МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних завдань

з навчальної дисципліни «Програмування на Python»

для студентів денної та заочної форм навчання

за спеціальностями «122 Комп’ютерні науки», та

«113 Прикладна математика»

Затверджено
редакційно-видавничою
радою університету,
протокол № 3 від 24.10.2024 р.

Харків
НТУ «ХПІ»
2024

Методичні вказівки до виконання лабораторних завдань з навчальної дисципліни «Програмування на Python» для студентів денної та заочної форм навчання за спеціальностями «122 Комп'ютерні науки», та «113 Прикладна математика»/ уклад.: М. І. Шаповалова. – Харків: НТУ «ХПІ», 2024. – 72 с.

Укладач: М. І. Шаповалова

Рецензент: Л. В. Розова

Кафедра математичного моделювання на інтелектуальних обчислень в інженерії.

ЗМІСТ

1. ВСТУП……4
2. Тема 1. Вступ до програмування мовою Python……..5
3. Завдання до лабораторної роботи №1……10
4. Тема 2. Колекції у Python. Цикли. Вийнятки……..12
5. Завдання до лабораторної роботи №2……20
6. Тема 3. Функції…..……21
7. Завдання до лабораторної роботи №3……25
8. Тема 4. Робота з файлами……27
9. Завдання до лабораторної роботи №4……30
10. Тема 5. Модуль Numpy……32
11. Завдання до лабораторної роботи №5……37
12. Тема 6. Модуль Matplotlib……38
13. Завдання до лабораторної роботи №6……45
14. Тема 7. Об`єктно орієнтоване програмування у Python……48
15. Завдання до лабораторної роботи №7……53
16. Тема 8. GUI. PyQt……..56
17. Завдання до лабораторної роботи №8……69
18. СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ ……71

ВСТУП

Методичні вказівки до дисципліни «Програмування на Python» розроблені для комплексного супроводу навчального процесу та допомоги студентам у систематичному вивченні основ програмування на мові Python. Вони включають як теоретичні матеріали, так і детальні рекомендації для виконання практичних завдань, що дозволяє поступово та структуровано опанувати навички, необхідні для ефективного програмування.

Основна мета цих методичних вказівок – допомогти студентам зрозуміти принципи роботи мови Python, розібратися в її синтаксисі та ключових конструкціях. Особлива увага приділяється таким базовим темам, як синтаксис мови, типи даних (числові, рядкові, логічні типи), оператори порівняння, цикли та логічні оператори. Ці теми є фундаментом для подальшого вивчення мови та необхідні для розуміння більш складних концепцій.

Курс також включає розгляд колекційних типів даних, таких як списки, кортежі, множини та словники, які дозволяють зберігати та опрацьовувати великі обсяги інформації ефективно та зручно. Завдяки детальному розгляду роботи з колекціями студенти навчаться застосовувати їх у реальних програмних рішеннях. У вказівках наводяться численні приклади, що демонструють, як правильно використовувати ці інструменти для вирішення різних типів завдань.

Друга частина курсу присвячена функціям та методам їх оптимізації через використання обгорток. Студенти навчаться створювати власні функції, що сприяє організації коду в модулі та повторному його використанню. Велике значення надається також обробці помилок — методи та інструменти, що розглядаються в рамках цієї теми, допоможуть студентам створювати програми, стійкі до помилок, та підвищувати їх надійність.

Ще одним важливим аспектом курсу є ознайомлення з об'єктно-орієнтованим програмуванням (ООП). У методичних вказівках розглядаються основні принципи ООП, такі як класи, об’єкти, спадкування та поліморфізм. Це дозволяє студентам зрозуміти, як структурувати великі програмні проекти, використовуючи ООП для побудови більш гнучких і масштабованих систем.

Окрема увага приділяється практичним аспектам роботи з бібліотеками Python, які знайшли широке застосування у наукових та інженерних розрахунках. У методичних вказівках представлені такі бібліотеки, як NumPy, Pandas та Matplotlib, що дозволяють ефективно виконувати складні розрахунки, а також візуалізувати отримані результати. Студенти навчаться використовувати ці бібліотеки для аналізу даних, побудови графіків та вирішення прикладних задач.

Для повноцінного оволодіння навичками програмування на Python також важливим є вивчення принципів створення графічного інтерфейсу користувача (GUI). У методичних вказівках розглянуто основи роботи з інструментами для розробки GUI, що дозволить студентам створювати власні застосунки з інтуїтивно зрозумілими інтерфейсами. Зокрема, увага приділяється використанню таких бібліотек, як Tkinter або PyQt.

Загалом, методичні вказівки є важливим інструментом для опанування програмування на Python. Вони не тільки сприяють ефективному засвоєнню матеріалу, але й надають практичні інструменти та приклади, які допоможуть студентам закріпити знання на практиці та застосовувати їх у реальних проектах. Кожен розділ вказівок містить чіткі інструкції, приклади коду та рекомендації щодо самостійного виконання завдань, що допомагає студентам у поглибленні знань та підготовці до подальшої професійної діяльності у сфері програмування.

Тема 1. Вступ до програмування мовою Python.

Зміст розділу:

1. Встановлення Python 3 та середовища розробки.
2. Типи даних.
3. Модуль math.
4. Робота з рядками.

Завантаження Python.

З'ясуйте розрядність вашої операційної системи.

Перейдіть на сайт https://www.python.org/downloads/

Оберіть версію Python. Завантажте файл з розширенням .exe відповідної розрядності.

Встановіть Python:

- відзначте рекомендований параметр Install launcher for all users;
- не забудьте встановити прапорець Add Python 3.x to PATH (це полегшить правильне налаштування системи);
- оберіть варіант налаштування установки Customize installation;
- вкажіть каталог установки C:\PythonX (де X – номер версії).

Перевіримо, чи Python успішно був встановлений на комп'ютер. Для цього натисніть сполучення клавіш Win+R на клавіатурі, введіть команду cmd і натисніть OK. У консольному вікні, що з'явилося, введіть команду
python -- version і натисніть Enter:

```
> python --version
  Python 3.10.7
```

Якщо отримали схожий результат, то Python відповідної версії успішно встановлений у вашій системі.

Встановлення середовища розробки PyCharm:
https://www.jetbrains.com/pycharm/

Запуск файлу на виконання (Python читає файли построково):

1. У терміналі, за допомогою команди python та назви файлу з розширенням, наприклад file.py
2. Через меню у середовищі розробки PyCharm, через RUN для відповідного файлу
3. Комбінацією клавіш Ctrl + Shift + F10

Типи даних.

Базові типи даних (вказуються неявно):

```
a = 5 # int
b = 7.0 # float
c = 2 > 4 # Boolean
d = "World" # string
e = 1.5 + 0.5j # complex
```

Для перевірки типу даних – type()
Для виводу результату у консоль – print()

```
print(type(a))
print(type(e))
print(a, b, c, d, e.real, e.imag)
```

Перетворення типів даних.

Для перетворення будь-якого значення на ціле число використовується функція int()

```
p = 2.3
print(int(p)) # 2
```

Для перетворення будь-якого значення на дійсне число використовується функція float()

```
p = 8
print(float(p)) # 8.0
```

Для перетворення будь-якого числового значення у рядок використовується функція str()

```
p = 78.5
print(str(p)) # '78.5'
```

Введення та виведення даних через термінал.

Функція input() повертає результат введення даних користувачем з клавіатури, який представляє собою рядок символів:

```
x = input('Введіть x\n')
#дані, що вводяться мають тип рядка
y = input('Введіть y\n')
x = int(x) #здійснюємо перетворення типів
y = int(y)
print(x+y) #додаємо два введені числа. Результат – сума
```

Символ \n – перенос на новий рядок.

Оператори виразів в Python і правила визначення старшинства (Табл. 1).

Таблиця 1 – Оператори виразів в Python.

Оператори	Опис
yield *x*	Підтримка протоколу send у функціях-генераторах
lambda args: expression	Створює анонімну функцію
x if *y* else *z*	Тримісний оператор вибору (значення *x* обчислюється, тільки якщо значення *y* істинно)
x or *y*	Логічна операція «АБО» (значення *y* обчислюється, тільки якщо значення *x* = хибність)
x and *y*	Логічний оператор «І» (значення *y* обчислюється, тільки якщо значення *x* = істина)

not *x*	Логічне заперечення
x in *y*, *x* not in *y*	Перевірка на входження (для ітерованих множин)
x is *y*, *x* is not *y*	Перевірка ідентичності об'єктів
x < *y*, *x* <= *y*, *x* > *y*, *x* >= *y*	Порівняння, перевірка на підмножину і над множину
x == *y*, *x* != *y*	Оператори перевірки на рівність
x | *y*	Бітова операція «АБО», об'єднання множин
x ^ *y*	Бітова операція «виключне АБО» (XOR), симетрична різниця множин
x & *y*	Бітова операція «І», перетин множин
x << *y*, *x* >> *y*	Зсув значення *x* вліво або вправо на *y* бітів
x, + *y*	Додавання, конкатенація
x – *y*	Віднімання, різниця множин
x * *y*	Множення, повторення
x % *y*	Залишок від ділення, формат
x / *y*, *x* // *y*	Ділення: справжнє і з округленням вниз
- *x*	Унарний знак «мінус»
~ *x*	Бітова операція «НЕ» (інверсія)
x ** *y*	Піднесення до степеню
x[i]	Індексація (в послідовності, відображеннях)
x[i:j:k]	Витяг зрізу
x(...)	Виклик (функцій, класів і інших об'єктів)
x.attr	Звернення до атрибуту
(...)	Кортеж, підвираз, вираз-генератор
[...]	Список, генератор списків
{...}	Словник, множина, генератор словників і множин

Стандартний модуль math.

Імпорт модулів – ключове слово import ім'я_модуля:

```
import math
sn =  math.sin; cs= math.cos; p = math.pi
```

Інший варіант інструкції from ім'я_модуля import *:

```
from math import *
a = 1
b = 2
x = sqrt(a*b)/(exp(a)*b)+a*exp((2*a)/b) print(x)
```

Функції в бібліотеці math.

До модуля входять змінні *pi* та *e*:

```
print(math.pi) # 3.141592653589793
print(math.e) # 2.718281828459045
```

Крім тригонометричних, математичних та алгебраїчних функцій, таких як:
acos(x), cosh(x), ldexp(x,y), sqrt(x), asin(x), exp(x), log(x), tan(x), atan(x), fabs(x), sinh(x), frexp(x), atan2(x,y), floor(x), pow(x,y), modf(x), ceil(x), fmod(x,y), sin(x), cos(x), log10(x), tanh(x)

у модулі містяться і особливі функції:

- **math.ceil(x)** – повертає округлене x як найближче ціле значення типу float, яке дорівнює або перевищує x (округлення "вгору").
- **math.copysign(x, y)** – повертає число x зі знаком числа y. На платформі, яка підтримує знак нуля copysign (1.0, -0.0) дасть -1.0.
- **math.fabs(x)** – повертає абсолютне значення (модуль) числа x. В Python є вбудована функція abs, але вона повертає модуль числа з тим же типом, що число, fabs же завжди повертає значення типу float.
- **math.factorial(x)** – повертає факторіал цілого числа x, якщо x не є цілим виникає помилка ValueError.
- **math.floor(x)** – на противагу ceil(x) повертає округлене x як найближче ціле значення типу float, менше або рівне x (округлення "вниз").
- **math.fmod(x, y)** – аналогічна функції fmod(x, y) бібліотеки C. Зазначимо, що це не те ж саме, що вираз Python x%y. Бажано використовувати при роботі з об'єктами float, в той час як x% y більше підходить для int.
- **math.frexp(x)** – являє число в експоненційному записі x= m*2e і повертає мантису m (дійсне число, модуль якого лежить в інтервалі від 0.5 до 1) і порядок e (ціле число) як пару чисел (m, e). Якщо x = 0, то повертає (0.0, 0)
- **math.fsum(iterable)** – повертає float суму від числових елементів ітерованого об'єкта.
- **math.isinf(x)** – перевіряє, чи є float об'єкт x плюс або мінус нескінченністю, результат відповідно True або False.
- **math.isnan(x)** – перевіряє, чи є float об'єкт x об'єктом NaN (not a number).
- **math.ldexp(x, i)** – повертає значення x*2i, тобто здійснює дію, зворотну функції math.frexp(x).

Рядки.

Рядок (String) – це складний тип даних, який представляє собою упорядковані послідовності символів, що використовуються для зберігання і представлення текстової інформації.

Рядок можна створити, застосувавши подвійні або одинарні лапки:

```
print("Це рядок")
print('І це рядок')
```

Кожний символ у рядку має унікальний порядковий номер — індекс. Нумерація символів починається з нуля. До конкретного символу в рядку можна звертатися за його індексом, зазначивши індекс у квадратних дужках a[0].

Функції для роботи з рядками.

Для визначення тих чи інших особливостей рядка, у бібліотеці Python введено наступні функції класу str:

- **str.isdigit()** — перевіряє чи складається рядок з цифр, повертає True, якщо виконуються обидві наступні умови:
  1. всі символи рядка є цифрами;
  2. у рядку є хоча б один символ.

  В іншому випадку функція повертає False.
- **str.islower()** — перевіряє чи рядок містить символи в нижньому регістрі.

- **str.isupper()** — перевіряє чи містить рядок символи у верхньому регістрі.
- **str.isnumeric()** — перевіряє чи містить рядок числові символи.
- **str.isspace()** — перевіряє чи є в рядку пробільні символи.
- **str.lstrip()** — видалення символів пробілів на початку рядка.
- **str.rstrip()** — видалення символів пробілів в кінці рядка.
- **str.strip()** — видалення символів пробілів на початку і в кінці рядка.

При виклику методів необхідно пам'ятати, що рядки в Python відносяться до категорії незмінних послідовностей, тобто всі функції і методи можуть лише створювати новий рядок.

```
msg = "Hello student ?"
msg[-1] = "!"   # помилка
msg = msg[0:-1] + "!"
print(msg)          #Hello student !
text = "red blue orange while"
words = text.split()
print(words)   # ['red', 'blue', 'orange', 'while']
new_text = text.split('e')
print(new_text) # ['r', 'd blu', ' orang', ' whil', '']
```

Службові символи. Зріз.

Екрановані послідовності – \
Спец символ для позначення шляху – **r**
Спец символ для переносу рядка – **\n**
Зріз – вибірка елементів з послідовності починаючи з __ закінчуючи __ індексом. Наприклад:

- від індексу 8 до кінця рядка – text[8:]
- від початку рядка до індексу 8 – text[:8]
- від 6 до 8 індексу – text[6:8]
- з кроком в 2 символи – text[::2]

Функції для роботи з рядками.

Метод **split()** – розділяє рядок на список підрядків по роздільнику.

- *sep* – необов'язковий параметр, що дозволяє встановити роздільник вручну. За замовчуванням будь-який пробіл є роздільником.
- *maxsplit* – необов'язковий параметр, що вказує на максимальну кількість розбиття, яке потрібно виконати. Якщо параметр вказано, то виконується не більше maxsplit розбиттів, тобто підсумковий список міститиме не більше maxsplit +1 елементів. Якщо maxsplit не вказано або дорівнює -1, то обмеження кількості сплітів немає.

```
num = '1, 2, 3, 4, 5'
# Вказуємо роздільник та обмежуємо кількість сплітів.
print(num.split(', ', 2))     # ['1', '2', '3, 4, 5']
```

len() – повертає кількість елементів у послідовності, тип int
coutn() – повертає кількість конкретних елементів у послідовності
find() – індекс першого знайденого елементу чи фрази у рядку. Діапазон пошуку як необов'язковий параметр: text.find('o', 7, 14)
capitalize() – перетворює текст, як у реченні.

upper() – перетворює рядок у верхній регістр
lower() – перетворює рядок у нижній регістр

Метод **format().**

Якщо фігурні дужки вихідного рядка порожні, то підстановка аргументів йде згідно з порядком їх слідування. Якщо в фігурних дужках вказані індекси аргументів, порядок підстановки може бути змінений

- рядок (‘{}текст{}’.format(x,y)) – де x та y – змінні
- рядок з позиціонуванням змінних (‘{0}текст{1}’.format(x,y))
- скорочений запис (f‘{x}текст{y}’)
- по ключу:

```
info = "This is a {subj}. It's {prop}."
print(info.format(subj="table", prop="small"))
# "This is a table. It's small."
```

Завдання до лабораторної роботи №1

1. Виведіть на екран № своєї групу, прізвище та ім’я. Створіть змінну a1 в яку запишіть ціле число, змінну a2 – дійсне число. Виведіть результат на екран за допомогою print() та перевірте тип даних – type().

2. Використовуючи функцію input() введіть значення двох змінних з клавіатури і виконайте над ними всі можливі арифметичні операції (+, -, *, /, %). Результат вивести за допомогою одного print() з відповідними текстовими коментарями до чисел, використовуючи format().

3. Напишіть програму для рішення задачі розрахунку оплати чеку в кафе. N друзів отримали в кафе рахунок на суму m гривень. Було прийнято рішення розділити оплату порівну. До оплати додали суму чайових яка склала 15 % від суми рахунку. Необхідно визначити суму оплати кожного з друзів. (Значення N і m введіть з клавіатури).

4. По варіантах (обрати одне завдання) обчислити значення виразу та вивести його на екран. Значення вводить користувач. Вважається, що користувач може вести лише число.

$$z=\left(\frac{e^{-x}-12.34}{\lg x-\cos x^3}\right)^{-0.5}$$

$$y=\frac{\sqrt{x-1}-tg(x+1)}{\arccos x+\ln x}+2,75$$

$$g=\frac{\sin x^2-\cos^4(x-1)^2}{arctg(x+2,6)+\sqrt[3]{\ln x}}$$

$$p=\frac{e^{-3x}+tg(3x-3)}{|\sin x|+\sqrt[4]{\cos x+\cos 2x}}$$

$$y=\frac{e^{-x}-4x-\ln^3 x}{\lg|x+1|+ctg(x^2-1)}$$

$$y=x+\frac{\sqrt{\arcsin x+arctg\pi x}}{\lg(13.4x+\pi)}$$

$$y=\operatorname{arc}tg\,x+\frac{e^{0,6x-1}-\sqrt{(x+6,1)^3}}{\ln x+tg^2 x}$$

$$y=\arcsin\left(\frac{x\cdot\ln x}{1+\cos x}+1\right)$$

$$u=0,3\cdot\lg e^{-x}+\frac{arctg\,x-\sin^2 x}{4\cdot\sqrt{\ln|x-1|}}$$

$$y=e^2\cdot\lg x^4\cdot\frac{(x-0,5)^2-\cos x}{\sqrt{|x+1|+|x|}}$$

$$y=\frac{e^{-x}-4\cdot\lg x}{\ln x-\cos|x+1|}$$

$$c=\sin^2 x-\frac{\sqrt{\lg|2-x|+\lg x}}{\sqrt[3]{tg\,x}+\sqrt{\cos^3 x}}$$

$$y=\arccos\left(\frac{x-\lg x}{1+\cos 3x}+1\right)$$

$$c=arctg\,x-\frac{\sqrt{\ln 4+\ln x}}{\sqrt[3]{\lg 2.4}+\sqrt[5]{\cos x^{-1}}}$$

$$y=\frac{\ln e^{-x}+\cos(x-1)}{\ln^3 x+\sqrt{sin3x+\cos 2x}}$$

$$y=3\cdot\lg x^4\cdot\frac{(x-1)^2-tg\,x}{\cos x+\sqrt{|x+1|+|x|}}$$

$$y=\frac{e^{-3x}+tg(4x-1)}{|\cos x|+\sqrt{\cos 2x}}$$

$$y=\frac{e^{-3x}+\ln^3(x-1)}{\ln|x+1|+tg(x^2-1)}$$

$$y=\sin\left(\frac{x+2.3\cdot\lg(x+1)}{\sqrt{2\ln x}+\cos x}\right)$$

$$u=\ln|1-x|+\frac{tg\,x-\sin^2 x}{1-\sqrt{\ln x}}$$

$$y=\frac{e^{-x}+tg(x-1)}{|\ln x|+\sqrt{sin\,x+\cos 2x}}$$

$$c=tg|x|-\frac{\sqrt{\ln 2+\ln x}}{\sqrt[3]{tg\,x}+\sqrt[4]{\cos x^{-1}}}$$

$$y=\frac{\sqrt{x+1}-\sin(x-\pi)}{\cos(x-3.1)+\ln^2 x}+x\cdot\lg x$$

$$y=tg\left(\frac{x+3\cdot\lg(x+1)}{\sqrt{\ln 4x}+\cos x}\right)$$

$$y=\frac{\sin^3 x-\cos(x-1)^2}{arctg(x+1)+\sqrt[3]{\lg x}}$$

$$c=lg|x|-\frac{\sqrt{\ln(x-1)+\ln x}}{\sqrt[3]{tg\,x}+\sqrt{\cos(x-\pi)}}$$

$$y=\frac{\sqrt{\sin(x+1)+arctg\pi x}}{\lg(x+2\pi)}-1$$

$$y=\frac{e^{-x}-x\cdot\sin x-\ln^2 x}{\lg|\cos x|+ctg(x^2-1)}$$

$$y=\arcsin x+\frac{e^{x-1}-\sqrt{(x+1)^5}}{\lg^3 x+tg\,x}$$

Тема 2. Колекції у Python. Цикли. Вийнятки.

Зміст розділу:

1. Списки list[].
2. Кортежі tuple().
3. Словники dict{}.
4. Множини set{}.
5. Умовний оператор if elif else.
6. Цикли. Оператор циклу for , while.
7. Оператори continue, break, pass.
8. Виключення (винятки).

Списки list[].

Список – являє собою послідовність впорядкованих елементів в квадратних дужках.

- **len()** – довжина списку, тобто кількість елементів в ньому.
- **append()** – додавання елементу до списку. Метод може отримувати тільки один параметр. Параметром методу може бути будь-який об'єкт: число, рядок, список і т.д.
- **extend()** – дозволяє розширити список. Вхідним параметром методу є **інший список**, який додається до даного. Список розширення може бути іменованим об'єктом або списком, взятим у квадратні дужки [ ].
- **insert()** – дозволяє вставити новий елемент в список із заданої позиції. Метод отримує два параметри. **Перший** параметр – позиція вставки, яка нумерується з 0. **Другий** параметр – ім'я об'єкту (значення), яке вставляється.
- **index()** – визначення індексу заданого конкретного елементу в списку.

```
my_list.insert(2, 777)
# вставка в позицію 2 нового числа 777
```

- **count()** – визначення кількості входжень заданого елементу у список.
- **pop()** – видаляє останній елемент із списку. При використанні індексу, видаляє елемент по ньому.
- **clear()** – видаляє усі елементи списку, та повертає пустий список.
- **sort()** – сортує елементи списку. Підтримує сортування по ключу.
- **reverse()** – «перевертає» список від останнього до першого елементу.

```
my_list = ['1', '22', '3333', '7700', '55555']
my_list.sort()
# ['1', '22', '3333', '55555', '7700']
my_list.sort(reverse=True)
# ['7700', '55555', '3333', '22', '1']
my_list.sort(key=len)
#['1', '22', '7700', '3333', '55555']
```

Кортежі tuple()

Кортеж – це група об'єктів різних типів, взята в круглі дужки яка не допускає зміни.

```
a = () # пустий кортеж
c = ('ab', 'abcd', 'cde', 'abc', 'def')
b = (1, 3, 7, 13, 'abc', True)
```

tuple() – створення кортежу з ітерованого об’єкту:

```
d = tuple('Hello')
# ('H', 'e', 'l', 'l', 'o')
f = tuple((3, 2, 0, -5))
# (3, 2, 0, -5) створення з іншого кортежу

A = (1, 2, 3)
B = (4, 5, 6)
C = A + B # C = (1, 2, 3, 4, 5, 6) конкатенація
A = (1, 2, 3) * 3
# A = (1, 2, 3, 1, 2, 3, 1, 2, 3) повторення задану кількість разів
```

Словники dict{}.

Словники – це вбудований тип даних, який базується на відображенні пар типу (ключ:значення).

- **dict.clear()** — очищує словник;
- **dict.copy()** — повертає копію словника;
- **dict.fromkeys(seq[, value])** — створює словник з ключами з seq та значенням value (за замовченням None);
- **dict.get(key[, default])(x)** — повертає значення ключа, а якщо його немає, то повертає default (за замовченням None)
- **dict.items()** — повертає пари (ключ, значення);
- **dict.keys()** — повертає ключі в словнику;
- **dict.pop(key[, default])** — видаляє ключ і повертає значення. Якщо ключа немає, повертає default (за замовченням породжує виключення);
- **dict.popitem()** — видаляє і повертає пару (ключ, значення). Якщо словник порожній, породжує виключення KeyError;
- **dict.setdefault(key[, default])** — повертає значення ключа, але якщо його немає, то не породжує виключення, а створює ключ зі значенням default (за замовченням None);
- **dict.update([other])** — обновляє словник, додаючи пари (ключ, значення) із other. Якщо є пари в dict та other з одним ключем, значення буде взято з other. Повертає None (не новий словник!);
- **dict.values()** — повертає значення у словнику.

Вбудовані методи словників.

❖ Приклад додавання у, зміни та видалення з словника:

```
dic = {'cat': 'кішка', 'dog': 'собака', \
          'bird': 'птах', 'mouse': 'миша'}
dic['elephant'] = 'бегемот'
dic['fox'] = 'лисиця'
print(dic)
# {'cat': 'кішка', 'dog': 'собака', 'bird': 'птах', 'mouse':
'миша', 'elephant': 'бегемот', 'fox': 'лисиця'}
dic['elephant'] = 'слон'
del(dic['bird'])
print(dic)
# {'cat': 'кішка', 'dog': 'собака', 'mouse': 'миша', 'elephant':
'слон', 'fox': 'лисиця'}
```

❖ Приклад оновлення та злиття списку:

```
d1 = {'A': 1, 'B': 2, 'C': 3}
d2 = {'C': 2, 'D': 4, 'E': 5}
d1.update(d2)
print(d1)
# {'A': 1, 'B': 2, 'C': 2, 'D': 4, 'E': 5}
d1 = {'A': 1, 'B': 2, 'C': 3}
d2 = {'C': 2, 'D': 4, 'E': 5}
print(dict(d2, ** d1))
# {'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5}
```

Множини set{}.

Множина – це невпорядкована колекція елементів, послідовність унікальних об'єктів, створюється за допомогою фігурних дужок {}, а також можна створити зі списку, використавши функцію set.

| — об'єднання; & — перетин; – — різниця; ^ — симетрична різниця

```
a = set('abcde')
b = set('cdefgh')
print(a) # {'a', 'e', 'c', 'b', 'd'}
print(b) # {'c', 'h', 'e', 'f', 'd', 'g'}
print(a-b) # {'a', 'b'}
print(a | b) # {'a', 'c', 'h', 'e', 'f', 'b', 'd', 'g'}
print(a & b) # {'e', 'c', 'd'}
print(a ^ b) # {'h', 'f', 'b', 'a', 'g'}
```

Операції над множинами:

- метод **add** приймає один аргумент будь-якого незмінного типу і додає його в множину, якщо такого елемента там ще немає;
- метод **update** приймає один аргумент множину і додає всі елементи цієї множини до даної множини.
- **discard** приймає одне значення як параметр, і видаляє це значення з множини. Якщо викликати discard для значення, якого немає у множині, в ній нічого не трапиться.
- **remove** так само приймає одне значення як аргумент і також видаляє це значення з множини. Але якщо такого значення немає у множині, метод remove видає помилку KeyError .
- **pop** видаляє одне значення з множини і повертає його. Для множини — як невпорядкованого набору — поняття останнього елемента немає, тому важко передбачити, який елемент буде вилучено методом pop. Спроба виклику pop для порожньої множини створить виняток KeyError.
- **clear** видаляє всі елементи множини, перетворюючи її на порожню. Це еквівалентне наданню значення = set(), яке створить нову порожню множину і замінить посилання на неї у змінній.
- **union** – об'єднує дві множини
- **intersection** – функція «перетину» двох множин, видає елементи, які є і у 1-шій і у 2-ій множині
- **difference** – функція «віднімання» двох множин, видає елементи, які є у 1-шій множині за винятком тих, що є у 2-ій множині

- **issubset** – перевіряє, чи є 1-ша множина підмножиною 2-гої.
- **issuperset** – функція перевіряє, чи є 1-ша множина над множиною 2-гої

Множини не підтримують звернення по індексу!

Умовний оператор if ...else

```
if <умова> :
     оператор 1 …
else:
     оператор 2 …
```

Приклад 1. Визначити чи ділиться число a на b. Числа вводяться користувачем з клавіатури.

```
a = int(input('Введіть перше число a = '))
b = int(input('Введіть друге число b = '))
res = a % b
if res==0 :
     print('Число ', a, ' ділиться на ', b)
else:
     print('Число ', a, ' не ділиться на ', b)
```

Приклад 2. Скласти програму, яка буде знаходити розв'язок наступного рівняння: x = a ÷ b, за умови, що b не дорівнює нулю.

```
a = float(input('a = ?'))
b = float(input('b = ?'))
if (b != 0):
     print(a/b)
```

Приклад 3. Перевірка існування трикутника. Перевірити умову, що сума двох сторін більше третьої.

```
a = float(input('a = ?'))
b = float(input('b = ?'))
c = float(input('c = ?'))
if a + b > c and a + c > b and b + c > a:
     print('Це трикутник.')
else:
     print('Це не трикутник.')
```

Приклад 4. Перевірити рівність дробу 0.

```
if x==0 and not(y==0):
     print('Дріб дорівнює нулю.')
# чи так
if x==0 and y!=0:
```

Множинне розгалуження (if… elif… else).

Приклад 5. Виконати перевірку введеного користувачем числа за умовою: якщо число дорівнює 0, то виводимо 0, якщо число лежить в діапазоні від 0 до 100 виводимо 1, якщо менше 0, то виводити 2, інакше повідомлення про помилку.

```
num1 = int(input())
if num1 == 0:
     result = 0
elif num1 >0 and num1 <100:
     result = 1
elif num1 <0:
     result = 2
else:
     print("Error")
print(result )
```

Цикл for та range().

```
spisok = [12, 44, 28, 30]
i = 0
for element in spisok:
     spisok[i] = element / 2
     i += 1
print(spisok)
# результат: [6 22 14 15]
```

Цикл for (цикл з параметром) призначений для перебору елементів структур даних і деяких інших об'єктів.

```
spisok = [12, 44, 28, 30]
for i in range(len(spisok)):
     spisok[i] = spisok[i] / 2
print(spisok)
# [6.0, 22.0, 14.0, 15.0]
```

Приклад 6. Вивести на екран слово «Привіт!» 10 разів:

```
for _ in range(10):
     print("Привіт!")
```

Цикл while.

```
while логічний_вираз:
     вираз 1 ...
     вираз n
```

Приклад 7. Надрукувати числа з діапазону [1, 10].

```
i = 1
while i<=10:
     print(i)
     i = i + 1
```

Оператори continue та break.

Оператор **continue** починає наступний прохід циклу, минаючи тіло циклу (for або while).

Приклад 8. Вивести усі парні числа від 6 до 30, окрім чисел у діапазоні від 14 до 22.

```
for i in range(6, 31, 2):
     if i > 12 and i < 26: # якщо число 12<i<26
          continue # перехід на нову ітерацію циклу
     print(i, end=" ") #6 8 10 12 26 28 30
```

Оператор **break** достроково переривает цикл. Здійснює вихід із циклу навіть у тому випадку, коли всі ітерації не виконано.

Приклад 9. Обчислити суму цілих чисел доти, поки не буде введено q.

```
  s = 0  # початкова сума
  while True:  # нескінченний цикл
     a = input("Введіть число: ")
     if a == "q":  # якщо введено символ q
          break # переривання нескінченного циклу
     a = int(a) # перетворення рядка в ціле число
     s = s + a # обчислення суми чисел
  print("Сума чисел s =", s)
  input()
```

Сенс такого запису з'являється лише разом із інструкцією break. Якщо під час виконання Python зустрічає інструкцію break всередині циклу, він відразу ж припиняє виконання цього циклу і виходить із нього. При цьому гілка else не виконуватиметься. Зрозуміло, інструкцію break у змозі викликати лише всередині інструкції if, тобто вона повинна виконуватися тільки при виконанні якоїсь особливої умови.

```
a = int(input())
while a != 0:
     if a < 0:
          print('Зустрілось число <0', a)
          break
     a = int(input())
else:
     print('Жодного від\'ємного числа')
```

Pass.

Pass – оператор-заглушка, рівноцінний відсутності операції.

У ході виконання даного оператора нічого не відбувається, тому він може використовуватися як заглушка в тих місцях, де це синтаксично необхідно, наприклад: в інструкціях, де тіло є обов'язковим.

Припустимо, у нас є цикл чи функція, які ми ще не реалізували, але хочемо зробити це у майбутньому. У Python у циклів і функцій не може бути "порожнього" тіла – інтерпретатор виведе помилку. Тому в тіло можна помістити інструкцію pass і все запрацює.

```
for x in [0, 1, 2]:
    pass
```

Обробка виняткових ситуацій (виключення).

Винятки – це сповіщення інтерпретатора, порушувані в разі виникнення помилки в програмному коді або при настанні якої небудь події. Якщо в коді не передбачено оброблення винятків, то програма переривається і виводиться повідомлення про помилку.

В програмі існує три типи помилок :

Синтаксичні – це помилки в імені оператора або функції, невідповідність закриваючих та відкриваючих лапок і т.д. Тобто помилки в синтаксисі мови.

Семантичні – це помилки в логіці роботи програми, які можна виявити тільки за результатами роботи скрипта. Як правило, інтерпретатор не попереджає про наявність помилки. А програма буде виконуватися, оскільки не містить синтаксичних помилок.

Помилки часу виконання – це помилки, які виникають під час роботи скрипта. Причиною є події, які не передбачені програмістом. Класичним прикладом служить ділення на нуль.

```
try:
    ...
except:
    ...
```

Це означає, що якщо всередині try буде помилка, тоді виконається блок except. Якщо в тілі try виключення не виникає, то тіло гілки except не виконується.

```
n = input("Введіть ціле число: ")
try:
    n = int(n)
    print("Число введено")
except:
    print("Введено не число.")
# Може бути
except ValueError:
    print("Введено не число.")
```

Приклад 10. Обробка винятку ділення на 0:

```
try:
    a = int(input('Введіть перше число: '))
    b = int(input('Введіть друге число: '))
    print(a / b)
except ZeroDivisionError as e:
    print(e)  # division by zero
# Отримали опис помилки
# Опрацюємо виняток, виведемо напис користувачу
except ZeroDivisionError:
    print('На нуль ділити неможна')
except Exception as e:
    print(e, type(e))
# Відловили усі помилки, показали їх тип. Програма не припинить роботу
```

Обробка виняткових ситуацій. Блок finally. Загальна схема.

```
try:
     print('Я впевнений, винятків не буде!')
except Exception:
     print('Виключення')
else:
# Будь-який код, який має бути виконаний, якщо виняток у блоці
try не було викликано, але для якого не повинно проводитися
обробка винятків
     print('Я буду виконаний, якщо у try не буде винятків.''Мої
винятки не будуть оброблятися.')
finally:
     print('Я буду виконаний у будь-якому випадку!')
Результат:
>>> Я впевнений, винятків не буде!
>>> Я буду виконаний, якщо у try не буде винятків.
Мої винятки не оброблятимуться.
>>> Я буду виконаний у будь-якому випадку!
```

https://pypi.org/ – документація по роботі з бібліотеками.

Requests – Модуль запитів дозволяє надсилати HTTP-запити за допомогою Python. HTTP-запит повертає об'єкт відповіді з усіма даними відповіді (вміст, кодування, статус тощо).

Для встановлення модуля вписати команду в терміналі

pip install requests

Time – модуль для роботи з часом. Функція sleep() призупиняє (затримує) виконання поточного потоку на задану кількість секунд.

Datetime – модуль надає класи для роботи з датами та часом. Використовуємо datetime.now(), щоб отримати поточну дату й час.

```
import requests
import time from datetime
import datetime
while True:
     try:
          a = requests.get('https://www.google.com.ua/')
          print(a)
          time.sleep(10)
          if a == '<Response [200]>' :
               pass
          elif a == '<Response [503]>' :
               print('Помилка сайту')
          else:
               print('Якась інша помилка')
     except requests.exceptions.ConnectionError
          error_time = datetime.now()
          print('Сервер лежить!\n' + str(error_time))
```

Основні винятки:

- **Exception** – те, на чому фактично будуються всі інші помилки;
- **AttributeError** – виникає, коли посилання атрибута чи присвоєння не можуть бути виконані;
- **IOError** – виникає в тому випадку, коли операція I/O (така як оператор виводу, вбудована функція open() або метод об'єкта-файлу) не може бути виконані, пов'язаною з I/O причиною: «файл не знайдено», або «диск заповнений», чи іншими словами.
- **ImportError** – виникає, коли оператор import не може знайти визначення модуля, або коли оператор не може знайти ім'я файлу, який має бути імпортований;
- **IndexError** – виникає, коли індекс послідовності знаходиться поза допустимим діапазоном;
- **KeyError** – виникає, коли ключ зіставлення (dictionary key) не знайдено у наборі існуючих ключів;
- **KeyboardInterrupt** – виникає, коли користувач натискає клавішу переривання (зазвичай Delete чи Ctrl+C);
- **NameError** – виникає, коли локальне чи глобальне ім’я не знайдено;
- **OSError** – виникає, коли функція отримує пов'язану із системою помилку;
- **SyntaxError** – виникає, коли синтаксична помилка зустрічається синтаксичним аналізатором;
- **TypeError** – виникає, коли операція чи функція застосовується до об'єкта невідповідного типу. Пов'язане значення є рядок, в якому наводяться докладні відомості про невідповідність типів;
- **ValueError** – виникає, коли вбудована операція чи функція отримують аргумент, тип якого правильний, але неправильне значення, і ситуація неспроможна бути описана точніше, як із виникненні IndexError;
- **ZeroDivisionError** – виникає, коли другий аргумент операції division чи modulo дорівнює нулю.

Завдання до лабораторної роботи №2

1. За допомогою конструкції if elif else описати перевірку введену користувачем точки (x та y координати), чи належить вона координатним осям, якщо так – то якій чверті, якщо не належить – вивести відповідне повідомлення.
2. Обчислити значення функції y, за умови, що при x<0 – вона дорівнює x; при 0<=x<1 вона дорівнює 0; при x>1 вона дорівнює 2*x. Користувач вводить x з клавіатури.
3. Написати калькулятор, який робить обчислення з двох чисел, залежно від значення параметра type, де type може бути '+' '-' '*' ‘/’. Приклад: calc(10, 10, '+')
4. У написаному калькуляторі обробіть помилку ділення на нуль (ZeroDivisionError), та помилку введення не числа (ValueError)
5. Створіть текстову змінну str в яку запишіть своє ім’я, групу в якій навчаєтесь, спеціальність. З створеного рядка виведіть на екран тільки назву групи. В рядку str замініть своє ім’я на прізвище і виведіть змінений рядок на екран. Виконайте розподіл рядка по пробілу та обчисліть кількість слів у вашому рядку.
6. Задайте два списки однакової довжини. Необхідно створити з них словник таким чином, щоб елементи першого списку були ключами, а елементи другого – відповідно значеннями нашого словника.
7. Відпрацюйте команди доповнення множини елементами, вилучення елементів, пошуку елемента. Результати вивести на екран.

Тема 3. Функції.

Зміст розділу:

1. Функції.
2. Функція як аргумент.
3. Лямбда функції.
4. Вкладені функції.

Створення функції.

Функція в Python – це об'єкт, який приймає аргументи і повертає значення.

Блок функції починається з ключового слова def , після якого йдуть назва функції і круглі дужки ( ). Будь-які аргументи, які приймає функція повинні знаходитися всередині цих дужок. Після дужок йде двокрапка : і з нового рядка з відступом починається тіло функції.

Вираз return припиняє виконання функції і повертає вказане після цієї команди значення:

```
def <назва> (параметри):
    <блок команд>
    return <результат>
```

Функції приймають неіменовані змінні позиційовано.

Можна використовувати параметр за замовчуванням в якості заміни при відсутності формального, вони завжди йдуть останніми.

Можна передавати іменовані параметри, без позиціонування.

У return записується необхідний результат. Функцію з прописаним return можна передавати в іншу функцію. Передається результат роботи функції не вона сама.

Приклад створення функції:

```
def func(a, b=5, c=10):
    print('a = ', a, ', b = ', b, ', c = ', c)
func(3, 7) # a = 3 , b = 7 , c = 10
func(0) # a = 0 , b = 5 , c = 10
func(25, c=5) # a = 25 , b = 5 , c = 5
func(c=1, a=100) # a = 100 , b = 5 , c = 1
func(-1, -5, 8) # a = -1 , b = -5 , c = 8
func()
# TypeError: func() missing 1 required positional argument: 'a'
func(b=0, c=5)
# TypeError: func() missing 1 required positional argument: 'a'
```

Документація до функцій.

Хорошим тоном при написанні функцій є створення документації. Для цього використовується багаторядковий коментар. Якщо у тіло функції додати ‘‘‘ ’’’, то PyCharm автоматично пропише усі необхідні змінні, нам залишається лише додати опис.

Для того, щоб викликати документацію функції, навіть якщо її було імпортовано, використовується help(ім’я функції).

```
def function(a, b, c):
    '''
    :param a:
    :param b:
    :param c:
    :return:
    '''
    return a+b+c
help(function)
```

Аргументи довільної довжини *args **kwargs.

Коли заздалегідь не відома кількість аргументів, які необхідно прийняти функції – слід використовувати аргументи довільної довжини.

***** спеціальний символ, що бере на себе усі об'єкти, які не увійшли в інші змінні при упаковуванні та віддає множину елементів при розпакуванні.

***args** – приймає у себе необмежену кількість неіменованих параметрів, пакуючи їх у кортеж

****kwargs** – приймає у себе необмежену кількість іменованих параметрів пакуючи їх у словник

!! *Порядок слідування* аргументів є важливим: спочатку позиційні аргументи, потім *args, а вже за ними **kwargs.

```
def func (positional, *args, **kwargs)
```

Використання ***** для процедури unpacking (розпакування):

- списку

```
a, *b, c = "st", 20, 30, 3.14, "Text", 1
print(a) # st
print(b) # [20, 30, 3.14, 'Text']
print(c) # 1
```

- рядку

```
*a, b, c = "Hello Tom"
print(a) # ['H', 'e', 'l', 'l', 'o', ' ', 'T']
print(b) # o
print(c) # m
```

- використання range

```
s = [4, 10]
print(list(range(1, 6))) # [1, 2, 3, 4, 5]
print(list(range(s))) # TypeError, список не ціле
print(list(range(*s))) # [4, 5, 6, 7, 8, 9]
```

Приклад 11. Використання аргументів довільної довжини:

```
def func(a, b, c, d):
    print(a, b, c, d)
a = ('hello', True, 78, [3,4,5])
```

```
func(*a) # hello True 78 [3, 4, 5]
def func(*args):
    print(sum(args)*0.06)
func(1, 33, 444) # 28.68
func(1, 4434, 4444, 6666) # 932.6999999999999
def func(**kwargs):
    print(kwargs)
func(a=1, b=2, c=3) # {'a': 1, 'b': 2, 'c': 3}
def func(*args,**kwargs):
    print(args)
    print(kwargs)
func(1,2,3,4,5, a=1, b=2, c=3) # (1, 2, 3, 4, 5)
# {'a': 1, 'b': 2, 'c': 3}
```

Map, filter, lambda.

Функція **map()** – застосовує задану функцію до кожного елемента ітерованого (списку, кортежу тощо) і повертає ітератор, що містить результати.

```
a = ['hello', 'abc', 'cat']
b = list(map(str.upper, a))
print(b) # ['HELLO', 'ABC', 'CAT']
```

Функція **filter()** – приймає іншу функцію перевірки з результатом типу Boolean до кожного елементу послідовності, відбираючи значення з True.

Функція **lambda** – анонімна функція без об'явлення, використовується для скорочення запису простих функцій. Особливості: кількість параметрів – будь яка, тіло містить лише 1 вираз, записується в одному рядку, може бути викликана негайно.

```
def sq(x):
    return x**2
a = [-2, -1, 5, 7]
b = map(sq, a)
print(b) # <map object>
c = list(map(sq, a))
print(c) # [4, 1, 25, 49]
# використовуючи filter
age = [11, 20, 3, 18, 40]
def is_adult(age):
    return age >= 18
f = filter(is_adult, age)
print(list(f)) # [20, 18, 40]
# використовуючи lambda функцію
is_adult = lambda age: age >= 18
f = filter(is_adult, age)
print(list(f)) # [20, 18, 40]
# скоротивши запис в 1 рядок
print(list(filter(lambda age: age >= 18, age)))
```

Область видимості.

В Python дві базових області видимості змінних:

- Глобальні змінні – оголошені поза будь-якої функції
- Локальні змінні – оголошені всередині тіла функції

Доступ до локальних змінних мають лише функції, в яких вони були оголошені, доступ до глобальних – можна отримати по всій програмі в будь-який функції.

Щоб змінити глобальну змінну всередині тіла функції – ключове слово global.

Ієрархія **LEGB**:

- Local – локальні змінні у рамках функції
- Enclosing – змінні, розташовані по ієрархії вище
- Global – змінні для усього проекту
- Bild-in – вбудовані об'єкти та змінні (print, list, …)

! Констант у Python не існує. Це глобальні змінні, що позначаються великими літерами (API_KEY)

```
a = 10 # глобальна змінна
API_KEY = '123qwe456' # константа
def nfunc(x):
    a = 7 # a - локальна змінна
    for i in range(x):
        n = i+1   # n - локальна змінна
        print(n)
print(n) # NameError: name 'n' is not defined
nfunc(5)
def nfunc(x):
    global a # змінюємо глобальну змінну у функції
    a = 7
    for i in range(x):
        n=i+1
    print(n)
print(a) # 10
nfunc(5)
print(a) # 7
```

Вкладені функції.

Вкладені функції – це означає, що всередині інструкції def може бути інша інструкція def. Кількість вкладень функцій довільна. В найбільш загальному випадку форма оголошення вкладених функцій наступна:

```
def Funk1(parameters1):
    def Funk2(parameters2):
        …
            def FunkN(parametersN):
                return
        return
    return
```

Приклад вкладеної функції:

```
name = 'Anna'
def print_1():
    print('Перша функція, ім\'я: ' + name)
```

```
        def print_2():
            name = 'alex'
            print('Друга функція, ім\'я: ' + name)
        print_2()
print_1()
# Перша функція, ім'я: Anna
# Друга функція, ім'я: alex
```

Декоратори.

Декоратор – функція обгортка, яка огортає іншу функцію у свій функціонал, без зміни основної функції.

```
def tagMaker(func):
    def wrapper(*args, **kwargs):
        print('<div>')
        res = func(*args, **kwargs)
        print('</div>')
        return res
    return wrapper
@tagMaker
def printText(text):
    print(text)
printText('Hello World')
```

Приклад 12. Визначити швидкість роботи функції.

```
import time
import datetime
def recTime(func):
    def wrapper(*args, **kwargs):
        start = datetime.datetime.now()
        func(*args, **kwargs)
        bone = datetime.datetime.now() - start
        print(f'Функція відпрацювала за {bone} сек')
    return wrapper
@recTime
def sfunc():
    time.sleep(3)
    print('Завершено')
sfunc()
#Завершено
#Функція відпрацювала за 0:00:03.015691 сек
```

Завдання до лабораторної роботи №3

1. По варіантах (обрати **одне** на вибір):

- Дано три дійсних числа a, b, c. Якщо a>b>c, кожне число збільшується удвічі, інакше кожне число зменшується на одиницю.

- Дано дві змінні цілого типу: A і B. Якщо їх значення не рівні, то присвоїти кожній змінній суму цих значень, а якщо рівні, то присвоїти змінним нульові значення. Вивести нові значення змінних A і B.
- Дано три цілих числа: A, B, C. Перевірити істинність висловлювання: «Число B знаходиться між числами A і C і є кратним 3».
- По введеному з клавіатури символу визначити, чи він розділовий знак?
- Дано два цілих числа: A, B. Перевірити істинність висловлювання: «Кожне з чисел A і B непарне їх різниця від'ємне число».
- Дано два цілих числа: A, B. Перевірити істинність висловлювання: «Хоча б одне з чисел A і B непарне»
- За введеним з клавіатури символом визначити, чи він відноситься до знаків арифметичних операцій.
- Дано ціле позитивне число. Перевірити істинність висловлювання: «Дане число є парним двозначним».
- Дано текстовий рядок. Якщо кількість символів рядка менше 15 до рядка додайте своє прізвище інакше надрукуйте назву групи.
- Дано двозначне число. Перевірити істинність висловлювання: «кожна цифра числа менша 6.
- За введеним з клавіатури значенням температури вивести: холодно, тепло чи спекотно на вулиці.
- Дано дві змінні цілого типу: A і B. Якщо їх значення не рівні, то присвоїти кожній змінній більше з цих значень, а якщо рівні, то присвоїти змінним нульові значення.
- Дано ціле число. Перевірити істинність висловлювання: «Дане число є двозначним і додатнім»
- За введеними координатами з'ясувати, до якої координатної чверті належить точка.
- За введеним з клавіатури значенням часу (ч.) надрукувати, до якої частини доби воно належить. Наприклад, 10 годин – ранок.
- Скласти програму, яка визначала б: яких оцінок більше отримано під час іспиту з Програмування "4" чи "5".

2. Дано список цілих чисел. Написати програму подвоєння елементів списку з використанням лямбда-функції та функції map().

3. Дано рядок з 20 слів. Використовуючи лямбда-функцію та функції map() всі слова записати з великої букви.

4. Розробити функцію яка приймає послідовність чисел a1, a2, ….an. Визначити в якій половині послідовності додатних елементів більше. Якщо n не парне то кількість елементів у першій половині заокруглити до більшого числа, другої половини до меншого.

5. На стадіоні є три категорії місць для сидіння: місця класу A коштують a грошових одиниць, місця класу B коштують b грошових одиниць, а місця класу C – c грошових одиниць. Напишіть першу функцію, яка запитує скільки продано квитків на кожний клас місць, і другу функцію, яка відображає суму отриманого доходу від продажу квитків на кожен клас окремо і загалом.

6. Напишіть декоратор, який вимірює швидкість роботи функції, при "читанні" кортежу та "читанні" списку. Оцінити, при яких значеннях N функція з перетворенням списку в кортеж працює швидше (теоретично кортеж 'читається' швидше, але перетворення теж йде час).

Тема 4. Робота з файлами.

Зміст розділу:
1. Файли.
2. Операції над файлами.
3. Робота з файлами.

Операції над файлами.

Існує кілька варіантів дій з файлами:
- відкриття (open);
- читання (read);
- запис (write);
- закриття (close).

Відкриття файлу.

Щоб почати роботу з файлом, його треба відкрити за допомогою функції **open().**

```
f = open(file, mode)
```

- Перший параметр – шлях до файлу.
- Другий аргумент – mode встановлює режим відкриття файлу в залежності від того, що з ним необхідно робити (Табл. 2).

Таблиця 2 – Режими відкриття файлу.

Режим	Можливості
r	Тільки читання (по замовчуванню)
w	Запис. Якщо не знайдений, то створюється новий
x	Запис. Якщо не знайдений, викликається виняток
a	Запис. Не стирає дані, а додає в кінець
t	Відкриття в форматі текстового файлу
b	Відкриття у вигляді бінарного файлу
+	Робота у варіанті і записи, і читання одночасно

Закриття файлу.

- метод **close()**, прописаний після всіх необхідних дій
- метод try/except (finally) – при появі операцій з винятками файл буде автоматично закритий
- інструкція with..as, яка спрощує обробку винятків, тому метод close() в цьому випадку буде не потрібен.

```
f = open('xyz.txt', 'r')
```

```
# fp = open('C:/xyz.txt','r')
f.close()
f = open('xyz.txt', 'r')
try:
# робота з файлом
except:
     f.close()

with open ('xyz.txt') as f:
# Робота з файлом
```

Читання файлу.

Для читання файлу він відкривається з режимом *r* (**Read**):

- read() – зчитує весь вміст файлу в один рядок
- read(n) – зчитує n байтів від положення маркера
- readline() – зчитує один рядок з файлу
- readlines() – зчитує всі рядки файлу в список

Приклад 13. Створіть директорію з назвою data в середині директорії, де знаходиться даний скрипт.

```
# Запис даних у текстовий файл
import os.path
# Модуль, який містить функції для роботи з шляхом у файловій системі
text = '''Hello! I am a text file. And I had been written with a Python script before you opened me, so look up the docs and try to delete me using Python, too.'''
def write_text_to_file(filename, text):
"""Функція для запису у файл filename рядка text"""   f          =
open(filename, "w")
     # Відкриття файлу для запису
     f.write(text) # Запис рядка text у файл
     f.close() # Закриття файлу
if                __name__                 ==              '__main__':
     write_text_to_file(os.path.join('data', \
                        'example01.txt'), text)

# Відкриття файлу для дозапису
import os.path
import datetime
text = '''This text was added in example 2!
Updated '''
if __name__ == '__main__':
     log_file = os.path.join('data', 'example01.txt')
     with open(log_file, 'a') as log:
          print('\n', text,str(datetime.datetime.now()),\
                    file=log)

# Відкриття файлу для зчитування
import os.path
```

```
def read_file(fname):
"""Функція для зчитування файлу fname та виведення його вмісту
на екран"""
     file = open(fname, 'r')
     # відкриття файлу для зчитування
     print('File ' + fname + ':')
     # виведення назви файлу
     # зчитування вмісту файлу по рядках
     for line in file:
          print(line, end='') # виведення рядка s

     file.close() # закриття файлу
if __name__ == '__main__':
# функція os.path.join з'єднує частини шляху у файловій системі
# необхідним роздільником
     read_file(os.path.join('data', 'example01.txt'))
```

Робота з файлами.

Копіювання файлу. Модуль **shutil** пропонує низку високорівневих операцій над файлами та колекціями файлів. Зокрема, передбачені функції, які підтримують копіювання та видалення файлів.

```
import shutil
     shutil.copyfile("C:\\mydoc.doc", "C:\\My
Documents\\mydoc_2.doc")
```

Перейменування файлу. Модуль **os** – забезпечує портативний спосіб використання залежних від операційної системи функцій. Для маніпулювання шляхами – модуль os.path; прочитати всі рядки в усіх файлах у командному рядку – модуль fileinput; для створення тимчасових файлів і каталогів перегляньте модуль tempfile.

```
import os os.rename("C:\\mydoc.doc\\testfile.txt",
"/home/user/test.txt")
```

Видалення файлу:

```
import os
os.remove("C:\\mydoc.doc \\testfile.txt")
```

Читання потрібного рядка з текстового файлу. Щоб прочитати рядок під певним номером – можна скористатися як стандартним читанням файлу в список, так і використовувати модуль **linecache:**

```
line = linecache.getline("C:\\boot.ini", 2)
# чи
line = open("C:\\boot.ini").readlines()
```

Перебір файлів у каталозі:

```
for filename in os.listdir("..\\plugins"):
     print(filename)
```

Перебір файлів у каталозі за маскою:

```
import glob
for filename in glob.glob("..\\plugins\\*.zip"):
     print(filename)
```

Порівняння файлів. Порівнювати файли можна як за змістом, так і за їх властивостями, що значно швидше, за допомогою **filecmp**:

```
import filecmp
similar = filecmp.cmp('C:\\file1.txt', 'C:\\file2.txt')
     print(similar)
```

Завдання до лабораторної роботи №4

1. Написати програму обчислення значень функції y=2*x-1 на проміжку від [-3,3] з кроком 0.5. Результати записати у файл.

2. Зчитайте текстовий рядок з файлу. Порахуйте скільки раз в ньому зустрічається символ введений користувачем з клавіатури. (Використайте цикл while).

3. Розробіть функції для здійснення наступних операцій зі списками:
   - Швидке сортування;
   - Пошук елементу за значенням;
   - Пошук перших п’яти мінімальних елементів;
   - Пошук середнього арифметичного;
   - Повернення списку, що сформований з початкового списку, але не містить повторів.

Списки зчитати з одного файлу з різних рядків. Результати записати в 1 файл з парам. «a».

4. Сформувати файл (або файли) у текстовому редакторі «Блокнот». Маємо текстовий файл (або файли).

Обрати один варіант:

- Переписати його рядки в інший файл. Порядок розташування рядків у другому файлі повинен: а) збігатися з порядком рядків в заданому файлі; б) бути зворотним по відношенню до порядку рядків в заданому файлі.
- Переписати його рядки в зворотному порядку (справа наліво) в інший файл. Порядок рядків у другому файлі повинен: а) збігатися з порядком рядків в заданому файлі; б) бути зворотним по відношенню до порядку рядків в заданому файлі.
- Отримати текст, в якому в кінці кожного рядка з заданого файлу доданий знак оклику.
- Переписати в інший файл ті його рядки, в яких є більше 30-ти символів.
- Переписати в інший файл всі його рядки з заміною в них символу «0» на символ «1» і навпаки.
- Всі парні рядки цього файлу записати в другий файл, а непарні в третій файл. Порядок проходження рядків зберігається.
- Два файли з однаковою кількістю рядків. Переписати зі збереженням порядку проходження рядки першого файлу в другий, а рядки другого файлу – в перший. Використовувати допоміжний файл.

- Два файли з однаковою кількістю рядків. З'ясувати, чи співпадають їх рядки. Якщо ні, то отримати номер першого рядка, в якому ці файли відрізняються один від одного.
- Вивести на екран: а) всі його рядки, які розпочинаються з літери «Т»; б) всі його рядки, які містять більше 30 символів; в) всі його рядки, в яких є більше трьох прогалин; г) всі його рядки, які містять в якості фрагмента заданий текст.
- Визначити: а) кількість рядків, що починаються з літер «А» чи «а»; б) в яких є рівно п‘ять літер «і».
- Визначити і вивести: а) довжину найдовшого рядка; б) номер найдовшого рядка (якщо таких рядків декілька, то номер першого із них); в) сам найдовший рядок (якщо таких рядків декілька, то перший із них).
- З'ясувати, чи є в ньому рядок, який розпочинається з літери «Т». Якщо так, то визначити номер першого з таких рядків.
- Визначити і вивести: а) перший символ першого рядка; б) п'ятий символ першого рядка; в) перші 10 символів першого рядка; г) перший символ другого рядка; д) k-й символ n-го рядка.
- У кожному рядку заданого файлу перші два символи є літерами. Вивести: а) слово, утворене першими літерами кожного рядка; б) слово, утворене другими літерами кожного рядка; в) послідовність символів, утворену 5-ми символами кожного рядка.
- Підрахувати кількість рядків у ньому.
- Підрахувати кількість символів в ньому.
- Підрахувати кількість символів в кожному рядку.
- Видалити з файлу третій рядок. Результат записати в інший файл.
- Видалити з файлу його останній рядок. Результат записати в інший файл.
- Видалити з файлу перший рядок, в кінці якого стоїть знак запитання. Результат записати в інший файл.
- Додати у файл рядок з дванадцяти рисок (----------), розмістивши їх: а) після п'ятого рядка; б) після останнього з рядків, в яких немає прогалини. Якщо таких рядків немає, то новий рядок необхідно додати після всіх рядків наявного файлу. В обох випадках результат записати в інший файл.
- Елементами файлу є числа. Видалити з нього п'яте число. Результат записати в файл.
- Елементами файлу є цілі числа. Всі парні числа записати в інший файл.
- Елементами файлу є окремі символи (цифри та літери). Всі цифри цього файлу записати в другій файл, а решта символи – в третій файл. Порядок слідування зберігається.
- Елементами файлу є окремі слова. Записати в інший файл слова, які розпочинаються на літеру «о» або «а».
- Елементами файлу є цілі числа. Видалити з нього число, записане після першого нуля (нулі у файлі обов'язково присутні). Результат записати в інший файл.
- Два текстові файли однакового розміру, елементами яких є числа. Отримати третій файл, кожен елемент якого дорівнює: а) сумі відповідних елементів заданих файлів; б) більшому із відповідних елементів заданих файлів.
- Два текстові файли однакового розміру, елементами яких є числа. Отримати третій файл, кожен елемент якого дорівнює: а) різниці відповідних елементів заданих файлів; б) меншому з відповідних елементів заданих файлів.
- Два текстові файли однакового розміру, елементами яких є окремі літери. Отримати третій файл, кожен елемент якого є поєднанням відповідних літер першого і другого файлів.
- Два текстові файли однакового розміру, елементами яких є окремі літери. Записати в третій файл всі співпадаючі елементи наявних файлів.

Тема 5. Модуль NumPy.

Зміст розділу:

1. Опис модуля NumPy.
2. Масиви.
3. Операції над масивами.

NumPy – це модуль з відкритим кодом для Python, що надає загальні математичні та числові операції в виді скомпільованих, швидких функцій. Вони об'єднуються у високорівневі пакети, які забезпечують функціональність, що порівнюється з можливостями MatLab.

NumPy (Numeric Python) надає базові методи для маніпулювання великими масивами і матрицями.

Швидкодія коду Python з використанням NumPy в 50 раз вища ніж коду написаному на «чистому» Python і порівняна з швидкодією комерційного пакету матричної алгебри MATLAB.

Головною особливістю NumPy є об'єкт array – масив, який подібний до списків Python. Однак з масивами можна виконувати числові операції з великими обсягами інформації в рази швидше і, головне, набагато ефективніше ніж із списками. Array приймає два параметри, список та тип масиву (Рис. 1).

```
import numpy as np
s = [1, 6, 8, 9, 10]
m = np.array(s)
print(m) # [ 1 6 8 9 10]
a = np.array([1, 5, 8, 9], float)
print(a) # [1. 5. 8. 9.]
print(type(a)) # <class 'numpy.ndarray'>
n = np.arange(0, 1, 0.2)
print(n) # [0. 0.2 0.4 0.6 0.8]
```

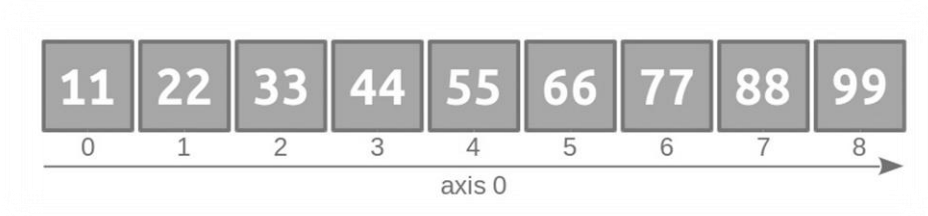

Рисунок 1 – Одновимірний масив.

Багатовимірні масиви. Двовимірний масив.

Перша вісь (індекс) – це рядки, друга вісь – це стовпці (Рис. 2).
Щоб отримати потрібний елемент , треба написати:
ім'я_масива[індекс_axis0, індекс_axis1].

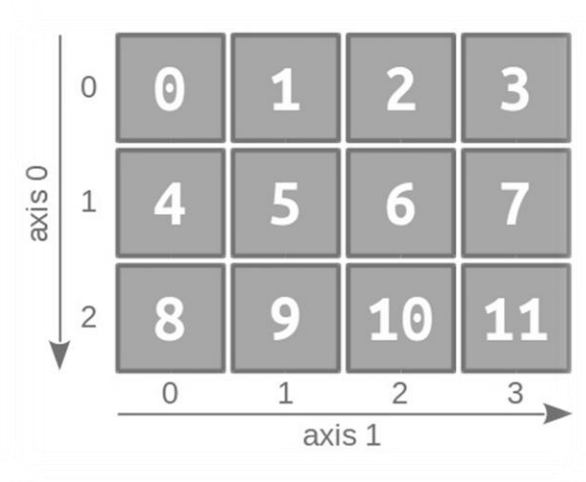

Рисунок 2 – Двовимірний масив.

Код, що описує створення двовимірного (Рис. 2) масиву:

```
a = np.array([[0, 1, 2, 3],
              [4, 5, 6, 7],
              [8, 9, 10, 11]])
print(a)
[[ 0 1 2 3]
[ 4 5 6 7]
[ 8 9 10 11]]
print(a[1, 1], a[2, 1], a[0, 3])  # 5 9 3
```

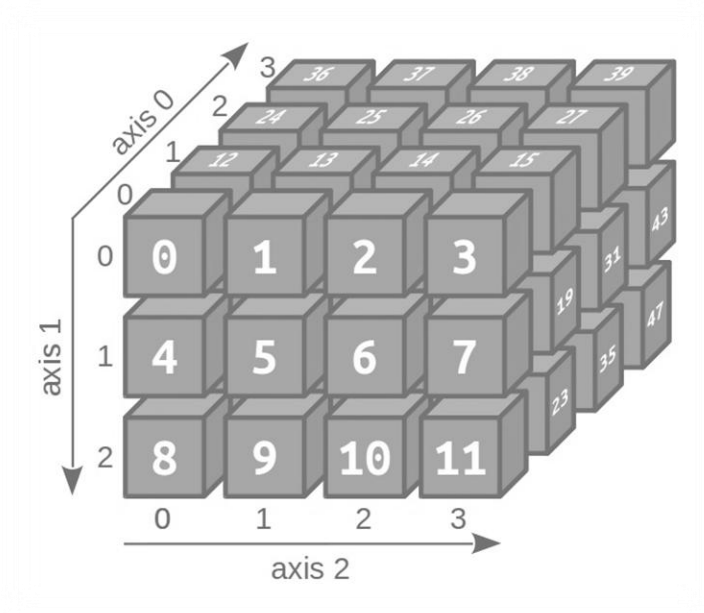

Рисунок 2 – Двовимірний масив.

Код, що описує створення тривимірного (Рис. 3) масиву:

```
a = np.arange(48).reshape(4, 3, 4)
print(a)
[[[ 0 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 31]
[32 33 34 35]]
[[36 37 38 39]
[40 41 42 43]
[44 45 46 47]]]
```

Заповнення масивів випадковими числами:

```
import numpy as np
one = np.random.rand(10)
# одновимірний масив випадкових значень
print(one)
two = np.random.rand(3, 4)
# двовимірний масив випадкових значень
print(two)
one_int = np.random.randint(10, size=7)
# одновимірний масив випадкових цілих чисел
print(one_int)
two_int = np.random.randint(10, size=(4, 4))
# двовимірний масив випадкових цілих чисел
print(two_int)
```

Для проходження по елементам масиву використовується конструкція:

```
for x in np.nditer(two):  # двовимірний масив
     print(x)
for y in two.flat: # двовимірний масив
     print(y)
for z in one: # одновимірний масив
     print(z)
for row in one:
     print('element', row) # друк по рядкам
```

numpy.nditer – ефективний багатовимірний об’єкт-ітератор для перебору масивів.

ndarray.flat – одновимірний ітератор по масиву. Це екземпляр numpy.flatiter, який діє аналогічно до вбудованого об'єкта ітератора, але не є його підкласом.

Для проходження по елементам у 2-мірному масиві, доцільно скористатися вкладеними циклами:

```
import numpy as np
A = np.array([9, 1, 7, 6, 3, 9, 5, 2, 0])
b = A.reshape(3, 3)
for x in b:
     for y in x:
          print('element', y)
```

Методи для роботи з масивами.

1. Зрізи (Array **slicing**) працюють з багатовимірними масивами аналогічно, як і з одновимірними, застосовуючи кожний зріз, як фільтр.
2. Метод **shape** повертає кортеж з кількістю рядків і стовбців у матриці.
3. Метод **dtype** повертає тип змінних, що зберігаються в масиві.
4. Метод **len** повертає кількість рядків у масиві.
5. Метод **in** використовується для перевірки на наявність елемента у масиві.
6. **Reshape** – переформатувати масив, що задає новий багатовимірний масив. Метод не модифікує оригінальний масив, а повертає новий.
7. Списки також можна створювати з масивів.
8. Можна ініціалізувати масив одним значенням для подальшої роботи з ним.
9. Можна також транспонувати масив, при цьому створюється новий масив.
10. Можна зчепити два і більше масивів за допомогою методу **concatenate**. Якщо ж масиви не одномірні то є можливість задати вісь по якій буде здійснюватися конкатенація. За замовчуванням конкатенація здійснюється за першим виміром.

```
import numpy as np
a = np.array([[1, 2, 3], [4, 5, 6]], float)
print(a, a[1, :], a.shape, a.dtype)
print(len(a))
print(2 in a, 0 in a)
b = np.array(range(12), int)
print(b, b.shape)
b = b.reshape((3, 4))
print(b, b.shape)
c = b.tolist()
print(c, type(c))
b.fill(0)
print(b, b.shape)
print(b.transpose(), b.shape)
d = b.flatten()
print(d, d.shape)
a = np.array([1, 2, 3], float)
b = np.array([9, 7, 5], float)
print(np.concatenate((a, b)), b.shape)
c = np.array([[1, 2, 3], [1, 2, 3]], float)
d = np.array([[9, 7, 5], [9, 7, 5]], float)
print(np.concatenate((c, d), axis=0))
print(np.concatenate((c, d), axis=1))
```

Математичні операції над масивами.

При виконанні стандартних математичних операцій над масивами – кількість рядків і стовпців першого масиву має бути рівною кількості рядків і стовпців другого масиву. Це означає, що масиви повинні бути однакового розміру при здійсненні бінарних математичних операцій над ними (додавання, віднімання, множення і тому подібне). Коли розмір одного масиву не співпадає з іншим – виникає виняток Exception.

В numpy також включена бібліотека стандартних математичних функцій, які можуть бути застосовані до кожного елементу до масиву: abs, sign, sqrt, log, log10, exp, sin, cos, tan, arcsin, arccos, arctan, sinh, cosh, tanh, arcsinh, arccosh і arctanh, пі та е.

Функції floor, ceil та rint повертають нижнє, верхнє і округлене значення.

```
import numpy as np
a = np.array([1, 2, 3], float)
b = np.array([1.1, 8.6, -57.5], float)
print(b - a)
print(a + b)
print(b / a)
print(a * b)
print(a % b)
print(b**a)
print(np.sqrt(a))
print(np.floor(b))
print(np.ceil(b))
print(np.rint(b))
```

Базові операції над масивами.

1. Елементи масиву можуть бути просумовані або перемножені за допомогою відповідних методів **sum** і **prod**.
2. Деякі функції надають можливість оперувати статистичними даними. Це такі функції, як середнє арифметичне (**mean**), дисперсія (**var**) і стандартне відхилення (**std**).
3. Можна знайти мінімальне (**min**) і максимальне (**max**) числове значення у масиві.
4. Функції **argmin** і **argmax** повертають індекс мінімального або максимального елемента у масиві.
5. Масиви можна відсортувати. Для цього є функція **sort**.
6. Унікальні елементи масиву можна отримати за допомогою функції **unique**.
7. Для двовимірного масиву діагональ можна отримати за допомогою функції **diagonal**.

```
import numpy as np
a = np.array([9, 2, 3, 9, 7, 8, 8, -5, 0], float)
print(a.sum())
print(a.prod())
print(a.mean())
print(a.var())
print(a.std())
print(a.min())
print(a.max()
print(a.argmin()
print(a.argmax())
a.sort()
print(a)
print(np.unique(a))
a = a.reshape(3, 3)
print(a)
print(a.diagonal())
```

Завдання до лабораторної роботи №5

1. Задано 2 одновимірних масиви. Виконайте з ними всі арифметичні операції. Використовуючи метод конкатенації (об'єднання) масивів створіть новий масив з двох попередніх і знайдіть максимальний, мінімальний елемент, суму елементів та їх добуток. Результати всіх операцій вивести на екран, додаючи коментарі.
2. Задано одновимірний масив із 15 елементів. Сформувати новий масив в якому кожен елемент заданого масиву зменшити на середнє значення та відсортувати отриманий масив за зростанням. Результати всіх операцій записати у файл.

3. Задано одновимірний масив з 20 елементів. Для ініціалізації використайте функцію random(). Перетворіть його у двовимірний. Кожен елемент масиву збільшити на 10. Результат виведіть у csv файл підписавши колонки (використовувати 1 рядок як шапку таблиці).
4. Задано двовимірний масив цілих чисел в діапазоні від -15 до 15. Створіть новий масив в якому всі числа менші 0 замініть на -1, більші 0 – на 1. Результати вивести на екран.

5. Застосуйте функції sort(), min(), sum(), mean() для двовимірного масиву розміром [4, 6]. Оцініть результат та опишіть його у звіті. Метод заповнення масиву даними виберіть на свій розсуд. Результати всіх операцій вивести на екран.

6. Напишіть програму NumPy для перетворення декартових координат на полярні координати випадкової матриці 10x3, що представляє декартові координати.

Тема 6. Модуль Matplotlib.

Зміст розділу:

1. Характеристика модуля Matplotlib.
2. Налаштування вигляду 2д графіків.
3. Панель інструментів.
4. Облік похибок error bars.
5. Графік розсіювання.
6. Графіки у полярних координатах.

Модуль Matplotlib.

Matplotlib – бібліотека на мові програмування Python для візуалізації даних двовимірною 2D графікою (3D графіка також підтримується).

- **plot()** – функція побудови графіку
- **show()** – функція показу графіку
- Функція **linspace**(start, stop, num=50, endpoint=True, retstep=False, dtype=None) – повертає одновимірний масив із зазначеної кількості елементів, значення яких рівномірно розподілені всередині заданого інтервалу:

start – число, початок послідовності.

stop – число, яке є кінцем послідовності, якщо endpoint = True. Якщо endpoint=False то дане число не включається до інтервалу, при цьому значення кроку між елементами послідовності змінюється.

num – ціле позитивне число (необов'язковий), за замовчуванням = 50. Визначає кількість елементів послідовності.

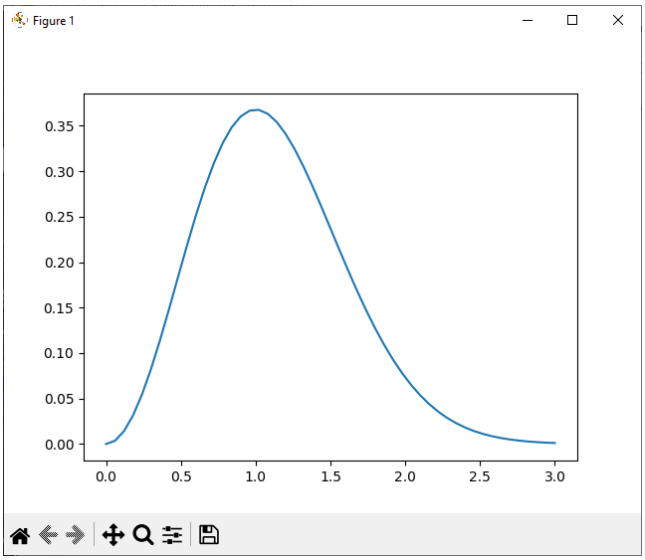

Рисунок 4 – Результат використання Linspace.

Linspace (Рис. 4).

```
import matplotlib.pyplot as plt
from numpy import *
def f(t):
    return t**2*exp(-t**2)
t = linspace(0, 3, 51) # 51 точка між 0 та 3
y = f(t)
```

```
plt.plot(t, y)
plt.show()
```

Налаштування вигляду графікі.

Модифікуємо зображення графіку з рисунку 4, додавши підписи осей абсцис *ylabel* та ординат *ylabel* , назви графіку *title* , та висвітлимо легенду *legend()*, задамо відображення *plot()* графіку функції *y* зеленим кольором пунктирною лінією *'g--'* (Рис. 5):

```
from numpy import *
import matplotlib.pyplot as plt
t = linspace(0, 3, 51)
y = t**2*exp(-t**2)
plt.plot(t, y, 'g--', label='t^2*exp(-t^2)')
plt.axis([0, 3, -0.05, 0.5])
# задання [xmin, xmax, ymin, ymax]
plt.xlabel('t') # позначення вісі абсцис
plt.ylabel('y') # позначення вісі ординат
plt.title('My first normal plot') # назва графіка
plt.legend() # вставка легенди (тексту в label)
plt.show()
```

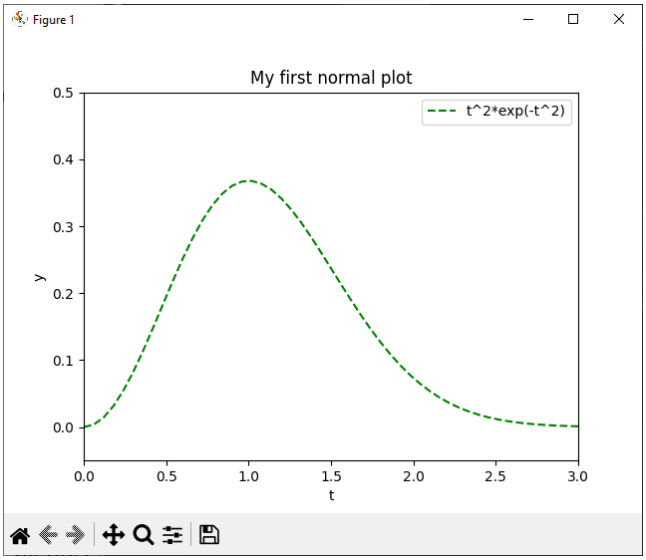

Рисунок 5 – Результат роботи коду.

Текстові параметри графіка наведені у Таблиці 3, іменовані параметри функції plot() – у Таблиці 4.

Таблиця 3 – Текстові параметри графіку.

title()	заголовок (назва) графіку
xlabel()	вісь абсцис
ylabel()	вісь ординат
grid()	підключення відображення сітки
legend()	легенда
text()	розміщення тексту на полі графіку

Таблиця 4 – Іменовані параметри функції відображення графіку.

Аргументи	Змінюваний параметр
color або c	Колір лінії
linestyle	Стиль лінії, використовуються позначення показані вище
linewidth	Товщину лінії (дійсне число)
marker	Вид маркера
markeredgecolor	Колір краю (edge) маркера
markeredgewidth	Товщину краю маркера
markerfacecolor	Колір самого маркера
markersize	Розмір маркера

Стовпчаста діаграма.

У matplotlib для реалізації стовпчастих діаграм використовується функція bar(), результат роботи програми зображено на Рисунку 6.

```
import matplotlib.pyplot as plt
import numpy as np
from numpy import *
color_rectangle = np.random.rand(10, 3) # RGB
index = np.arange(5)
values = [5, 7, 3, 4, 6]
plt.bar(index, values, color=color_rectangle)
plt.xticks(index+0.4, ['A', 'B', 'C', 'D', 'E'])
plt.show()
```

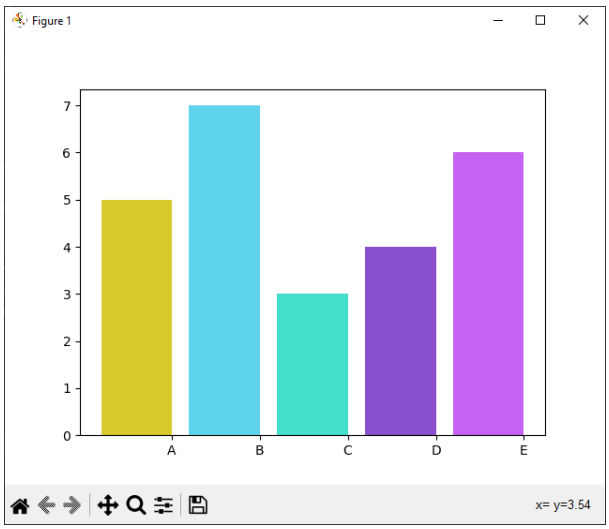

Рисунок 6 – Стовпчаста діаграма.

Як і лінійні графіки, стовпчасті діаграми широко використовуються для одночасного відображення великих наборів даних (Рис. 7). Але у випадку із багаторядними працює особлива структура.

Для групової діаграми необхідно реалізувати поділ простору, використовуючи індекс (для зручності його ширина дорівнює 1), на кількість стовпців, які до нього

відносяться. Також рекомендується додавати порожній простір, який виступатиме розподілом між категоріями.

```
import matplotlib.pyplot as plt
import numpy as np
index = np.arange(5)
values1 = [5, 7, 3, 4, 6]
values2 = [6, 6, 4, 5, 7]
values3 = [5, 6, 5, 4, 6]
bw = 0.3
plt.axis([0, 5, 0, 8])
plt.title('A Multiseries Bar Chart', fontsize=20)
plt.bar(index, values1, bw, color='b')
plt.bar(index+bw, values2, bw, color='g')
plt.bar(index+2*bw, values3, bw, color='c')
plt.xticks(index+1.5*bw, ['A', 'B', 'C', 'D', 'E'])
plt.show()
```

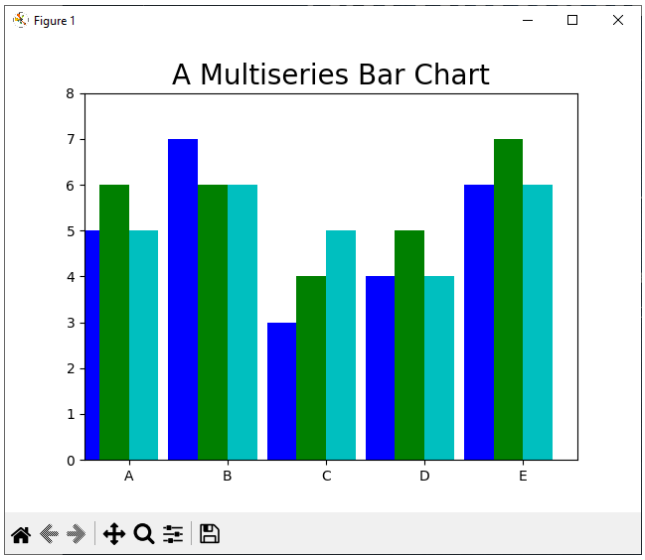

Рисунок 7 – Групова стовпчаста діаграма.

Збереження файлу.

Для збереження файлу зображення графіку використовують наступний код:

```
from numpy import *
import matplotlib.pyplot as plt
t = linspace(0, 1, 51)
y = t * exp(-t ** 2)
plt.plot(t, y)
plt.savefig('name_of_file.png')
```

Для кастомізації графіку, можна використовувати параметри розташування легенди в області відображення. У таблиці 5 наведено коди та текстові представлення команд розташування легенди.

Таблиця 5 – Розташування легенди.

Місце	String	Code
Найліпший варіант	best	0
Угорі праворуч	upper right	1
Угорі ліворуч	upper left	2
Унизу ліворуч	lower left	3
Унизу праворуч	lower right	4
Праворуч	right	5
Посередині ліворуч	center left	6
Посередині праворуч	center right	7
Посередині внизу	lower center	8
Посередині вгорі	upper center	9
Посередині	center	10

Панель інструментів.

Панель складається із семи кнопок:

- *будинок*, повертає оператора з будь-якого моменту перегляду до того вигляду, з якого починався перегляд зображення

- *стрілки*, дозволяють оператору переміщатися між виглядами, тобто на відміну від першої кнопки, що повертає програму виключно до першого вигляду, що не залежить від користувача, дають йому можливість порівнювати, наприклад, різні масштаби наближення до якоїсь точки

- *четверта* кнопка (із блакитним хрестом) має два можливі режими:

режим pan – натиснувши цю кнопку, а потім натиснувши в межах графіка ліву кнопку миші, користувач може переміщувати графік у межах вікна

- режим *zoom* – натиснувши праву кнопку миші, користувач може змінювати масштаб по горизонталі або вертикалі, рухаючись у відповідній площині праворуч або ліворуч, угору або вниз

- *п'ята* кнопка, дозволяє наближати або видаляти обрану область, відповідно виділяючи її лівою або правою кнопкою миші

- *шоста* кнопка, викликає меню налаштувань вікна

- *сьома* кнопка, дозволяє зберегти рисунок у зручному форматі.

Таблиця 6 – Гарячі клавіши панелі інструментів.

Гаряча клавіша	Які функції виконує
h	home, перша кнопка
c або ←	друга кнопка
v або →	третя кнопка
p	pan, четверта кнопка
o	п'ята кнопка
утримуючи x	pan та zoom тільки по горизонталі
утримуючи y	pan та zoom тільки по вертикалі
утримуючи Ctrl	кнопка для збереження пропорцій
g	додавання сітки
l	логарифмічна шкала

Облік похибок. Error bars.

У практичній науці експеримент навіть за максимальної точності вимірювань завжди вносить похибки. Для того, щоб врахувати це й указати можливий розкид навколо значення, яке вважається істинним, уводять error bars, які на кривій для отриманих точок показують своєрідний довірчий інтервал (Рис. 8).

```
import matplotlib.pyplot as plt
import numpy as
np x = np.arange(0, 4, 0.2)
y = np.exp(-x)
e1 = 0.1 * np.abs(np.random.randn(len(y)))
plt.errorbar(x, y, yerr=e1, fmt='.-')
plt.show()
```

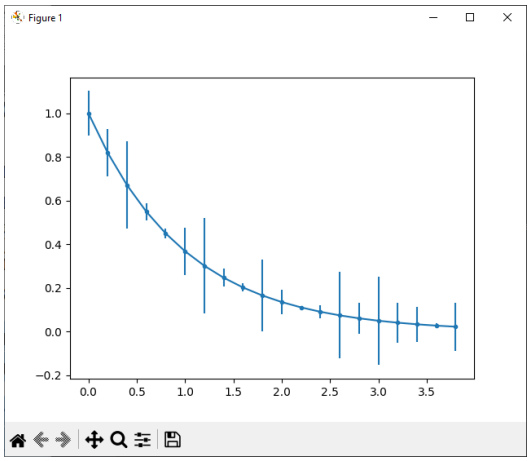

Рисунок 8 – Error bars.

Отже, за допомогою функції arange() створено масив точок x від 0 до 4 із кроком 0,2. Далі отримали масив точок y, відповідальних за значення функції exp(-x). Ці дії нічим не відрізняються від тих, що виконувалися раніше, коли будувалися графіки кривих. Наступна команда вносить випадкові помилки: визначається розмір масиву y, встановлюється для нього відповідний масив розподілених за гауссовим розподілом навколо нуля випадкових чисел, функція abs() бере від них модуль так, щоб усі числа були додатними, а також помножується на 0,1 таким чином, щоб ці помилки не були занадто великими для отримуваного графіка.

Далі видно, що одержана послідовність указується у функції errorbar() у вигляді аргументу yerr та послідовно накладається на отриманий графік y(x), вигляд якого визначається параметром fmt. Якщо помилки є не тільки для y-значень, але і для x, то використовується еквівалентний аргумент xerr. Відповідні "зарубки" матимуть вигляд хрестів.

Переданий аргумент fmt може мати значення None, тоді на екран виводяться тільки error bars, без графіка. Функція errorbar() також має інші іменовані параметри: ecolor та elinewidth, що визначають відповідно колір та товщину лінії, яка показує інтервал, capsize визначає ширину обмежувальних "кришечок" (у пікселях).

Розглянутий облік помилок є симетричним відносно істинної точки, але можливі й несиметричні відхилення. Їх можна вказати в тій самій функції за допомогою списку з двох послідовностей: *першої* – для від'ємних відхилень, *другої* – для додатних. Робиться це в такій формі:

plt.errorbar(x, y, yerr=[e1,e2], fmt='.-')

Графік розсіювання.

Графік розсіювання дозволяє зображувати одночасно дві множини даних, які не утворюють криву, а саме – двомірну множину точок. Кожна точка має дві координати. Графік розсіювання часто використовується для визначення зв'язку між двома величинами та дозволяє знайти більш точні межі вимірювань (Рис. 9).

Для створення таких графіків у модулі matplotlib.pyplot є своя функція **scatter().** Вона приймає дві послідовності та зображує їх на площині у вигляді маркерів (за замовчуванням, вони круглі та сині). Їх можна змінювати за допомогою іменованих параметрів тієї самої функції:

s визначає розмір маркерів і може бути як одним числом для всіх, так і містити масив значень;

c визначає колір маркерів (колір може бути одним для всіх або вказуватись множиною);

marker визначає тип маркера.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randn(1000)
y = np.random.randn(1000)
size = 50*np.random.randn(1000)
colors = np.random.rand(1000)
plt.scatter(x, y, s=size, c=colors) plt.show()
```

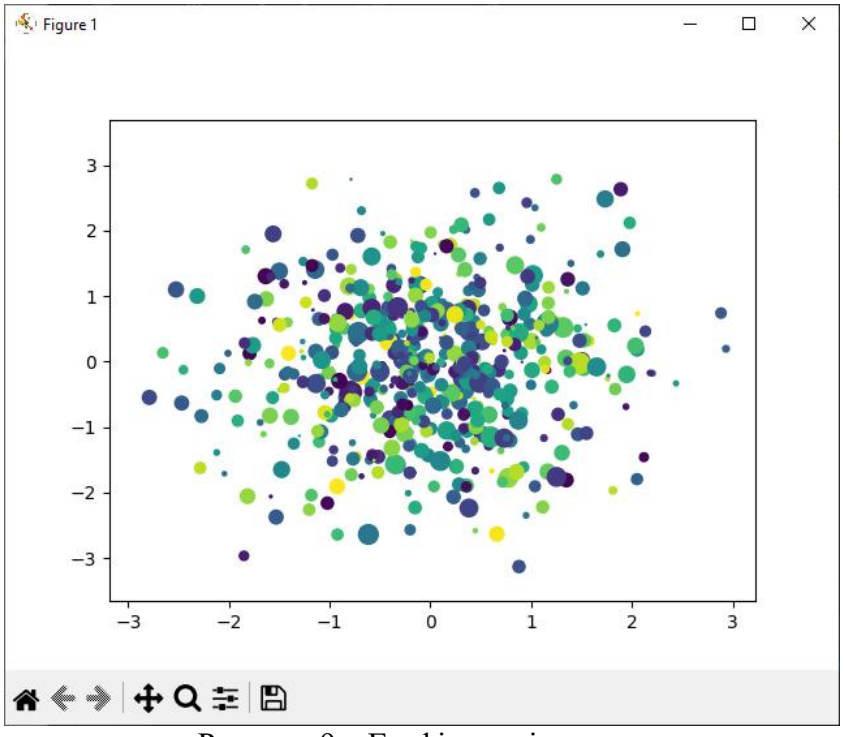

Рисунок 9 – Графік розсіювання.

Полярні координати.

Крім найбільш часто використовуваної декартової системи координат, досить широко застосовують полярну систему координат, яка є досить зручною при виконанні різних радіальних завдань. Координати точок у ній вказують за допомогою радіус-вектора ρ, що виходить із початку координат, та кута θ.

Кут можна вказувати в радіанах або градусах (Matplotlib використовує градуси). Для побудови графіків у полярних координатах використовують функцію **polar().** В аргументах першими йдуть кути, потім радіуси (Рис. 10).

```
import matplotlib.pyplot as plt
import numpy as np
theta = np.arange(0., 2., 1./180.)*np.pi # 360 точок
plt.polar(3*theta, theta/5) # спіраль
plt.polar(theta, np.cos(4*theta)) # рівняння квітки
plt.polar(theta, [1.4]*len(theta))# коло
plt.show()
```

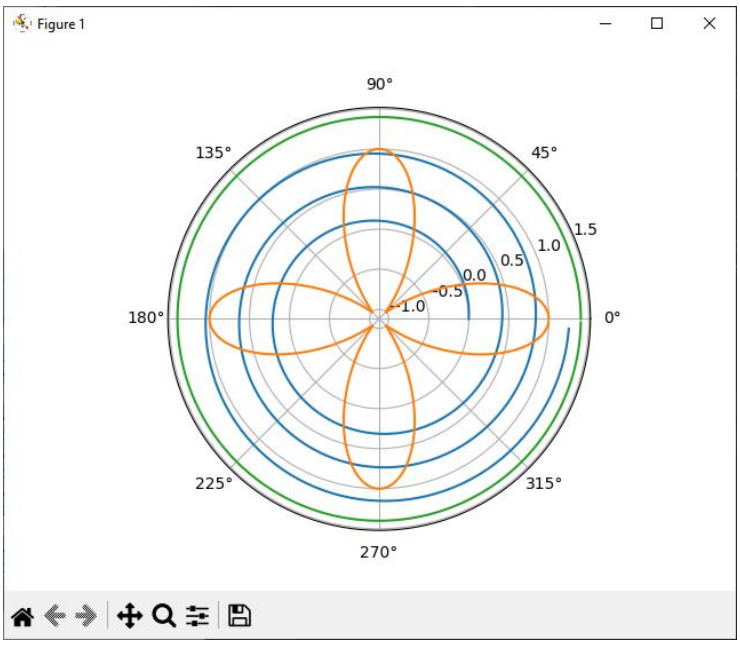

Рисунок 10 – Графік у полярних координатах.

Текст, примітки.

Крім тексту в назвах, підписах до осей, легенді, можна безпосередньо вставляти його в графік за допомогою простої функції **text**(x, y, text), де x і y координати, а text – рядок. Ця функція вставляє текст у позицію відповідно до вказаних координат.

Можлива вставка й у координатах графіка, в яких за (0, 0) беруть нижній лівий кут, а за (1, 1) – правий верхній, за допомогою функції **figtext**(x, y, text).

Текстові функції вставляють зазвичай текст у графік, але часто виникає потреба саме вказати, виділити якийсь екстремум, незвичайну точку. Для цього використовують примітки: **annotate**('annotation', xy=(x1, y1), xytext=(x2, y2)). Тут замість annotation пишеться текст примітки, замість (x1, y1) координати потрібної точки, замість (x2, y2) – координати місця, куди необхідно вставити текст.

Завдання до лабораторної роботи №6

1. Написати програму побудови лінійного графіку. Значення x задати у вигляді списку. Список значень у отримайте використовуючи функцію, яка збільшить значення x на 2. (можна вибрати інший варіант поділить на 1.5, x^2 , …). Задати параметри діаграми:

назву, підписи осей, легенду, сітку. Стиль графіку, колір та розмір текстів задайте самостійно.

2. Напишіть програму побудови графіків функцій y =|x|, y = x^3 , y = (1/2)x, де x[-6;6], на одному полі, задайте легенду розмістивши її в нижньому правому кутку, різний колір та стиль ліній графіків (https://matplotlib.org/stable/api/markers_api.html – маркери можна подивитись тут). Підписи осей відобразити червоним кольором, розмір шрифту – 14, стиль за бажанням. Назву (Графіки математичних функцій) відобразити синім кольором розмір шрифту 16, жирний. Отримане зображення зберегти у файл з розширенням .png і .pdf

3. Написати програму побудови графіків функції розміщуючи їх:
а) на одному полі; б) на різних полях однієї фігури (Функція supplot()).
Стилі графіків вибрати на свій розсуд. Використати **примітки** на графіках. Функцію обрати на з:

1) Y(x)=x*sin(5*x), x=[-2...5]; Y(x)=5*sin(1/x)*cos(x^2)^3, x=[-4...4]

2) Y(x)=3*sin(1/x)+cos(x^2)^2, x=[-6...6; Y(x)=1/x*sin(5*x), x=[-5...5]

3) Y(x)=2^x*sin(10x), x=[-3...3]; Y(x)=x^(1/2)*sin(10*x), x=[0...5]

4) Y(x)=15*sin(10*x)*cos(3*x), x=[-3...3]; (x)=5*sin(10*x)*sin(3*x), x=[0...4]

5) Y(x)=sin(10*x)*sin(3*x)/(x^2), x=[0...4]; Y(x)=5*sin(10*x)*sin(3*x)/(x^(1/2)), x=[1...7]

6) Y(x)=5*cos(10*x)*sin(x)/(x^(1/2)), x=[0...5]; Y(x)=cos(10*x)*sin(3*x)/(x^(1/2)), x=[0...10]

7) Y(x)=-5*cos(10*x)*sin(3*x)/(x^x), x=[0...5]; Y(x)=5*sin(10*x)*sin(3*x)/(x^x), x=[0...8]

8) Y(x)=x^sin(10*x), x=[1...10]; (x)=cos(10*x)*sin(3*x)/(x^(1/2)), x=[0...10]

9) Y(x)=x^cos(x^2), x=[0...10]; Y(x)=5*sin(10*x)*sin(3*x)/(x^x), x=[0...8]

10) Y(x)=10*cos(x^2)/x^2, x=[0...4]; Y(x)=(1/x)*cos(x^2+1/x), x=[1...10]

11) Y(x)=sin(x)*(1/x)*cos(x^2+1/x), x=[-2...2]; Y(x)=5*sin(x)*cos(x^2+1/x)^2, x=[1...10]

12) Y(x)=5*sin(1/x)*cos(x^2+1/x)^2, x=[1...4]; Y(x)=5*sin(1/x)*cos(x^2)^3, x=[-4...4]

13) Y(x)=(x^3)*cos(x^2), x=[-2...2]; Y(x)=(x^3)+cos(15*x), x=[-2...2]

14) Y(x)=(3^x)+cos(15*x), x=[-1...2]; Y(x)=10*cos(x^2)/x^2, x=[0...4]

15) Y(x)=sin(10*x)*sin(3*x)/(x^2), x=[0...4]; Y(x)=5*sin(10*x)*sin(3*x)/(x^(1/2)), x=[1...7]

4. Написати програму побудови стовпчикової гістограми частоти появи голосних літер у тексті (текст зчитується із текстового файлу, розмір тексту до 100 слів). Отриману гістограму зберегти у .png файл.

5. За даними таблиці написати програму побудови кругової діаграми. Для даної діаграми відобразити назву (взяти з заголовку таблиці), значення, легенду. Задати колір для фірми Грант зелений інші на сій вибір. Одну із частин розмістити зовні діаграми. Результат зберегти у файл.

Валовий дохід підприємств				
Полісся	Грант	Mipica	AT trading	Trade F
10%	45%	19%	11%	15%

6. Написати програму побудови групової горизонтальної гістограми за даними таблиці. Всі параметри гістограми мають бути задані. Стиль – за власним бажанням. Результат зберегти у файл.

Валовий дохід підприємств				
Полісся	Грант	Mipica	AT trading	Trade F
10%	45%	19%	11%	15%

Тема 7. Об’єктно-орієнтоване програмування у Python.

Зміст розділу:

1. Принципи об’єктно-орієнтоване програмування у Python.
2. Створення класу.
3. Наслідування та поліморфізм.

Об’єктно-орієнтоване програмування (ООП) – це технологія програмування, яка ґрунтується на понятті класів, об’єктів та успадкуванні елементів базових класів похідними класами.

Клас — це спеціальна конструкція, яка використовується для групування пов'язаних змінних та функцій.

Об'єкт — це екземпляр деякого класу об'єктів або просто класу.

Принципи ООП:

- *абстракція* це – представлення інформації у термінах його інтерфейсу з користувачем
- *інкапсуляція* – об’єднання даних та функцій (методів) їх опрацювання в єдиній інформаційній структурі, яка називається класом
- *успадкування* класів – імпорт властивостей базових класів у похідні класи
- *поліморфізм* – полягає у використанні однакових інтерфейсів для роботи з різними за функціональністю об’єктами

Створення класу.

Клас складається з оголошення (інструкція class), імені класу та тіла класу, що містить атрибути та методи.

class <ім'я_класу> :
<тіло класу>

Загальний вигляд команди створення екземпляра (об'єкта) класу:

ім’я_об’єкта = ім’я_класу()

Класи збирають в собі набори даних (змінних) разом з наборами функцій, що на них діють. Мета полягає в тому, щоб досягти більш модульного коду за допомогою групування змінних і функцій, в невеликі вузли, що легко модифікувати.

За замовчуванням у Python для посилання на об'єкт використовується ім'я **self**. Змінна self зв'язується з об’єктом, до якого було застосовано даний метод, і через цю змінну отримуємо доступ до атрибутів об'єкта. Коли цей же метод застосовується до іншого об'єкта, то self зв'яжеться вже з саме цим іншим об'єктом, і через цю змінну будуть викликатись тільки його поля.

```
class Adder:
      n = 1
      def add(self, v):
            return v + self.n
a = Adder()
b = Adder()
a.n = 10
print(a.add(3)) # 13 print(b.add(4)) # 5
```

Тут від класу Adder створюється два об'єкта – a та b. Для об'єкта a заводиться власне поле n. Об'єкт b, не має такого поля, отже успадковує n від клас Adder. У методі add() вираз self.n – це звернення до поля n, переданого об'єкта, і не важливо, на якому рівні наслідування його буде знайдено.

Необхідність конструкторів пов'язана з тим, що часто об'єкти повинні мати власні властивості одразу. Припустимо маємо клас Person, об'єкти котрого обов'язково повинні мати ім'я. Якщо клас буде описано наступним способом:

```
class Person():
     def set_name(self, name)
          self.name = name
#то створення об'єкта можливе без полів. Для встановлення імені
метод set_name() необхідно викликати окремо:
p1 = Person()
p1.name = 'Jane Doe'
print(p1.name) # 'Jane Doe'
#Наявність конструктора не дозволить створити об'єкт без полів:
class Person():
     def __init__(self, name):
          self.name = name
p1 = Person('Jane Doe')
print(p1.name) # 'Jane Doe'
```

Тут при виклику класу в круглих дужках передаються значення, котрі будуть присвоєні параметрам метода init(). Перший параметр – self – посилання на сам щойно створений об'єкт.

Розглянемо приклад створення власних класів та методів.

Як приклад опишемо за допомогою класу, коментар із сайту, коментар матиме 3 атрибути: дата написання, кількість вподобайок та текст коментаря. А як функції, які в класах називаються методами, будемо виводити коментар у консоль і редагувати.

```
class CommentFromWebsite():
"""Коментарі з сайту"""
     def __init__(self, data, text, likes):
          self.data = data
          self.text = text
          self.likes = likes

# Створюэмо екземпляр класу
new_coment = CommentFromWebsite('05/08/2021', 'Перший коментар',
'100')
print(type(new_coment))
# <class '__main__.CommentFromWebsite'>
print(new_coment.data) # 05/08/2021 print(new_coment.text) #
Перший коментар
```

Створимо клас. Для цього використовуємо ключове слово клас. Далі йде назва класу. Назву прийнято записувати, в CamelCase, далі дужки та двокрапка. Не обов'язково, але можна писати документацію, тобто опис класу – це робиться так само, як функції – три подвійні лапки спочатку та наприкінці опису.

Тепер потрібно здійснити ініціалізацію класу. Ініціалізація – це функція, яка свідчить, що клас створено з певними атрибутами. Іншими словами, конструктор класу. Метод __init__ потрібний для того, щоб була можливість прописати атрибути класу в момент створення класу. Можна обійтися без конструктора, записавши атрибути класу як змінні. Метод __init__ виконується щоразу під час створення нового екземпляру класу, та відповідає за базовий функціонал під час створення об'єкта. Це зарезервоване ім'я методу у системі. Він визначається одним обов'язковим параметром self і параметрами атрибутів, яких може бути безліч.

Пописуємо self, а також дату, текст та лайки – це будуть атрибути нашого класу, тобто ці змінні прийматимуть функції ініціалізації екземпляра класу __init__ для створення класу. Атрибут self – це посилання на екземпляр класу, який вказує на методи, та означає, що робота йде над екземпляром класу. Це обов'язковий параметр у визначенні методу, і його підписують не тільки у __init__, а й у всіх методах класу.

Далі, використовуючи посилання на об'єкт self, прописуємо параметри, що ведуть в функцію init, перевизначаючи їх. Використовую таку конструкцію: self ТОЧКА Атрибут = вхідний параметр (ім'я атрибута та вхідного параметра – однакові). Дещо дивна конструкція, і це пов'язано з тим, що мова Python часто не має строгих правил для написання різних конструкцій і базується на договорі між програмістами. І тут саме такий випадок. Перший параметр self іменується за домовленістю, як і перевизначення вхідних параметрів за допомогою конструкції: self.атрибут = атрибут. Тепер при створенні екземпляру класу передаватимемо лише ті параметри, які потрібні, self передавати не потрібно.

Спробуємо створити екземпляр класу. Нехай це буде new_coment, пишемо ім'я класу та у дужках передаємо атрибути: дата, текст, кількість лайків. Наступним рядком виводимо тип цього об'єкта. Як бачимо елемент класу комент. Так як клас має атрибути – можемо прочитати їх, звернувшись до об'єкта через точку, передавши атрибут. Так само можемо прочитати й інші атрибути об'єкта.

```
new_coment.text = 'Новий коментар!'
print(new_coment.text) # Новий коментар!
new_coment2 = CommentFromWebsite('24/02/2022', 'Другий коментар',
'800')
print(new_coment2.text) # Другий коментар
```

Отже, якщо можемо звернутися до атрибута об'єкта, щоб його прочитати, значить, можемо змінити значення, переназначивши його. Звернімося до атрибуту об'єкта і надамо нове значення. Створимо другий екземпляр класу. Так як є клас, що є конструктором, можемо створити необмежену кількість екземплярів класів за допомогою конструктора. Але кількість дій, які хочемо зробити з об'єктом, обмежена завданням. Тому всі необхідні функції можемо писати в самому класі та застосовувати їх до будь-якого екземпляра класу.

```
class CommentFromWebsite():
"""Коментарі з сайту"""
    def __init__(self, data, text, likes):
        self.data = data
        self.text = text
        self.likes = int(likes)

    def showComment(self):
    """Вивести коментар у консоль"""
        print(f'\nКоментар з сайту,\nдата: {self.data},'
          f'\nтекст коментаря: {self.text},'
          f'\nвподобайок: {self.likes}')
```

```
    def changeLikes(self):
    """Додаємо лічильник вподобайок"""  # int у self
        self.likes = self.likes + 1
    def changeComment(self, new_text):
    """Змінити текст коментаря"""
        self.text = new_text
```

Пропишемо третій методи для виведення всієї інформації в консоль, збільшення кількості лайків та зміни тексту. Метод – це звичайна функція, правила написання якого такі ж, як і функції, але він відноситься до класу. Створюємо def з обов'язковим параметром посилання на об'єкт self, коментарі, та функцію print з форматуванням рядка.

```
# Створюємо екземпляр класу
new_coment = CommentFromWebsite('05/08/2021', 'Перший коментар',
'100')
new_coment2 = CommentFromWebsite('24/02/2022', 'Другий коментар',
'800')
new_coment.showComment()
new_coment2.showComment()
new_coment2.changeLikes()
new_coment2.showComment() new_coment.changeComment('Новий!!!')
new_coment.showComment()
Коментар з сайту, дата: 05/08/2021, текст коментаря: Перший
коментар, вподобайок: 100
Коментар з сайту, дата: 24/02/2022, текст коментаря: Другий
коментар, вподобайок: 800
Коментар з сайту, дата: 24/02/2022, текст коментаря: Другий
коментар, вподобайок: 801
Коментар з сайту, дата: 05/08/2021, текст коментаря: Новий!!!,
вподобайок: 100
```

Як бачимо, замість змінних, у фігурних дужках йде конструкція посилання self.атрибут – тому що потрібно звернутися саме до об'єкта класу, а не змінних усередині класу. Якщо просто написати дату, то інтерпретатор не знайде таку змінну. Запис self.дата ідентично запису – new_coment.data.

Наступна функція – зміна лічильника лайків. При виклику кількість лайків буде збільшено на один. Для коректної роботи функції значення лайків необхідно примусово перевести в цілочисельний тип даних. Функція зміни тексту крім обов'язкового параметра self, прийматиме параметр нового тексту.

Тепер, використовуючи точковий запис, можемо звернутися до будь-якого методу об'єкта. Візьмемо перший метод, який виводить у консоль весь коментарі, пишемо ім'я екземпляра класу ТОЧКА і метод, так само, як викликали функції.

Далі застосуємо 2-й спосіб, збільшуємо кількість лайків, потім виведемо коментар повністю.

Наступний метод – змінює коментар. Якщо просто запустити метод без аргументів буде помилка, так як очікується обов'язковий аргумент текст. Незалежно від того, скільки в класі методів, можна звернутися до методу через точку до будь-якого з них і він спрацює з будь-яким екземпляром класу.

Наслідування та поліморфізм.

Працюючи над новим класом необов'язково складати його з нуля у разі, коли новий клас та його об'єкти схожі з об'єктами первинного класу. Розглянемо приклад.

Є клас машин з атрибутами, бренд, модель, рік випуску та пробіг. Метод showCar водить всі атрибути машини. Метод drowCar – додає пройдені кілометри до показника пробігу.

Створюємо екземпляр CarsClassз з параметром s_car = CarsClass('Mercedes', 'GLS', '2022', 10), виводимо усю інформацію про машину. Тепер віртуально «поїдемо» цією машиною 100 км. Для цього метод drowCar передаємо значення 100. Ще раз виведемо інформацію. Клас підходить до абсолютно будь-якого автомобіля, так як у кожної машини є бренд, рік випуску та пробіг.

```
class CarsClass():
"""Клас автомобілей"""
     def __init__(self, brand, model, year, probig):
     """Ініціалізація атрибутів"""
          self.brand = brand
          self.model = model
          self.year = year
          self.probig = int(probig)
     def showCar(self):
     """Показати інформацію про машину"""
          print(f'{self.brand}, {self.model}, {self.year} рік,
{self.probig} km')
     def drowCar(self, km):
     """ Метод поїздки на авто """
          self.probig = self.probig + km
s_car = CarsClass('Mercedes', 'GLS', '2022', 10)
s_car.showCar()    #    Mercedes,    GLS,    2022    рік,    10    km
s_car.drowCar(100)
s_car.showCar() # Mercedes, GLS, 2022 рік, 110 km
```

Якщо хочемо зробити новий клас електромобілі з атрибутом показника заряд акумулятора, то просто додати атрибут батареї цьому класу неможна (не у всіх авто є батарея). Треба створити новий клас для електромобілів, і найголовніше, так як електромобіль це теж автомобіль і має ті ж атрибути – не обов'язково копіювати клас автомобіля, а можна провести успадкування атрибутів з методів CarsClass у клас електромобіль, доповнивши своїми атрибутами.

```
class ElectroCar(CarsClass):
"""Клас електрокарів, ініціалізація атрибутів"""
     def __init__(self, brand, model, year, probig):
          super().__init__(brand, model, year, probig)

tesla = ElectroCar('Tesls', 'T', '2017', 10000) tesla.showCar()
# Tesls, T, 2017 рік, 10000 km
```

Застосування методів і атрибутів одного класу в іншому називається **успадкування**, той хто приймає – нащадок, а той хто передає – батьківський клас. Створюємо клас електромобіль, який наслідуємо від основного класу CarsClass – ElectroCar. Для того, щоб успадкувати всі атрибути та методи батьківського класу, потрібно вказати у дужках, від

кого успадковуємо (CarsClass). Після двокрапки описується клас. Є можливість доповнити конструктор (базовий метод init), переписавши його під потреби нового класу. Передамо self обов'язково, і також приймаємо всі параметри батьківського класу (через кому). Далі потрібно використовувати конструкцію super() з усіма атрибутами батька.

Функція super() – це спеціальна функція, що допомагає пов'язати нащадка та батька, внаслідок чього, дочірній клас електро отримує всі атрибути класу батька.

Створимо новий екземпляр класу електрокар, використовуючи атрибути, які успадкували від батька, нехай це буде Тесла. Далі ініціалізуємо екземпляр класу, звертаючись до дочірнього класу електрокар. Передаємо в нього ті самі параметри, які доступні і в батьківському класі і в класі-спадкоємцю. Створюємо екземпляр електрокару, та викликаємо метод класу батька до екземпляра дочірнього класу (наприклад, showCar).

Наслідування заощадило місце та час, що дозволило автоматично успадкувати методи за допомогою декількох рядків коду.

```
class ElectroCar(CarsClass):
"""Клас електрокарів, ініціалізація атрибутів"""
     def __init__(self, brand, model, year, probig):
          super().__init__(brand, model, year, probig)
               self.battery = 100 # потім видалити
     def description_battery(self):
     """Вивід інформації стосовно батареї"""
          print('Це авто має потужність'+str(self.battery)+'%')
tesla = ElectroCar('Tesls', 'T', '2017', 10000)
tesla.description_battery()
tesla.showCar() # Це авто має потужність 100 %
s_car.description_battery()
s_car.showCar()
#   AttributeError:   'CarsClass'   object   has   no   attribute
'description_battery'
```

Тепер розширимо функціонал класу ElectroCar, додавши до неї специфічні функції електромобілів. Нехай це буде ємність батареї. В init додаємо значення батареї за замовчуванням (100). Так як self.battery не передається в аргументах методу, цей атрибут буде доданий до екземпляра автоматично з параметром 100. Створимо метод description_battery(), що виводить ємність батареї.

Викличемо метод, який буде вводити повідомлення про стан батареї. Цей метод не працюватиме з основного класу Car, він не універсальний і працює тільки для нащадка. У такий спосіб працює успадкування.

```
class ElectroCar(CarsClass):
"""Клас електрокарів, ініціалізація атрибутів"""
     def __init__(self, brand, model, year, probig):
          super().__init__(brand, model, year, probig)
          self.battery = 100 # потім видалити
     def description_battery(self):
     """Вивід інформації стосовно батареї"""
          print('Це авто має потужність'+str(self.battery)+' %')

     def showCar(self):
     """Показати інформацію про машину"""
          print(f'{self.brand}, {self.model}')
tesla = ElectroCar('Tesls', 'T', '2017', 10000)
```

```
tesla.description_battery()
# Це авто має потужність 100 %
tesla.showCar() # Tesls, T
```

Перевизначення методів класу батька у нащадка. Для цього необхідно створити метод з такою самою назвою. І далі змінити його поведінку, наприклад, видалити з виводу рік і пробіг. Тоді звертаючись до об'єкта класу ElectroCar з методом showCar – бачимо лише бренд та модель. При цьому об'єкт класу Car працює з методом класу Car та видає повну інформацію. Тобто, якщо перевизначаємо батьківський метод в його нащадку – ігнорується метод батька, а береться метод, який знаходиться в нащадку. Це є поліморфізм, зміна функціоналу у дочірніх класах.

Використання об'єктів одних класів як атрибути інших класів.

Ця можливість дозволяє дробити та сегментувати код, розділяючи класи з великою кількістю атрибутів та методів на дрібніші, але логічно пов'язані класи.

У прикладі про автомобілі, був введений показник батареї у клас ElectroCar. Але логіка поведінки батареї може розвиватись, і в якийсь момент обсяг коду вимагатиме винесення всіх атрибутів і методів, пов'язаних тільки з батареєю в окремий клас. Тоді для взаємодії є два шляхи, або множинне спадкування, або використання екземпляру класу як атрибуту.

Подивимося, як поводиться екземпляр класу, створимо новий клас Battery():

```
class Battery():
"""Клас батареї"""
    def __init__(self, battery = 100):
        self.battery = battery
    def description_battery(self):
    '''Виводить інформацію про батарею'''
        print('Це авто має потужність ' +
str(self.battery) + ' %')

class ElectroCar(CarsClass):
"""Клас електрокарів"""
    def __init__(self, brand, model, year, probig):
        super().__init__(brand, model, year, probig)
        self.battery = Battery()
    def showCar(self):
    """Показати інформацію про машину"""
        print(f'{self.brand}, {self.model}')
tesla = ElectroCar('Tesls', 'T', '2017', 10000)
tesla.showCar() # Tesls, T
akum = Battery()
akum.description_battery()
# Це авто має потужність 100 %
```

Це самостійний клас, який нічого не успадковує, ініціалізуємо його і перенесемо значення, яке передавали за замовченням. Переносимо методи, що належать до цього класу. Даний клас у коді повинен бути обов'язково вищим за місце використання екземпляру. У класі ElectroCar аргументам прописуємо параметри Battery. Запис self.battery = Battery(), ідентичний створенню екземпляру класу. При ініціалізації ElectroCar буде створено екземпляр класу Battery() і наданий аргументу у класі ElectroCar. Тобто як атрибут будь-якого класу можемо використовувати об'єкт іншого класу.

Щоб звернутися до значення Battery(), можемо використовувати конструкцію: akum.description_battery(). Звертаємося до екземпляру одного класу, до його атрибуту, який у свою чергу сам є об'єктом іншого класу, а далі звертаємося до методів, доступних іншому класу. Таким чином, можемо використовувати екземпляри одного класу, як атрибути в іншому класі.

Також важливо відзначити, що класи у Python підтримують множинне спадкування – коли при створенні одного класу використовуються два класи батьків. Дочірній клас може прийняти атрибути класу батька. Але є важливий момент, потрібно враховувати перетин атрибутів батька, щоб уникнути випадкового перевизначення. Наприклад, якщо у одного з батьків буде такий же, як і у другому, і в такому випадку краще перевизначити батьківські атрибути безпосередньо в дочірньому.

Завдання до лабораторної роботи №7

1. Оголосіть клас Point3D для точок із трьома координатами x, y, z. Створіть кілька екземплярів цього класу і через них виведіть у консоль значення x, y, z. Далі, зробіть такі маніпуляції: поміняйте будь-яке значення координати в класі Point3D і подивіться як це вплине на відображені величини екземплярів класу; видаліть координату z у класі Point3D і переконайтеся, що вона буде відсутня у всіх примірниках. Поміняйте координату в будь-якому екземплярі класу і подивіться на результат.

2. Створіть клас Student. Опишіть метод name що визначає ПІБ студента, метод marks який визначає оцінки з дисциплін і метод average_marks що обчислює середнє значення.

3. Створіть клас Triangle, що буде містити дані про трикутник. Опишіть методи обчислення периметра та площі цього трикутника. Клас Triangle, буде містити три поля – довжини сторін трикутника – a, b, c. Опишіть метод perimeter, що визначає периметр трикутника та метод square, що обчислює площу зазначеного трикутника використовуючи формулу Герона. (використати правило перевірки існування трикутника).

4. Напишіть клас з назвою Circle, який містить два методи: для обчислення площі круга та довжину кола за введеним радіусом.

5. Напишіть клас Car, який надає для створених екземплярів такі атрибути даних автомобіля: марку виготовлення автомобіля, модель автомобіля, рік автомобіля, швидкість (початкове значення 0). Клас також повинен мати наступні методи: accelerate (метод повинен щоразу додавати 10 до значення атрибуту даних про швидкість), brake (метод повинен віднімати 10 від значення атрибута даних швидкості кожного разу, коли він викликається), get_speed (метод повинен повернути поточну швидкість). Створіть екземпляр класу Car і викличте метод accelerate шість разів. Після кожного виклику методу accelerate отримайте поточну швидкість автомобіля і надрукуйте її значення. Потім викличте метод brake п’ять разів. Після кожного виклику методу brake отримайте поточну швидкість автомобіля та надрукуйте її значення.

6. Напишіть програму, в якій є головний клас з текстовим полем. У головному класі може бути метод присвоювання значення полю: без аргументів і з одним текстовим аргументом. Об'єкт головного класу створюється передачею текстового аргументу конструктору. За підсумками головного класу створюється класу нащадок. У класі-нащадці потрібно додати числове поле. У конструктора класу-нащадка два аргументи.

Тема 8. GUI. PyQt.

Зміст розділу:

1. Модуль PyQt.
2. Компоненти графічного інтерфейсу користувача.
3. Робота з меню.

Будувати графічний інтерфейс користувача Graphical User Interface (GUI) для програм мовою Python можна за допомогою відповідних бібліотек компонентів графічного інтерфейсу або, використовуючи кальку з англійської, бібліотек віджетів.

Майже всі сучасні графічні інтерфейси загального призначення будуються за моделлю WIMP – Window, Icon, Menu, Pointer (вікно, іконка, меню, покажчик). Усередині вікон рисуються елементи графічного інтерфейсу, які для стислості називатимемо віджетами (widget – штука).

PyQt — це бібліотека, яка дозволяє використовувати фреймворк Qt GUI з Python. Сам Qt написаний мовою C++. Використовуючи його з Python, ви можете створювати програми набагато швидше, не жертвуючи при цьому швидкістю.

Перетворення на «.py»

Розроблена форма графічного інтерфейсу зберігається у файлі із розширенням *.ui. Для редагування такого файлу через PyQt 5 його спочатку необхідно перетворити на файл ".py".

pip install pyqt5
pip install pyqt5-tools

Перейдіть до папки, де збережено файл із розширенням ".ui". Виконайте команду:

pyuic5 -x [filename].ui -o [filename].py

чи

python -m PyQt5.uic.pyuic -x [filename].ui -o [filename].py

Далі у вас буде Python файл, який можна редагувати через будь-який текстовий редактор

PushButton, QCheckBox, QComboBox, QLabel and QSlider widgets.

Qt постачається з великим вибором доступних віджетів і навіть дозволяє створювати власні власні та налаштовані віджети.

Спочатку давайте розглянемо деякі з найпоширеніших віджетів PyQt. Наступний код створює діапазон віджетів PyQt і додає їх до макета вікна, щоб ви могли бачити їх разом.

```
import sys
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QApplication, QCheckBox, QComboBox,
QDateEdit, QDateTimeEdit, QDial, QDoubleSpinBox, QFontComboBox,
QLabel, QLCDNumber, QLineEdit, QMainWindow, QProgressBar,
QPushButton, QRadioButton, QSlider, QSpinBox, QTimeEdit,
QVBoxLayout, QWidget
# Підклас QMainWindow для налаштування головного вікна програми

class MainWindow(QMainWindow):
    def __init__(self):
```

```
            super().__init__()
            self.setWindowTitle("Widgets App")
            layout = QVBoxLayout()
            widgets    =    [QCheckBox,    QComboBox,    QDateEdit,
     QDateTimeEdit,    QDial,    QDoubleSpinBox,    QFontComboBox,
QLCDNumber,   QLabel,   QLineEdit,   QProgressBar,   QPushButton,
QRadioButton, QSlider, QSpinBox, QTimeEdit]
            for w in widgets:
                 layout.addWidget(w())
            widget = QWidget()
            widget.setLayout(layout)
# Встановити центральний віджет вікна. За замовчуванням віджет
розшириться, щоб зайняти весь простір у вікні.
            self.setCentralWidget(widget)
app = QApplication(sys.argv)
window = MainWindow()
window.show()
app.exec()
```

Компоненти графічного інтерфейсу користувача наведені на Рисунку 11:

- QCheckBox – Прапорець.
- QComboBox – Розкривний список.
- QDateEdit – Для редагування дати.
- QDateTimeEdit – Для редагування часу і дати.
- QDial – Поворотний диск.
- QDoubleSpinbox – Цифровий лічильник для поплавців.
- QFontComboBox – Список шрифтів.
- QLCDNumber – LCD-дисплей.
- QLabel – Просто мітка, не інтерактивна.
- QLineEdit – Введіть рядок тексту.
- QProgressBar – Індикатор виконання.
- QPushButton – Кнопка.
- QRadioButton – Набір перемикачів лише з одним активним елементом.
- QSlider – Повзунок.
- QSpinBox – Цілочисельний спінер.
- QTimeEdit – Для редагування часу.

Розглянемо деякі з найбільш часто використовуваних віджетів і розглянемо їх більш детально.

Щоб розглянути роботу віджетів, знадобиться проста програма для їх розміщення (Рис. 12). Наступний код викликає вікно. Будемо його доповнювати елементами:

```
import sys
from PyQt5.QtWidgets import (QMainWindow, QApplication, QLabel,
QCheckBox, QComboBox, QListWidget, QLineEdit, QLineEdit, QSpinBox,
QDoubleSpinBox, QSlider )
from PyQt5.QtCore import Qt

class MainWindow(QMainWindow):
     def __init__(self):
          super(MainWindow, self).__init__()
```

```
        self.setWindowTitle("My App")

app = QApplication(sys.argv)
w = MainWindow()
w.show()
app.exec()
```

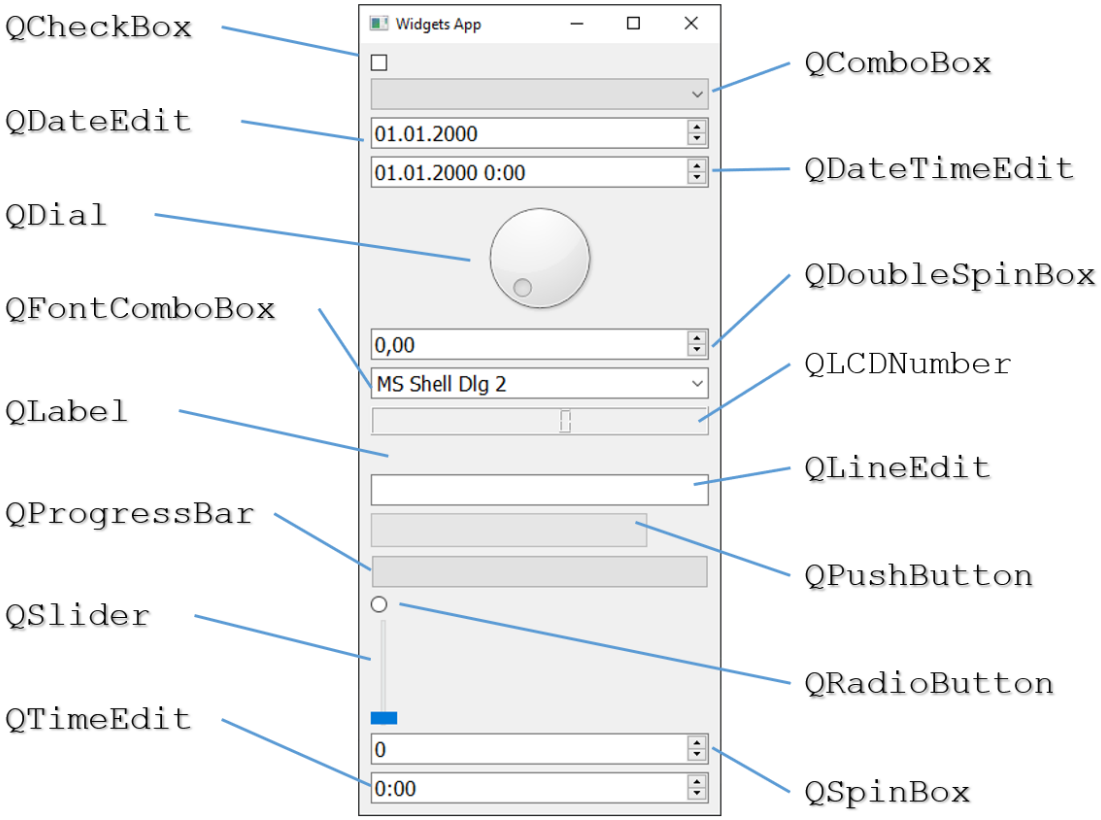

Рисунок 11 – Компоненти графічного інтерфейсу користувача.

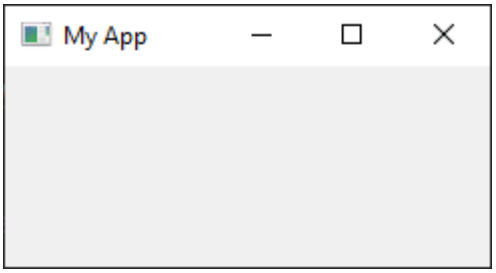

Рисунок 12 – Вікно для розміщення віджетів.

QLabel.

QLabel – простий однорядковий фрагмент тексту, Встановлення тексту відбувається шляхом передачі str під час його створення (Рис. 13). Підтримується налаштування параметрів шрифту. Для зміни властивостей шрифту віджета, треба отримати поточний шрифт, оновити його, а потім знову застосувати. Це гарантує, що шрифт буде відповідати умовам робочого простору.

Вирівнювання вказується за допомогою прапора з Qt. namespace. Можна поєднувати прапорці за допомогою прямої лінії (|), при цьому використовувати лише прапорець вертикального чи горизонтального вирівнювання.

```
class MainWindow(QMainWindow):
     def __init__(self):
          super(MainWindow, self).__init__()
          self.setWindowTitle("My App")
          widget = QLabel("Hello")
          widget.setText("Next text")
          font = widget.font()
          font.setPointSize(30)
          widget.setFont(font)
          widget.setAlignment(Qt.AlignHCenter | Qt.AlignVCenter)

          self.setCentralWidget(widget)
```

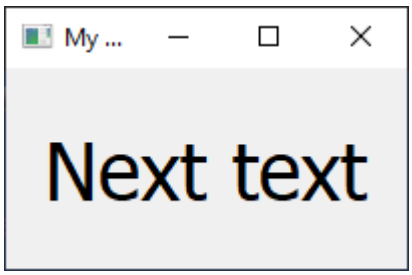

Рисунок 13 – Використання віджету QLabel.

Qlabel. QPixmap.

QLabel можна використовувати для відображення зображення за допомогою .setPixmap(). За замовчуванням зображення масштабується, зберігаючи співвідношення сторін. Якщо необхідно, щоб воно розтягувався та масштабувався, чи щоб повністю відповідати вікну, необхідно встановити .setScaledContents(True) для QLabel (Рис. 14).

```
class MainWindow(QMainWindow):
     def __init__(self):
          super(MainWindow, self).__init__()
          self.setWindowTitle("My App")
          widget = QLabel("Hello")
          widget.setPixmap(QPixmap("demo.png"))
          widget.setScaledContents(True)
          self.setCentralWidget(widget)
```

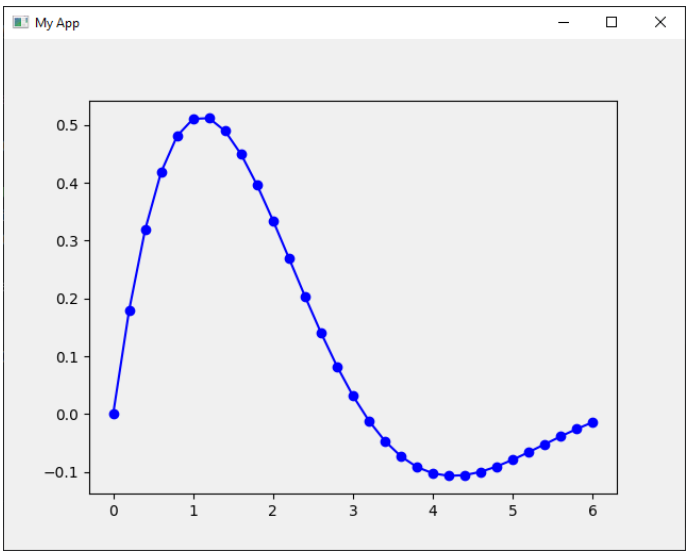

Рисунок 14 – Використання віджету Qlabel, QPixmap.

QCheckBox.

QCheckBox() – представляє користувачеві прапорець (Рис. 15). Однак, як і з усіма віджетами Qt, існує ряд параметрів для зміни поведінки віджетів. Можна встановити стан прапорця програмним шляхом за допомогою .setChecked або .setCheckState. Перший приймає або True або False, що представляє позначене або не позначене поле відповідно. Однак за допомогою .setCheckState, визначається певний перевірений стан.

Прапорець, який підтримує частково позначений стан (Qt.PartiallyChecked), зазвичай називають «тристаном», який не є ні увімкненим, ні вимкненим. Прапорець у цьому стані відображається як сірий прапорець і зазвичай використовується в ієрархічних структурах прапорців, де піделементи пов’язані з батьківськими прапорцями.

```
class MainWindow(QMainWindow):
     def __init__(self):
          super(MainWindow, self).__init__()
          self.setWindowTitle("My App")
          widget = QCheckBox()
          widget.setCheckState(Qt.Checked)
# For tristate: widget.setCheckState(Qt.PartiallyChecked)
# Or: widget.setTriState(True)
          widget.stateChanged.connect(self.show_state)
          self.setCentralWidget(widget)

     def show_state(self, s):
          print(s == Qt.Checked)
          print(s)
```

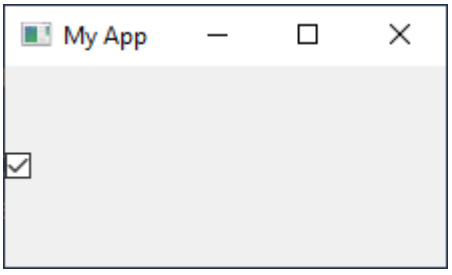

Рисунок 15 – Використання віджету QCheckBox.

QComboBox.

QComboBox — це розкривний список (Рис. 16), що дозволяє вибрати один елемент зі списку, а поточний вибраний елемент буде показано як мітку на віджеті. Комбінований список підходить для вибору з довгого списку параметрів. Є можливість додати елементи до QComboBox, передавши список рядків у .addItems(). Елементи будуть додані в тому порядку, в якому вони надані.

```
class MainWindow(QMainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        self.setWindowTitle("My App")
        widget = QComboBox()
        widget.addItems(["One", "Two", "Three"])
# Sends the current index (position) of the selected item.
        widget.currentIndexChanged.connect(
self.index_changed )
# There is an alternate signal to send the text.
        widget.currentTextChanged.connect(
self.text_changed )

        self.setCentralWidget(widget)
    def index_changed(self, i): # i is an int
        print(i)
    def text_changed(self, s): # s is a str
        print(s)
```

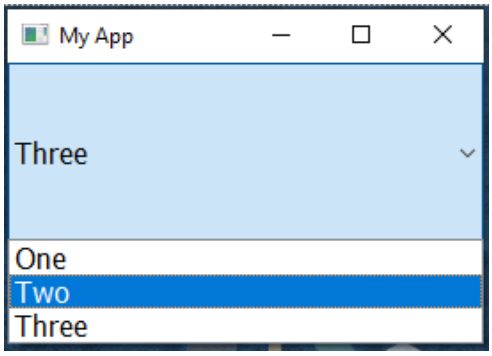

Рисунок 16 – Використання віджету QComboBox.

QComboBox також можна редагувати, дозволяючи користувачам вводити значення, яких зараз немає у списку, і /або вставляти їх, або просто використовувати як значення. Щоб зробити вікно доступним для редагування:

widget.setEditable(True)

Щоб визначити, як обробляється вставка, необхідно встановити прапорець. Такі позначки зберігаються в самому класі QComboBox, наприклад:

widget.setInsertPolicy(QComboBox.InsertAlphabetically)

Щоб обмежити кількість елементів, дозволених у полі, використовуйте .setMaxCount, наприклад:

widget.setMaxCount(10)

QListWidget.

Цей віджет схожий на QComboBox, за винятком того, що параметри представлені у вигляді списку елементів, який можна прокручувати (Рис. 17). Він також підтримує вибір кількох елементів одночасно. QListWidget пропонує сигнал currentItemChanged, який надсилає QListWidgetItem (елемент віджета списку), і сигнал currentTextChanged, який надсилає текст поточного елемента.

```
class MainWindow(QMainWindow):
     def __init__(self):
          super(MainWindow, self).__init__()
          self.setWindowTitle("My App")
          widget = QListWidget()
          widget.addItems(["One", "Two", "Three"])
          widget.currentItemChanged.connect(self.index_changed)
          widget.currentTextChanged.connect(self.text_changed)
          self.setCentralWidget(widget)
     def index_changed(self, i):
# Not an index, i is a QListWidgetItem
          print(i.text())
     def text_changed(self, s): # s is a str
          print(s)
```

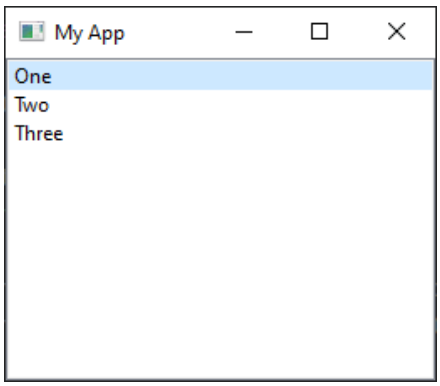

Рисунок 17 – Використання віджету QListWidget.

QLineEdit.

Віджет QLineEdit — це просте однорядкове поле для редагування тексту, у яке користувачі можуть вводити дані (Рис. 18). Вони використовуються для полів форми або налаштувань, де немає обмеженого списку дійсних введених даних. Наприклад, під час введення електронної адреси або імені комп'ютера. Можна ввести обмеження на максимальну довжину тексту під час редагування рядка.

QLineEdit має низку сигналів, доступних для різних подій редагування, зокрема, коли натиснуто клавішу повернення (користувачем), якщо користувач змінив вибір. Є також два сигнали редагування: один, коли текст у полі було відредаговано, і інший, коли його було змінено. Тут розрізняють редагування користувача та програмні зміни. Сигнал textEdited надсилається лише тоді, коли користувач редагує текст. Крім того, можна виконати перевірку введення за допомогою маски введення, щоб визначити, які символи підтримуються widget.setInputMask.

```
class MainWindow(QMainWindow):
```

```
    def __init__(self):
        super(MainWindow, self).__init__()
        self.setWindowTitle("My App")
        widget = QLineEdit()
        widget.setMaxLength(10)
        widget.setPlaceholderText("Enter your text")
        #widget.setReadOnly(True) # uncomment this to make
readonly
        widget.returnPressed.connect(self.return_pressed)
        widget.selectionChanged.connect(self.selection_changed)

        widget.textChanged.connect(self.text_changed)
        widget.textEdited.connect(self.text_edited)
        self.setCentralWidget(widget)
    def return_pressed(self):
        print("Return pressed!")
        self.centralWidget().setText("BOOM!")
    def selection_changed(self):
        print("Selection changed")
        print(self.centralWidget().selectedText())
    def text_changed(self, s):
        print("Text changed...")
        print(s)
    def text_edited(self, s):
        print("Text edited...")
        print(s)
```

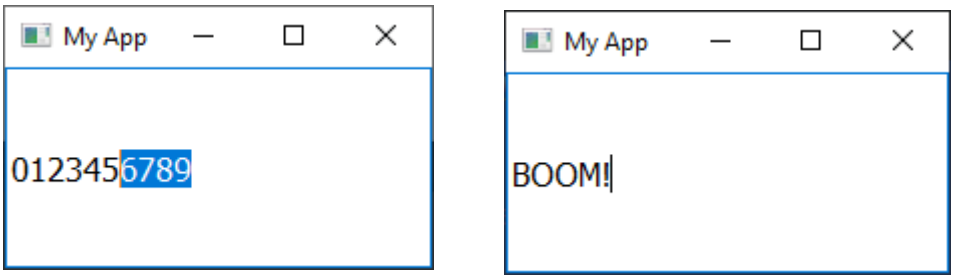

Рисунок 18 – Використання віджету QLineEdit.

QSpinBox та QDoubleSpinBox.

QSpinBox надає невелике числове вікно введення зі стрілками для збільшення та зменшення значення (Рис. 19). Підтримує цілі числа, а відповідний віджет QDoubleSpinBox підтримує числа з плаваючою точкою. setMinimum і setMaximum – використовуються для встановлення діапазону прийнятних значень, чи setRange, щоб встановити обидва одночасно. Анотація типів значень підтримується як префіксами, так і суфіксами, які можна додати до числа, наприклад, для валютних маркерів або одиниць, використовуючи .setPrefix і .setSuffix відповідно.

Натискання стрілок вгору та вниз на віджеті збільшить або зменшить значення у віджеті на суму, яку можна встановити за допомогою .setSingleStep.

QSpinBox, і QDoubleSpinBox мають сигнал .valueChanged, який запускається щоразу, коли їхнє значення змінюється. Необроблений сигнал .valueChanged надсилає числове значення (або int, або float), тоді як .textChanged надсилає значення у вигляді рядка, включаючи символи префіксу та суфіксу. За бажанням можна вимкнути введення тексту

під час редагування рядка вікна обертання, установивши для нього режим лише для читання. widget.lineEdit().setReadOnly(True).

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("My App")
        widget = QSpinBox()
        # Or: widget = QDoubleSpinBox()
        widget.setMinimum(-10)
        widget.setMaximum(3)
        # Or: widget.setRange(-10,3)
        widget.setPrefix("$")
        widget.setSuffix("c")
        widget.setSingleStep(3)
        # Or e.g. 0.5 for QDoubleSpinBox
        widget.valueChanged.connect(self.value_changed)
        widget.textChanged.connect(self.value_changed_str)
        self.setCentralWidget(widget)
    def value_changed(self, i):
        print(i)
    def value_changed_str(self, s):
        print(s)
```

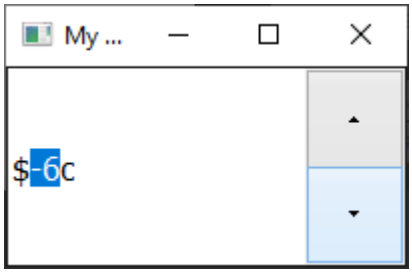

Рисунок 19 – Віджети QSpinBox та QDoubleSpinBox.

QSlider.

QSlider надає віджет слайд-бару, який функціонує всередині так само, як QDoubleSpinBox (Рис. 20). Замість відображення поточного значення в числовому вигляді, воно представлене положенням повзунка вздовж довжини віджета. Це часто корисно, коли забезпечується регулювання між двома крайовими значеннями, але там, де не потрібна абсолютна точність. Найпоширенішим використанням цього типу віджетів є керування гучністю. Є додатковий сигнал .sliderMoved, який спрацьовує щоразу, коли повзунок переміщується, і сигнал .sliderPressed, який випромінює щоразу, коли повзунок клацає.

Можна створити повзунок із вертикальною чи горизонтальною орієнтацією, передаючи орієнтацію під час створення. Наприклад: widget = QSlider(Qt.Vertical), чи widget = QSlider(Qt.Horizontal).

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("My App")
        widget = QSlider()
        widget.setMinimum(-10)
```

```
        widget.setMaximum(3) # Or: widget.setRange(-10,3)
        widget.setSingleStep(3)
        widget.valueChanged.connect(self.value_changed)
        widget.sliderMoved.connect(self.slider_position)
        widget.sliderPressed.connect(self.slider_pressed)
        widget.sliderReleased.connect(self.slider_released)
        self.setCentralWidget(widget)
    def value_changed(self, i):
        print(i)
    def slider_position(self, p):
        print("position", p)
    def slider_pressed(self):
        print("Pressed!")
    def slider_released(self):
        print("Released")
```

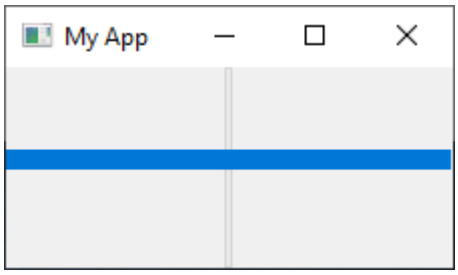

Рисунок 20 – Слайдер.

QDial.

QDial — це обертовий віджет, який функціонує так само, як повзунок, але виглядає як аналоговий циферблат (Рис. 21). Це виглядає гарно, але з точки зору інтерфейсу користувача не дуже зручно. Однак вони часто використовуються в аудіододатках як представлення реальних аналогових циферблатів. Сигнали такі ж, як і для QSlider, і зберігають ті самі назви (наприклад, .sliderMoved).

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("My App")
        widget = QDial()
        widget.setRange(-10, 100)
        widget.setSingleStep(1)
        widget.valueChanged.connect(self.value_changed)
        widget.sliderMoved.connect(self.slider_position)
        widget.sliderPressed.connect(self.slider_pressed)
        widget.sliderReleased.connect(self.slider_released)
        self.setCentralWidget(widget)
    def value_changed(self, i):
        print(i)
    def slider_position(self, p):
        print("position", p)
    def slider_pressed(self):
        print("Pressed!")
```

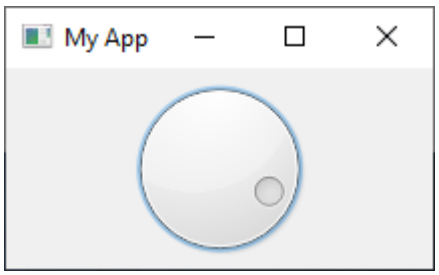

Рисунок 21 – Віджет QDial.

Панелі інструментів.

Панелі інструментів — це панелі піктограм і/або тексту, які використовуються для виконання звичайних завдань у програмі, доступ до яких через меню був би громіздким (Рис. 22). У Qt панелі інструментів створюються з класу QToolBar. Для початку створюється екземпляр класу, а потім викликаєте .addToolbar у QMainWindow. Передача рядка як першого параметра в QToolBar встановлює назву панелі інструментів, яка використовуватиметься для ідентифікації панелі інструментів в інтерфейсі користувача.

QAction — це клас, який надає спосіб опису абстрактних інтерфейсів користувача.

Створимо функцію, яка прийматиме сигнал від QAction. Під час створення екземпляра можна передати мітку для дії та/або піктограму. Необхідно передати будь-який QObject, що буде служити батьківським елементом для дії. Передаємо self як посилання на головне вікно (тут батьківський елемент передається як останній параметр).

Встановимо підказку про стан — цей текст відображатиметься на панелі стану QStatusBar. Підключаємо сигнал .triggered до спеціальної функції. Цей сигнал спрацьовуватиме кожного разу, коли QAction запускається (або активується).

```
class MainWindow(QMainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        self.setWindowTitle("My Awesome App")
        label = QLabel("Hello!")
        label.setAlignment(Qt.AlignCenter)
        self.setCentralWidget(label)
        toolbar = QToolBar("My main toolbar")
        self.addToolBar(toolbar)
        button_action = QAction("Your button", self)
        button_action.setStatusTip("This is your button")

    button_action.triggered.connect(self.onMyToolBarButtonClick)
        toolbar.addAction(button_action)
        self.setStatusBar(QStatusBar(self))
    def onMyToolBarButtonClick(self, s):
        print("click", s)
```

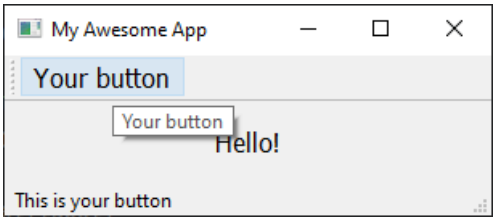

Рисунок 22 – Робота з панеллю інструментів.

Далі ввімкнемо перемикач QAction, натискання вмикає його, повторне натискання – вимикає. Для цього викликаємо setCheckable(True) для об'єкта QAction (Рис. 23).

```
def onMyToolBarButtonClick(self, s):
        print("click", s)
```

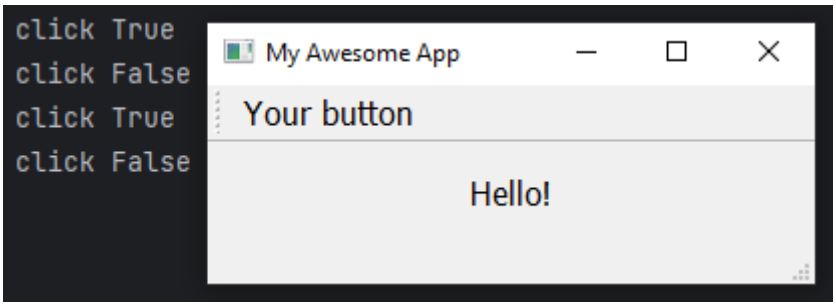

Рисунок 23 – Робота перемикача дії.

Можна створити об'єкт QIcon, передавши назву файлу класу. Щоб додати піктограму до QAction (і, отже, кнопку) – передаємо її як перший параметр під час створення QAction. Не забуваємо повідомити панелі інструментів, наскільки великі піктограми передаються, інакше вона буде оточена великою кількістю відступів. Зробити це можна, викликавши .setIconSize() з об'єктом QSize (Рис. 24).

```
class MainWindow(QMainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        self.setWindowTitle("My Awesome App")
        label = QLabel("Hello!")
        label.setAlignment(Qt.AlignCenter)
        self.setCentralWidget(label)
        toolbar = QToolBar("My main toolbar")
        toolbar.setIconSize(QSize(32, 32))
        self.addToolBar(toolbar)
        button_action = QAction(QIcon("demo.png"), "Your
button", self)
        button_action.setStatusTip("This is your button")

    button_action.triggered.connect(self.onMyToolBarButtonClick)
        button_action.setCheckable(True)
        toolbar.addAction(button_action)
        self.setStatusBar(QStatusBar(self))
    def onMyToolBarButtonClick(self, s):
        print("click", s)
```

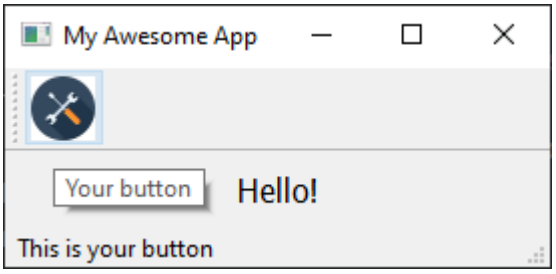

Рисунок 23 – Встановлення піктограми у панель інструментів.

Меню є ще одним стандартним компонентом інтерфейсу користувача (Рис. 24). Зазвичай вони знаходяться у верхній частині вікна та дозволяють отримати доступ до всіх стандартних функцій програми. Існує кілька стандартних меню — наприклад, Файл, Правка, Довідка. Меню можуть бути вкладеними для створення ієрархічних дерев функцій, і вони часто підтримують і відображають комбінації клавіш для швидкого доступу до своїх функцій.

Створемо меню .menuBar() у QMainWindow та додамо меню на панель, викликаючи .addMenu(), передаючи назву меню «&Файл». Амперсанд визначає швидку клавішу переходу до цього меню при натисканні Alt.

Тут грає роль сила Action, можемо повторно використати вже існуючий QAction, щоб додати ту саму функцію до меню. Щоб додати дію, викликаємо .addAction, передаючи одну з визначених дій.

```
class MainWindow(QMainWindow):
     def __init__(self):
          super().__init__()
          self.setWindowTitle("My App")
          label = QLabel("Hello!")
          label.setAlignment(Qt.AlignCenter)
          self.setCentralWidget(label)
          toolbar = QToolBar("My main toolbar")
          toolbar.setIconSize(QSize(16, 16))
          self.addToolBar(toolbar)
          button_action    =    QAction(QIcon("img.png"),    "&Your
button", self)
          button_action.setStatusTip("This is your button")

     button_action.triggered.connect(self.onMyToolBarButtonClick)
          button_action.setCheckable(True)
          toolbar.addAction(button_action)
          toolbar.addSeparator()
          button_action2    =    QAction(QIcon("img.png"),    "Your
&button2", self)
          button_action2.setStatusTip("This is your button2")

     button_action2.triggered.connect(self.onMyToolBarButtonClick
)
          button_action2.setCheckable(True)
          toolbar.addAction(button_action2)
          toolbar.addWidget(QLabel("Hello"))
          toolbar.addWidget(QCheckBox())
          self.setStatusBar(QStatusBar(self))
          menu = self.menuBar()
          file_menu = menu.addMenu("&File")
          file_menu.addAction(button_action)
```

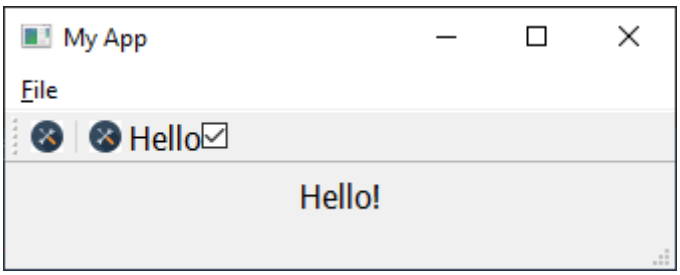

Рисунок 24 – Меню.

Додамо роздільник до меню (Рис. 25), який відображатиметься як горизонтальна лінія в меню, а потім додамо ще QAction. Щоб додати підменю, створюємо нове меню, викликавши addMenu() у батьківському меню.

```
menu = self.menuBar()
file_menu = menu.addMenu("&File")
file_menu.addAction(button_action)
file_menu.addSeparator()
file_submenu                =                file_menu.addMenu("Submenu")
file_submenu.addAction(button_action2)

button_action.setShortcut(QKeySequence("Ctrl+p"))
```

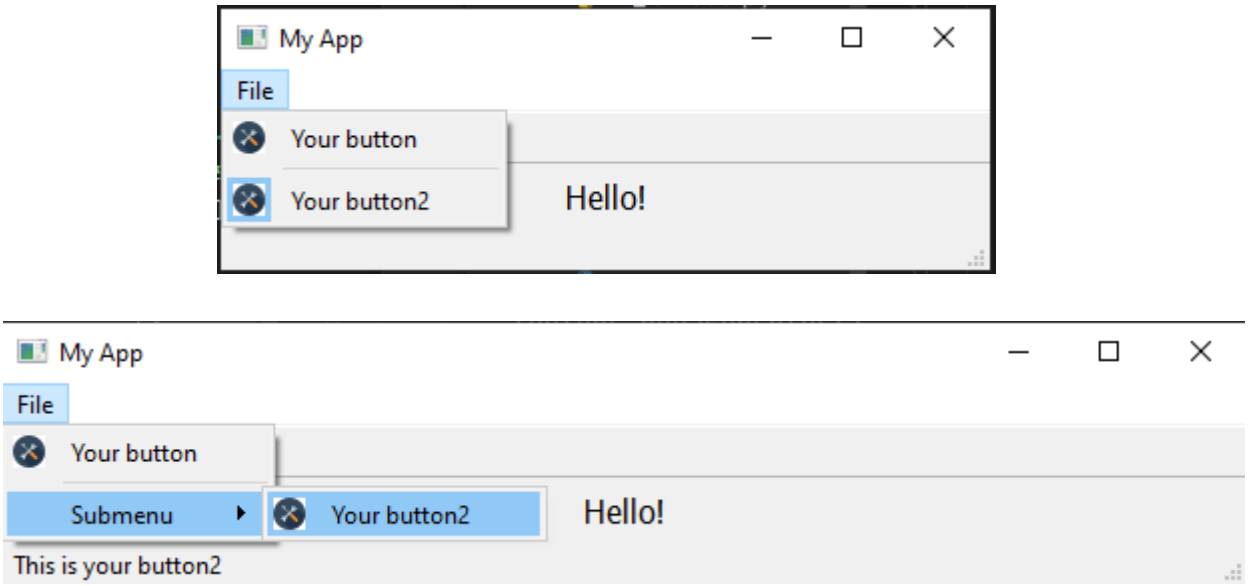

Рисунок 25 – Додавання підменю до головного меню.

Завдання до лабораторної роботи №8

1. Не використовуючи дизайнер, написати програму, яка виводить у вікні Ваше ім’я та прізвище, при натисканні на кнопку, випадковим чином змінюється колір напису.

2. Створити у Qt Designer форму з полями «Студент», «Дисципліна», «Оцінка» та полями для введення даних. Для вводу числових даних використовувати QSpinBox. Дизайн

вікна на Ваш розсуд. При натисканні на кнопку, інформація зберігається у csv файл у відповідні стовпчики. Використовувати параметр зберігання з дозаписом інформації.

3. Додати до створеної програми – меню з 2-4 пунктами. Додати QFileDialog для вибору файлу зображення і вставлення його у вікно (QtGui.QPixmap).

4. Додати на форму QComboBox, інформація до полів якого заноситься після натискання на кнопку. В залежності від обраного QRadioButton відповідна інформація зчитується із одного з трьох стовпчиків csv файлу, записаного у пункті 2. Якщо обрано стовпчик “Оцінка”, активується QProgressBar (значення якого змінюється в залежності від обраної оцінки. Діапазон – від 0 до 5).

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Програмування числових методів мовою Python : підруч./ А. В. Анісімов, А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий ; за ред. А. В. Анісімова. – К. : Видавничо-поліграфічний центр "Київський університет", 2014. – 640 с.
2. Fabio Nelli «Python Data Analytics», 2015 —350 p.
3. Beau Curtin «Django Cookbook Web Development with Django Step by Step Guide 2-nd Edition», 2016 —153.
4. Stewart A. «Python Programming for Beginners», 2016 — 115 p.
5. Креневич А.П. Python у прикладах і задачах. Частина 1. Структурне програмування Навчальний посібник із дисципліни "Інформатика та програмування" – К.: ВПЦ "Київський Університет", 2017. – 206 с.
6. Основи програмування. Python. Частина 1 [Електронний ресурс]: підручник для студ. спеціальності 122 "Комп’ютерні науки", спеціалізації "Інформаційні технології в біології та медицині" / А. В. Яковенко ; КПІ ім. Ігоря Сікорського. – Електронні текстові данні (1 файл: 1,59 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2018. – 195 с.
7. Замуруєва О. В., Кримусь А. С., Ольхова Н. В. Об’єктно-орієнтоване програмування в Python : курс лекцій. Луцьк : Вежа-Друк, 2018. – 64 с.
8. Васильєв О. М. Програмування мовою Python. Тернопіль: Навчальна книга – Богдан, 2019. – 504с
9. Костюченко А.О. Основи програмування мовою Python: навчальний посібник. Ч.: ФОП Баликіна С.М., 2020. -180 с.
10. Fitzpatrick M. Create Simple GUI Applications, with Python & Qt5. The hands-on guide to building desktop apps with Python. – 2020. – 264 p.
11. Matthes E. Python Crash Course, 3rd Edition: A Hands-On, Project-Based Introduction to Programming 3rd Edition. . – 2023. – 552 p. ISBN 978-1718502703
12. https://peps.python.org/pep-0008/ (Керівництво по стилю для коду Python)

Навчальне видання

Методичні вказівки
до виконання лабораторних завдань
з навчальної дисципліни «Програмування на Python»
для студентів денної та заочної форм навчання
за спеціальностями «122 Комп’ютерні науки», та
«113 Прикладна математика»

Укладач:

ШАПОВАЛОВА Марія Ігорівна

Відповідальний за випуск доц. Водка О.О.

Роботу до видання рекомендував доц. Федоров В.О.

В авторській редакції

План 2024 р., поз. 894

Підписано до видання 24.10.2024

Гарнітура Times New Roman. Обсяг - 3,5 др. арк.

Електронна версія