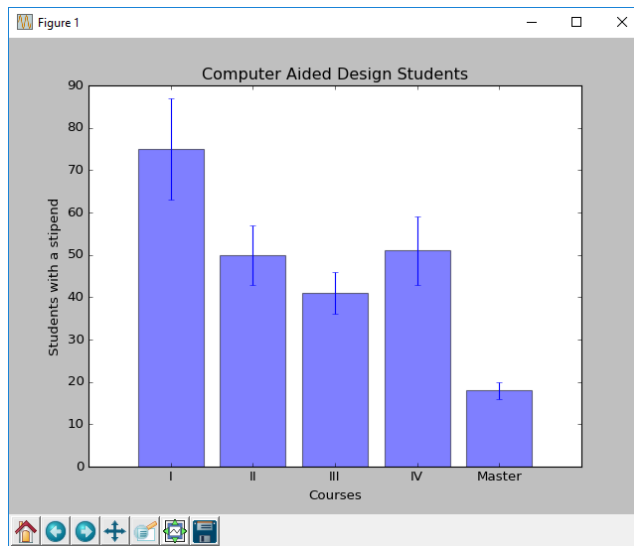


Інший випадок використання стовпцевих діаграм, це ввід результатів в горизонтальному положенні, тобто, коли стовпчики будуть розміщуватися горизонтально, що досить зручно, для представлення виконаної роботи. Для цього достатньо замінити команду `bar()` на `barh()`, зберігши всі параметри такі ж самі.

Крім того, доволі часто потрібно представляти графіки із певною точністю визначення даних, які презентуються. Для цього використовують візуальне представлення похибки визначення параметрів, а на стовпцевій діаграмі ці похибки можна зобразити за допомогою такого коду:

```
classes = ('I', 'II', 'III', 'IV', 'Master')
y_pos = arange(len(classes))
students = [75, 50, 41, 51, 18]
stud_err = [12, 7, 5, 8, 2]
title('Computer Aided Design Students')
xlabel('Courses')
ylabel('Students with a stipend')
bar(y_pos, students, align='center', alpha=0.5, yerr=stud_err)
xticks(y_pos, classes)
show()
```

Результат виконання коду:



Як можна бачити, додатково було введено масив значень для похибки `stud_err`, який присвоюється змінній `yerr` і передається у функцію `bar()`.

### Гістограми

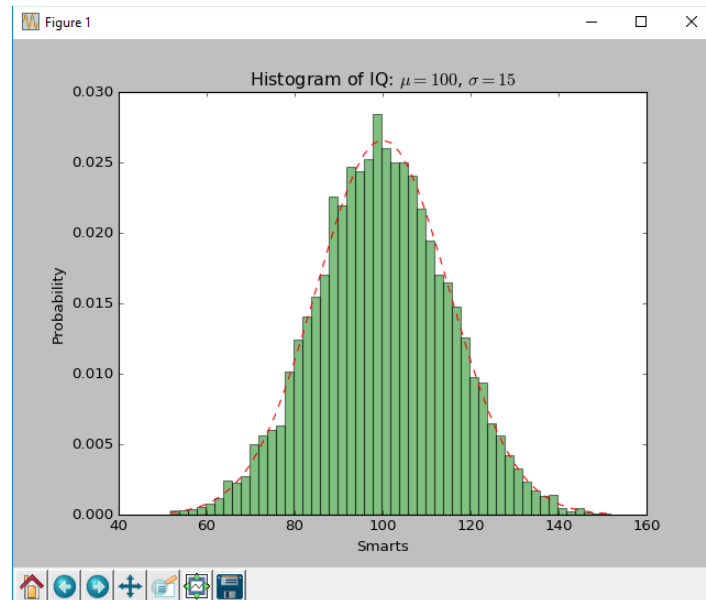
Гістограми – графіки, які показують частоту або появу значень. У Matplotlib, функція `hist()` використовується для розрахунку і представлення гістограм. Для початку ви повинні передати масив значень. Ви можете контролювати кількість клітинок у гістограмі, вказавши їх таким чином: `hist(values, numcells)`. Як варіант в гістограмі можна вказати `bins` `hist(values, bins)`, де `bins` є список, який зберігає значення `bin`. Функція `hist()` повертає значення кортежу ймовірностей, кількості однакових сегментів (`bins`) та патчів (`patches`). Патчі використовуються для утворення стовпчиків. Приклад використання функції `hist()` показано в коді:

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

# вхідні дані
mu = 100 # математичне очікування
sigma = 15 # стандартне відхилення
x = mu + sigma * np.random.randn(10000)

num_bins = 50
# формування гістограми
n, bins, patches = plt.hist(x, num_bins, normed=1, facecolor='green', alpha=0.5)
# додає обвідну для гістограми
y = mlab.normpdf(bins, mu, sigma)
plt.plot(bins, y, 'r--')
plt.xlabel('Smarts')
```

```
plt.ylabel('Probability')
plt.title(r'Histogram of IQ:  $\mu=100$ ,  $\sigma=15$ ')
plt.show()
```



Гістограми вимагають багато додаткових вхідних параметрів, які дозволяють виводити дані у спосіб, який найбільше підходить для нас. Як і стовпцеві діаграми, гістограми можуть виводитися в горизонтальному та вертикальному представленні.

### Кругові діаграми

Кругові діаграми є простим у використанні, як гістограми та стовпцеві діаграми. Функція, яка реалізує кругові діаграми називається *pie(x)*, де *x* є значення, яке буде представлятися.

```
classes = 'I', 'II', 'III', 'IV', 'Master'
students = [75, 50, 41, 51, 18]
sum = np.sum(students)
size = []
for i in range(len(students)):
    size.append(int((students[i]/sum)*100))
    print(size)
colors = ['yellowgreen', 'mediumpurple', 'lightskyblue', 'lightcoral', 'magenta']
explode = (0, 0, 0, 0, 0.05) # пропорції зсуву клина

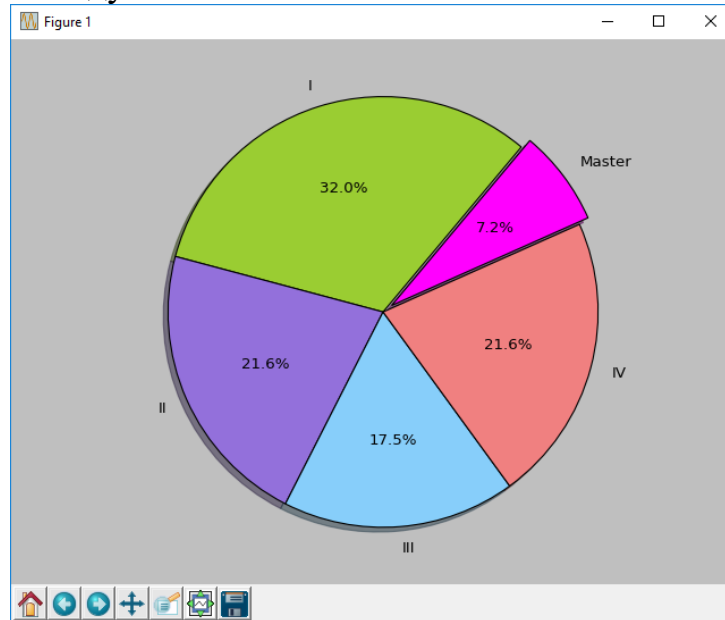
plt.pie(size, # data
        explode=explode, # параметри зсуву сектора
        labels=classes, # назви секторів
        colors=colors, # масив кольорів
        autopct='%1.1f%%', # друк значень в клинах
```

```

        shadow=True, # мінь
        startangle=50 # початковий кут
    )
plt.axis('equal')
plt.show()

```

Результат виконання коду:



### Стовбурові графіки

Стовбурові графіки зображаються лінією, яка починається в точці з координатою  $(x, 0)$  і до  $(x, y)$  для кожного  $(x, y)$  значення. Стовбурові ділянки використовуються для позначення дискретних даних і популярні для побудови профільованих даних, як приклад.

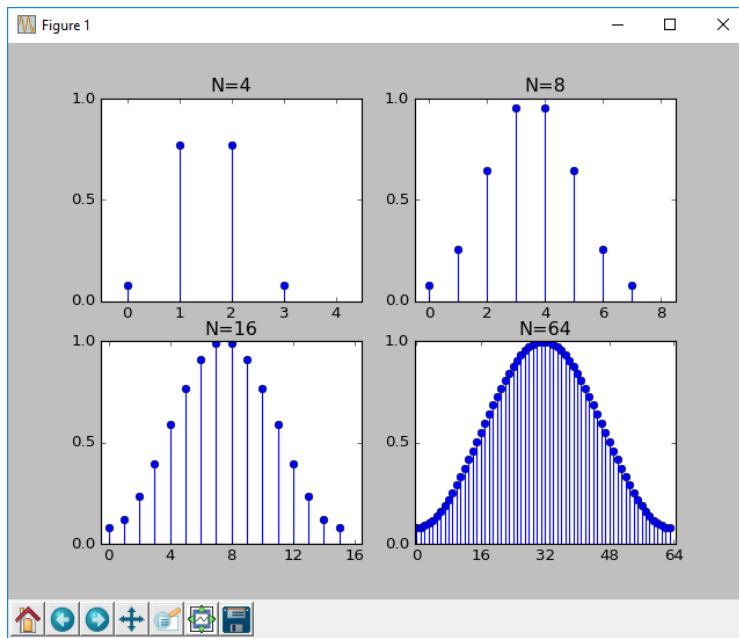
```

from pylab import *
N = [4, 8, 16, 64]
for i, n in enumerate(N):
    subplot(2, 2, i+1)
    stem(arange(n), hamming(n))
    xticks(arange(0, n+1, n/4))
    yticks([0, 0.5, 1])
    xlim(-0.5, n+0.5)
    title('N=%d' % n)
show()

```

Результати виконання коду:





У попередньому прикладі було використано функції *hamming()*, щоб створити вікно Хеммінга, яке використовується в фільтрації даних.

### Графіки 3D

Від недавнього часу бібліотека Matplotlib почала підтримувати 3D графіки. З новими версіями Matplotlib також прийшли 3D графіки за допомогою модуля *mplot3d* ([http://matplotlib.org/mpl\\_toolkits/mplot3d/tutorial.html](http://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html)), який є частиною Matplotlib.

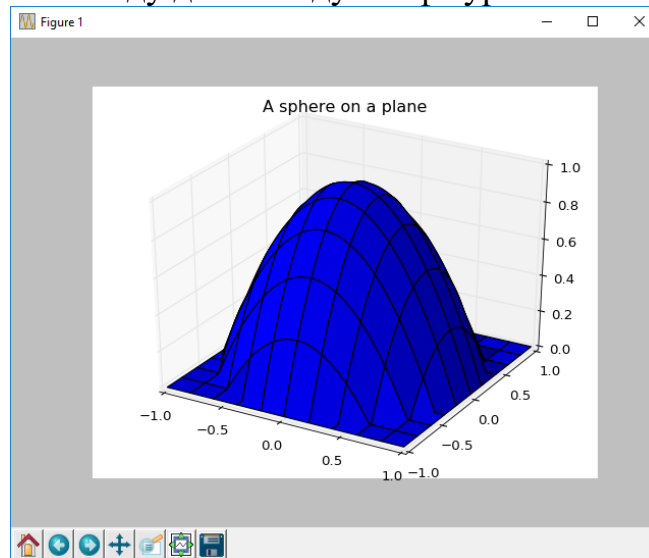
Приклад використання цього модуля покажемо на малюванні фактичної 3D моделі ділянки сфери. Функція *plot\_surface()* малює графік в 3D.

```
from pylab import *
from mpl_toolkits.mplot3d import Axes3D
x = linspace(-1, 1, 100)
y = linspace(-1, 1, 100)
u, v = meshgrid(x, y)
fig = figure()
z = 1-u**2-v**2
z[nonzero(z<0)] = 0
ax = fig.gca(projection='3d')
ax.plot_surface(u, v, z)
title('A sphere on a plane')
show()
```

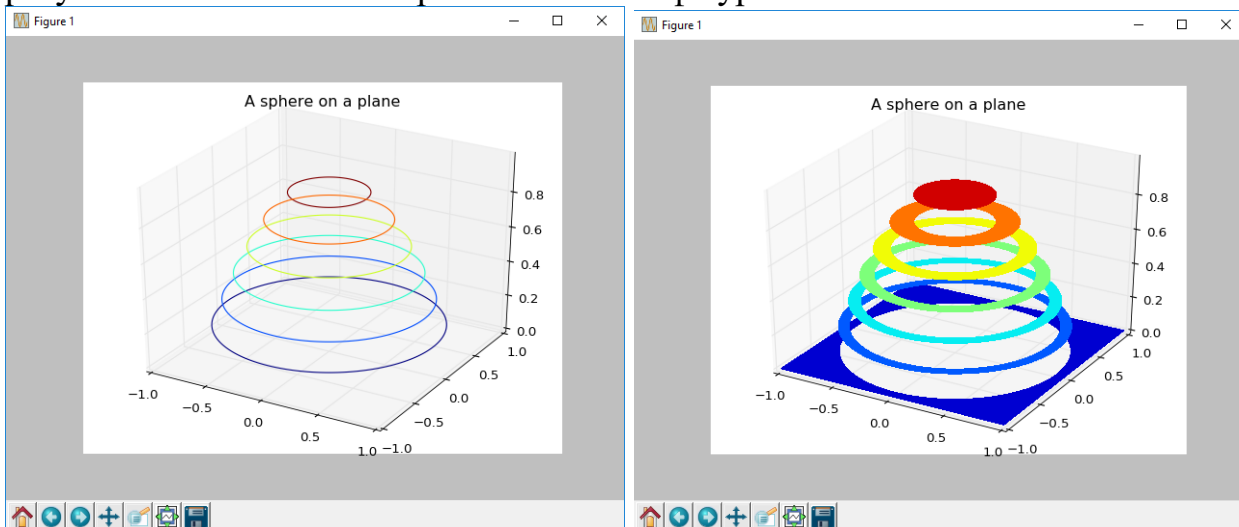
Як можна бачити з коду, після імпортування PyLab, ми імпортуємо *mplot3d* інструментарій. Для створення 3D ділянку, перше, що ви повинні зробити, це створити об'єкт *Axes3D* з параметром *projection='3d'*. Це можна зробити за

допомогою команди `ax = fig.gca(projection='3d')`. Змінна `ax` є об'єктом `Axes3D` і для того щоб отримати доступ до 3D-функції, ви повинні використовувати додаткові функції `Axes3D` (наприклад, `ax.plot_surface()`).

Результат виконання коду для виводу 3D фігури показано нижче.



До інших 3D графіків відносяться також `contour()` і `contourf()` (просто замініть `ax.plot_surface()` на `ax.contour()` або `ax.contourf()` і запусить програму на виконна). В результаті ми зможемо отримати такі 3D фігури.



### Відображення об'єктів на карті

Розглянемо відображення графічних об'єктів довільної форми на прикладі виводу даних із файлу на екран. Як фон фігури було вибрано зображення із назвою `lab3_map.png`. Хотілося би наголосити, що для коректного відображення даних необхідно попередньо встановити бібліотеки `SciPy` та `NumPy`.

Отже, для відображення даних було використано код:

```
from pylab import *
```

```

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

def main():
    # завантажуюємо карту
    fig = plt.figure(figsize=(10, 6))
    img = mpimg.imread('lab3_map.png')
    imgplot = plt.imshow(img)
    plt.axis('off')

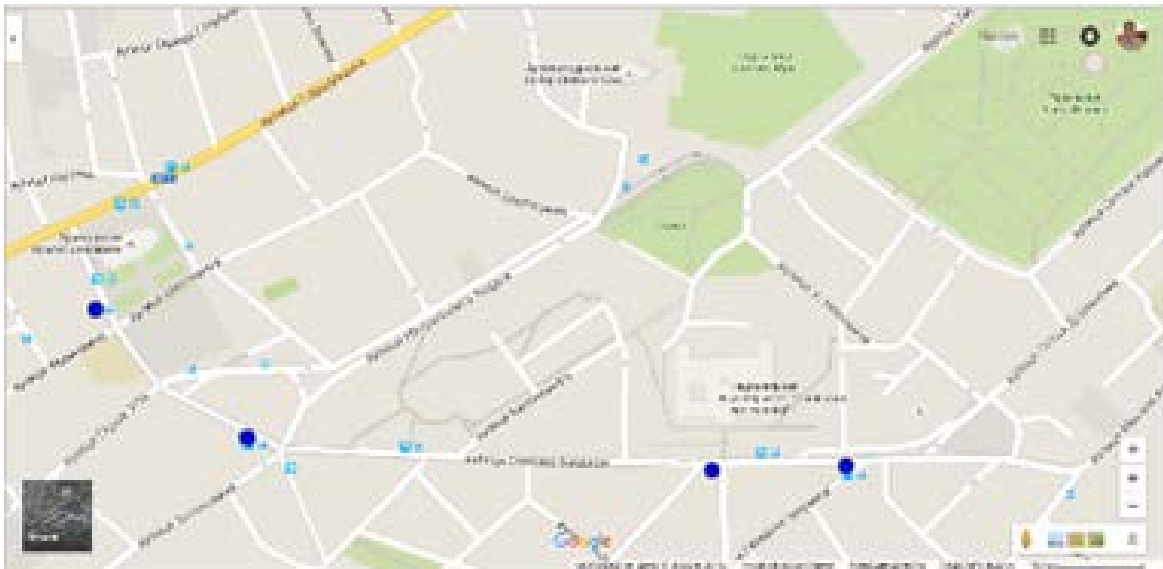
    # читаємо значення координат зупинок з файлу
    with open('train_stops.txt') as tstops:
        train_stops = tstops.read()
        train_stops = train_stops.split('\n')
        x = [row.split(',')[0] for row in train_stops]
        y = [row.split(',')[1] for row in train_stops]
        stops = plot(x, y, marker='o', linestyle='', picker=5)
        cid = fig.canvas.mpl_connect('button_press_event', onclick) # значення
координат мишки
        print('Mouse coordinats:', onclick)
        fig.canvas.mpl_connect('pick_event', onpick)
    plt.show()

def onclick(event):
    if event.xdata != None and event.ydata != None:
        print (event.xdata, event.ydata)

def onpick(event):
    thisline = event.artist
    xdata = thisline.get_xdata()
    ydata = thisline.get_ydata()
    ind = event.ind
    points = tuple(zip(xdata[ind], ydata[ind]))
    print('onpick points:', points)
main()

```

Результат виконання коду:



Як видно із рисунку, на ньому відображена карта місцевості, яка була завантажена із файлу та має розмір 10х6 дюймів. Координати для крапок було зчитано із файлу `train_stops.txt` та відображено за допомогою функції `plot()`. Один із важливих параметрів функції `plot()` є `picker`, який у нашому випадку рівний 5. Цей параметр відповідає за кількість пікселів, які будуть враховувати, коли курсор мишки буде знаходитися в межах реального значення точки зчитаної з файлу.

Щоб отримати дані координат можна використати дві функції: `onclick()` та `onpick()`. Функція `onclick()` повертає значення координат при натисканні клавішею мишки та виводить їх значення на екран. Функція `onpick()` працює подібним чином, проте повертає значення лише тоді, коли курсор мишки знаходиться на відстані в 5 пікселів від реального значення точки (задається параметром `picker`).

### 3. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Переваги пакету Matplotlib.
2. Що таке інтерактивні графіки?
3. Назвіть головні параметри збереження графіків у файл.
4. Які є доступні маркери при виводі графіків?
5. Як відбувається вивід підграфіків?
6. Записати вираз  $\frac{3\pi}{7}$ .
7. Що таке гістограми, де вони використовуються?
8. Спосіб представлення кругових діаграм.

### 4. ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Ознайомитись з теоретичною частиною лабораторної роботи.
2. Одержати індивідуальне завдання (див. Варіанти індивідуальних завдань). При необхідності уточнити завдання на виконання у викладача.

3. Скласти програму на мові програмування Python відповідно до поставленого завдання. Виконана програма повинна повністю реалізувати розв'язок поставленої задачі.
4. Запустити програму та зберегти результат виконання.
5. Оформити звіт відповідно до встановлених вимог.

## **5. ЗМІСТ ЗВІТУ**

1. Мета роботи.
2. Відповіді на контрольні запитання.
4. Індивідуальне завдання, отримане у викладача, згідно з варіантом.
5. Код програми для реалізації завдання.
5. Аналіз отриманих результатів.
6. Ґрунтовні висновки.
7. Список використаної літератури.

## **6. СПИСОК ЛІТЕРАТУРИ**

1. *Лутц М.*, Изучаем Python. - К.:Символ-Плюс , 2011. - 1280 с.
2. *Бизли Д.*, Python. Подробный справочник, 4-е издание. - К.:Символ-Плюс, 2012. - 864 с.
3. Офіційний сайт Python: <https://www.python.org/doc/>

## ВАРІАНТИ ІНДИВІДУЛЬНИХ ЗАВДАНЬ

### Завдання №1

Згенерувати файл, в якому записані числові значення залежності інтенсивності сигналу  $I$  від частоти  $f$  в діапазоні від 10 ГГц до 70 ГГц з кроком 0.2 ГГц. Закон зміни інтенсивності:  $I(f) = \sin(2\pi ft + \varphi)$ .

- а) вивести вміст файлу на екран;
- б) показати на графіку мінімальне та максимальне значення функції, перетину з осями координат, нулі функції;
- в) вивести на одному графіку два графіки залежності інтенсивності, які зсунуті по фазі на величину  $\pi/2$ ;
- г) на одному графіку в різних вікнах зобразити: 1 – закон зміни інтенсивності сигналу; 2 – закон зміни інтенсивності з фазою  $\pi/4$ ; 3 – закон зміни інтенсивності з фазою  $\pi/8$ ;
- д) з використанням кругової діаграми, вивести кількість від'ємних та додатних значень закону зміну інтенсивності в діапазонах 10-30 ГГц, 31-50 ГГц, 51-70 ГГц.

### Завдання №2

Для вибраного мікрорайону Львова (узгоджується індивідуально з викладачем) необхідно розробити програму, яка би:

- а) на карті в довільному порядку зобразити зупинки автобусів, маршрутних таксі та трамваїв. Кожен елемент необхідно представити різним кольором. Дані про розташування зупинок необхідно загрузати з файлу. У лівому верхньому куті вивести відповідність кольору до зупинки;
- б) на основі отриманих даних про зупинки, необхідно провести лінії відповідного кольору, які з'єднують зупинки;
- в) з використанням методів апроксимації, програмованими методами провести обвідну, яка проходить через вибрані точки;
- г) написати програму, яка би визначала відстань від однієї зупинки до іншої, в залежності від того, куди ми клацнули мишкою на карті з врахуванням існуючого маршруту.

# ЛАБОРАТОРНА РОБОТА №4

на тему «Наукове представлення даних в Python»

## 1. МЕТА РОБОТИ

Розглянути можливості наукового представлення даних та графіків з використанням засобів NumPy та SciPy в Python.

## 2. ТЕОРЕТИЧНА ЧАСТИНА

### Змінні і функції

Пакет NumPy надає нам дві корисні допоміжні функції. Будемо називати їх допоміжними функціями, тому що вони не потрапляють до будь-якої конкретної категорії чисельного аналізу чи обробки сигналів.

При роботі в інтерактивному середовищі, постійно виникає потреба визначати змінні. Часом важко пригадати, які змінні визначено і що вони означають. Функція `who()` друкує список поточних масивів NumPy:

```
from pylab import *  
who()
```

Результат виконання:

Upper bound on total bytes = 0

```
from pylab import *  
up, down = arange(10), arange(10, 0, -1)  
who()
```

Виведе:

Name	Shape	Bytes	Type
down	10	40	int32
up	10	40	int32

Upper bound on total bytes = 80

Функція `lookfor()` відмінно підходить для пошуку всередині рядків документації. Тому, для пошуку функцій, які виконують чисельну інтеграцію, виконайте наступне:

```
lookfor('integrate')
```

В результаті виведе:

Search results for 'integrate'

-----

numpy.trapz

Integrate along the given axis using the composite trapezoidal rule.

### SciPy

Бібліотека SciPy (<http://www.scipy.org/>) є науковою бібліотекою для Python з

відкритим вихідним кодом. Ідея SciPy схожа до Octave-Forge (<http://octave.sourceforge.net/>), що забезпечує додаткові пакети для GNU Octave - (<http://www.octave.org>) та інструменти, які покращують MATLAB (<http://www.mathworks.com>). SciPy побудований на базі NumPy, тому вимагає NumPy, щоб працювати належним чином.

SciPy поділена на декілька модулів, деякі з яких наведено в таблиці 4.1.

Таблиця 4.1. Пакети SciPy

Пакет	Опис пакету
Fftpack	Швидке перетворення Фур'є
Integrate	Вбудовані функції, в тому числі звичайні диференціальні рівняння
Interpolate	Інтерполяція функцій
Linalg	Лінійна алгебра
Optimize	Оптимізація функцій, включаючи алгоритми пошуку коренів
Signal	Обробка сигналів
Special	Спеціальні функції (Airy, Bessel, etc.)

Для початку, розглянемо SciPy модулі, які відносяться до чисельного аналізу і обробки сигналів. Додаткові SciPy модулі включають в себе матриці (модуль `scipy.sparse`), статистику (модуль `scipy.stats`) та багато іншого. Для перегляду повного списку доступних модулів, виконайте `import SciPy`, а потім `help(SciPy)`.

Щоб імпортувати конкретний модуль SciPy, виконайте команду `import scipy.modulename`. Наприклад, щоб імпортувати `scipy.linalg`, виконайте:

```
import scipy.linalg
```

Інший спосіб підключення модуля `linalg` є:

```
from scipy import linalg
```

### *Лінійна алгебра*

SciPy і NumPy надають нам багато функцій, щоб мати справу з такими темами як: вирішення систем лінійних рівнянь, операції з матрицями і векторами та розкладання матриці.

### *Вирішення систем лінійних рівнянь*

Щоб вирішити систему лінійних рівнянь, ми спочатку записуємо задачу в матричному вигляді.

$$\begin{aligned} 2x + 3y &= 10 \\ 3x - y &= -1.5 \end{aligned}$$

Ми починаємо з визначення матриці  $M$  і вектора  $V$ . Матриця складається з коефіцієнтів  $x$  та  $y$ , які знаходяться в першому ряду і записуються як  $[2, 3]$ . В



другому ряду [3, -1], відповідно.

```
from pylab import *  
M = array([[ 2, 3], [3, -1]])
```

Далі, ми визначаємо вектор результату [10, -1.5]:

```
V = array([10, -1.5])
```

Тепер все що потрібно це використати функцію *solve()*, яка обчислить систему рівнянь, та функція *print()*, щоб вивести результат на екран:

```
print(solve(M, V))
```

В результаті виконання отримаємо [0.5, 3. ]. Результат означає, що значення *x* дорівнює 0,5, а *y* дорівнює 3.

Це рішення також можливо досягти шляхом обчислення зворотної матриці *M* і множення на вектор *V*:

```
print(dot(inv(M), V))
```

Було використано дві функції: *inv()* і *dot()*. Функція *inv()* обчислює обернену матрицю, а функція *dot()* виконує скалярний добуток. Якщо помножити *inv(M)\*V* отримали б множення елемента на елемент:

```
array([[ 0.90909091, -0.40909091],  
       [ 2.72727273,  0.27272727]])
```

### Вектор і матричні операції

Так як і функція *dot()*, функція *vdot()* повертає скалярний добуток двох векторів. Так що, якщо ви зацікавлені тільки у значенні *x* із попереднього прикладу, можна написати *dot(inv(M)[0], V)*, а результат буде 0.5.

Функція *inner(v1, v2)* буде виконувати скалярний добуток; тобто, помножить кожен елемент в *v1* на відповідний елемент в *v2*, а потім складе їх разом:

```
V1 = array([10, -1.5])  
V2 = array([1, 2])  
sum = 0  
for i in range(len(V1)):  
    sum += V1[i]*V2[i]  
print(sum)
```

В результаті отримаємо 7.0. Проте, цикл можна не використовувати, якщо використати функцію *inner(V1, V2)*, яка поверне такий же результат.

Крім того, функція *inner()* також працює на матрицях:

```
M = array([[2, 3], [3, -1]])  
result = inner(M, inv(M))  
print(M)  
print(result)
```

Результат виконання:

```
[[ 2  3]  
 [ 3 -1]]
```

```
[[ 1.00000000e+00  1.11022302e-16]
 [ 5.55111512e-17  1.00000000e+00]]
```

Схожим чином функція *outer()* здійснює векторний добуток двох векторів чи матриць:

```
V1 = array([10, -1.5])
V2 = array([1, 2])
print(outer(V1, V2))
```

Отримаємо: ([[ 10. , 20. ],  
[ -1.5, -3. ]])

Функція *transpose()* буде переставляти осі, а *conjugate()* буде переставляти осі і відкине уявну частину матриці або вектора:

```
V1 = array([10, -1.5])
V2 = array([1, 2])
print(outer(V1, V2))
print(outer(V2, V1))
print(all(outer(V1, V2) == transpose(outer(V2, V1))))
print(conjugate(V1+1j*V2))
```

Отримаємо:

```
[[ 10. 20.]
 [ -1.5 -3. ]]
[[ 10. -1.5]
 [ 20. -3. ]]
True
[ 10.0-1.j -1.5-2.j]
```

Функція *det(M)* поверне детермінант матриці M:

```
print(det(array([[2, 3], [3, -1]])))
```

Отримаємо:

```
-11.0
```

### Розкладання матриць

Розкладання матриці є переписуванням матриці в специфічній формі. Є багато розкладань в тому числі LU розкладання, сингулярне розкладання, і QR-розкладання. Модуль лінійної алгебри NumPy підтримує деякі матричні розкладання через функції, що показані в таблиці 4.2.

**Таблиця 4.2. Деякі функції розкладання матриць**

Функція	Опис функції
cholesky(m)	Розклад Холецького
eig(m)	Власний(спектральний) розклад
qr(m)	QR розклад

svd(m)

Сингулярний розклад

Наступний код здійснює спектральний розклад і перевіряє результат:

```
A = array([[1, 2], [0, 1]])
L, v = eig(A) # calculate eigenvalues and eigenvectors
print("Verification (should be 0) = ", det(A - eye(2)*L)) # verify eigenvalues
# (should be zero)
print("1st = ", dot(A, v[:, 0]) - L[0]*v[:, 0]) # verify 1st eigenvector (should be 0)
print("2nd = ", dot(A, v[:, 1]) - L[1]*v[:, 1]) # verify 2nd eigenvector (should be 0)
```

Результат виконання:

```
Verification (should be 0) = 0.0
1st = [0. 0.]
2nd = [2.22044605e-16  0.00000000e+00]
```

Було створено матрицю (A) і розраховано її власні значення  $\lambda_{1,2}$  (збережені у векторі L) і власні вектори  $v_{1,2}$  (збережені в матриці v); це робиться в рядку `L, v = eig(A)`. Після оцінки власних значень, вони можуть бути перевірені шляхом розрахунку детермінанту  $\det(A - \lambda \cdot I)$ , який повинен бути рівний нулю; це робиться в рядку: `det(A - eye(2)*L)`. Крім того, для кожного власного вектора  $\lambda$ ,  $\lambda \cdot v$  має дорівнювати  $A \cdot v$ ; це перевіряється в останніх двох рядках:

```
dot(A, v[:,0]) - L[0]*v[:,0] і
dot(A, v[:,1]) - L[1]*v[:,1].
```

### Додаткові функції лінійної алгебри

Додаткові функції лінійної алгебри доступні в модулі `scipy.linalg`. Щоб отримати доступ до функцій лінійної алгебри в SciPy, виконайте `import scipy.linalg from scipy` або `from SciPy import linalg`.

### Чисельне інтегрування

Чисельне інтегрування це процес чисельного розрахунку певного інтеграла. Є багато випадків, коли чисельне інтегрування є важливе. Приклади включають в себе обчислення площі фігури або площі області під графіком, а також рішення диференціальних рівнянь.

Розглянемо використання модуля на прикладі обчислення площі половини круга з радіусом 1. Ми вже знаємо, що ця область буде  $\pi/2$ . Тому в певному розумінні, обчислення площі половини круга буде еквівалентно обчисленню чисельного значення  $\pi$ .

По-перше, створимо два вектори  $x$  та  $y$ . Ці два вектори задовольняють рівнянню кола  $x^2 + y^2 = 1$ . З використанням імпортованого модуля PyLab:

```
from pylab import *
N = 7
x = linspace (-1, 1, N)
y = sqrt(1-x**2)
```

```
print (x**2 + y**2)
```

В результаті отримаємо: [ 1. 1. 1. 1. 1. 1. 1.]

Змінна N була вибрана довільно; N це число точок у векторах x і y. Останній результат показує, що всі точки у векторах x і y задовольняють рівнянню кола,  $x^2+y^2=1$ .

Для візуалізації чисельного інтегрування, я виводжу на графік прямокутники, які апроксимують площу кола:

```
figure()
dx = x[1]-x[0]
for i in range(len(x)-1):
    rect = Rectangle((x[i], 0), dx, 0.5*(y[i]+y[i+1]))
    gca().add_patch(rect)
```

```
title('Approximating the area of half a circle')
axis('equal')
show()
```

Площа області під кривою (тобто інтеграл, який ми обраховуємо) – приблизно дорівнює сумі цих прямокутників. Площа кожного прямокутника є  $(y[i]+y[i+1])*dx/2$ , так що загальна сума може бути записана наступним чином:

```
print(dx*(sum(y[:-1]+y[1:])))
```

В результаті отримаємо: 2.9175533787759904

Результат було помножено на 2, так що ми можемо порівняти результат з  $\pi$  замість  $\pi/2$ . Очевидно, що чим більше N, тим ближче це значення буде до  $\pi$ :

```
for N in [5, 10, 20, 100]:
    x = linspace(-1, 1, N)
    dx = x[1]-x[0]
    y = sqrt(1-x**2)
    est_pi = dx*sum(y[:-1]+y[1:])
    print("N=%d, estimated pi is %f" % (N, est_pi))
```

Отримаємо:

```
N=5, estimated pi is 2.732051
N=10, estimated pi is 3.019232
N=20, estimated pi is 3.101560
N=100, estimated pi is 3.138218
```

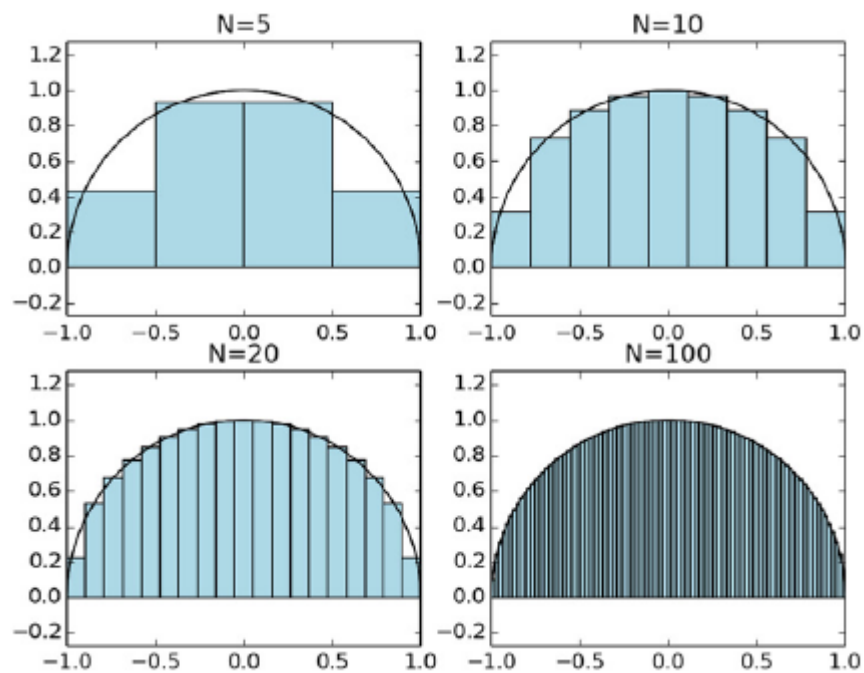
Як ви можете бачити, для N = 100, точність становить приблизно 1%. Це можна побачити на рисунку знизу і відповідно код, який дозволяє вивести результати на екран.

```
from pylab import *
N=1000
x1 = linspace(-1, 1, N)
y1 = sqrt(1-x1**2)
```

```

for i, N in enumerate([5, 10, 20, 100]):
    subplot(2, 2, i+1)
    x = linspace(-1, 1, N)
    y = sqrt(1-x**2)
    dx = x[1]-x[0]
    for i in range(len(x)-1):
        gca().add_patch(Rectangle((x[i], 0), dx, 0.5*(y[i]+y[i+1])),
            fc='lightblue'))
    axis('equal')
    title('N=%d' % N)
    plot(x1, y1, 'k', lw=1)
show()

```



При розрахунку площі круга, було вибрано значення, які розподілені рівномірно. Якщо ви хочете використовувати нерівномірно розподілені значення, реалізація є більш складною. Крім того, метод використовує прямокутники для апроксимації площі області під кривою, але в цьому конкретному прикладі (і багатьох інших), краще підійшли б трапеції, що приводять нас до застосування функції *trapz*(y, x). Функція приймає вектори y та x і повертає чисельний інтеграл. Нижче наведено виконання інтегрування нерівномірно розташованих значень x, використовуючи функцію *trapz*():

```

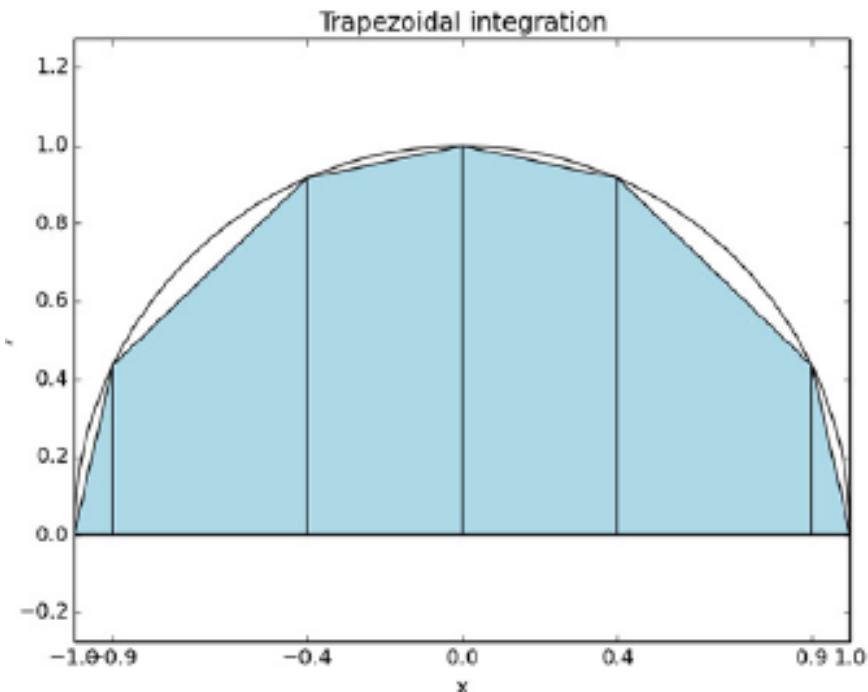
x = array([-1, -0.9, -0.4, 0.0, 0.4, 0.9, 1])
y = sqrt(1-x**2)
print(trapz(y, x)*2)

```

Отримаємо: 2.9727951234089831

Розглянемо візуальне представлення інтегрування трапезіями на прикладі записаного коду та виводу результатів:

```
from pylab import *
figure()
title('Trapezoidal integration')
xlabel('x')
ylabel('y')
# plot the "ideal" circle
N = 1000
x_circle = linspace(-1, 1, N)
y_circle = sqrt(1-x_circle**2)
plot(x_circle, y_circle, 'k', lw=1)
# non-evenly spaced values
x = array([-1, -0.9, -0.4, 0.0, 0.4, 0.9, 1])
y = sqrt(1-x**2)
for i in range(len(x)-1):
    # add trapezoids
    gca().add_patch(Polygon([[x[i], 0], [x[i], y[i]], [x[i+1],y[i+1]], \
        [x[i+1], 0]], fc='lightblue'))
xticks(x)
axis('equal')
show()
```



*Інтерполяція та апроксимація кривих*

Інтерполяція і апроксимація кривої мають справу з вибором функцій, що підходять під відомі дискретні значення. Є декілька причин, чому ми хочемо, щоб функції підходили під відповідні точки даних:

- Підбір відомої функції, під зібрані експериментальні дані. Це може бути корисно у визначенні інших параметрів експерименту.
- Оцінка чисельного значення функцій на додаткових точках (крім заданих).

Інтерполяція знаходить ефективні рішення, які зроблені спеціально для конкретного завдання. Замість написання таблиці для всіх можливих значень, можна придумати многочлен інтерполяції, який є більш ефективним, хоча і з можливою втратою точності. В інших випадках, ви можете спробувати самостійно реалізувати відому функцію, наприклад функцію *sqrt()*, замість використання алгоритму, що присутній в бібліотеці для збільшення продуктивності (при можливій втраті точності).

### Кусково-лінійна інтерполяція

Повернемося до нашого прикладу півкола. На цей раз, ми обмежимося четвертиною кола; тобто, для додатних значень  $x$  і  $y$ . Почнемо з обчислення значення  $y$  для  $x$ , що дорівнює  $0, 0.2, \dots, 1$ . Збережемо результати у векторах  $xp$  і  $yp$ :

```
xp = linspace(0, 1, 6)
print(xp)
yp = sqrt(1-xp**2)
```

Отримаємо: ([ 0. , 0.2, 0.4, 0.6, 0.8, 1. ])

Нам необхідно обчислити значення  $y$  для  $x$  рівних  $0.1, 0.3, \dots, 0.9$ , враховуючи  $xp$  та  $yp$ . Для цього ми використаємо функцію *interp(x, xp, yp)*. Функція повертає значення кусково-лінійної функції, що визначена за  $xp$  та  $yp$  в заданій точці  $x$ . Це означає, що функція *interp()* повертає значення точки на лінії, що з'єднує дві суміжні  $(xp, yp)$  точки. Це називається кусково-лінійною інтерполяцією:

```
xi = arange(0.1, 1.0, 0.2)
yi = interp(xi, xp, yp)
```

Вектор  $y_i$  містить інтерпольовані значення в точках  $0.1, 0.3, \dots, 0.9$ .

Далі показано візуалізацію кусково-лінійної інтерполяції для чверті круга:

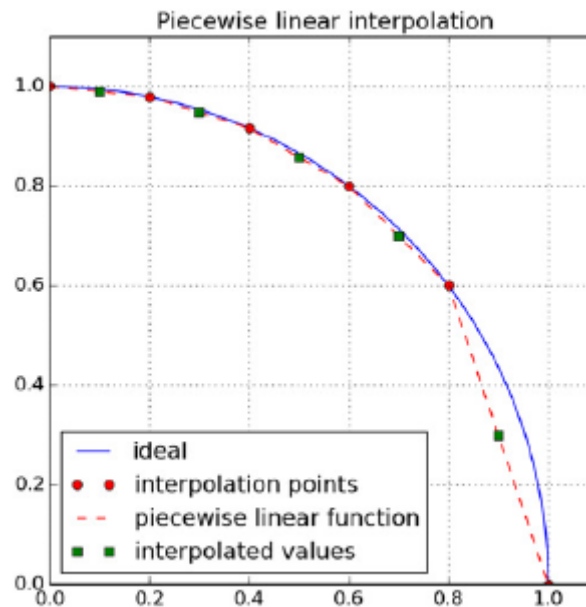
```
from pylab import *
figure()
hold(True)
x = linspace(0, 1, 500)
y = sqrt(1-x**2)
xp = linspace(0, 1, 6)
yp = sqrt(1-xp**2)
xi = arange(0.1, 1.0, 0.2)
yi = interp(xi, xp, yp)
plot(x, y, 'b', label='ideal')
```

```

plot(xp, yp, 'or', label='interpolation points')
plot(xp, yp, '--r', label='piecewise linear function')
plot(xi, yi, 'sg', label='interpolated values')
legend(loc='best')
grid()
axis('scaled')
axis([0, 1.1, 0, 1.1])
title('Piecewise linear interpolation')
show()

```

На рисунку показано результат виконання кусково-лінійної інтерполяції:



Значення  $x_p$  і  $y_p$  обчислюються програмно в цьому прикладі; але насправді, ці значення можуть братись з раніше визначених даних. З графіка можна побачити, що інтерпольовані значення на 0,9 є менш точними за інші інтерпольовані значення. Як правило, чим більше точок додається, тим точнішим буде результат.

### Многочлени

Многочлени це математичні вирази, які включають в себе суму цілих степеней змінної, помноженої на коефіцієнт. Прикладом може бути  $2x^2 + x - 1$ , або навіть  $x$ . При цьому  $\sin(x)$  не є многочленом. Многочлени такі важливі через те, що вони включають лише основні операції: додавання, віднімання і множення (піднесення до степеня може бути реалізовані з використанням декількох множень). Ця властивість дозволяє дуже легко реалізувати їх в обчисленнях. Розклад в ряд Тейлора є яскравим прикладом трансформації функція в многочлен, який легко обчислюється.

Щоб мати можливість працювати з многочленами в NumPy і SciPy, ми



представляємо многочлен у вигляді вектора. Перший елемент у векторі є коефіцієнт з найвищом степенем, а останній елемент є коефіцієнт з найнижчим степенем 0. Таким чином, щоб представити многочлен  $x^2 + 3x + 2$  використаємо такий запис:

```
p = array([1, 3, 2])
```

Для розв'язку рівняння  $x^2 + 3x + 2 = 0$ , використайте функцію *roots(p)*, яка повертає значення коренів:

```
p = array([1, 3, 2])  
print(roots(p))
```

Якщо ви хочете скласти многочлен з його коренів замість коефіцієнтів, то використовується функція *poly()*:

```
p = poly([-2, -1])  
print(p)
```

Отримаємо: [1. 3. 2]

Для додавання і віднімання многочленів використовуються функції *polyadd()* і *polysub()*:

```
p1 = poly([-2, -1])  
p2 = array([1, 0, 0, 0])  
print(polyadd(p1, p2))
```

Отримаємо: [ 1. 1. 3. 2.]

В попередньому прикладі було проведено додавання поліномів  $x^2 + 3x + 2$  та  $x^3$ , а в результаті отримали  $x^3 + x^2 + 3x + 2$ .

Множення і ділення многочленів здійснюється з використанням функцій *polymul()* і *polydiv()*. Значення, що повертається з *polydiv()* є часткою і залишком:

```
p = polymul(array([1, 2]), array([1, 3]))  
print(p)  
print(polydiv(p, array([1, 3])))
```

Отримаємо:

```
[1 5 6]  
(array([ 1., 2.]), array([ 0.]))
```

Інтеграція і диференціювання в многочленах здійснюється використовуючи функції *polyint()* і *polyder()*, відповідно:

```
p = poly([-1j, 1j])  
print(p)  
print(polyder(p))  
print(polyint(p))
```

В результаті отримаємо:

```
[ 1.  0.  1.]  
[ 2.  0.]  
[ 0.33333333  0.          1.          0.          ]
```

У першому рядку було створено многочлен з комплексних чисел; створений многочлен зберігається в змінній  $p$  і дорівнює  $x^2 + 1$ . Використовуючи функцію `polyder()` було обчислено похідну  $p$  і отримали  $2x$ . Використовуючи функцію `polyint()`, було обчислено інтеграл  $p$  і отримали  $\frac{1}{3}x^3 + x$ .

### Використання многочленів

Так чому ж многочлени настільки важливі? Основна причина в тому, що їх можна використовувати щоб апроксимувати функції із експериментально отриманих даних, а також із аналітичних функцій. А так як поліноми вимагають тільки множення і додавання, то реалізації поліномів у вбудованій системі є дуже простою.

Поліноми підбираються до даних за допомогою функції `polyfit(x, y, n)`. Враховуючи вектор точок  $x$  і вектор точок  $y$ , функція `polyfit()` поверне многочлен ступеня  $n$  (найвищий степінь  $x$ ), що найкраще відповідає множині точок даних. Ще одною корисною функцією є `polyval(p, x)`; ця функція повертає значення многочлена в точці  $x$  ( $x$  може бути вектором).

### Приклад: лінійна регресія

Відомим алгоритмом побудови кривої є лінійна регресія. Ідея полягає в тому, щоб намалювати пряму лінію таким чином, що загальна відстань усіх точок до лінії була мінімальною.

Для цього прикладу, було створено пряму лінію, а потім додамо "шум вимірювання" до значень. Аналізуючи нові "зашумлені" дані було проаналізовано поліном першого порядку, який найкраще відповідає даним. Далі, отримані результати порівнюються з відомими правильними значеннями:

```
from pylab import *
# визначення кількості даних
N = 100
start = 0
end = 1

A = rand()+1
B = rand()

# рівня лінії y = A*x + B
x = linspace(start, end, N)
y = A*x + B
y += randn(N)/10

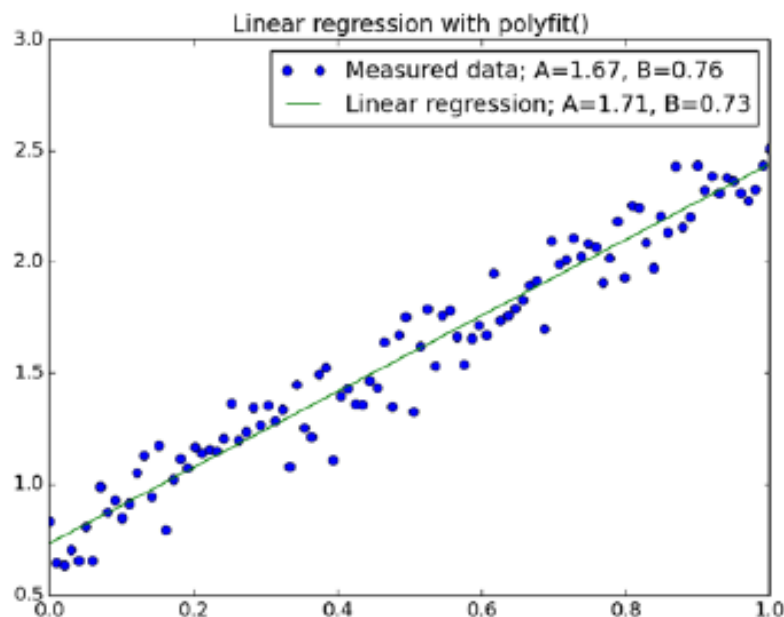
# лінійна регресія
p = polyfit(x, y, 1)
```

```

figure()
title('Linear regression with polyfit()')
plot(x, y, 'o',
      label='Measured data; A=%.2f, B=%.2f' % (A, B))
plot(x, polyval(p, x), '-',
      label='Linear regression; A=%.2f, B=%.2f' % tuple(p))
legend(loc='best')
show()

```

Спершу було вибрано значення для А (від 1 до 2) і В (від 0 до 1), а потім побудовано лінію з шумом використовуючи функцію *randn()*. Далі, використовуючи функцію *polyfit()*, з використанням полінома першого ступеня, що є прямою лінією. В результаті було отримано графік із даними та відповідною інтерпольованою лінією.



### Приклад: лінійна регресія нелінійних функцій

У тих випадках, коли функція, яку ви намагаєтеся апроксимувати не є лінійною, існує можливість все ж виконати лінійну регресію. Розглянемо це на прикладі апроксимації експоненційних даних.

```

from pylab import *
# number of data points
N = 100
start = 0
end = 2

```

```

A = rand()+0.5

```

```

B = rand()

# our linear line will be:
#  $y = B \cdot \exp(A \cdot x) = \exp(A \cdot x + \log(B))$ 

x = linspace(start, end, N)
y = exp(A*x+B)
y += randn(N)/5

# linear regression
p = polyfit(x, log(y), 1)
figure()
title(r'Linear regression with polyfit(),  $y = Be^{Ax}$ ')
plot(x, y, 'o',
      label='Measured data; A=%.2f, B=%.2f' % (A, exp(B)))
plot(x, exp(polyval(p, x)), '- ',
      label='Linear regression; A=%.2f, B=%.2f' % (p[0], exp(p[1])))
legend(loc='best')
show()

```

Регресія виконується при виклику функції *polyfit()*. Було передано  $x$  і  $\log(y)$  відповідні значення, що дозволило використати лінійну регресію до  $\log(y)$  або експоненційну регресію до  $y$ . Результати цієї регресії:

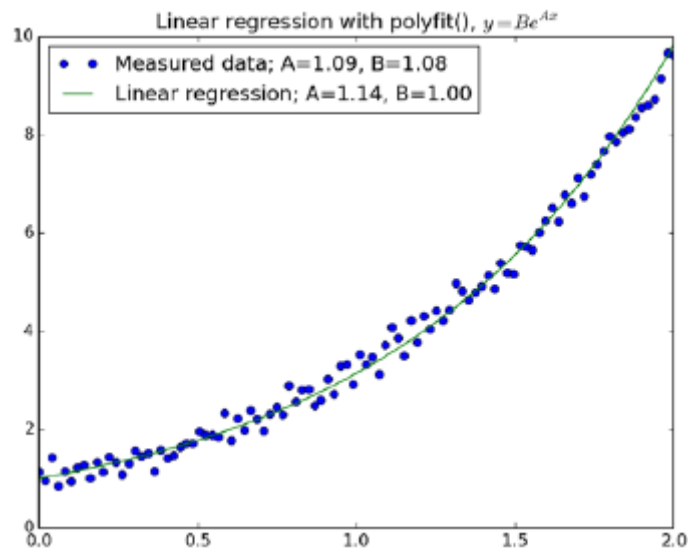


Рис. 4.5. Апроксимація експоненціальних даних

*Приклад: наближення функцій многочленами з*

Ще один набір завдань, що вирішуються за допомогою функції *polyfit()* є наближення функцій з використанням інтерполяції. Це здійснюється простою

реалізацією відомих функцій. В цьому прикладі, ми будемо апроксимувати функцію  $\sin(x)$ .

Ідея полягає в тому, щоб створити поліном, який проходить через відомі точки інтерполяції, тобто розрахувати значення  $\sin(x)$  для відомих  $n$  значень  $x$ , а потім створити багаточлен  $n-1$  ступеня, який проходить через всі ці точки.

Почнемо з вибору набору точок від 0 до  $\pi/2$ ; це будуть наші точки інтерполяції. Значення за межами цього діапазону можна буде обчислити за допомогою тригонометричних тотожностей і функції інтерполяції. Ми вибираємо п'ять точок для інтерполяції, таким чином, вирішуючи, що степінь інтерполяційного многочлена буде 4. Після того, як точки вибрані, ми обчислюємо синус цих точок.

Для цього прикладу було вибрано значення синуса, які можуть бути легко обчислені за допомогою функції `sqrt()`. Значення, які були вибрані для інтерполяції: 0, 30, 45, 60, і 90 градусів. Ці значення були вибрані тому що їх точні значення синусів: 0,  $1/2$ ,  $\sqrt{2}/2$ ,  $\sqrt{3}/2$  і 1, відповідно. У векторній формі, це виглядає наступним чином:

```
values = [0, pi/6, pi/4, pi/3, pi/2]
sines = sqrt(arange(5))/2
print(sines)
```

Отримаємо: [ 0. , 0.5, 0.70710678, 0.8660254, 1. ]

З цими даними інтерполяцію виконати буде просто:

```
values = [0, pi/6, pi/4, pi/3, pi/2]
sines = sqrt(arange(5))/2
p = polyfit(values, sines, len(values)-1)
print(p)
```

Отримаємо: [ 2.87971125e-02, -2.04340696e-01, 2.13730075e-02,  
9.95626184e-01, 1.52055217e-16]

Тому, для реалізації  $\sin(x)$ , нам потрібно зберегти значення  $p$  дане раніше та написати просту підпрограму для обчислення значення  $\sin(x)$  з використанням полінома. Якщо використовувати NumPy, то просто викличемо `polyval()`.

Давайте виведемо різницю між нашою реалізацією  $\sin(x)$  і вбудованою в Python функцією  $\sin(x)$ :

```
values = [0, pi/6, pi/4, pi/3, pi/2]
sines = sqrt(arange(5))/2
p = polyfit(values, sines, len(values)-1)

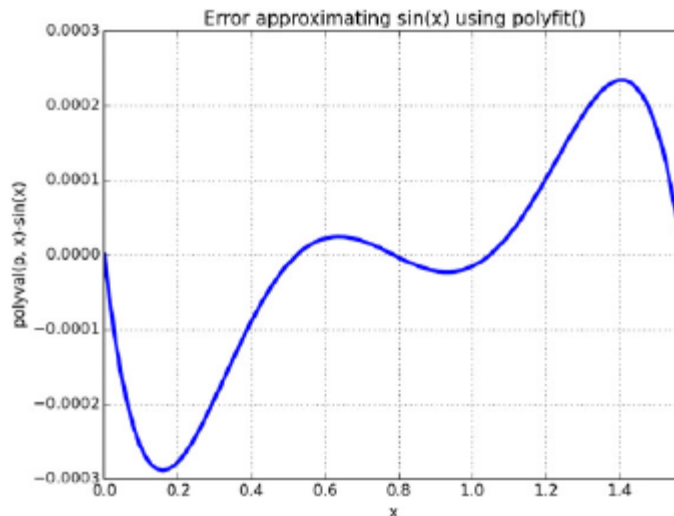
figure()
x = linspace(0, pi/2, 100)
plot(x, polyval(p, x)-sin(x), label='error', lw=3)
grid()
```

```

ylabel('polyval(p, x)-sin(x)')
xlabel('x')
title('Error approximating sin(x) using polyfit()')
xlim(0, pi/2)
show()

```

Результат виконання коду:



Як ми можемо бачити з рисунку, результати є доволі точними; абсолютна похибка є меншою за 0.003

### Сплайн-інтерполяція

Модуль `scipy.interpolate` додає додаткові функції інтерполяції. Одною з них є функція `spline(xp, yp, x)`. Зверніть увагу, що аргументи функції `spline()` впорядковані інакше, ніж в функції `interp()`. Сплайн-інтерполяція є кусково-поліноміальною інтерполяцією, що дотримується певних правил, щоб отримати згладжені результати.

Порівняємо два типи інтерполяції за допомогою коду:

```

from scipy.interpolate import spline
from pylab import *

xp = linspace(0, 1, 6)
yp = sqrt(1-xp**2)
xi = linspace(0, 1, 100)
yi = interp(xi, xp, yp)
ys = spline(xp, yp, xi)
figure()
hold(True)
plot(xi, yi, '--', label='piecewise linear', lw=2)
plot(xi, ys, '-', label='spline', lw=2)

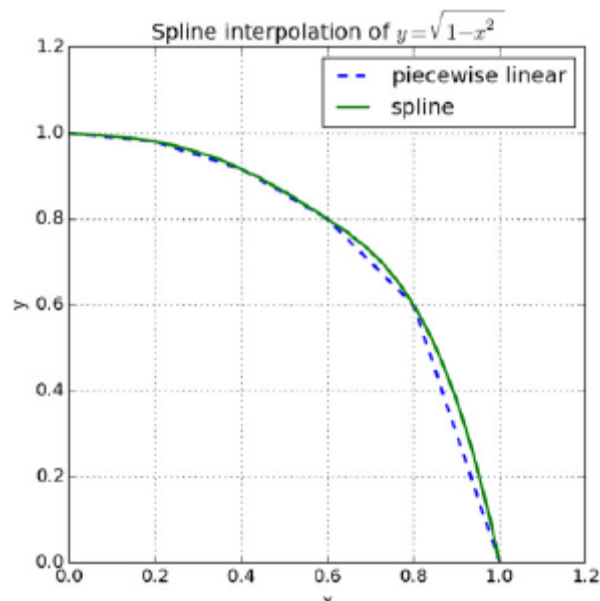
```

```

legend(loc='best')
grid()
title(r'Spline interpolation of  $y=\sqrt{1-x^2}$ ')
xlabel('x')
ylabel('y')
axis('scaled')
axis([0, 1.2, 0, 1.2])
show()

```

На рисунку показано порівняння кусково-лінійної інтерполяції з інтерполяцією сплайнами. Сплайн-інтерполяція виглядає "гладшою".



### Обробка сигналів

До сих пір в цій главі, ми мали справу з чисельним аналізом. Далі розглянемо можливості Python для обробки сигналів. Обробка сигналу являє собою величезне поле, яке має справу з сигналами, тобто значеннями, які змінюються з плином часу. Функціональні можливості обробки будемо розглядати з модулем *scipy.signal*.

### Функції *find()*, *nonzero()*, *where()* і *select()*

Функція *find(умова)*, знаходить індекси в масиві, для яких виконується умова:

```

from pylab import *
squares = arange(10)**2
print('squares = ', squares)
I = find(squares < 50)
print('I = ', I)
squares2 = squares[I]
print('squares2 = ', squares2)

```

Отримаємо:

```
squares = [ 0  1  4  9 16 25 36 49 64 81]
```

```
I = [0 1 2 3 4 5 6 7]
squares2 = [ 0  1  4  9 16 25 36 49]
```

У попередньому прикладі було створено вектор, який містить квадрати значень чисел 0-9. Далі було знайдено всі індекси вектора, які задовольняють умові, коли значення менші 50. Зверніть увагу, що повернені значення є вектором індексів. Якщо потрібно значення, а не індекси, необхідно отримати доступ до вихідного масиву за допомогою *squares[I]*.

Функція *nonzero(послідовність)* повертає ненульові елементи заданої послідовності. За принципом дії нагадує *find(послідовність!=0)*. Проте, функція *nonzero()* повертає 2-D матрицю, тоді як *find()* повертає індекси заданої послідовності.

```
from pylab import *
A = eye(3)
print('A = ', A)
I1 = find(A!=0)
# print('A[I1] = ', A[I1])
print('I1 = ', I1)
print('A.ravel()[I1] = ', A.ravel()[I1])
I2 = nonzero(A)
A[I2] = 3
print('A = ', A)
```

Отримаємо:

```
A = [[ 1.  0.  0.]
      [ 0.  1.  0.]
      [ 0.  0.  1.]]
I1 = [0 4 8]
A.ravel()[I1] = [ 1.  1.  1.]
A = [[ 3.  0.  0.]
      [ 0.  3.  0.]
      [ 0.  0.  3.]]
```

Як видно із вище наведеного прикладу було створено 2D матрицю A та використано функцію *find(A != 0)* щоб знайти відмінні від нуля елементи в матриці A. Зверніть увагу, що значення, яке повертається змінній I1 є 1D масив; це означає, що масив 2D індексується. Тому, спроба вивести результат виконання A[I1] викликає помилку програми, тому була закоментована для правильної роботи програми. Щоб отримати доступ до значення A, необхідно спочатку вирівняти матрицю A використовуючи функцію *ravel()*. Проте, якщо використовувати функцію *nonzero()*, можна отримати доступ до 2D елементів, як показано в коді лінії A[I2] = 3.

Функція *where(умова, x, y)* приймає три масиви однакового розміру: *умова*, *x* і



у, а далі оцінює кожен елемент згідно умови. Якщо елемент має значення True, то повертається значення відповідного елементу з *x*. Якщо елемент має значення False, то повертається значення відповідного елементу *y*:

```
from pylab import *
up = arange(10)
print('up = ', up)
down = arange(10, 0, -1)
print('down = ', down)
highest = where(up > down, up, down)
print('highest = ', highest)
```

В результаті виконання отримаємо:

```
up = [0 1 2 3 4 5 6 7 8 9]
down = [10 9 8 7 6 5 4 3 2 1]
highest = [10 9 8 7 6 5 6 7 8 9]
```

Функція *select(умова, Vals, default = 0)* додає додаткові властивості до функції *where()* дозволяючи декілька умов. Функція приймає список умов, зазначених в *умова* і повертає відповідний елемент пов'язаний з *Vals*, якщо умова виконана; якщо жодна з умов не виконується, то вибирається значення за замовчуванням:

```
up = arange(10)
ramp = select([up < 4, up > 7], [4, 7], up)
print('ramp = ', ramp)
```

Отримаємо:

```
ramp = [4 4 4 4 4 5 6 7 7 7]
```

Перші три елементи *up* є меншими 4, тому умова  $up < 4$  виконана, в результаті чого відбувається вибір значення 4. Останні три елементи більші ніж 7, в результаті чого вибираються величини 7. Значення, які більші або дорівнюють 4 але менші ніж 7 залишаються без змін, так як за замовчуванням встановлюються рівним *up*. Ця функція називається *clipping* і доступна як метод NumPy ndarray об'єкте і як автономна функція *clip()*.

#### Приклад: Просте виявлення сигналу в шумі

Виявлення сигналів в присутності шуму грає важливу роль в великій кількості різного роду прикладних застосувань. Наприклад, воно використовується в системах зв'язку, в виявленні сигналів, таких як радіо- чи телевізійних передач та виділити їх від шуму. Воно також використовується в медицині для виявленням сигналу ЕКГ, а також у багатьох інших областях.

Розглянемо роботу виявлення сигналу на прикладі виявлення його з побудованого чистого сигналу. Під сигналом, я маю на увазі одновимірний масив (вектор), де значення зберігаються як функція залежності від часу. Наша мета буде виявити "події", які будуть представлені вузькими трикутниками, розміщеними

випадковим чином в часі. Там може бути кілька подій в сигналі.

Для створення трикутного імпульсу використаємо функцію `signal.triang()`. Функція генерує трикутне вікно заданого розміру. Помістимо випадковим чином трикутні імпульси в векторі сигналу, як показано в прикладі нижче.

```
from pylab import *
from scipy import signal
# задаємо параметри сигналу
n = 100
t = arange(n)
y = zeros(n)
num_pulses = 3
pw = 11
amp = 20
for i in range(num_pulses):
    loc = floor(rand()*(n-pw+1))
    y[loc:loc+pw] = signal.triang(pw)*amp

# Виводимо графік на екран
figure(1)
title('Signal')
xlabel('t')
ylabel('y')
# встановлюємо межі зміни по осі ординат
axes = gca()
#axes.set_xlim([0,20])
axes.set_ylim([-5,25])
plot(t, y)

# додавляємо до сигналу шум
y += randn(n)
figure(2)
title('Signal and noise')
xlabel('t')
ylabel('y')
plot(t, y)
show()
```

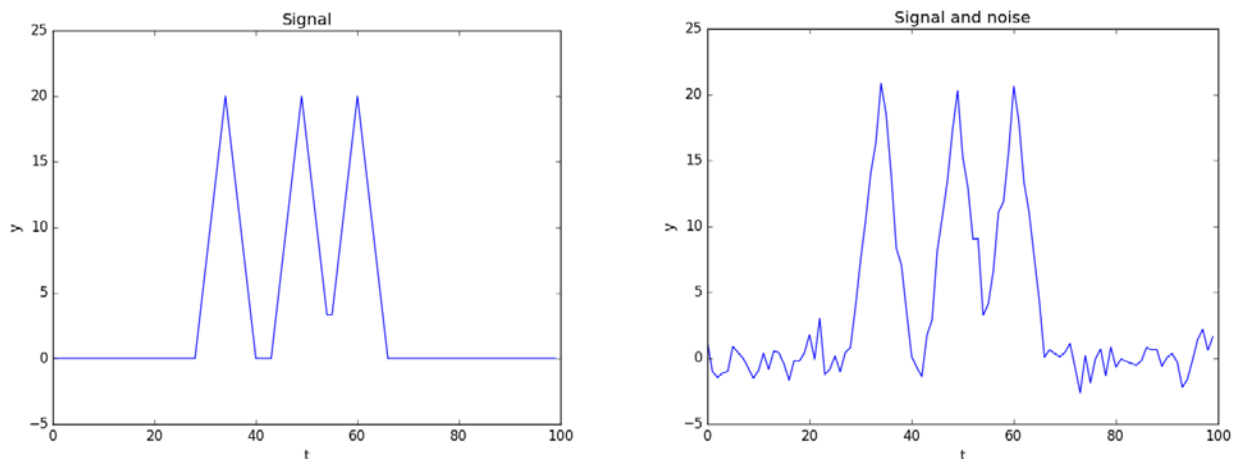
По-перше, було визначено деякі параметри для формування сигналу. Число точок в сигналі  $n$  дорівнює 100. Кількість трикутних імпульсів `num_pulses` було задано рівно 3. Кожен трикутний імпульс буде генеруватися з використанням `pw=11` точок. Максимальне значення піку сигналу амплітуди є `amp`.

Після того, як задали всі параметри було створено два вектора: вектор часу  $t$ ,

та вектор значень  $y$ . Вектор  $t$  є деяка довільна мітка часу; в цьому прикладі, він збільшує значення, починаючи від 0 і до  $n-1$ . Вектор  $y$  спочатку встановлений на рівні нуля.

Після цього випадковим чином було згенеровано трикутні імпульси. Змінна, де буде розміщений трикутний імпульс  $loc$  генерується випадковим чином за допомогою функції  $rand()$ , яка генерує значення між 0 і 1. Таким чином було вибрано значення від 0 до  $n - pw + 1$ , забезпечуючи розміщення піків в межах вектора  $y$ . Після того, як були встановлені піки за допомогою функції  $randn()$  було добавлено шум до сигналу. Функція розподілу шуму представлена як гауссовий розподіл, або "білий шум". Було використано нормальний розподіл з дисперсією 1 і середнє 0. Відзначимо, що функція  $randn()$  відрізняється від  $rand()$ .

Результат виконання описано вище коду для згенерованих сигналів показані нижче.

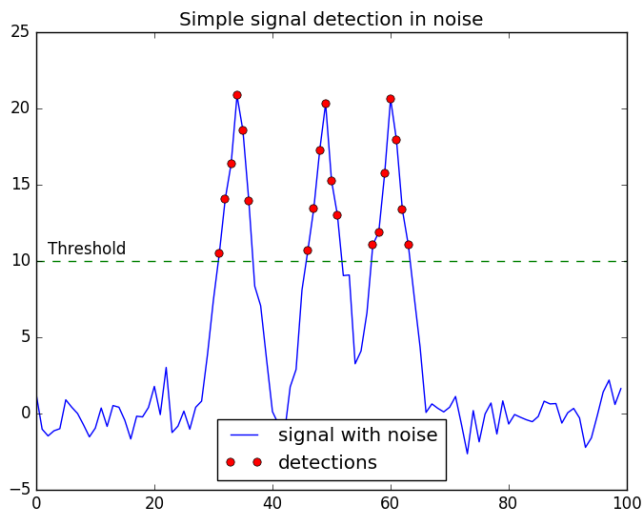


В процесі виконання завдання було показано, що можна спостерігати спостерігати один або два піки замість трьох. Це нормально, так як ми хотіли продемонструвати певну хаотичність результатів.

В попередньому прикладі ми показали, як створити сигнал. Тепер давайте його виявимо. Для виявлення, ми будемо використовувати простий алгоритм: щоразу, коли значення вище встановленого порогу, ми вважаємо це як подія або виявлення. Для нашого випадку ми встановили поріг на рівні  $amp/2$  і використали функцію  $find()$ , як показано в прикладі нижче.

```
# детектування сигналів
thr = amp/2
I = find(y > thr)
# вивід результатів детектування
figure(3)
hold(True)
plot(t, y, 'b', label='signal with noise')
plot(t[I], y[I], 'ro', label='detections')
plot([0, n], [thr, thr], 'g--')
```

```
# Бувід результатів
text(2, thr+.2, 'Threshold', va='bottom')
title('Simple signal detection in noise')
legend(loc='best')
show()
```



### Функції *diff()* і *split()*

Інший набір функцій, які корисно використовувати при виявленні сигналу є *diff()* і *split()*. Функція *diff(v)* повертає вектор, що складається з різниць елементів в *v*. Функція *split(v, індекси)* розділяє вектор за індексами. Відмінність першого порядку визначається як  $out[n] = a[n+1] - a[n]$  вздовж даної осі, а відмінності вищих порядків обчислюються за допомогою виклику функції *diff()* рекурсивно:

```
from pylab import *
import random
v = []
for i in range(10):
    value = random.randint(2, 10)
    v.append(value)
print('Initial array:', v)
print('Function diff():', diff(v))
print('Function split():', split(v, [4, 8]))
```

Результат виконання:

Initial array: [9, 8, 4, 5, 10, 8, 5, 9, 6, 3]

Function diff(): [-1 -4 1 5 -2 -3 4 -3 -3]

Function split(): [array([9, 8, 4, 5]), array([10, 8, 5, 9]), array([6, 3])]

### Виявлення сигналу в шумі, Частина 2

У попередньому прикладі було продемонстровано просте виявлення сигналу за допомогою функції `find()`. Крім того, було відображено усі точки, які лежать вище певного порогу. У багатьох випадках нас не надто цікавлять точки, які розміщені під встановленим порогом, оскільки поріг вибирається довільно. Більший інтерес маємо щодо точок, які розміщені над пороговим значенням.

Скористаємося попередньою програмою, для генерування та визначення точок вище порогового значення. На відміну від попереднього ми будемо визначати максимальне значення серед значень, які лежать вище порогового значення. Нижче представлено код, який дозволяє провести такі розрахунки:

```
J = find(diff(I) > 1)
for K in split(I, J+1):
    ytag = y[K]
    peak = find(ytag==max(ytag))
    plot(peak+K[0], ytag[peak], 'sg', ms=7)
```

Ідея алгоритму полягає в наступному: ми розділили виявлення на окремі групи; і в кожній групі, ми знаходимо вершину і відображаємо її.

■ **Примітка** Оскільки обидва шум і сигнал вибираються випадковим чином, фактичні числові значення можуть відрізнитися від тих, які представлені в цьому прикладі. Проте, концепції та ідеї, як і раніше актуальні.

Перша проблема розділення використовує індекси виявлених значень. Група вважається одним знайденим, якщо індекси є послідовними. Кожен раз, коли є стрибок індексів, це означає нову групу:

```
print('find(y>thr) = ', find(y > thr))
```

Отримаємо масив [31 32 33 34 35 36 46 47 48 49 50 51 57 58 59 60 61 62 63]

Таким чином, група [31 32 33 34 35 36] одна група, група [46 47 48 49 50 51] друга група, а [57 58 59 60 61 62 63] є третьою групою.

Функція `diff()` повертає значення, яке не рівне 1, якщо появляється нова група.

Функція `print('diff(I) = ', diff(I))`,

повертає масив [ 1 1 1 1 1 10 1 1 1 1 6 1 1 1 1 1].

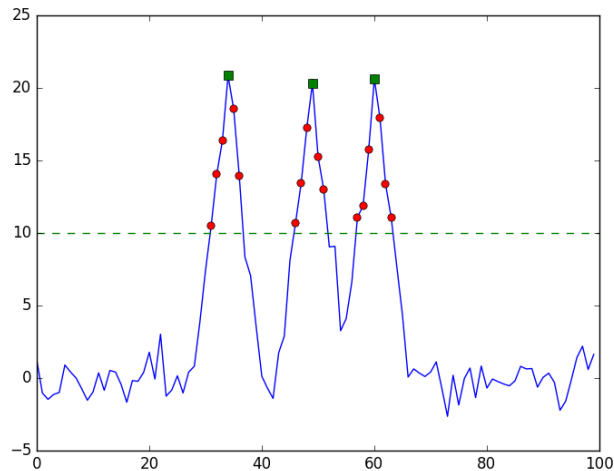
Функція `print('find(diff(I) > 1) = ', J)`, повертає масив [ 5 11].

Таким чином ми хочемо розділити наші дані на 6 елементі та завершити на 12 елементі. Для цього використовуємо функцію `split()`.

```
print('split(I, J+1) = ', split(I, J+1))
```

```
array([31, 32, 33, 34, 35, 36], dtype=int64), array([46, 47, 48, 49, 50, 51], dtype=int64),
array([57, 58, 59, 60, 61, 62, 63], dtype=int64)]
```

Для того аби знайти пік на вибраному наборі даних можна використати функцію `find(ytag==max(ytag))`. Результат виконання алгоритму програми показаний нижче.



## Фільтрація

Однією з причин того, щоб перетворити сигнал з часової області в сигнал в частотній області, є те, що фільтрація в частотній області є набагато простішою. Фільтр являє собою операцію, яка змінює сигнал. Принцип роботи фільтра полягає у тому аби дозволити деякі частоти бути присутніми в сигналі, в той час як іншим частотам заборонити. Фільтри використовуються в різних прикладних задачах, починаючи від аудіо і до радіолокаційним системам.

Фільтри поділяються за своєю поведінкою. Фільтр, який пропускає низькі частоти і зупиняє високі частот називається фільтром нижніх частот (ФНЧ). Аналогічно, фільтр верхніх частот (ФВЧ) пропускає тільки високі частоти. Існують також і інші категорії, такі як смуговий фільтр (дозволяє тільки певну смугу частот), режекторний фільтр (не дозволяє нічого, крім певної смуги частот), і вузькосмугових фільтрів (пропускає тільки дуже мале число частот).

## Розробка фільтру

Конструкція фільтра є доволі складною темою, тому ми коротко розглянемо лише найважливіші нюанси з проектування фільтрів в Python з використанням SciPy.

Модуль *scipy.signal* включає в себе декілька функцій, щоб допомогти створити фільтр. Функція *iirdesign()* використовується для проектування рекурсивного фільтра (IIR фільтр). Інші корисні конструктивні рекурсивних фільтрів включають *butter()*, *cheby1()*, *cheby2()* і *ellip()*. Нерекурсивний фільтр (FIR фільтр) предствлений функціями *remez()* і *firwin()*. Не будемо детально заглиблюватися в їх опис, але при необхідності можна скористатися онлайн допомогою бібліотеки SciPy. Крім того, якщо необхідно переглянути частотні характеристики фільтра, використовуйте функції *freqz()* і *freqs()*.

Для прикладу нижче показано код для низькочастотного фільтра Баттерворта (IIR фільтр) і виведені графічно його частотні характеристики.

```

from pylab import *
from scipy import signal

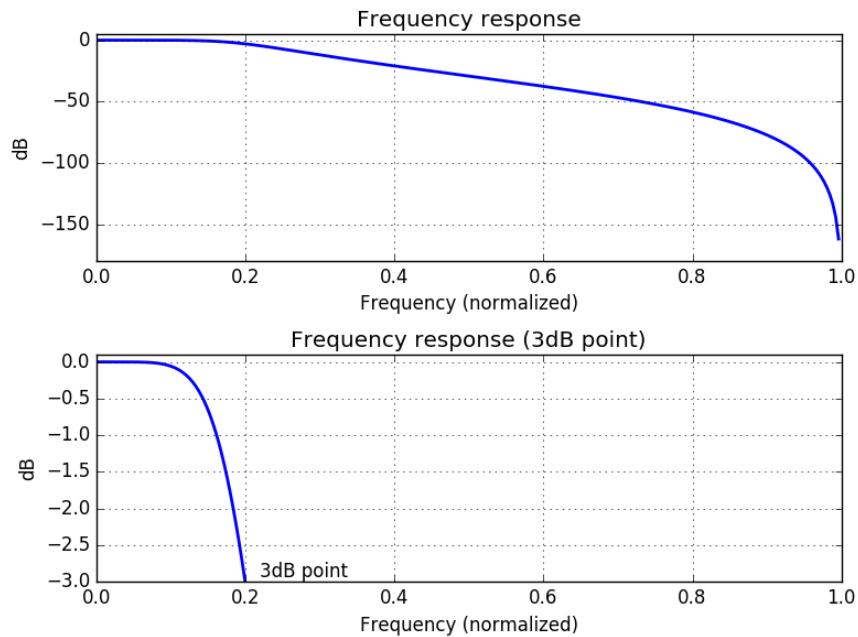
N = 256 # number of points for freqz
Wc = 0.2 # 3dB point
Order = 3 # filter order
# design a Butterworth filter
[b, a] = signal.butter(Order, Wc)
# calculate the frequency response
[w, h] = signal.freqz(b, a, N)
# plot the results
figure()
subplot(2, 1, 1)
plot(arange(N)/float(N), 20*log10(abs(h)), lw=2)
title('Frequency response')
xlabel('Frequency (normalized)')
ylabel('dB')
ylim(ylim()[0], ylim()[1]+5)
grid()

subplot(2, 1, 2)
plot(arange(N)/float(N), 20*log10(abs(h)), lw=2)
title('Frequency response (3dB point)')
xlabel('Frequency (normalized)')
ylabel('dB')
text(Wc+.02, -3, '3dB point', va='bottom')
ylim([-3, 0.1])
grid()

tight_layout()# subplot_tool()
show()

```

Було використано дві функції: *butter()* і *freqz()*. Функція *butter()* розробляє рекурсивний фільтр (IIR фільтр) із заданими параметрами (порядок і частота зрізу), а функція *freqz()* повертає частотну характеристику. Зверніть увагу, що частотна характеристика є комплексним числом, тому графік амплітуди було показано в дБ абсолютного значення  $20 \cdot \log_{10}(\text{abs}(H))$ , як показано нижче.



### ***Приклад: ковзаюче середнє***

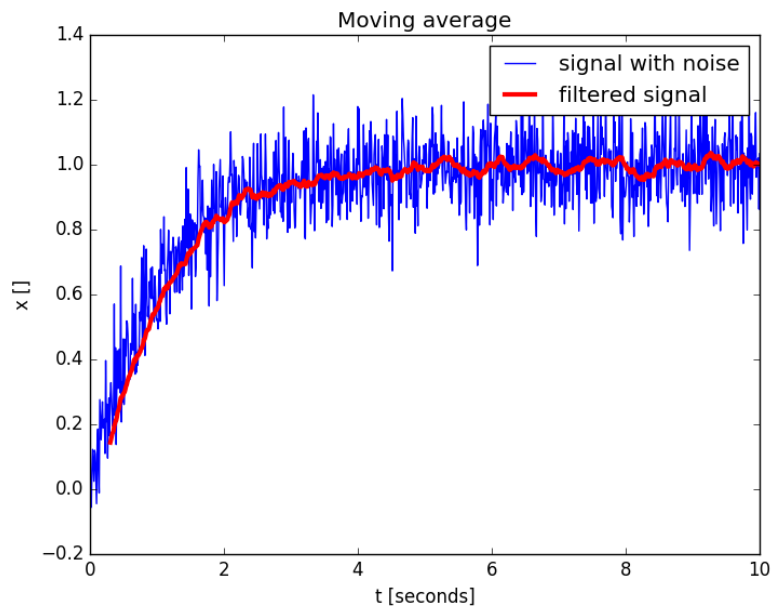
У багатьох випадках, фільтрація використовується для "згладжування" сигналу. Один із найпростіших способів для цього є використання алгоритму ковзаючого середнього. Для кожної із двох послідовних точок, ми обчислюємо середнє і використовуємо отримані значення замість них. Точки накладаються одна на одну, так що результат використання алгоритму для вектора [1, 2, 0, 2] буде [1,5, 1, 1]. Але навіть зупинятися лише на двох зразках? Ковзаюче середнє значення може бути виконане в декількох точках, повертаючи середнє значення цих точок. Нижче показано приклад програми, як цей алгоритм можна реалізувати з SciPy в Python.

```
from pylab import *
N = 1024
t = linspace(0, 10, N)
x = 1-exp(-t) + randn(N)/10
W = 32 # num points in moving average
xf = zeros(len(x)-W+1)
for i in range(len(x)-W+1):
    xf[i] = mean(x[i:i+W])
#xf = signal.lfilter(ones(W)/W, 1, x)
plot(t, x)
hold(True)
plot(t[W-1:], xf, lw=3, color = 'red')
title('Moving average')
legend(['signal with noise', 'filtered signal'])
xlabel('t [seconds]')
```



```
ylabel('x []')  
show()
```

Результат виконання коду програми:



### 3. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Особливості пакетів NumPy та SciPy.
2. Що таке інтерполяція та апроксимація?
3. Регресія. Для чого вона використовується?
4. Наближення функції.
5. Використання функцій *find()*, *nonzero()*, *where()*, *select()*?
6. Порівняння функцій *diff()* і *split()*.
7. Застосування фільтрів в обробці сигналів. Принцип роботи ФНЧ та ФВЧ.
8. Алгоритм роботи ковзаючого середнього.
9. Медіанний фільтр. Алгоритм роботи.

### 4. ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Ознайомитись з теоретичною частиною лабораторної роботи.
2. Одержати індивідуальне завдання (див. Варіанти індивідуальних завдань). При необхідності уточнити завдання на виконання у викладача.
3. Скласти програму на мові програмування Python відповідно до поставленого завдання. Виконана програма повинна повністю реалізувати розв'язок поставленої задачі.
4. Запустити програму та зберегти результат виконання.
5. Оформити звіт відповідно до встановлених вимог.

## **5. ЗМІСТ ЗВІТУ**

1. Мета роботи.
2. Відповіді на контрольні запитання.
4. Індивідуальне завдання, отримане у викладача, згідно з варіантом.
5. Код програми для реалізації завдання.
5. Аналіз отриманих результатів.
6. Ґрунтовні висновки.
7. Список використаної літератури.

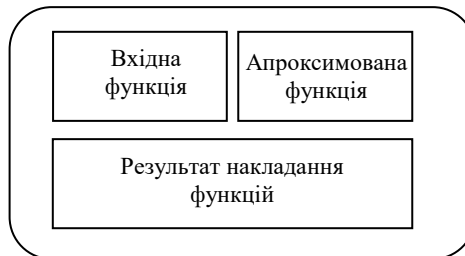
## **6. СПИСОК ЛІТЕРАТУРИ**

1. *Лутц М.*, Изучаем Python. - К.:Символ-Плюс , 2011. - 1280 с.
2. *Бизли Д.*, Python. Подробный справочник, 4-е издание. - К.:Символ-Плюс, 2012. - 864 с.
3. Офіційний сайт Python: <https://www.python.org/doc/>

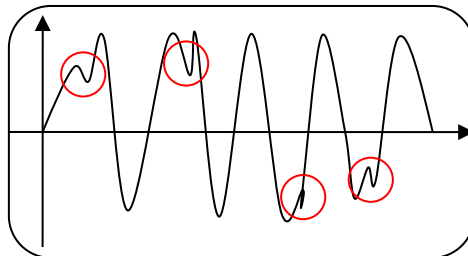
## ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №4

### Завдання №1

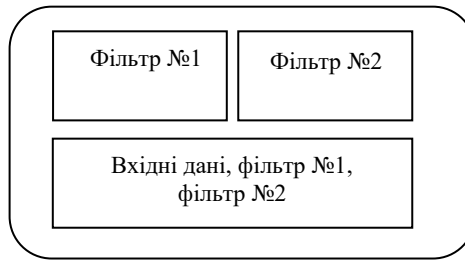
1. Відкрити та візуально відобразити дані з файлу для вашого варіанту. У заданому файлі є 4 колонки, де 1 – номер виміру, 2 – кут повороту з енкодера, 3 – кут повороту зразка, 4 – інтенсивність сигналу. Необхідно показати залежність кута повороту зразка від інтенсивності сигналу.
2. Застосувати фільтр низьких частот (для відсікання шумів в сигналі) та фільтр високих частот (для відсікання постійної складової сигналу). Параметри для ФНЧ та ФВЧ вибрати самостійно, шляхом аналізу вхідних даних. Результат фільтрування вивести в графічному вікні.
3. Апроксимувати задану функцію з використанням засобів python. В одному графічному вікні вивести: вхідні дані, отриманий результат апроксимації, та результат накладання двох графіків.



4. Знайти кількість нулів функції (точки перетину з віссю абсцис). Відобразити їх візуально. У разі, якщо інтенсивність сигналу приймає тільки додатні або від'ємні значення, необхідно відняти постійну складову від сигналу аби отримати додатні та від'ємні значення, а відповідно і точки перетину їх з віссю абсцис.
5. Знайти та візуально показати усі мінімальні та максимальні значення функції, для даних, які знаходяться по обидві сторони від осі абсцис. Підрахувати та вивести їх значення у вигляді таблиці в окремому файлі.
6. Врахувати точки перегину функції, які помилково можуть вважатися як максимуми та мінімуми функції (червоні кружечки на схематичному рисунку). Підрахувати та вивести їх значення у вигляді таблиці в окремому файлі. Порівняти значення із отриманими в п.5. Проаналізувати результат.



7. Застосувати 2 типи різних фільтрів (вибрати самостійно) для фільтрації вхідних даних для їх графічного представлення. Результат фільтрування показати графічно.



8. Порівняти кількість отриманих максимумів функції без фільтрування та з фільтруванням вхідної функції.

*Варіанти індивідуальних завдань до завдання №1:*

1. Variant\_1.txt
2. Variant\_2.txt
3. Variant\_3.txt
4. Variant\_4.txt
5. Variant\_5.txt
6. Variant\_6.txt
7. Variant\_7.txt
8. Variant\_8.txt
9. Variant\_9.txt
10. Variant\_10.txt
11. Variant\_11.txt
12. Variant\_12.txt
13. Variant\_13.txt
14. Variant\_14.txt
15. Variant\_15.txt

# ЛАБОРАТОРНА РОБОТА №5

на тему «Методи класифікації: навчання з вчителем (лінійна регресія, k-найближчих сусідів, метод опорних векторів)»

## 1. МЕТА РОБОТИ

Ознайомитися із можливостями мови програмування Python для вирішення задач класифікації та реалізації стандартних алгоритмів на основі модулів наукової обробки даних.

## 2. ТЕОРЕТИЧНА ЧАСТИНА

### 2.1. Вступ до задач класифікації

**Задача класифікації** – формалізована задача, яка містить множину об'єктів (ситуацій), поділених певним чином на класи. Задана кінцева множина об'єктів, для яких відомо, до яких класів вони відносяться. Ця множина називається *вибіркою*. До якого класу належать інші об'єкти невідомо. Необхідно побудувати такий алгоритм, який буде здатний класифікувати довільний об'єкт з вхідної множини.

Для вирішення таких задач класифікації існує багато алгоритмів. В даній лабораторній роботі розглянемо алгоритми навчання з вчителем. Крім того, для практичного представлення результатів виконання алгоритмів в лабораторній роботі використано модулі `numpy`, `scipy`, `pandas` та `sklearn`.

Для установки даних модулів в Python на ОС Windows існують вже готові інтерпретатори WinPython, Miniconda або Anaconda, які мають вже преінстальовані необхідні пакети модулів, а також наявна можливість встановлення необхідних модулів через командний рядок за допомогою `pip`.

### 2.2. Зчитування та вивід даних

*Використання модулів в Python на прикладі Pandas*

Імпорт даних модулів для запуску як правило виглядає наступним чином (за загальноприйнятими аббревіатурами)

```
import scipy as sp
import numpy as np
import pandas
import csv
```

Модуль *pandas* є потужним інструментом для аналізу даних. Пакет дає можливість будувати звідні таблиці, виконувати групування, рисувати графіки на отриманих наборах даних. Pandas дає можливість широкого вибору джерел завантаження файлів, наприклад SQL, текстові файли, Excel файли, HTML.

Для прикладу завантажимо текстовий файл з масивами факторів, які будуть застосовані для подальшого пояснення функцій.

```
from pandas import read_csv
data1 = read_csv("Data1.txt", delimiter='\t', header = 0)
data2 = read_csv("Data2.txt", delimiter=',', header = 0)
print(data1)
```

```
print(data2)
```

В результаті виконання програми отримали такі набори даних з файлів

Для *Data1.txt*

Number	Value	Result
1	3	5
2	6	3
3	2	9
4	1	12

Для *Data2.txt*

Parameter	Function
8	34
7	12
5	3
5	7

Зверніть увагу, що вхідні дані, які записані у файлі, були зчитані за допомогою функції `read_csv()`, де параметри `delimiter` визначає спосіб розділення даних (кома, дві крапки, табуляція, т.д), а `header` визначає чи враховувати перший рядок даних.

Крім того, існує ще декілька цікавих способів для того аби відкрити та відобразити інформацію з файлу. Для прикладу, коли потрібно вивести тільки одну колонку з файлу, це можна зробити за допомогою такого коду:

```
with open('Data1.txt') as inf:
    reader = csv.reader(inf, delimiter="\t")
    second_col = list(zip(*reader))[2]
    print(second_col)
```

або

```
FileOpen = open("Data1.txt", "r")
for columns in (raw.strip().split() for raw in FileOpen):
    print(columns[1])
```

Для того аби додати нову колонку використовуєть функція `insert()`

```
DataInsert=[2,4,6,10]
data1.insert(1,'New Data', DataInsert)
print(data1)
```

У функції `insert()` перший параметр –позиції, куди будемо вставляти колонку (починаючи з 0), друга відповідає за ім'я нової колонки, третя відповідає масиву значень (типу `list`). Результат виконання функції:

Number	New Data	Value	Result
1	2	3	5
2	4	6	3
3	6	2	9
4	10	1	12

### 2.3. Лінійна регресія. Типовий алгоритм та його реалізація.

**Лінійна регресія** – це метод моделювання залежності між скаляром  $y$  та векторною (у загальному випадку) змінною  $X$ . У випадку, якщо змінна  $X$  також є скаляром, регресію називають простою.

При використанні лінійної регресії взаємозв'язок між даними моделюється за допомогою лінійних функцій, а невідомі параметри моделі оцінюються за вхідними даними. Подібно до інших методів регресійного аналізу лінійна регресія повертає розподіл умовної імовірності  $y$  в залежності від  $X$ , а не розподіл спільної імовірності  $y$  та  $X$ , що стосується області мультиваріативного аналізу.

При розрахунках параметрів моделі лінійної регресії як правило застосовується метод найменших квадратів, але також можуть бути використані інші методи. Так само метод найменших квадратів може бути використаний і для нелінійних моделей. Тому МНК та лінійна регресія хоч і є тісно пов'язаними, але не є синонімами.

Розглянемо застосування лінійної регресії на прикладі задачі.

**Задача 1.** Наявні дані виробітку на 1 робітника  $Y$  (тис. грн) і товарообігу  $X$  (тис. грн) в 12 магазинах за квартал. На основі вказаних даних необхідно знайти:

- коефіцієнт кореляції середнього виробітку 1 робітника від товарообороту;
- скласти рівняння регресії цієї залежності.

Магазини												
	1	2	3	4	5	6	7	8	9	10	11	12
$x$	10	14	21	23	27	32	39	45	55	61	62	68
$y$	3.8	4.8	5.9	6.1	6.2	6.3	6.6	7.4	8.5	9.7	10.5	12.4

Розв'яжемо дану задачу методом найменших квадратів (лінійна регресія). Зв'язок між параметрами можна виразити функцією  $Y=aX+b$ . Для визначення параметрів  $a$ ,  $b$  згідно із методом найменших квадратів складемо розрахункову таблицю та знайдемо відповідний масив  $x^2$ ,  $y^2$ ,  $x*y$  та суму елементів.

```
from pylab import *
import numpy as np
import math
#ініціалізуємо два масиви з вхідними даними
dataX=[10,14,21,23,27,32,39,45,55,61,62,68]
dataY=[3.8,4.8,5.9,6.1,6.2,6.3,6.6,7.4,8.5,9.7,10.5,12.4]
dataX2=[]
dataY2=[]
dataXY=[]
for i in range(len(dataX)):
    XX=dataX[i]* dataX[i]
    YY=dataY[i]* dataY[i]
```

```

XY=dataX[i]* dataY[i]
dataX2.append(XX)
dataY2.append(YY)
dataXY.append(XY)
#обчислюємо необхідні значення сум
sumX=sum(dataX)
sumY=sum(dataY)
sumX2=sum(dataX2)
sumY2=sum(dataY2)
sumXY=sum(dataXY)
dataX.append(sumX)
dataY.append(sumY)
dataX2.append(sumX2)
dataX2.append(sumY2)
dataXY.append(sumXY)

```

В результаті виконання отримаємо таблицю значень для решти параметрів:

Магазини													
	1	2	3	4	5	6	7	8	9	10	11	12	Сума
X	10	14	21	23	27	32	39	45	55	61	62	68	<b>457</b>
Y	3.8	4.8	5.9	6.1	6.2	6.3	6.6	7.4	8.5	9.7	10.5	12.4	<b>88,2</b>
X <sup>2</sup>	100	196	441	529	729	1024	1521	2025	3025	3721	3844	4624	<b>21779</b>
Y <sup>2</sup>	14,44	23,04	34,81	37,21	38,44	39,69	43,56	54,76	72,25	94,09	110,3	153,8	<b>716,3</b>
XY	38	67,2	123,9	140,3	167,4	201,6	257,4	333	467,5	591,7	651	843,2	<b>3882,3</b>

```

#обчислюємо коефіцієнти a і b лінійної регресії
a=np.array([[sumX2,sumX],[sumX,len(dataX)-1]])
print("a=", a)
b=np.array([sumXY,sumY])
print("b=", b)
#Реалізуємо рішення даної системи рівнянь за допомогою модуля numpy
Y=np.linalg.solve(a,b)
print("Y=", Y)
print("Рівняння регресійної залежності = ", Y[0],"*x +", Y[1])

```

Отримаємо масив Y з параметрами a і b. Отримане рівняння  $a \cdot x + b$  і буде регресійною залежністю. Побудуємо графік за отриманими результатами та порівняємо їх з теоретичними.

```

dataRegression=[]
for i in range(len(dataX)-1):
    Regression=dataX[i]*Y[0]+Y[1]
    dataRegression.append(Regression)
dataX.pop()

```

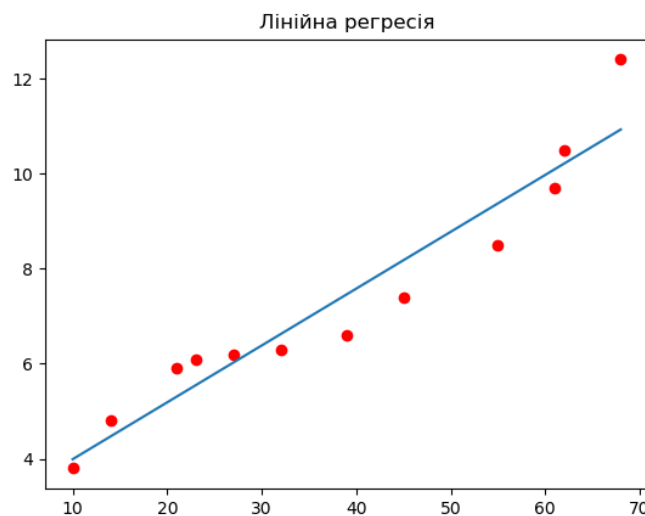


```

dataY.pop()
print("len(dataX)=", len(dataX))
print("len(dataY)=", len(dataY))
#рисуємо графік лінійної залежності
figure()
title(u'Лінійна регресія')
plot(dataX,dataY, 'ro')
plot(dataX,dataRegression)
show()

```

В результаті виконання ми отримаємо та графічну залежність, де буде показано наші взідні дані та графік лінійної залежності.



Отриманий графік наочно показує теоретичні та очікувані результати.

```

#обчислюємо коефіцієнт кореляції
corr=(len(dataX)*sumXY-sumX*sumY)/(math.sqrt(len(dataX)*sumX2-
sumX*sumX)*math.sqrt(len(dataX)*sumY2-sumY*sumY))
print("len(dataX)*sumXY-sumX*sumY = ", len(dataX)*sumXY-sumX*sumY)
print("corr = ", corr)

```

*Лінійний коефіцієнт кореляції* – кількісна оцінка і міра тісноти зв'язку двох змінних. Коефіцієнт кореляції приймає значення в інтервалі від -1 до +1. Вважають, що якщо цей коефіцієнт не більше 0,3, то зв'язок слабкий: від 0,3 до 0,7 – середній; більше 0,7 – міцний, або тісний. Коли коефіцієнт дорівнює 1, то зв'язок функціональний, якщо він дорівнює 0, то говорять про відсутність лінійного зв'язку між ознаками.

$$r_{xy} = \frac{\sum xy - \frac{\sum x \sum y}{n}}{\sqrt{\left(\sum x^2 - \frac{(\sum x)^2}{n}\right) \left(\sum y^2 - \frac{(\sum y)^2}{n}\right)}}$$

Отримуємо близько 0.959. Отже зв'язок сильний, пряма залежність.

## 2.4 Алгоритм k-найближчих сусідів (kNN)

Метод k-найближчих сусідів (англ. k-nearest neighbor method) — простий непараметричний класифікаційний метод, де для класифікації об'єктів у рамках простору властивостей використовуються відстані (звичайно евклідові), пораховані до усіх інших об'єктів. Вибираються об'єкти, до яких відстань найменша, і вони виділяються в окремий клас.

Метод k-найближчих сусідів – метричний алгоритм для автоматичної класифікації об'єктів. Основним принципом методу найближчих сусідів є те, що об'єкт присвоюється того класу, який є найбільш поширеним серед сусідів даного елемента. Сусіди беруться, виходячи з множини об'єктів, класи яких уже відомі, і, виходячи з ключового для даного методу значення k, вираховується, який клас є найчисленнішим серед них. Кожен об'єкт має кінцеву кількість атрибутів (розмірностей). Передбачається, що існує певний набір об'єктів з уже наявною класифікацією.

Покажемо приклад реалізації алгоритму kNN із створення програми Python на прикладі класифікації фруктів. Логіка програми буде побудована таким чином, аби повністю відповідати необхідним крокам для реалізації алгоритму, а саме:

Для кожної точки в нашому наборі даних:

1. обчислити відстань між  $X$  (шукана точка) та поточною точкою
2. відсортувати відстані в порядку зростання
3. вибрати  $k$  варіантів з найнижчими відстанями до  $X$ .
4. знайти клас більшості серед цих предметів
5. вибрати клас більшості в якості нашого передбачення для класу  $X$ .

Реалізацію алгоритму покажемо на прикладі класифікації фруктів за вибраними параметрами: вага та колір. Отже, розпочнемо написання алгоритму із визначення відстані між поточними точками із введених даних, до деякої шуканої.

```
def distance(fruit1, fruit2):
    Xa = fruit1[0] - fruit2[0]
    Xb = fruit1[1] - fruit2[1]
    dist = (Xa**2 + Xb**2)**0.5
    return dist
```

Ви обчислюєте відстані між об'єктами вхідної вибірки за допомогою Евклідової відстані, де відстань між двома векторами  $X_a$  і  $X_b$  в математичному записі задається так:

$$d = \sqrt{(x_{A_0} - x_{B_0})^2 + (x_{A_1} - x_{B_1})^2}$$

Для прикладу, відстань між точками (0,0) та (1,2) обчислюється як

$$\sqrt{(1 - 0)^2 + (2 - 0)^2}$$

Якщо нам потрібно знайти відстань для випадку, коли є 4 атрибути в нашій вхідній вибірці, то відстань між точками (1,0,0,1) та (7,6,9,4) можна буде розрахувати так

$$\sqrt{(7 - 1)^2 + (6 - 0)^2 + (9 - 0)^2 + (4 - 1)^2}$$

За умовою задачі, у нас є такий набір даних:

```
dataset = [  
    # weight, color, type  
    [303, 3, "banana"],  
    [370, 1, "apple"],  
    [298, 3, "banana"],  
    [277, 3, "banana"],  
    [377, 4, "apple"],  
    [299, 3, "banana"],  
    [382, 1, "apple"],  
    [374, 4, "apple"],  
    [303, 4, "banana"],  
    [309, 3, "banana"],  
    [359, 1, "apple"],  
    [366, 1, "apple"],  
    [311, 3, "banana"],  
    [302, 3, "banana"],  
    [373, 4, "apple"],  
    [305, 3, "banana"],  
    [371, 3, "apple"],  
]
```

Необхідно визначити, який тип невідомого фрукту `unknown_fruit = [373, 1]`

Знайдемо відстані від шуканої точки, до заданих даних:

```
for i in range(0,17):  
    print(dataset[i], "Distance = ", distance(dataset[i], unknown_fruit))
```

Далі ці відстані необхідно відсортувати в порядку зростання.

```
sorted_dataset = sorted(dataset, key=lambda fruit: distance(fruit, unknown_fruit))  
for j in range(0,17):  
    print(sorted_dataset[j]),
```

Вибираємо  $k = 3$ , для випадку, коли хочемо ідентифікувати наш клас базуючись на перших 3 елементах з вхідного набору даних, які мають найменші відстані до шуканого фрукту для класифікації.

```
from collections import Counter  
#бере тільки перші k даних  
top_k = sorted_dataset[0:k]
```

```

class_counts= Counter(fruit for (weight, color, fruit) in top_k)

# вибір класу з найбільшою кількістю голосів за нього
classification= max(class_counts, key=lambda cls: class_counts[cls])

# вивід результату
print("Our class = ", classification)

```

В результаті виконання ми отримаємо відповідь, чи наш новий фрукт є банан чи яблуко, в залежності від його вхідних параметрів: ваги та кольору.

## 2.5. Метод опорних векторів (SVM)

Метод опорних векторів, відомий в англійській літературі як support vector machine (SVM), є машинним алгоритмом, котрий навчається на прикладах та використовується для класифікації об'єктів. Наприклад, SVM може розрізнити аварійний режим роботи електромеханічної системи та класифікувати його за наявності попередніх досліджень, можливих за технологічними вимогами режимів роботи. Такий підхід розкриває значні можливості для побудування адаптивних систем автоматичного керування.

Для прикладу, розглянемо набір даних, що зображені на рис.1. Коли поставити завдання провести пряму (площину), яка би відокремила одні дані від інших (класифікувати), то таких прямих можна провести дуже багато, і кожна з яких буде правильним класифікатором, адже завдання було виконано (рис. 1б). Проте, якщо поставити завдання, поділити дані у найбільш оптимальний спосіб, то ми би отримали лінію на рис. 1в.

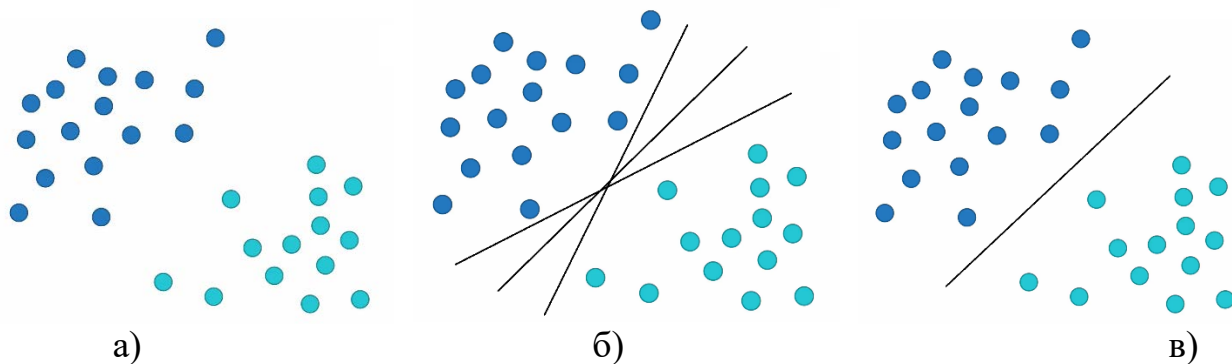


Рис. 1. Можливі способи поділу вхідних даних

Тобто, лінія яка зображена на рис. 1в і є вихідним рішенням задачі класифікації для заданого набору даних.

З рис. 2 можна бачити, що для наших ліній, найширша лінія є для випадку в, тобто виконується підхід «widest road approach».

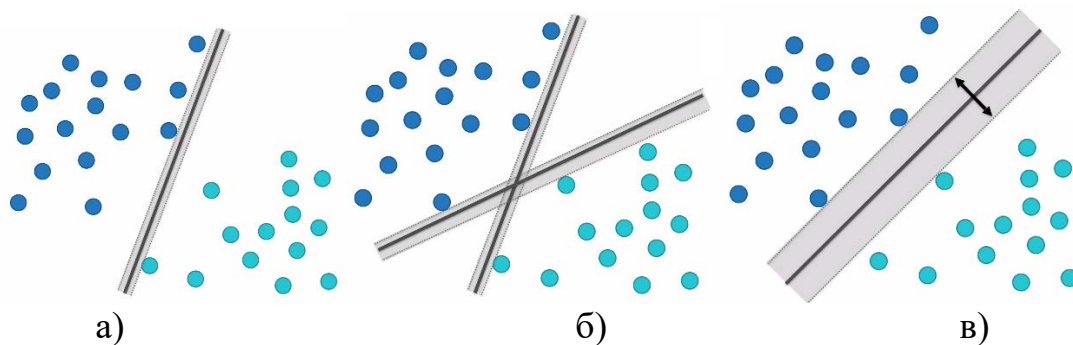


Рис. 2. Приклад можливої ширини класифікатора

Давайте подивимося на цю ширину вулиці під іншим кутом. Будемо розглядати варіант, коли це не найширша вулиця, а найширші межі (margins). Тобто відстань між точками (даними) і цією лінією є найбільшою (рис. 3а). Оранжевим коліром показані опорні вектори, а лінія, яка найкраще ділить дані називається гіперплощиною (hyperplane).

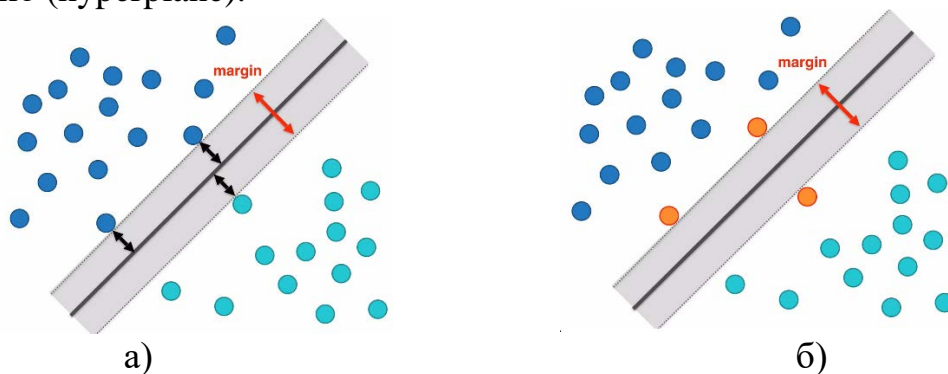


Рис. 3. Межі і гіперплощина

Наше завдання, знайти спосіб визначити найширші межі, тобто ширину вулиці.

### Приклад застосування SVM для класифікації на Python

*Завдання:* класифікувати рецепти приготування капкейків і мафінів. Коли отримуємо новий рецепт, то визначити, чи це рецепт для приготування капкейку чи мафіна.

*Необхідні кроки*

1. Знайти дані
2. Застосувати модель класифікації
3. Проаналізувати результати

Для отримання даних, в інтернеті було знайдено 20 рецептів, по 10 на кожен із капкейки і мафіни, знайдено спільні параметри та приведено в одну таблицю, де показано лише найважливіші параметри двох рецептів.

Type	Flour	Milk	Sugar	Butter	Egg	Baking Powder	Vanilla	Salt
Muffin	55	28	3	7	5	2	0	0
Muffin	47	24	12	6	9	1	0	0
Muffin	47	23	18	6	4	1	0	0
Muffin	45	11	17	17	8	1	0	0
Muffin	50	25	12	6	5	2	1	0
Muffin	55	27	3	7	5	2	1	0
Muffin	54	27	7	5	5	2	0	0
Muffin	47	26	10	10	4	1	0	0
Muffin	50	17	17	8	6	1	0	0
Muffin	50	17	17	11	4	1	0	0
Cupcake	39	0	26	19	14	1	1	0
Cupcake	42	21	16	10	8	3	0	0
Cupcake	34	17	20	20	5	2	1	0
Cupcake	39	13	17	19	10	1	1	0
Cupcake	38	15	23	15	8	0	1	0
Cupcake	42	18	25	9	5	1	0	0
Cupcake	36	14	21	14	11	2	1	0
Cupcake	38	15	31	8	6	1	1	0
Cupcake	36	16	24	12	9	1	1	0
Cupcake	34	17	23	11	13	0	1	0

Як видно із таблиці, у нас є лише 2 типи даних, по відношенню до яких потрібно провести класифікацію рецептів, ну і відповідно різні складники рецепту.

Наступний крок це застосування класифікатора SVM. Спершу нам необхідно імпортувати всі бібліотеки, які будемо використовувати в нашому прикладі.

```
# ***** Step 1: Імпортуємо бібліотеки *****
```

```
# Пакети для аналізу
```

```
import pandas as pd
```

```
import numpy as np
```

```
from sklearn import svm
```

```
# Пакети для візуалізації
```

```
from matplotlib import pyplot as plt
```

```
import seaborn as sns; sns.set(font_scale=1.2)
```

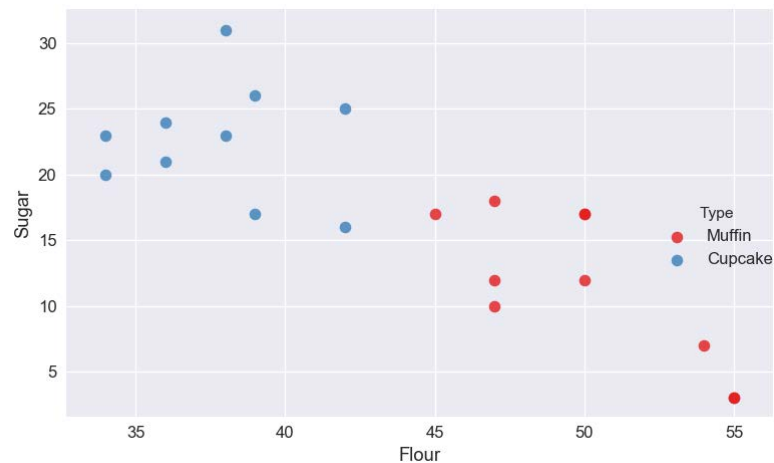
```
# Pickle package
```

```
import pickle
```

Наступний крок імпортувати наші дані та відобразити їх. Оскільки найбільший вплив на рецепт мають сама цукор та борошно.

```
# ***** Step 2: Імпортуємо дані *****
# Зчитуємо інгредієнти в мафінів та капкейків
recipes = pd.read_csv('muffin_data.csv', delimiter=';')

# Відображаємо 2 інгредієнти
sns.lmplot('Flour', 'Sugar', data=recipes, hue='Type', palette='Set1', fit_reg = False,
scatter_kws={"s": 70});
plt.show()
```



```
# ***** Step 3: Підготуємо дані *****
# Specify inputs for the model
# ingredients = recipes[['Flour', 'Milk', 'Sugar', 'Butter', 'Egg', 'Baking Powder',
'Vanilla', 'Salt']].as_matrix()
ingredients = recipes[['Flour', 'Sugar']].as_matrix()
type_label = np.where(recipes['Type']=='Muffin', 0, 1)

# Feature names
recipe_features = recipes.columns.values[1:].tolist()
print(recipe_features)

# ***** Step 4: Fit the Model *****
# Fit the SVM model
model = svm.SVC(kernel='linear')
print(model.fit(ingredients, type_label))
```

В результаті отримаємо:

```
['Flour', 'Milk', 'Sugar', 'Butter', 'Egg', 'Baking Powder', 'Vanilla', 'Salt']
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
max_iter=-1, probability=False, random_state=None, shrinking=True,
tol=0.001, verbose=False)
```



Отриманий результат не дає бажаного результату для розуміння, тому давайте представимо цей результат на графі. Для цього, нам необхідно створити гіперплощину для наших даних, а також показано межі нашого класифікатора.

*# \*\*\*\*\* Step 5: Візуалізуємо результат \*\*\*\*\**

*# Get the separating hyperplane*

```
w = model.coef_[0]
```

```
a = -w[0] / w[1]
```

```
xx = np.linspace(30, 60)
```

```
yy = a * xx - (model.intercept_[0]) / w[1]
```

*# Plot the parallels to the separating hyperplane that pass through the support vectors*

```
b = model.support_vectors_[0]
```

```
yy_down = a * xx + (b[1] - a * b[0])
```

```
b = model.support_vectors_[-1]
```

```
yy_up = a * xx + (b[1] - a * b[0])
```

*# Plot the hyperplane*

```
sns.lmplot('Flour', 'Sugar', data=recipes, hue='Type', palette='Set1', fit_reg=False,
scatter_kws={"s": 70})
```

```
plt.plot(xx, yy, linewidth=2, color='black');
```

```
plt.show()
```

*# Look at the margins and support vectors*

```
sns.lmplot('Flour', 'Sugar', data=recipes, hue='Type', palette='Set1', fit_reg=False,
scatter_kws={"s": 70})
```

```
plt.plot(xx, yy, linewidth=2, color='black')
```

```
plt.plot(xx, yy_down, 'k--')
```

```
plt.plot(xx, yy_up, 'k--')
```

```
plt.scatter(model.support_vectors_[0], model.support_vectors_[1],
```

```
s=80, facecolors='none');
```

```
plt.show()
```

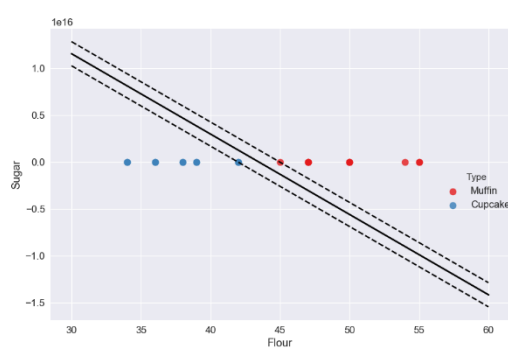
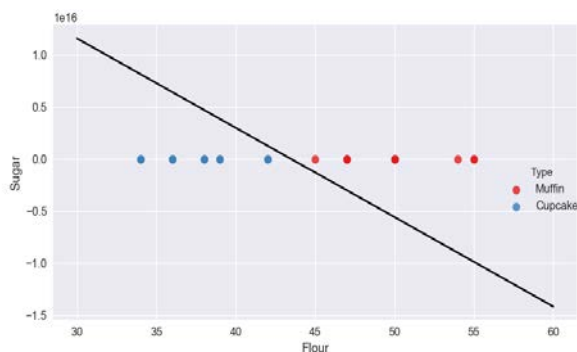


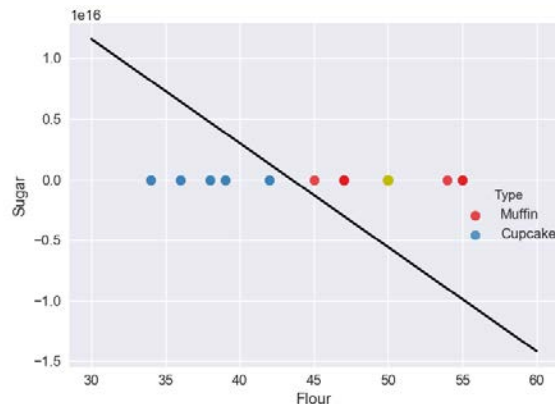
Рис. 4. Результат застосування алгоритму SVM



Як видно, то зліва у нас вийшов класифікатор SVM, а справа показано межі.

Далі, давайте протестуємо роботу нашого класифікатора. Для цього введено додаткову точку в наші дані.

```
# ***** Step 6: Передбачити новий випадок *****  
# Plot the point to visually see where the point lies  
sns.lmplot('Flour', 'Sugar', data=recipes, hue='Type', palette='Set1', fit_reg=False,  
scatter_kws={"s": 70})  
plt.plot(xx, yy, linewidth=2, color='black')  
plt.plot(50, 20, 'yo', markersize='9');  
plt.show()
```



З рисунку легко побачити, до якого типу належить новий рецепт, проте нам необхідно аби наш класифікатор мав можливість самостійно визначити це. Для цього створимо функцію, яка буде нам давати відповідь, до якого класу належить наш новий рецепт – мафін чи капкейк.

```
# Create a function to guess when a recipe is a muffin or a cupcake
```

```
def muffin_or_cupcake(flour, sugar):  
    if(model.predict([[flour, sugar]])==0):  
        print('You\'re looking at a muffin recipe!')  
    else:  
        print('You\'re looking at a cupcake recipe!')
```

Як видно, функція має лише 2 параметри – борошно та цукор, а результат буде, який це рецепт. Якщо протестувати роботу програми, то результат класифікації при введених значеннях борошна та цукру, отримаємо, що ми маємо рецепт випікання мафіна.

```
# Predict if 50 parts flour and 20 parts sugar
```

```
muffin_or_cupcake(50, 20)
```

Результат виконання

You're looking at a muffin recipe!

### **3. КОНТРОЛЬНІ ЗАПИТАННЯ**

1. Для чого використовуються модулі pandas, skapy, sklearn ?
2. Якими найзручнішими способами можна ввести стартові блоки даних для обробки для подальшої класифікації ?
3. У чому полягає задача регресії?
4. Завдання та принцип дії лінійної регресії.
5. Алгоритм роботи k-найближчих сусідів.
6. Алгоритм роботи методу опорних векторів.
7. Алгоритм роботи методу лінійної регресія.

### **4. ЛАБОРАТОРНЕ ЗАВДАННЯ**

1. Ознайомитись з теоретичною частиною лабораторної роботи.
2. Одержати індивідуальне завдання (див. Варіанти індивідуальних завдань). При необхідності уточнити завдання на виконання у викладача.
3. Скласти програму на мові програмування Python відповідно до поставленого завдання. Виконана програма повинна повністю реалізувати розв'язок поставленої задачі.
4. Запустити програму та зберегти результат виконання.
5. Оформити звіт відповідно до встановлених вимог.

### **5. ЗМІСТ ЗВІТУ**

1. Мета роботи.
2. Відповіді на контрольні запитання.
3. Індивідуальне завдання, отримане у викладача, згідно з варіантом.
4. Код програми для реалізації завдання.
5. Аналіз отриманих результатів.
6. Ґрунтовні висновки.
7. Список використаної літератури.

### **6. СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ**

1. *Лутц М.*, Изучаем Python. - К.:Символ-Плюс , 2011. - 1280 с.
2. *Бизли Д.*, Python. Подробный справочник, 4-е издание. - К.:Символ-Плюс, 2012. - 864 с.
3. Офіційний сайт Python: <https://www.python.org/doc/>
4. Офіційний сайт машинного навчання: <http://scikit-learn.org>

## ВАРІАНТИ ІНДИВІДУАЛЬНИХ ЗАВДАНЬ

### до лабораторної роботи №5

#### Задача 1.

Вам, як генеральному директору компанії, доручили відкрити магазин для продажу електроніки в одну із міст України. У вас вже є побудована мережа магазинів, а отже вам відома статистика продажів та кількість населення в містах (Табл. 1.1). Використовуючи запропоновану статистику даних та з використанням методу регресії для оцінки параметрів функції, яка може передбачити прибуток компанії, розробити:

- а) модель лінійної регресії для оцінки параметрів;
- б) оцінити ефективність відкривання магазинів в запропонованих містах для вашого варіанту (див. Табл. 1.2);
- с) запропонувати 3 міста (на свій вибір) для відкриття магазинів з можливістю найбільшого прибутку;
- д) запропонувати 3 міста (на свій вибір) для відкриття магазинів з найшвидшим терміном повернення інвестиції.

**Таблиця 1.1. Статистика прибутку компанії**

	Прибуток компанії, тис. грн за рік	Населення, тис. чоловік	Рік відкриття	Початкові інвестиції, тис. грн	К-сть прац.	Область
<b>M1</b>	260	729,1	2009	750	5	Львівська
<b>M2</b>	140	227,1	2011	540	4	Івано-Франківська
<b>M3</b>	190	217,1	2012	580	5	Тернопільська
<b>M4</b>	240	216,1	2013	430	4	Волинська
<b>M5</b>	150	115,9	2013	350	3	Закарпатська
<b>M6</b>	140	85,5	2014	320	3	Закарпатська
<b>M7</b>	170	76,9	2011	470	2	Львівська
<b>M8</b>	136	67,9	2013	420	2	Львівська
<b>M9</b>	90	69,1	2013	390	3	Волинська
<b>M10</b>	145	67,6	2013	500	2	Івано-Франківська
<b>M11</b>	118	61,4	2014	310	2	Івано-Франківська
<b>M12</b>	107	59,9	2014	420	3	Львівська
<b>M13</b>	92	53,3	2013	340	2	Волинська
<b>M14</b>	89	38,9	2014	390	2	Волинська
<b>M15</b>	46	29,4	2015	320	2	Львівська
<b>M16</b>	67	28,6	2014	250	2	Тернопільська

**Таблиця 1.2. Варіанти індивідуальних завдань**

	Населення, тис. чоловік	Початкові інвестиції, тис. грн	Кількість працівників	Область
<b>B1</b>	220	250	1-5	Львівська
<b>B2</b>	210	250	1-5	Івано-Франківська
<b>B3</b>	120	250	1-5	Тернопільська
<b>B4</b>	85	250	1-5	Волинська
<b>B5</b>	34	250	1-5	Закарпатська
<b>B6</b>	67	250	1-5	Закарпатська
<b>B7</b>	54	250	1-5	Львівська
<b>B8</b>	90	250	1-5	Львівська
<b>B9</b>	110	250	1-5	Волинська
<b>B10</b>	75	250	1-5	Івано-Франківська
<b>B11</b>	91	250	1-5	Івано-Франківська
<b>B12</b>	84	250	1-5	Львівська
<b>B13</b>	54	250	1-5	Волинська
<b>B14</b>	35	250	1-5	Волинська
<b>B15</b>	37	250	1-5	Львівська

### Задача 2.

На підприємстві працює система сортування виготовленої продукції. До основних параметрів згідно яких відбувається класифікації зразків належать: діаметр, колір, вага, товщина та матеріал зразка (DataTypes.xlsx).

Визначити до якого типу належить заданий згідно вашого варіанту зразок (Табл. 2.1), використовуючи класифікацію за допомогою методу:

- а) k-найближчих сусідів;
- б) опорних векторів.

**Таблиця 2.1. Варіанти завдань**

	Вага, г	Діаметр, см	Колір зразка	Товщина зразка, см	Матеріал зразка
B1	250	4	білий	3	дерево
B2	120	6	жовтий	2	дерево
B3	230	3	зелений	2	метал
B4	150	5	червоний	4	пластик
B5	94	2	червоний	2	пластик
B6	145	3	рожевий	3	дерево
B7	172	5	жовтий	1	метал
B8	290	5	блакитний	2	метал
B9	131	3	червоний	3	дерево
B10	49	2	зелений	2	дерево
B11	83	4	блакитний	3	дерево
B12	112	3	зелений	5	пластик
B13	154	2	рожевий	3	метал
B14	350	4	жовтий	4	метал
B15	102	7	червоний	1	пластик

НАВЧАЛЬНЕ ВИДАННЯ

*Андрущак Назарій Анатолійович*

## **МЕТОДИ ТА ЗАСОБИ КОМП'ЮТЕРНОГО НАВЧАННЯ**

### **ЛАБОРАТОРНИЙ ПРАКТИКУМ**

для студентів другого (магістерського) рівня вищої освіти  
спеціальності 122 «Комп'ютерні науки»  
спеціалізація «Системне проектування»

**Редактор**

Андрущак Н.А.

**Комп'ютерне верстання**

Андрущак Н.А.