

І.М. Штунов

# ПАЙТОН ДЛЯ ЕКОНОМІСТІВ

НТУ «ДП»  
2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ  
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»



**І.М. Пістунов**

# **ПАЙТОН ДЛЯ ЕКОНОМІСТІВ**

Дніпро  
НТУ «ДПУ»  
2025

УДК 004.4:330

ПЗ4

*Затверджено вченою радою як навчальний посібник  
для студентів спеціальності 051 Економіка  
(протокол № \_\_ від \_\_.\_\_.2025).*

**Рецензенти:**

*Н.К. Васильєва*, д-р екон. наук, проф., завідувач кафедри інформаційних систем і технологій Дніпровського державного аграрно-економічного університету;

*К.Ф. Ковальчук*, д-р екон. наук, проф., декан факультету економіки та менеджменту Інституту промислових та бізнес технологій Українського державного університету науки і технологій.

**ПЗ4** Пістунов І.М. Пайтон для економістів. навч. посібник. Дніпро : НТУ «ДП», 2025. 231 с.

Подано опис інсталяції інтерпретатора, порядок підключення бібліотек. Описана структура мови програмування, методи створення програм, типи даних, типи масивів даних, основні оператори мови, такі, як IF, WHILE, FOR, INPUT, OUTPUT. Описані основні функції базової комплектації мови, а також найбільш затребувані можливості таких бібліотек як PUNDAS, TURTLE, STATGRAF, тощо. Окремо пояснюється принцип створення незалежних від інтерпретатора програм типу \*.EXE. Також наведено методика вирішення оптимізаційних задач та задач прогнозування.

Посібник містить значну кількість прикладів програм, що пояснюють окремі теми і можуть слугувати частинами більших програм, що вирішують практичні задачі.

Призначено для студентів закладів вищої освіти і молодих вчених, а також може слугувати підручником для школярів, що факультативно вивчають Пайтон.

Посібник базується на літературних джерелах вітчизняних та зарубіжних авторів та на досвіді викладання дисциплін «Мова програмування Пайтон», «Аналіз даних з використанням Пайтон» та «Пайтон для економістів» за спеціальністю 051 Економіка в Національному технічному університеті «Дніпровська політехніка».

УДК 004.4:330

© І.М. Пістунов, 2024

© НТУ« Дніпровська політехніка», 2024

# ЗМІСТ

ВСТУП.....	6
Порядок засвоєння матеріалу .....	8
Набуті компетенції.....	9
Розділ 1. БАЗОВІ ПОЛОЖЕННЯ .....	11
1.1. Історія виникнення мови Пайтон .....	11
1.2. Інсталяція інтерпретатора та бібліотек підпрограм .....	12
1.3. Порядок написання програм через інтерпретатор, Блокнот та PyCharm .....	19
1.4. Структура програми, вкладені оператори, коментарі.....	26
Розділ 2. ДАНІ МОВИ ПРОГРАМУВАННЯ ПАЙТОН .....	33
2.1. Тексти (літерали) та операції з ними .....	33
2.2. Числа цілі, дробові, з плаваючою точкою та операції з ними	41
2.2.1. Цілі числа.....	42
2.2.2. Дійсні числа .....	44
2.2.3. Вбудовані функції Python.....	46
2.2.4. Правила запису формул у Пайтоні.....	51
2.3. Масиви даних та операції з ними .....	52
2.3.1. Списки (list) .....	53
2.3.2. Індеси і зрізи .....	58
2.3.3. Кортежі (tuple).....	60
2.3.4. Словники (dict) і робота з ними. Методи словників .	63
Розділ 3. ОСНОВНІ ОПЕРАТОРИ МОВИ ПАЙТОН.....	69
3.1. Логічний оператор IF.....	71



3.2. Оператор циклу .....	75
3.2.1. WHILE.....	75
3.2.2. FOR.....	78
3.3. Оператори вводу та виводу даних.....	81
3.4. Підпрограми-функції .....	89
3.5. Створення GUI додатку.....	94
3.6. Компіляція текстів програм у *.exe файл.....	106
3.7. Обробка винятків у Пайтон.....	109
Розділ 4. БІБЛІОТЕКИ ПАЙТОН.....	114
4.1. Економічні та фінансові розрахунки.....	117
4.2. Операції вводу-виводу з файлами Excel та виведення на друк.....	119
4.3. Зчитування даних зі сторінок в Інтернеті .....	124
4.4. Графіка .....	137
4.4.1. TURTLE .....	137
4.4.2. MATPLOTLIB .....	141
4.5. Статистичний аналіз .....	145
4.5.1. Описова статистика. Кореляція.....	145
4.5.2. Закони розподілу. Статистичні критерії.....	147
4.6. Регресійний аналіз. Економетрика.....	150
4.7. Прогнозування часових рядів .....	154
4.8. Оптимізація .....	158
4.9. Теорія ігор.....	165
4.10. Кластеризація, класифікація .....	170
4.11. Нейронні сітки.....	174
4.11.1. Регресійні моделі .....	174
4.11.2. Кластеризація. Класифікація .....	177
4.12. Імітаційне моделювання.....	181
4.13. Нечіткі множини .....	185

Розділ 5. ІНДИВІДУАЛЬНІ ЗАВДАННЯ 1 .....	194
5.1. Завдання №1. Банківська позика .....	195
5.2. Завдання №2. Логічні оператори, булевий тип даних .....	195
5.3. Завдання №3. Конкатенація і повторення ряд .....	196
5.4. Завдання №4. Програма "Вгадай число від 1 до N" .....	198
5.5. Завдання № 5. Робота з файлами .....	198
5.6. Завдання №6. Визначити індекси елементів масиву (списку), значення яких належать заданому діапазону .....	199
5.7 Завдання №7. Малювання з використанням черепашої графіки .....	199
5.8. Завдання №8. Робота з списками в Python .....	200
5.9. Завдання №9. Знайдення оптимального рішення .....	201
5.10. Екзаменаційне завдання. Прогнозування. Теорія ігор. Нечіткі моделі .....	209
Розділ 6. ІНДИВІДУАЛЬНІ ЗАВДАННЯ 2 .....	211
6.1. Завдання №1. Створення GUI додатку .....	212
6.2. Завдання №2. Робота з комплексом Python в Інтернеті .....	214
6.3. Завдання №3. Робота з файлами типу *CSV .....	216
6.4. Завдання №4. Побудова графіків різних типів .....	217
6.5. Завдання №5. Побудова графіків зі складною формою представлення .....	221
6.6. Завдання №6. Побудова графіків різних типів для даних української економіки .....	225
ВИСНОВКИ .....	228
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ .....	230

# ВСТУП

Python є однією з найпопулярніших мов програмування у світі, і він має безліч корисних функцій та бібліотек, які можуть бути використані економістами. Ось кілька причин, чому Python корисний для економістів:

1. Аналіз даних: Python має потужні бібліотеки, такі як NumPy, Pandas та Matplotlib, які дозволяють економістам обробляти, аналізувати та візуалізувати економічні дані. Завдяки цим бібліотекам ви можете легко і ефективно виконувати операції з даними, такі як завантаження, фільтрування, сортування, обчислення статистики, побудова графіків тощо.
2. Економетрика: Python надає потужні бібліотеки для економетричного моделювання, такі як StatsModels та Scikit-learn. Ви можете застосовувати різні методи регресії, часового ряду, класифікації та кластеризації для аналізу економічних даних та прогнозування.
3. Веб-скрапінг: Python має потужні бібліотеки, такі як Beautiful Soup та Scrapy, які дозволяють збирати дані з веб-сторінок. Економісти можуть використовувати ці бібліотеки для отримання економічних даних з інтернету, наприклад, цін на товари або фінансові показники.
4. Моделювання та оптимізація: Python має багато бібліотек для моделювання та оптимізації, такі як SciPy та PuLP. Ці бібліотеки дозволяють економістам будувати та розв'язувати економічні моделі, наприклад, лінійне програмування, оптимізацію портфеля або моделі споживчого вибору.
5. Машинне навчання: Python є популярним інструментом для машинного навчання та штучного інтелекту. Із використанням

бібліотек, таких як Scikit-learn, TensorFlow та PyTorch, економісти можуть застосовувати методи машинного навчання для аналізу економічних даних, прогнозування ринкових тенденцій або моделювання ризиків.

6. Біг-дата: Python може працювати з практично необмеженими масивами даних, чого не може, наприклад програма Excel, одна сторінка якої обмежена у 64 тисячі рядків.

Python має простий синтаксис, що дозволяє швидко вивчати мову програмування економістам без попереднього досвіду в програмуванні. Його активна спільнота розробників також регулярно розширює функціональність мови та надає нові бібліотеки, що спрощують роботу з економічними даними та аналізом.

Python використовують вчені різних напрямків – і медики, і механіки, і фізики. Для кожного з цих наукових напрямків у Python знайдеться своє програмне забезпечення.

Основні властивості мови Python:

- гнучка - підходить для розробки програм самого різного призначення і складності: від простих навчальних програм до професійних проектів.
- складається з нескладних для освоєння конструкцій – на відміну від інших мов, про які ви могли чути (C, PHP, Pascal).
- не вимагає компіляції, тобто програми на Python можна запускати на будь-яких платформах (Windows, Linux, MacOS, ...).
- підтримує безліч стилів програмування, на її прикладі можна освоїти будь-яку парадигму (ООП, структурне програмування, функціональне програмування).
- містить ще безліч корисних можливостей.

Мова програмування Python використовується при розробці таких відомих інтернет-сервісів, як YouTube, Google, Yahoo !; Python застосовують при розробці програм в NASA (космічне агентство США) і CERN (Європейська

організація з ядерних досліджень), а також, добре підходить для обробки економічних даних, отриманих з мережі Internet.

Цей посібник адаптовано спеціально для економістів, виключені описи тих можливостей, в яких немає потреби для цього напрямку. Але, маючі базові знання, при бажанні, кожен економіст може стати програмістом, поглибивши їх через численні довідники та відеоуроки, що їх можна знайти на YouTube і в Інтернеті.

Всі приклади, наведені у тексті посібника, є робочими. Їх копіювання прискорить розробку ваших програм.

У посібнику будуть описані прийоми використання Пайтон для реалізації різних економіко-математичних методів без конкретного їх опису.

Для опису цих методів звертайтеся до електронних посібників, посилання і Інтернеті на які подано у списку літератури.

### **Порядок засвоєння матеріалу**

1. Студент має ознайомитися з текстом чергового пункту посібника, увімкнувши інтерпретатор Пайтон (IDLE).
2. Побачивши приклад, його потрібно скопіювати до інтерпретатора і перевірити порядок роботи.
3. Варто спробувати змінити текст прикладу, для отримання іншого результату, навіть, якщо це призведе до збою в програмі.
4. Кожна змістовна частина посібника супроводжується індивідуальним завданням розробки певної програми.
5. На початку тексту програми треба вмістити дані щодо номеру завдання, назви завдання, номеру групи та імені студента. Наприклад

```
# Лабораторна робота №2  
# студ. гр. 051-99-1  
# Косач-Лариси Петрівни
```



6. Текст кожної програми має супроводжуватися коментарями, що описують той чи інший етап розрахунку. Наприклад

```
print('Привіт усім', 12**14); print(12.45678**123.54677) # Тут пишете коментар
# Тут пишете коментар
```
7. Файли з текстом програм треба розміщати у хмарному сховищі, вказаному викладачем, що проводить практичні заняття.
8. Файли з текстом програм у своїй назві мають містити номер завдання та ім'я студента і номер групи.
9. Сума оцінок по індивідуальним завдання складає результат вивчення цієї мови програмування.
10. Якщо практична робота після перевірки не зарахована, треба виправити помилки згідно з зауваженнями викладача. Доопрацьована практична робота надається для повторної перевірки разом з першим варіантом.
11. Студент, що не виконав практичні роботи, до іспиту не допускається.
12. Кожну програму потрібно розробляти з урахуванням некомпетентності оператора (захист від «дурня»), тому вона має містити підказки про момент введення даних, перевірку на правильність типу введених даних, повідомлення оператора про невірні дані, спробу знову ввести дані, опис результатів розрахунків.
13. Відсутність вимог з п. 12 автоматично зменшує оцінку до рівня не вище «добре».

### **Набуті компетенції**

1. Уміти виконати економічний розрахунок бідь якої складності.
2. Уміти проваджувати сучасні методи та моделі рішення економічних задач.
3. Уміти розраховувати ризики фінансових ринків та інвестиційних інструментів, на основі розумінні принципів інженерії та застосуванні методів математичного та статистичного аналізу.

4. Уміти отримувати дані з сайтів Інтернету, проводити їх статистичну розробку.
5. Уміти виводити дані у текстових форматах, а також у форматі Excel.
6. Уміти створювати автономні програми, які працюватимуть без підтримки інтерпретатора Пайтон.
7. Використовувати аналітичний та інструментарій мови програмування Python для обґрунтування економічних рішень

# Розділ 1.

## БАЗОВІ ПОЛОЖЕННЯ

*Вивчивши матеріали цього розділу студенти узнають, як інстальювати Пайтон та його доданки на своєму комп'ютері.*

### 1.1. Історія виникнення мови Пайтон

Історія створення мови програмування Python пов'язана з ім'ям Гвідо ван Россума, голландського програміста. Вона розпочалася в кінці 1980-х років, коли ван Россум працював у центрі досліджень CWI (Centrum Wiskunde & Informatica) у Нідерландах.

У 1989 році ван Россум почав працювати над новою мовою програмування, яка була б простою та ефективною для розв'язання різноманітних завдань. Він вирішив використовувати мову ABC, але розробити її у більш об'ємну та потужну мову.

Основними цілями створення Python були зручність для програмістів та читабельність коду.

У лютому 1991 року ван Россум випустив першу версію Python (версія 0.9.0) і назвав її на честь популярного теле-шоу "Monty Python's Flying Circus". Він обрав цю назву, бо вважав, що це буде наймовірною назвою для мови програмування.

Python швидко здобув популярність серед програмістів через свою простоту в освоєнні, читабельність коду та велику кількість корисних функцій. З



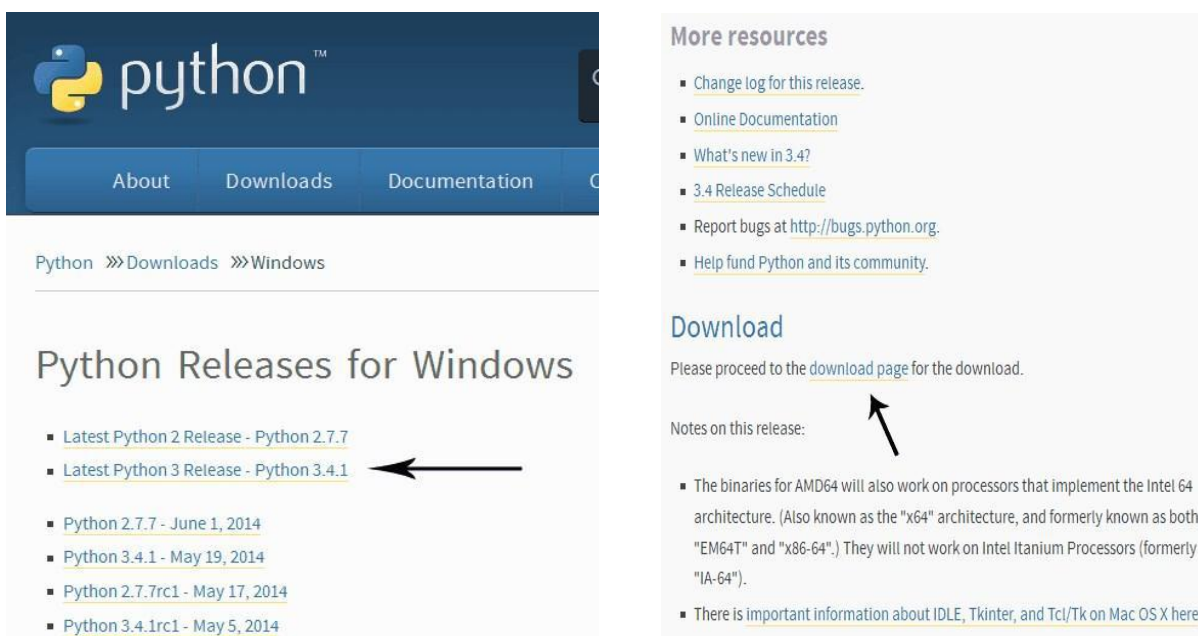
часом мова постійно розвивалася, з'являлися нові версії з покращеннями та доповненнями.

Сьогодні Python – одна з найпопулярніших мов програмування у світі. Вона використовується в різноманітних галузях, таких як веб-розробка, наукові дослідження, штучний інтелект, аналіз даних, розробка ігор та багато іншого. Велика спільнота розробників по всьому світу активно підтримує та розвиває мову, що робить її дуже динамічним та перспективним інструментом для програмістів.

## 1.2. Інсталяція інтерпретатора та бібліотек підпрограм

Завантажувати Python треба з офіційного сайту. Не рекомендується завантажувати інтерпретатор Python з інших сайтів або через торрент, в них можуть бути віруси. Програма безкоштовна. Потрібно зайти на <https://python.org/downloads/windows/>, та обрати "latest python release" і взяти найсвіжішу версію Python.

З'являється сторінка з описом даної версії Python (англійською). Потім крутимо в самий низ сторінки та відкриваємо "download page", як це показано стрілками на рисунках внизу.

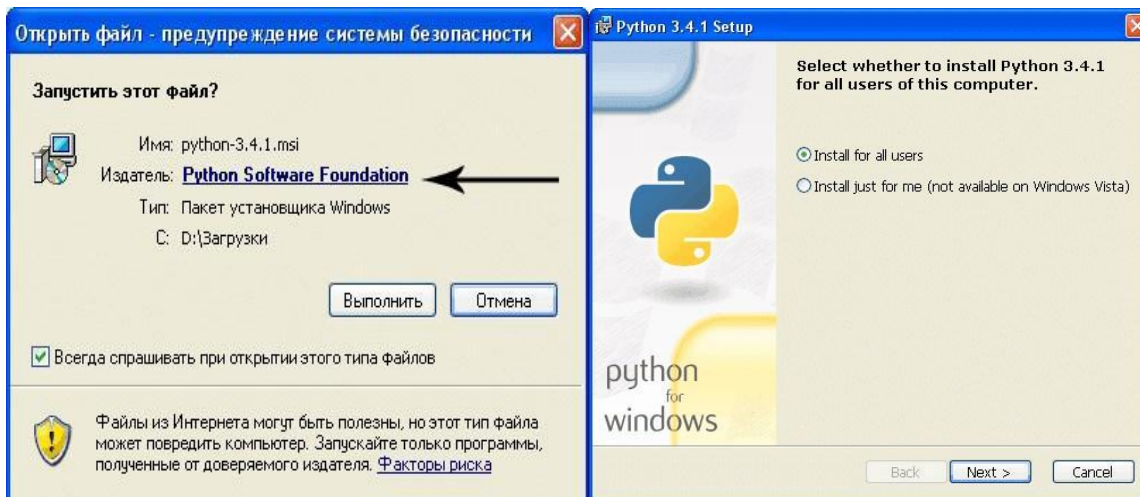


The screenshot shows the Python website's 'Downloads' page for Windows. The page title is 'Python Releases for Windows'. It lists several releases, with 'Latest Python 3 Release - Python 3.4.1' highlighted by a black arrow. To the right, there is a 'More resources' section with links like 'Change log for this release', 'Online Documentation', and 'What's new in 3.4?'. Below that is a 'Download' section with the text 'Please proceed to the download page for the download.' and 'Notes on this release:'. A black arrow points to the 'download page' link in the 'Download' section.

Ви побачите список файлів, які можна завантажити. Нам потрібен Windows x86 MSI installer (якщо система 32-х бітна), або Windows x86-64 MSI installer (якщо система 64-х бітна), які показані стрілками. Більше з файлів нічого не потрібно.

Version	Operating System	Description	Date	MD5 Sum	File Size
<a href="#">Mac OS X 64-bit/32-bit installer</a>	Mac OS X	for Mac OS X 10.6 and later		316a2f83edff73bbbc2c84390bee2db	22776248
<a href="#">Mac OS X 32-bit i386/PPC installer</a>	Mac OS X	for Mac OS X 10.5 and later		534f8ec2f5ad5539f9165b3125b5e959	22692757
<a href="#">XZ compressed source tarball</a>	Source release			6cafc183b4106476dd73d5738d7f616a	14125788
<a href="#">Gzipped source tarball</a>	Source release			26695450087f8587b26d0b6a63844af5	19113124
<a href="#">Windows debug information files</a>	Windows			9ce29e8356cf13f88e41f7595c2d7399	36744364
<a href="#">Windows x86 MSI installer</a>	Windows			4940c3fad01ffa2ca7f9cc43a005b89a	24408064
<a href="#">Windows debug information files for 64-bit binaries</a>	Windows			44a2d4d3c62a147f5a9f733b030490d1	24129218
<a href="#">Windows help file</a>	Windows			6ff47ff938b15d2900f3c7311ab629e5	7297786
<a href="#">Windows x86-64 MSI installer</a>	Windows	for AMD64/EM64T/x64, not Itanium processors		25440653f27ee1597fd6b3e15eee155f	25104384

Клацаємо по скачаному файлу правою кнопкою мишки і чекаємо, поки *Python* завантажиться. Потім відкриваємо завантажений файл. Якщо він підписаний *Python Software Foundation*, значить файл скачано вірно. Встановлюємо доступ для всіх користувачів або тільки для одного (на ваш розсуд).



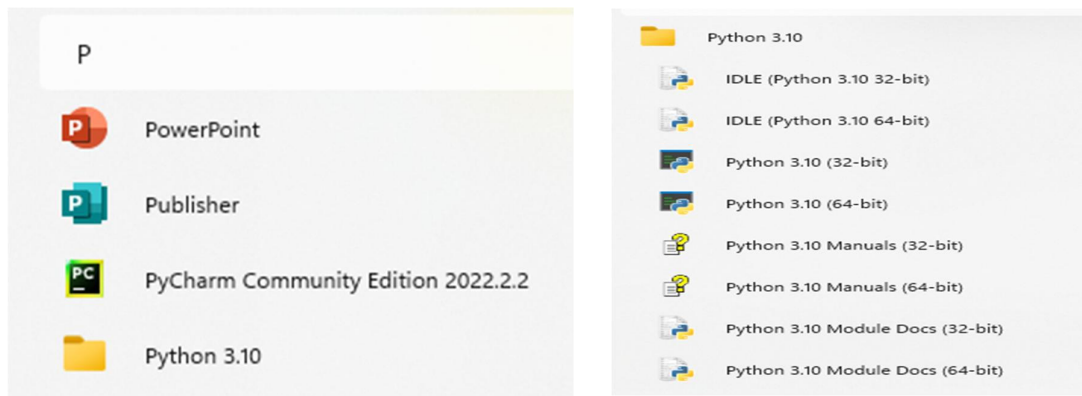


Можна довільно обрати папку для установки, але краще Пайтон працює на диску C:\, де розташовані всі програми вашого комп'ютера.

Можна довільно обрати компоненти, які будуть встановлені, але краще залишити компоненти за замовчуванням.



Після закінченні інсталяції пакету, ви отримаєте повідомлення, про його успішність. І на кнопку «Windows» ви отримаєте доступ до папки Python.



Мова Python цікава ще й тим, що більшість операцій в ній не потрібно програмувати. Для цього існують так звані бібліотеки підпрограм (Extension Packages). На сайті <https://pypi.python.org/pypi> їх вже налічується 127091.

Більш зручним для отримання цих бібліотек є сайт <http://www.lfd.uci.edu/~Egohlke/pythonlibs/> звідкіль можна взяти потрібну бібліотеку для того виду операційної системи, яка встановлена на вашому комп'ютері.

Ось так виглядає частина списку бібліотек:

**Index by date:** scikit-umfpack jupyter matplotlib scikit-image lxml pandas pyrsistent rasterio psutil gensim pillow peewee cffi pillow-simd cobra gpy openpiv chaco enable indexed\_gzip spglib yarl mod\_wsgi dulwich mercurial cython tiffiffle numpy orange scandir ode openexr discretize scipy pypmc chainer netcdf4 pymatgen regex twisted zodbpickle zope.interface fiona gdal brotli pycopg javabridge sfepy pyopengl fastcache bsddb3 ruamel.yaml multidict python-ldap faultfinder debug-information-files pymol babel czifile numexpr bokeh mkl-service spyder astropy kiwisolver numba llvmlite sqlalchemy scs veusz scikit-learn mysqlclient shapely vtk h5py cx\_oracle fabio fastparquet spectrum tornado chompack mpi4py pycairo pymongo moderngl minepy cvxpy fastcluster pywin32 tensorflow quantlib wrapt yt pycuda wordcloud gr opencv fixx gvar ad3 aiohttp pulp

А так – список доступних версій конкретної бібліотеки, який ви побачите, обравши якусь зі списку.

**NumPy**, a fundamental package needed for scientific computi

Numpy+MKL is linked to the Intel® Math Kernel Library  
numpy.core directory.

[numpy-1.13.3+mkl-cp27-cp27m-win32.whl](#)

[numpy-1.13.3+mkl-cp27-cp27m-win\\_amd64.whl](#)

[numpy-1.13.3+mkl-cp34-cp34m-win32.whl](#)

[numpy-1.13.3+mkl-cp34-cp34m-win\\_amd64.whl](#)

[numpy-1.13.3+mkl-cp35-cp35m-win32.whl](#)

[numpy-1.13.3+mkl-cp35-cp35m-win\\_amd64.whl](#)

[numpy-1.13.3+mkl-cp36-cp36m-win32.whl](#)

[numpy-1.13.3+mkl-cp36-cp36m-win\\_amd64.whl](#)

У назві файлів можна побачити тип операційної системи та її розрядність, наприклад win\_32 або win\_64.

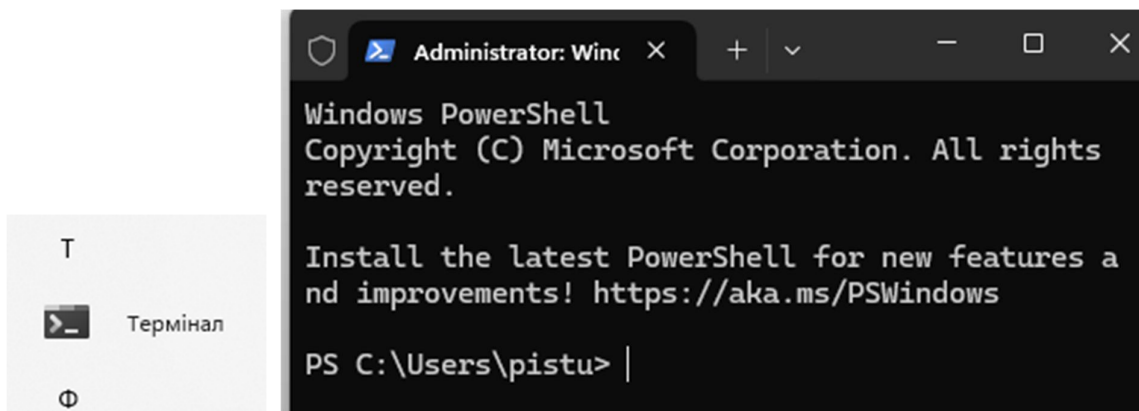
Всі бібліотеки мають свій опис, що саме вони роблять. Опис можна прочитати наприклад на сайті <https://pypi.python.org/pypi>.

Кожен пакет бібліотеки потрібно підключити до основного пакету Python.

Для того, щоб ця бібліотека діяла, необхідно скористатися спеціальною програмою-інсталятором **PIP**, яка автоматично постачається разом з інсталяційним пакетом Python.

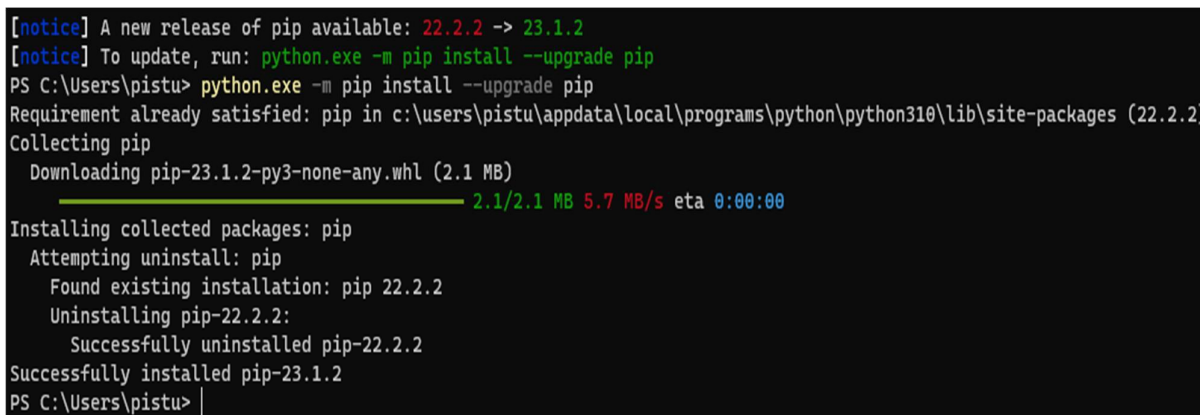
Установка бібліотеки виконується через Термінал, доступ до якого також здійснюється через кнопку «Windows». Всі програми, причому ваш комп'ютер має працювати в режимі «Адміністратор».

Вікно, що відкриється, має курсор, куди і буде записуватися команда інсталяції бібліотеки.



В деяких випадках, при першому звертанні до програми PIP, система запропонує скачати новішу версію – A new release of pip available. Погоджуйтеся на це, скориставшись інструкцією, що вказана нижче після запрошення, так, як показано нижче.

`python.exe -m pip install --upgrade pip`



Сама процедура установки нової бібліотеки зводиться до написання простої команди типу: **pip install pandas** або **pip install matplotlib**. Це приклади інсталяції бібліотек PUNDAS та MATPLOTLIB.

На рисунках, наведених нижче показано приклад такої роботи програми PIP для вказаних бібліотек.

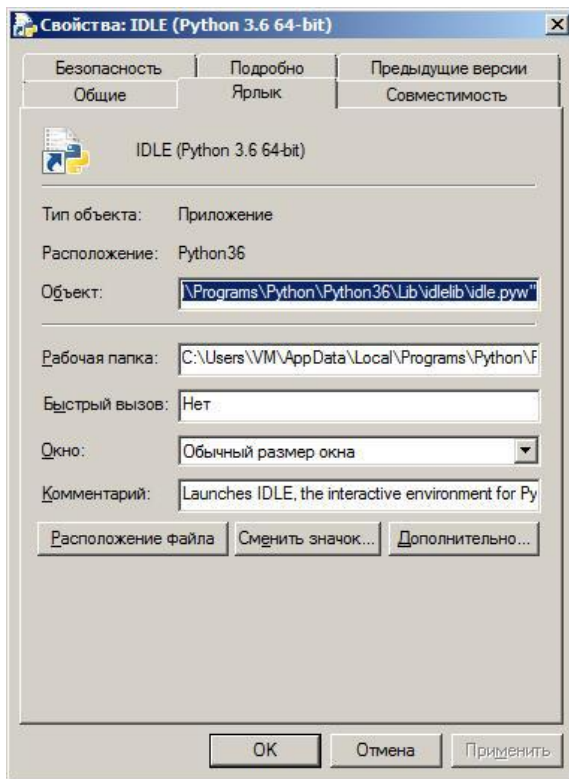
```
PS C:\Users\pistu> pip install pandas
Collecting pandas
  Downloading pandas-2.0.1-cp310-cp310-win_amd64.whl (10.7 MB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.7/10.7 MB 11.3 MB/s eta 0:00:00
Collecting python-dateutil>=2.8.2 (from pandas)
  Downloading python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 247.7/247.7 kB 14.8 MB/s eta 0:00:00
Collecting pytz>=2020.1 (from pandas)
  Downloading pytz-2023.3-py2.py3-none-any.whl (502 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 502.3/502.3 kB 10.5 MB/s eta 0:00:00
Collecting tzdata>=2022.1 (from pandas)
  Downloading tzdata-2023.3-py2.py3-none-any.whl (341 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 341.8/341.8 kB 10.7 MB/s eta 0:00:00
Collecting numpy>=1.21.0 (from pandas)
  Downloading numpy-1.24.3-cp310-cp310-win_amd64.whl (14.8 MB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 14.8/14.8 MB 11.5 MB/s eta 0:00:00
Collecting six>=1.5 (from python-dateutil>=2.8.2->pandas)
  Downloading six-1.16.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: pytz, tzdata, six, numpy, python-dateutil, pandas
Successfully installed numpy-1.24.3 pandas-2.0.1 python-dateutil-2.8.2 pytz-2023.3 six-1.16.0 tzdata-2023.3
```

```
PS C:\Users\pistu> pip install matplotlib
Collecting matplotlib
  Downloading matplotlib-3.7.1-cp310-cp310-win_amd64.whl (7.6 MB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.6/7.6 MB 10.2 MB/s eta 0:00:00
Collecting contourpy>=1.0.1 (from matplotlib)
  Downloading contourpy-1.0.7-cp310-cp310-win_amd64.whl (162 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 163.0/163.0 kB 9.5 MB/s eta 0:00:00
Collecting cyclor>=0.10 (from matplotlib)
  Downloading cyclor-0.11.0-py3-none-any.whl (6.4 kB)
Collecting fonttools>=4.22.0 (from matplotlib)
  Downloading fonttools-4.39.3-py3-none-any.whl (1.0 MB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 10.6 MB/s eta 0:00:00
Collecting kiwisolver>=1.0.1 (from matplotlib)
  Downloading kiwisolver-1.4.4-cp310-cp310-win_amd64.whl (55 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.3/55.3 kB ? eta 0:00:00
Requirement already satisfied: numpy>=1.20 in c:\users\pistu\appdata\local\programs\python\python310\lib\site-packages (from matplotlib) (1.24.3)
Collecting packaging>=20.0 (from matplotlib)
  Downloading packaging-23.1-py3-none-any.whl (48 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 48.9/48.9 kB ? eta 0:00:00
Collecting pillow>=6.2.0 (from matplotlib)
  Downloading Pillow-9.5.0-cp310-cp310-win_amd64.whl (2.5 MB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.5/2.5 MB 8.4 MB/s eta 0:00:00
Collecting pyparsing>=2.3.1 (from matplotlib)
  Downloading pyparsing-3.0.9-py3-none-any.whl (98 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.3/98.3 kB ? eta 0:00:00
Requirement already satisfied: python-dateutil>=2.7 in c:\users\pistu\appdata\local\programs\python\python310\lib\site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\pistu\appdata\local\programs\python\python310\lib\site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Installing collected packages: pyparsing, pillow, packaging, kiwisolver, fonttools, cyclor, contourpy, matplotlib
Successfully installed contourpy-1.0.7 cyclor-0.11.0 fonttools-4.39.3 kiwisolver-1.4.4 matplotlib-3.7.1 packaging-23.1 pillow-9.5.0 pyparsing-3.0.9
```

Така проста команда спрацює у випадку, коли інсталяційний пакет бібліотеки покладено у папку розташування програми PIP. Щоб знайти це місце розташування, натисніть ліву кнопку мишки на іконку програми IDLE у пакеті Python, яку ви маєте розмістити на робочому столі свого комп'ютера, та оберіть пункт «Властивості».

Там ви знайдете шлях до папки, що містить PIP.





Якщо ви вже скачали потрібну бібліотеку з вказаного вище сайту у якусь іншу папку, у прикладі це папка Install1 на диску D:\, то ви маєте вказати повний шлях до неї, наприклад:

D:\Install1\pandas-0.21.0rc1-cp36-cp36m-win\_amd64.whl.

Тобто загальний вигляд команди буде такий:

```
pip install D:\Install1\ pandas-0.21.0rc1-cp36-cp36m-win_amd64.whl.
```

Коли Пайтон встановлено, можна, але не обов'язково, встановити програму PyCharm, яка підключається до пакету Пайтона та дозволяє полегшити написання програм.

PyCharm Community Edition – це безкоштовне інтегроване середовище розробки (IDE) для мови програмування Python. Воно призначене для полегшення та прискорення процесу створення, налагодження та тестування Python-програм.

Серед можливостей середовища є низка інструментів для підвищення продуктивності праці програмістів: IDE підтримує автозавершення, що значно скорочує час, витрачений на написання коду. Інструмент також включає в себе функції статичного аналізу коду, які допомагають виявити потенційні помилки і підвищити якість програми, а PyCharm Community Edition забезпечує інтеграцію з системами контролю версій, такими як Git, що полегшує спільну роботу над проектами.



На додаток до базових функцій, PyCharm Community Edition надає розробникам можливість створювати віртуальні середовища для управління ізоляцією проекту та залежностями; IDE також включає в себе інструменти тестування коду та профілювання продуктивності.

#### Особливості PyCharm Community Edition

Безкоштовна ліцензія, доступна для всіх користувачів

Базовий набір інструментів для розробників (аналізатор, інтерпретатор, відладчик, підсвічування синтаксису, термінал, VCS, створення віртуальних середовищ, автозавершення, інтеграція з Git).

Інтуїтивно зрозумілий інтерфейс зі зручною навігацією та підсвічуванням синтаксису. Налаштовується за допомогою плагінів і тем.

#### Недоліки PyCharm Community Edition

Бракує деяких функцій, доступних у платній версії PyCharm.

Загалом, PyCharm Community Edition - це потужний безкоштовний інструмент для розробки на Python, який пропонує широкий спектр можливостей і зручний інтерфейс. Він ідеально підходить як для початківців, так і для досвідчених розробників, які хочуть підвищити продуктивність і поліпшити якість коду.

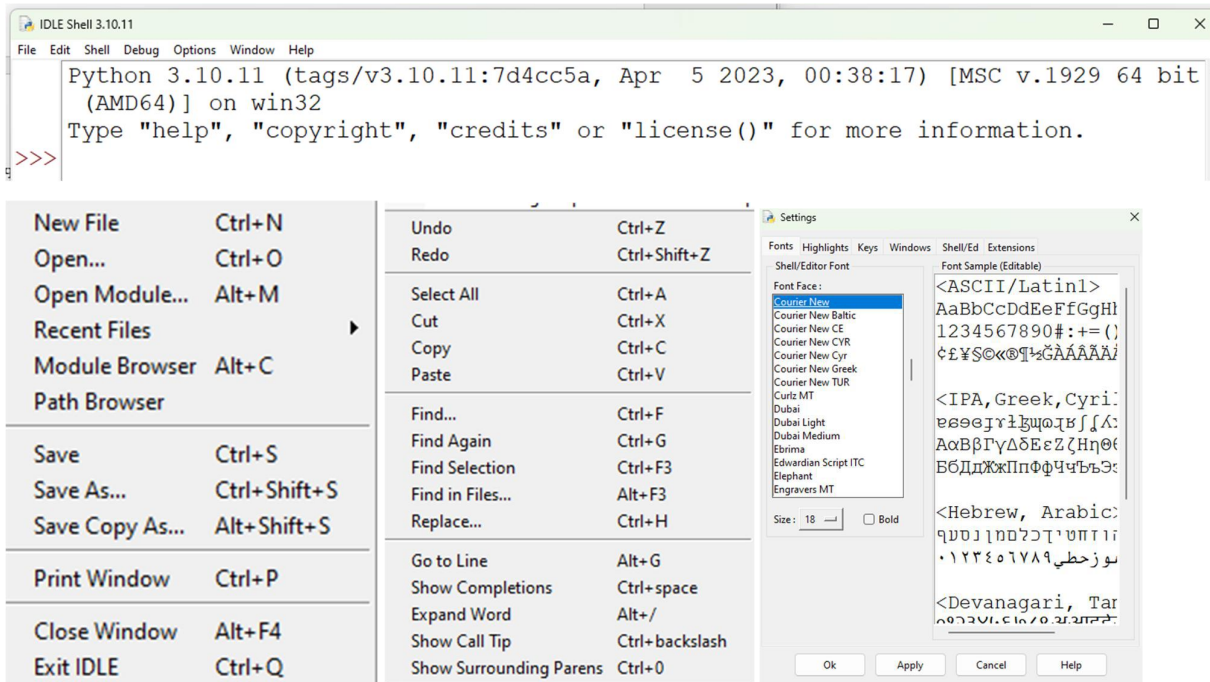
Існує платна та безоплатна версії програми, друга має обмежені можливості, але для початківця її буде цілком достатньо.

### **1.3. Порядок написання програм через інтерпретатор, Блокнот та PyCharm**

Після завантаження та установки Python відкриваємо IDLE (середовище розробки на мові Python, що поставляється разом з дистрибутивом). Ця аббревіатура означає (Integrated Development and Learning Environment) – інтегроване середовище для роботи і навчання.

Це програмне середовище є інтерпретатором, тобто, після написання якоїсь команди і натискання Enter, ця команда негайно виконується.

Головне меню IDLE мало відрізняється від меню будь якого текстового редактора. Пункт меню File забезпечує створення нових файлів, або відкриття старих з програмою. Пункт Edit теж має однакові з текстовими редакторами можливості копіювання, вставлення, вирізання частин тексту. Пошук певних фрагментів тексту тощо. За пунктом Options треба обрати Settings для налаштування приємного для вашого ока розміру та типу шрифту.

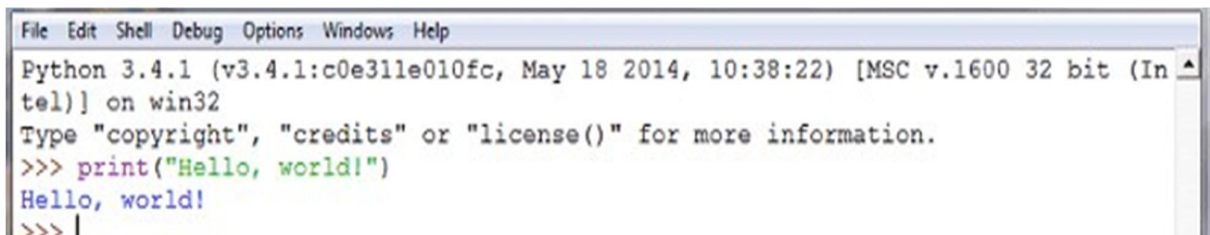


Традиційно, першою програмою у нас буде "hello world".

Щоб написати "hello world" на python, достатньо всього одного рядка:

```
print ("Hello world!")
```

Вводимо цей код в IDLE і натискаємо Enter. Результат видно на зображенні нижче.



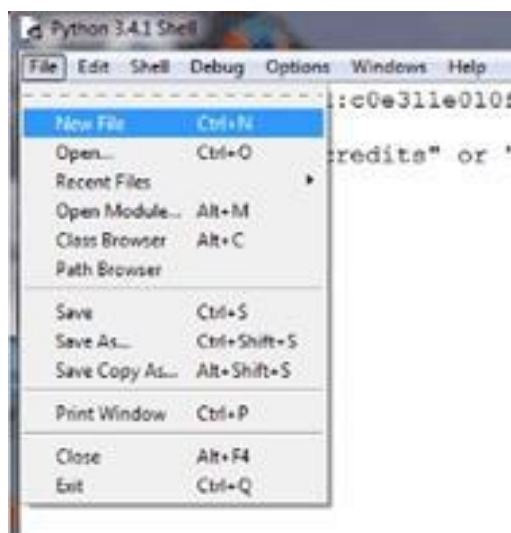
З інтерактивним режимом ми трохи познайомилися, можете з ним ще попрактикуватися, наприклад, написати:

```
print (3 + 4)
print (3 * 5)
print (3 ** 2)
```

Але, все-таки, інтерактивний режим не буде основним. В основному, ви будете зберігати програмний код у файлі та запускати вже файл.

Це тому, що програма складається з декількох команд, які мають виконуватися водночас, а не при натисканні Enter для кожної команди. Другою причиною є можливість використання створеної програми декілька разів, бо після закриття IDLE вся команди у ній пропадають. А програма, збережена у файлі, може бути виконана після її відкриття в IDLE.

Для того, щоб створити нове вікно з програмою, в інтерактивному режимі IDLE виберіть File → New File (або натисніть Ctrl + N).



Меню вікна нового файлу, де буде міститися програма не відрізняється від меню IDLE, у ньому теж діють ті самі пункти меню і ті самі гарячі клавіші. Але цей файл, при своєму виконанні буде видавати результати розрахунку в IDLE.

При наборі тексту програми варто одразу увімкнути нумерацію рядків через головне меню Options – Show Line Numbers.

У вікні, введіть наступний код:

```
name = input ("Як Вас звати?")
print ("Привіт,", name)
```

Перший рядок друкує питання ("Як Вас звати?"), очікує, поки ви не надрукуєте що-небудь і не натиснете *Enter* і зберігає введене значення в змінній *name*.

У другому рядку ми використовуємо функцію *print* для виведення тексту на екран, в даному випадку для виведення "Привіт," і того, що зберігається в змінній "name".

Тепер натиснемо F5 (або виберемо в меню IDLE Run → Run Module) і переконаємося, що те, що ми написали, працює. Перед запуском IDLE запропонує нам зберегти файл. Збережемо туди, куди вам буде зручно, після чого програма запуститься.

Ви повинні побачити щось на зразок цього (на скріншоті зліва – файл з написаної вами програмою, праворуч – результат її роботи).

```
*05.py - D:/02 Навчальний процес/00 Електронна_...
File Edit Format Run Options Window Help

name=input("Як ваше ім'я ?")
print("Привіт", name)

Python 3.6.0a3 Shell
File Edit Shell Debug Options Window Help
Python 3.6.0a3 (v3.6.0a3:f3edf13dc339, Jul 11 2016, 21:40:
(AMD64) on win32
Type "copyright", "credits" or "license()" for more inform
>>>
RESTART: D:/02 Навчальний процес/00 Електронна_.../Елект
05.py
Як ваше ім'я ? Ігор
Привіт Ігор
>>>
```

Якщо текст програми містить помилки, IDLE повідомляє про це текстом англійською мовою зі вказанням, у якому рядку тексту знаходиться ця помилка і що це за помилка..

```
Traceback (most recent call last):
  File "E:\02 Навчальний процес\& Пайтон\Блокнот.py", line 2, in
<module>
    name = iienput ( "Як Вас звати?")
NameError: name 'iienput' is not defined
```

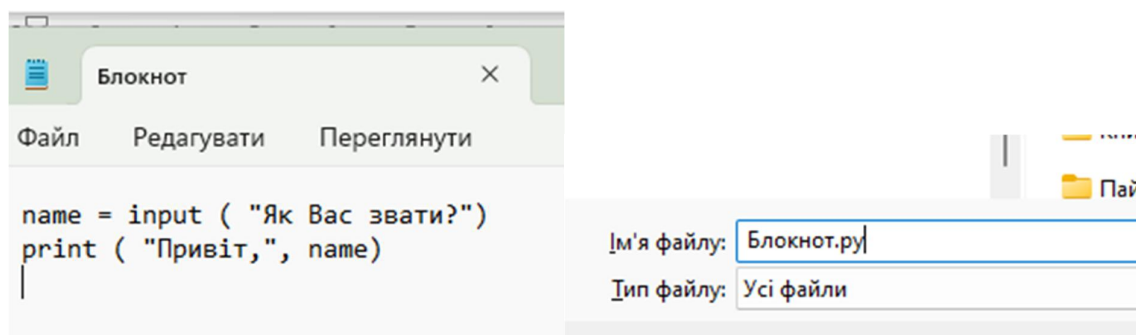
Зауважимо, що гарячі клавіші працюють тільки, якщо у вас на комп'ютері увімкнена мова ENG.

Тепер цей файл можна зберегти з будь яким іменем. При цьому його розширення буде \*.py. І наступного разу цю програму можна буде викликати через IDLE гарячими клавішами Ctrl + O.

Ще одна особливість відкривання файлів цією командою. Якщо ви її використовуєте через IDLE, то початкове вікно файл-менеджера покаже вам папку, де знаходиться програмне забезпечення Пайтона. Якщо ви це зробите через вікно файлу з програмою, то файл менеджер покаже вам папку, звідкіля цей файл було взято.

Оскільки програма на Пайтоні є простим текстом, її можна створювати на вбудованому у Windows редакторі Блокнот. Нижче подано приклад такої програми.

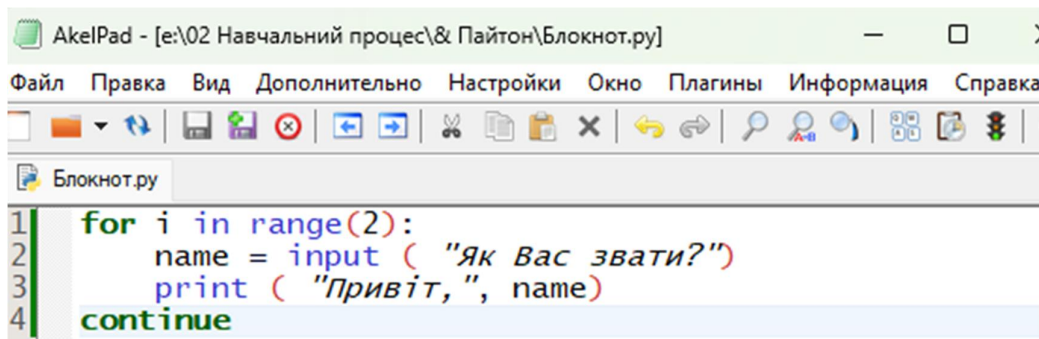
Зауважимо, що при збереженні написаної програми потрібно типом файлу обирати – Всі файли, а в імені файлу після крапки додавати префікс ру.



На відміну від редактора під управлінням IDLE, гарячі клавіші працюють і при увімкненій мові UKR.

Але цей редактор не розфарбовує окремі елементи тексту, що робить його гірше доступним для перевірки правильності коду. Також, тут відсутні підказки при написанні команд, які з'являються при роботі з редактором під управлінням IDLE.

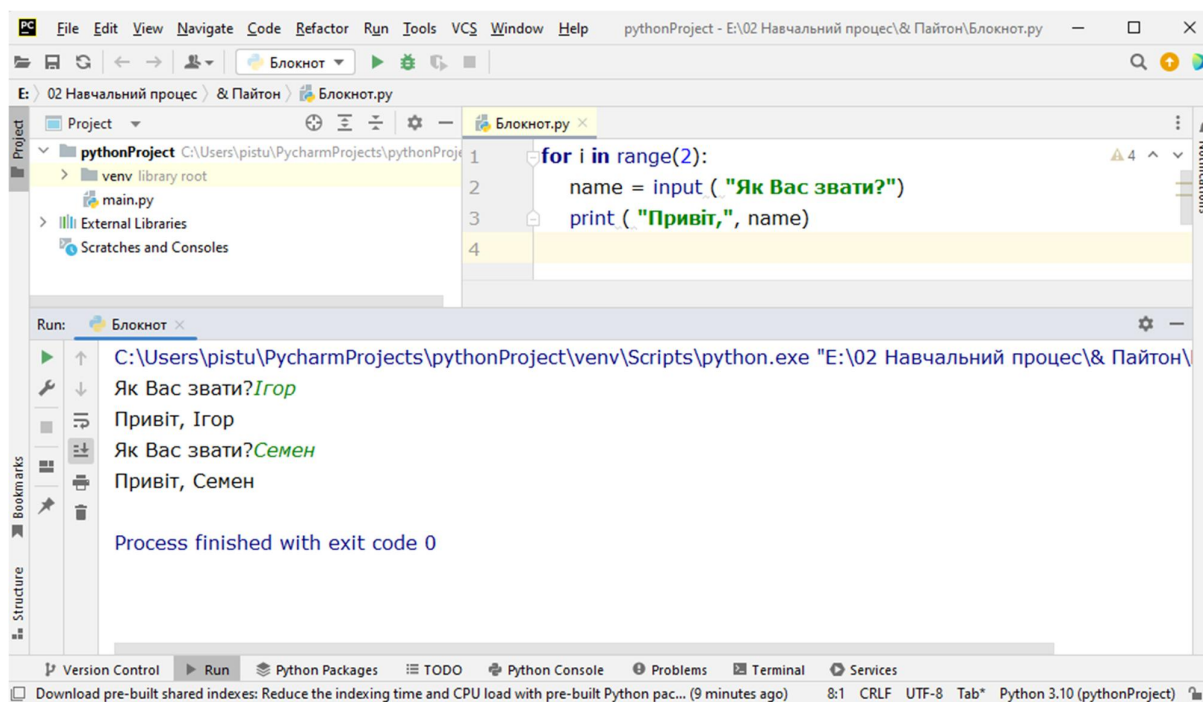
В якості альтернативи Блокноту, можна скористатися текстовим редактором AkeIPud файл-менеджера Total Comander. Цей редактор вмикається натисканням кнопки F4.






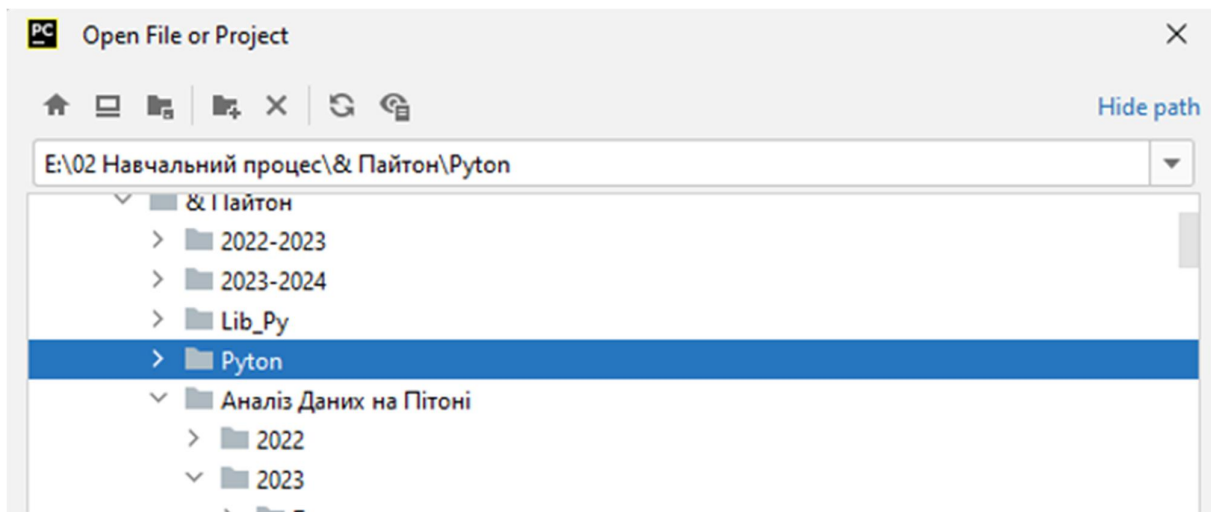
Як видно з прикладу, цей редактор подає кольорову розмітку тексту програми, показує номери рядків тексту, а також підтримує гарячі клавіші при увімкненій мові UKR. Але підказок до формату команд теж не дає.

Розглянемо можливості PyCharm. Інтерфейс програми складається з головного меню і трьох вікон. Перше – це список файлів, з якими працює програма, друге, текст файлу (закладки над цим вікном дозволяють перемикаватися між різними відкритими файлами), а третє – результат роботи програми.



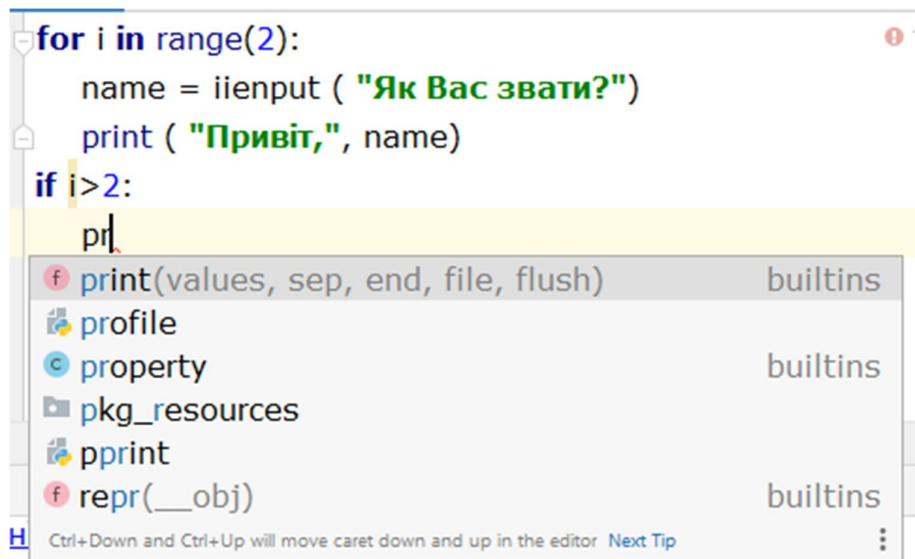
Гарячі клавіші у PyCharm теж працюють при увімкненій мові UKR.

Щоб відкрити новий файл гарячі клавіші Ctrl+O не працюють, потрібно натиснути мишкою іконку , або обрати по меню File – Open. Відкриється вікно папки, в якій були вміщені файли минулого разу.



Всі інші гарячі клавіші та пункти меню аналогічні редактору IDLE.


Набираючи текст ви отримуєте підказки по імені команд у вигляді списку, я кого можна одразу обрати необхідну. Це прискорює процес програмування.



При спробі виконати програму, яка містить явні помилки, перевірки синтаксису не відбувається, програма одразу виконується, а вже потім з'являється повідомлення про помилку.

```
File "E:\02 Навчальний процес\& Пайтон\Блокнот.py", line 2, in <module>
  name = ienput ( "Як Вас звати?")
NameError: name 'ienput' is not defined. Did you mean: 'input'?

Process finished with exit code 1
```

Зате збереження набраного тексту виконується автоматично і постійно, варто змінити хоча б один символ у програмі. Тому іконка запису програми на диск , як і пункт меню File- Save As виглядають зайвими.

Досвід практичного застосування безоплатної версії цієї програми показує, що інколи програми, що створені і перевірені на редакторі IDLE, не завжди працюють на PyCharm, про що ніде не повідомляється.

Висновок цього пункту:

1. Тексти програм можна писати на всіх, описаних вище редакторах.
2. Перевірку програми на працездатність (відладку) треба виконувати тільки на IDLE.
3. Для швидкого перегляду тексту програми найзручніше використати редактор AkelPud файл-менеджера Total Comander.

#### **1.4. Структура програми, вкладені оператори, коментарі**

Програма – це набір інструкцій для виконання комп'ютером.

Програма пишеться, як звичайний текст, використовуючи спеціальні слова, частіше, англійської мови.

Ці спеціальні слова називаються командами, або операторами, або скриптами, або тегами, або кодами. Наприклад print – друкувати, do – виконувати, тощо.

Команди заставляють комп'ютер виконувати різні операції з даними, що містяться в різних комірках пам'яті комп'ютера.

Ці комірки називаються змінними або параметрами. Щоб їх відрізнити одне від одного, програмісти надають їм імена. Ці імена можуть складатися з довільного набору великих і маленьких букв латинського алфавіту та цифр.

Зміна регістру букви сприймається комп'ютером як різні імена. Наприклад, Name і name – це різні імена. Або A1 та a1 – теж.

Команди записуються тільки маленькими літерами. Кожна команда частіше за все закінчується круглими дужками, в яких знаходять імена змінних, написані через кому, та деякі параметри команди. Але бувають виключення. Наприклад, команда for дужок не має, а команда print – має.

```
1 for i in range(2):
2     name = input("Як Вас звати?")
3     print("Привіт,", name)
```

Для зручності, редактор розмальовує команди у різні кольори. Ті команди що мають дужки – фіолетовим. І такі команди іще називаються «функція». Ті команди, що не мають дужок – помаранчевим, і називаються «оператор». Текст зображується зеленим. Така кольорова розмітка дозволяє швидко орієнтуватися у тексті, знаходити необхідні фрагменти та шукати помилки.

Всі команди записуються з першої позиції рядка. Кожна команда записується, як правило, в окремому рядку.

Кінець рядка є кінцем інструкції (крапка з комою не потрібно).

Іноді можливо записати кілька інструкцій в одному рядку, розділяючи їх крапкою з комою. Ось два приклади вірного і рівноцінного написання команд.

	File	Edit	Format	Run	Options
1	a = 1				
2	b = 2				
1	a = 1; b = 2; print(a, b)				
3	print(a, b)				

Але не робіть це занадто часто! Пам'ятайте про зручність читання коду.

Одна команда може бути вкладена в іншу, але синтаксис їх вкладення залежить від типу команди.

Якщо це команда типу функція (пам'ятаєте, фіолетового кольору), то інша команда вставляється в неї як і проста змінна. Наприклад, тут в команду (функцію) `print` вставляється завдання надрукувати значення змінних `a` та `b`, а також, значення логарифма `b`. Тобто, в одну функцію вставлена іще одна.

```
3|print (a, b, log(b))
```

Якщо це команда типу «оператор» (пам'ятаєте, помаранчеві), то вкладення в неї інших операторів виконується за наступним правилом: вкладені інструкції об'єднуються в блоки по величині відступів. Відступ може бути будь-яким, головне, щоб в межах одного вкладеного блоку відступ був однаковий. І про зручність читання коду не забувайте. Відступ в 1 пробіл, наприклад, не найкраще рішення. Використовуйте 4 пробіли (або знак табуляції кнопкою Tab).

Один відступ – перший рівень вкладення, два – другий.

Вкладені інструкції в Python записуються відповідно до одного й того ж шаблону, коли основна інструкція завершується двокрапкою, слідом за яким розташовується вкладений блок коду, зазвичай з відступом під рядком основний інструкції.

Основна інструкція:

Вкладений блок інструкцій

Повторимо приклад, в якому інструкція першого рівня `for`, містить 2 вкладених оператора типу функція – `input` та `print`.

```
1|for i in range(2):
2|    name = input ("Як Вас звати?")
3|    print ("Привіт,", name)
```

Припустимо записувати одну інструкцію в декількох рядках. Досить її укласти в пару круглих, квадратних або фігурних дужок, Нижче подано приклад двох рівноцінних способів запису інструкції для оператора end, об'єднаних дужкою.

```
1 a=2;b=4;c=8;d=10
2 if (a == 1
3     and b == 2
4     and c == 3
5     and d == 4):
6     print('spam' * 3)
```

```
1 a=2;b=4;c=8;d=10
2 if (a == 1 and b == 2 and c == 3 and d == 4):
3     print('spam' * 3)
```

Тіло складової інструкції може розташовуватися в тому ж рядку, що і тіло основної, якщо тіло є складової інструкції і не містить складових інструкцій. Нижче подано два рівноцінних варіанти запису вкладених операторів: в оператор if, вкладено функцію print.

<pre>File Edit Format Run Options Window Help 1 x=1 2 y=2 3 if x&gt; y: 4     print(x)</pre>	<pre>File Edit Format Run Options Window Help 1 x=1 2 y=2 3 if x&gt; y: print(x)</pre>
--	--

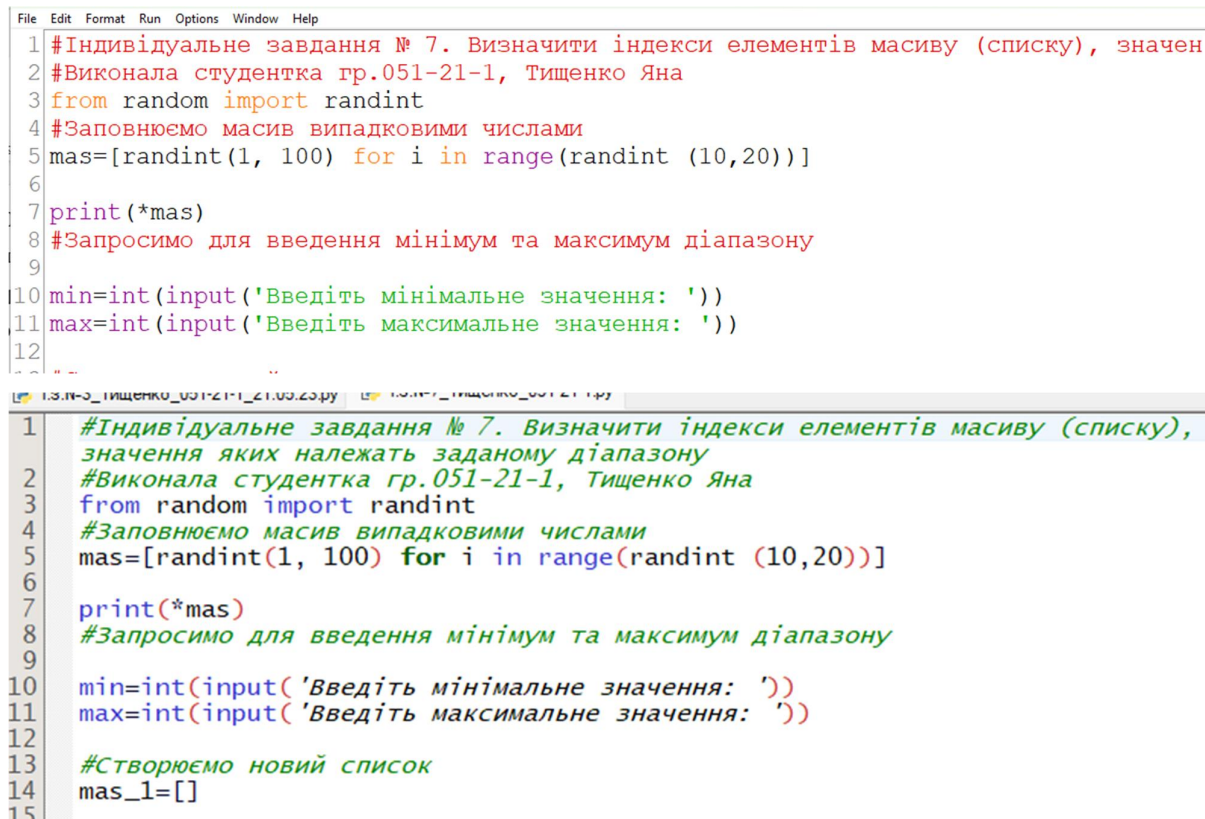
Знак # означає коментар, що не обробляється Python і потрібен тільки для того, щоб описати складні елементи програм. В редакторі IDLE коментарі позначаються червоним, а в редакторі AkelPad файл-менеджера Total Commander – зеленим.

Коментарі в програмі дуже важливі і необхідні для самого програміста, адже варто програмі стати довшою за 10-15 рядків, як ви в ній вже будете плутатися. Тому



програма повинна мати опис усі змінних, мети розрахунку, методів, що в ній застосовуються, тощо.

У прикладі однієї і тієї ж програми, записаної в різних редакторах, ви бачите виконання всіх вимог.



```
File Edit Format Run Options Window Help
1 #Індивідуальне завдання № 7. Визначити індекси елементів масиву (списку), значен
2 #Виконала студентка гр.051-21-1, Тищенко Яна
3 from random import randint
4 #Заповнюємо масив випадковими числами
5 mas=[randint(1, 100) for i in range(randint (10,20))]
6
7 print (*mas)
8 #Запросимо для введення мінімум та максимум діапазону
9
10 min=int(input('Введіть мінімальне значення: '))
11 max=int(input('Введіть максимальне значення: '))
12
13
14
15
1 #Індивідуальне завдання № 7. Визначити індекси елементів масиву (списку),
2 значення яких належать заданому діапазону
3 #Виконала студентка гр.051-21-1, Тищенко Яна
4 from random import randint
5 #Заповнюємо масив випадковими числами
6 mas=[randint(1, 100) for i in range(randint (10,20))]
7
8 print(*mas)
9 #Запросимо для введення мінімум та максимум діапазону
10
11 min=int(input('Введіть мінімальне значення: '))
12 max=int(input('Введіть максимальне значення: '))
13
14 #Створюємо новий список
15 mas_1=[]
```

Тепер, коли програміст захоче створити нову програму, де буде визначення індексів елементів масиву, він легко знайде необхідні команди вставить їх у нову програму.

Інколи студенти запитують про вплив пробілів між окремими елементами інструкції: назвою (типу print або for), дужками, комами, тощо.

Відповідь на це запитання проста:

1. Всі слова, що означають тип команди, імена комірок (змінні), текст, який ви плануєте показувати оператору вашої програми пишуться без пробілів. Не можна написати **p r i n t** замість **print** або **n a m e** замість **name**.
2. Поміж всіма цими елементами може бути довільна кількість пробілів.

Ось приклад, що це ілюструє. Нижче подано два варіанти однієї програми, що виводить два числа і текст. У першому відсутні пробіли між елементами, у другому – вставлено по 2 - 4 пробіли. Результат роботи програми від цього не змінився.

```
File Edit Format Run Options Window Help
1 a=1; b=2
2 print(a,b, 'Це перевірка')
3

File Edit Format Run Options Window Help
1 a = 1; b = 2
2 print ( a , b , 'Це перевірка' )
3

===== RESTART: E:/02
1 2 Це перевірка

===== RESTART: E:/02
1 2 Це перевірка
|
```

### Контрольні запитання

1. Які основні кроки потрібно виконати для інсталяції Python на комп'ютері?
2. Які є доступні варіанти для встановлення Python на операційну систему Windows, macOS та Linux?
3. Як перевірити, чи встановлено Python на комп'ютері, і яка версія встановлена?
4. Що таке інтерпретатор Python і які він має функції?
5. Які основні бібліотеки Python ви знаєте і для чого вони використовуються?
6. Які є різниці між встановленням бібліотек підпрограм за допомогою pip та вручним завантаженням та встановленням їх з джерела?

7. Як перевірити, чи встановлено певну бібліотеку підпрограм у вашому середовищі Python?
8. Якщо виникає проблема з встановленням бібліотеки підпрограм, які кроки ви зазвичай вживаєте для її вирішення?
9. Що таке віртуальне середовище Python і які його переваги?
10. Які є основні кроки для створення віртуального середовища Python, і як воно допомагає у керуванні залежностями вашого проекту?
11. Що таке блок коду в мові програмування Python і як ви його визначаєте?
12. Які ключові слова використовуються для визначення початку та кінця блоків коду в Python?
13. Які типи вкладених операторів ви знаєте в Python, і як вони використовуються?
14. Як ви розумієте поняття вкладеного оператора у Python, і чому вони використовуються?
15. Що таке коментарі в програмі Python, і як вони допомагають програмістам?
16. Як ви використовуєте коментарі для документування вашого коду, і чому це важливо для розробників?

*Опанувавши матеріали розділу студенти навчилися встановлювати Пайтон на власному комп'ютері, узнали структуру програм на Пайтоні та вивчили програми, що дозволяють писати Python-коди.*

# **Розділ 2.**

## **ДАНІ МОВИ ПРОГРАМУВАННЯ ПАЙТОН**

*Вивчення матеріалів цього розділу допоможе студентам оперувати з текстом, числами, та їх масивами.*

Кожен економіст оперує з інформацією, яку необхідно обробити, обрахувати і прийняти рішення. Мова програмування Пайтон має значну кількість різних типів даних, але для потреб економістів достатньо тільки два: тексти та числа.

### **2.1. Тексти (літерали) та операції з ними**

Тексти (літерали) в Python – впорядковані послідовності символів, що використовуються для зберігання і представлення текстової інформації, тому за допомогою рядків можна працювати з усім, що може бути представлено в текстовій формі. Цифри теж можуть бути у цьому тексті.

Рядки можуть бути представлені в апострофах і в лапках, наприклад

```
S = 'spam "s'
```

```
S = " spam's "
```

Для української мови актуальним є виведення тексту з апострофами. Для цього варто використати лапки для позначення початку і закінчення тексту, але апостроф ставити там, де вимагає орфографія. Не можна змішувати різні лапки та апострофи для позначення одного і того ж тексту, ось приклад.

```

>>> 'ljhg"
SyntaxError: incomplete input
>>> 'lkjg{"ljh'
'lkjg{"ljh'
>>> "ljhg'kj"
"ljhg'kj"

```

Рядки в апострофах і в лапках – одне і те ж. Причина наявності двох варіантів в тому, щоб дозволити вставляти в літерали рядків символи лапок і апострофів, не використовуючи екранування.

Для виконання додаткових ефектів при виводі тексту, використовують спеціальні символи, які починаються зі знака «зворотній слеш». Їх назвали екранованими послідовностями. Вони дозволяють вставити символи, які складно ввести з клавіатури.

Екранована послідовність	Призначення
\n	Переведення рядка
\a	Дзвінок
\b	Повернення курсора рядку на одну позицію ліворуч
\f	Переведення сторінки
\r	Повернення каретки
\t	Горизонтальна табуляція
\v	Вертикальна табуляція
\0	Символ Null (не є ознакою кінця рядку)

Перевіримо, як вони працюють.

Як видно з прикладу, реально працюють тільки символи переведення рядка та горизонтальної табуляції.

Якщо перед лапками, що відкривають рядок символів, стоїть

```

>>> print(';sjk\n;wej')
;sjk
;wej
>>> print(';sjk\a;wej')
;sjk!;wej
>>> print(';sjk \b;wej')
;sjk ;wej
>>> print(';sjk\f;wej')
;sjk␣;wej
>>> print(';sjk\r;wej')
;sjk;wej
>>> print(';sjk\t;wej')
;sjk      ;wej
>>> print(';sjk\v;wej')
;sjk!;wej

```





слова «Україна» має номер нуль, а останній – номер 6. Та коли починається підрахунок під кінця рядку, остання «а» вважається першим символом, а передостання «н» – другим.

З текстового рядка можна витягнути якусь його частину, або як кажуть, «витягнути зріз рядка».

Оператор вилучення зрізу: [X: Y]. X – початок зрізу, а Y – закінчення; символ з номером Y в зріз не входить. За замовчуванням перший індекс дорівнює 0, а другий – довжині рядка. Тому, коли один або і всі індекси не вказані, Пайтон їх обирає згідно вказаного правила.

```
>>> s='Україна вільна назавжди!'
>>> s[8:13]
'вільн'
>>> s[:15]
'Україна вільна '
>>> s[8:]
'вільна назавжди!'
>>> s[:]
'Україна вільна назавжди!'
```

Після того, як було вказано діапазон зрізу, можна через двокрапку задати крок, з яким потрібно витягувати зріз. Знак кроку вказує на порядок, у якому будуть рахуватися символи: якщо мінус, то підрахунок іде з кінця рядку.

```
>>> s[:]
'Україна вільна назавжди!'
>>> s[::-1]
'!иджвазан аньлів анїаркУ'
>>> s[8::-2]
'влн аажі'
```

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. Коли ми хочемо змінити якийсь символ в уже існуючому тесті, потрібно витягти методом зрізу ті частини тексту, що

залишаються незмінними і виконати конкатенацію цих частин з новим СИМВОЛОМ.

```
>>> s='Україна вільна назавжди!'
>>> s=s[:6]+' - це країна вільних людей'
>>> s
'Україн - це країна вільних людей'
```

Повний набір функцій та методів обробки рядків представлено у наступній таблиці, але пам'ятаючи приклад з екранованими послідовностями, очевидно, що не всі функції можуть працювати точно за описом. Кожну функцію перед застосуванням варто перевірити на маленькому прикладі.

Функція або метод	Призначення
S = 'str'; S = "str"; S = ''str''; S = ""str""	Літерали рядків
S = "s\np\ta\nbbb"	Екрановані послідовності
S = r"C:\temp\new"	Не форматовані рядки (пригнічують екранування)
S = b"byte"	Рядок байтів
S1 + S2	Конкатенація (складання рядків)
S1 * 3	Повтор рядків
S[i]	Звернення по індексу
S[i:j:step]	Витяг зрізу
len(S)	Довжина рядка
S.find(str, [start],[end])	Пошук підрядка в рядку. Повертає номер першого входження або 1
S.rfind(str, [start],[end])	Пошук підрядка в рядку. Повертає номер останнього входження або 1
S.index(str, [start],[end])	Пошук підрядка в рядку. Повертає номер першого входження або викликає ValueError
S.rindex(str, [start],[end])	Пошук підрядка в рядку. Повертає номер останнього входження або викликає ValueError
S.replace(шаблон, замена)	Заміна шаблону
S.split(символ)	Розбиття рядку по роздільнику
S.isdigit()	Визначає, чи складається рядок із цифр
S.isalpha()	Визначає, чи складається рядок із букв

Функція або метод	Призначення
<code>S.isalnum()</code>	Визначає, чи складається рядок із цифр чи з букв
<code>S.islower()</code>	Визначає, чи складається рядок із символів в нижньому регістрі
<code>S.isupper()</code>	Визначає, чи складається рядок із символів в верхньому регістрі
<code>S.isspace()</code>	Визначає, чи складається рядок із символів, що не відображаються в тексті (пробіл, символ Переведення у сторінки ( <code>'\f'</code> ), “новий рядок” ( <code>'\n'</code> ), “Переведення каретки” ( <code>'\r'</code> ), “горизонтальна табуляція” ( <code>'\t'</code> ) и “вертикальна табуляція” ( <code>'\v'</code> ))
<code>S.istitle()</code>	Визначає, чи починаються слова в рядку з заголовної букви
<code>S.upper()</code>	Перетворення рядку до верхнього регістру
<code>S.lower()</code>	Перетворення рядку до нижнього регістру
<code>S.startswith(str)</code> <code>S.endswith(str)</code>	Визначає, чи починаються рядок <code>S</code> з шаблону <code>str</code> Визначає, чи закінчується рядок <code>S</code> з шаблону <code>str</code>
<code>S.join(список)</code> <code>ord(символ)</code>	Збирання рядку зі списку з розділювачем <code>S</code> Переведення символу у код ASCII
<code>chr(число)</code> <code>S.capitalize()</code>	Переведення коду ASCII у символ Переведення першого символу рядка у верхній регістр, а всі інших – у нижній
<code>S.center(width, [fill])</code>	Повертає відцентрований рядок, по краям якого стоїть символ <code>fill</code> (пробіл за замовчуванням)
<code>S.lstrip([chars])</code> <code>S.rstrip([chars])</code>	Видаляє символи пробілів на початку рядка Видаляє символи пробілів у кінці рядка
<code>S.strip([chars])</code>	Видаляє символи пробілів на початку та у кінці рядка
<code>S.swapcase()</code>	Переведення символу нижнього регістру у верхній, а верхнього – у нижній
<code>S.title()</code>	Першу букву кожного слова Переведення у верхній регістр, а всі інші – у нижній

Досить часто виникають ситуації, коли потрібно зробити рядок, підставивши в нього деякі дані, отримані в процесі виконання програми (призначене для користувача введення, дані з файлів і т.д.). Підстановку даних

можна зробити за допомогою форматування рядків. Форматування можна зробити за допомогою оператора %, і методу *format*.

Специфікація цього оператора:

специфікація ::= [заповнювач] [вирівнювання] [знак] [#] [0] [ширина] [,]  
[. точність] [тип]

Пояснення специфікації:

заповнювач ::= символ крім '{' або '}'

вирівнювання ::= "<" | ">" | "=" | "^"

знак ::= "+" | "-" | ""

ширина ::= integer

точність ::= integer

тип ::= "b" | "C" | "D" | "E" | "E" | "F" | "F" | "G" | "G" | "N" | "O" | "S" | "X" | "X" |  
"%"

Вирівнювання проводиться за допомогою символу-заповнювача.

Доступними параметрами вирівнювання є:

Флаг	Значення
'<'	Символи, що заповнюють, будуть праворуч (вирівнювання об'єкту по лівому краю) (за замовчуванням).
'>'	Вирівнювання об'єкту по правому краю.
'='	Заповнювач буде після знаку, але перед цифрами. Працює тільки з числовими типами.
'^'	Вирівнювання по центру.

Опція “знак” використовується тільки для чисел і може приймати наступні значення:

Флаг	Значення
'+'	Знак має бути використаний для всіх чисел.
'-'	'-' для від'ємних, нічого для додатних,
'Пробіл'	'-' для від'ємних, пробіл для додатних,

Поле “тип” може приймати наступні значення:

Тип	Значення
'd', 'i', 'u'	Десяткове число.
'o'	Число в восьмеричній системі числення.
'x'	Число в шестнадцятковій системі числення (букви у нижньому регістрі).
'X'	Число в шестнадцятковій системі числення (букви у верхньому регістрі).
'e'	Число з плаваючою точкою з експонентою (експонента у нижньому регістрі).
'E'	Число з плаваючою точкою з експонентою (експонента у верхньому регістрі).
'f', 'F'	Число з плаваючою точкою (звичайний формат).
'g'	Число з плаваючою точкою з експонентою (експонента у нижньому регістрі), якщо воно менше, за -4 або точність, інакше – звичайний формат.
'G'	Число з плаваючою точкою з експонентою (експонента у верхньому регістрі, якщо воно менше, за -4 або точність, інакше – звичайний формат.
'c'	Символ (рядок із одного символу або число – код символу).
's'	Рядок.
'%'	Число множиться на 100, відображається число з плаваючою точкою, а за ним знак %.

Форматування рядків за допомогою методу *format* виконується, якщо для підстановки потрібно тільки один аргумент, то значення – сам аргумент:

```
>>> 'Hello, {}'.format('Vasya')
```

```
'Hello, Vasya!'
```

А якщо кілька, то значеннями будуть усі аргументи з рядками підстановки (звичайних або іменованих):

```
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'
>>> '{2}, {1}, {0}'.format('a', 'b', 'c')
'c, b, a'
```

```

>>> '{0}, {1}, {0}'.format('абра', 'кад')
'абра, кад, абра'
>>> '{0} {1} {0}'.format('абра', 'кад')
'абра кад абра'
>>> '{0}{1}{0}'.format('абра', 'кад')
'абракадабра'

```

Як ви помітили з цього прикладу, коми та пробіли поміж фігурних дужок теж друкуються, тому їх потрібно ставити тільки з необхідності.

Будь який набір символів можна перетворити на тест командою `str`, наприклад

```

>>> a=4.95e-12
>>> b=str(a)
>>> b[3]
'5'

>>> b[-4]
'e'

```

Тут число з плаваючою точкою в експоненціальному форматі, який означає  $4,95 \cdot 10^{-12}$ , було перетворено у тестовий формат, з яким можна робити всі операції, як і з текстом.

## 2.2. Числа цілі, дробові, з плаваючою точкою та операції з ними

Числа є основою інформації, що її обробляють економісти. У більшості випадків таким є числа цілі та дробові. Останні мають дробову частину десяткового дробу, відділену від цілої частини комою. Але у Пайтоні використовується англійська система позначення десяткового дробу, в якій десятковий дріб відділяється від цілої частини крапкою. На це потрібно постійно вважати при створенні програми. Не часто економісти використовують числа з



плаваючою точкою типу  $4,95 \cdot 10^8$ , коли потрібно показати дуже великі числа. Але їх опис також буде подано у цьому пункті.

### 2.2.1. Цілі числа

До формату цілого числа можна привести будь яке число, скориставшись функцією `int()`. Наприклад, якщо у нас є дробове число, або текст, що містить тільки цифри, їх можна перетворити у ціле число.

```
>>> a=4.5
>>> print(int(a))
4
>>> a='285'
>>> print(int(a))
285
```

Числа в Python нічим не відрізняються від звичайних чисел. Вони підтримують набір звичайнісіньких математичних операцій:

<code>x + y</code>	Додавання
<code>x - y</code>	Віднімання
<code>x * y</code>	Множення
<code>x / y</code>	Ділення
<code>x // y</code>	Ціла частина від ділення
<code>x % y</code>	Решта від ділення
<code>-x</code>	Зміна знаку числа
<code>abs(x)</code>	Модуль числа
<code>divmod(x, y)</code>	Пара ( <code>x // y</code> , <code>x % y</code> )
<code>x ** y</code>	Зведення в степінь
<code>pow(x, y[, z])</code>	$x^y$ по модулю (якщо модуль задано)
<code>help([object])</code>	виклик вбудованої довідкової системи.
<code>abs()</code> , <code>fabs()</code>	Обчислити абсолютне значення цілих, дійсних або комплексних елементів масиву. Для речових даних <code>fabs</code> працює швидше.

Також потрібно відзначити, що числа в Python, на відміну від багатьох інших мов, підтримують довгу арифметику (проте, це вимагає більше пам'яті).

Наприклад:

```
>>> 255+168
423
>>> 18*466
8388
>>> 3**150
369988485035126972924700782451696644186473100389722
973815184405301748249
>>> 20//3
6
>>> 20%3
2
```

Над цілими числами також можна виконувати бітові операції

$x   y$	Побітове або
$x \wedge y$	Побітове або, що виключає
$x \& y$	Побітове і
$x \ll n$	Бітовий зсув уліво
$x \gg y$	Бітовий зсув управо
$\sim x$	Інверсія бітів

Системи числення у мові Python: двійкова, вісімкова, десятична та шістнадцяткова. Якщо потрібно Переведення числа з однієї системи числення в іншу, то Python для цього надає кілька функцій:

`int([object], [основа системи числення])` – перетворення до цілого числа в десятковій системі числення. За замовчуванням система числення десяткова, але можна задати будь-яку основу від 2 до 36 включно.

`bin(x)` – перетворення цілого числа в двійковий рядок.

`hex(x)` – перетворення цілого числа в шістнадцятковий рядок.

`oct(x)` – перетворення цілого числа в вісімковий рядок.

Приклади:

```

>>> bin(8)
      '0b1000'
>>> hex(8)
      '0x8'
>>> oct(8)
      '0o10'

```

### 2.2.2. Дійсні числа

Дійсні числа мають тип float. Цією функцією можна перевести у формат дійсного числа ціле число або текст, що містить тільки цифри.

```

>>> a='40'
>>>
>>> print(float(a))
      40.0
>>> a=40
>>> print(float(a))
      40.0

```

Дійсні числа підтримують ті ж операції, що і цілі. Однак (через представлення чисел в комп'ютері) дійсні числа неточні, і це може привести до помилок:

```

>>> 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
      + 0.1
      0.9999999999999999

```

Для потреб економістів достатньо тільки два десяткових знаки, які означають суму в копійках. Тому при розрахунках сум варто скористатися функцією round(число, точність числа).

Наприклад, якщо число має велику мантису, її можна округлити до потрібної кількості знаків. У випадку розрахунків з матеріальними активами,

округлення робити не варто, оскільки зміна кількості у декілька тон, якщо розрахунок ведеться у тисячах тон, округлення може викликати втрату грошей.

```
>>> round(4.456987, 2)
4.46
>>> a=13
>>> b=7
>>> round(a/b, 2)
1.86
>>> a/b
1.8571428571428572
```

Простенькі приклади роботи з числами:

```
>>> c = 150
>>> d = 12.9
>>> c + d
162.9
>>> p = abs(d - c) # Модуль числа
>>> print(p)
137.1
>>> round(p) # Округлення 137
```

Крім стандартних виразів для роботи з числами (а в Python їх не так вже й багато), в складі Python є кілька корисних модулів.

Модуль **math** надає більш складні математичні функції.

Модуль **random** реалізує генератор випадкових чисел і функції випадкового вибору.

```
>>> math.pi
3.141592653589793
>>> import random
>>> random.random()
0.48701030555228886
>>> math.cos(10)
-0.8390715290764524
>>> math.exp(98)
3.637970947608805e+42
```

Більше подробиць щодо вбудованих модулів буде описано у розділі 4.

### 2.2.3. Вбудовані функції Python

Більшість цих функцій потрібно викликати через модуль `math`.

Наприклад, тригонометричні та степеневі функції.

`abs(x)` – повертає абсолютну величину (модуль числа).

`all(послідовність)` – повертає `True`, якщо всі елементи істинні (або, якщо послідовність порожня).

`any(послідовність)` – повертає `True`, якщо хоча б один елемент - істина.

Для порожньої послідовності повертає `False`.

`ascii(object)` – як `repr()`, повертає рядок, що містить представлення об'єкта, але замінює не ASCII символи на екрановані послідовності.

`bin(x)` – перетворення цілого числа в двійковий рядок.

`bool(x)` – перетворення до типу `bool`, що використовує стандартну процедуру перевірки істинності. Якщо `x` є помилковим або опущений, повертає значення `False`, в іншому випадку вона повертає `True`.

`bytearray ([джерело [, кодування [помилки]])` – перетворення до `bytearray`. `Bytearray` – змінна послідовність цілих чисел в діапазоні  $0 \leq X < 256$ . Викликана без аргументів, повертає порожній масив байтів.

`bytes ([джерело [, кодування [помилки]])` – повертає об'єкт типу `bytes`, який є незмінною послідовністю цілих чисел в діапазоні  $0 \leq X < 256$ . Аргументи конструктора інтерпретуються як для `bytearray()`.

`callable(x)` – повертає `True` для об'єкта, що підтримує виклик (як функції).

`chr(x)` – повертає односимвольний рядок, код символу якої дорівнює `x`.

`classmethod(x)` – створює зазначену функцію методом класу.

`compile(source, filename, mode, flags = 0, dont_inherit = False)` – компіляція в програмний код, який згодом може виконатися функцією `eval` або `exec`. Рядок не повинен містити символів повернення каретки або нульові байти.

`delattr(object, name)` – видаляє атрибут з ім'ям `'name'`.

`complex([real [, imag]])` – перетворення до комплексного числа.

`dict ([object])` – перетворення до словника.

`dir ([object])` – список імен об'єкта, а якщо об'єкт не вказано, список імен в поточній локальній області видимості.

`divmod (a, b)` – повертає часткове і залишок від ділення `a` на `b`.

`enumerate (iterable, start = 0)` – повертає ітератор, при кожному проході надає кортеж з номера і відповідного члена послідовності.

`eval(expression, globals = None, locals = None)` – виконує рядок програмного коду.

`exec(object [, globals [, locals]])` – виконує програмний код на Python.

`filter(function, iterable)` – повертає ітератор з тих елементів, для яких `function` повертає істину.

`float ([X])` – перетворення до числа з плаваючою крапкою. Якщо аргумент не вказано, повертається `0.0`.

`format(value [, format_spec])` – форматування (зазвичай форматування рядка).

`getattr (object, name, [default])` – витягує атрибут об'єкта або `default`.

`frozenset ([послідовність])` – повертає незмінну множину.

`globals ()` – словник глобальних імен.

`hasattr (object, name)` – чи має об'єкт атрибут з ім'ям `'name'`.

`hash (x)` – повертає хеш зазначеного об'єкта.

`help()` – вбудований довідник з Пайтону



```

>>> help ()

Welcome to Python 3.10's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the internet at https://docs.python.org/3.10/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules. To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics". Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help> |

```

Останній загального довідника запрошує задати запитання щодо конкретної функції чи оператора.

`hex (x)` – перетворення цілого числа в шістнадцяткову рядок.

`id (object)` – повертає "адреса" об'єкта. Це ціле число, яке гарантовано буде унікальним і постійним для даного об'єкта протягом терміну його існування.

`input ([prompt])` – повертає введений користувачем рядок.

`int ([object], [підставу системи числення])` – перетворення до цілого числа.

`isinstance (object, ClassInfo)` – істина, якщо об'єкт є екземпляром `ClassInfo` або його підкласом. Якщо об'єкт не є об'єктом даного типу, функція завжди повертає брехня.

`issubclass (клас, ClassInfo)` – істина, якщо клас є підкласом `ClassInfo`. Клас вважається підкласом себе.

`iter(x)` – повертає об'єкт ітератора.

`len(x)` – повертає число елементів в зазначеному об'єкті.

`list([object])` – створює список.

`locals ()` – словник локальних імен.

`map(function, iterator)` – ітератор, що вийшов після застосування до кожного елементу послідовності функції `function`.

`max(iter, [args ...] * [, key])` – максимальний елемент послідовності.

`memoryview ([object])` – створює об'єкт `memoryview`.

`min(iter, [args ...] * [, key])` – мінімальний елемент послідовності.

`next(x)` – повертає наступний елемент ітератора.

`object()` – повертає безликий об'єкт, який є базовим для всіх об'єктів.

`oct(x)` – перетворення цілого числа в вісімковий формат.

`open(file, mode = 'r', buffering = None, encoding = None, errors = None, newline = None, closefd = True)` – відкриває файл і повертає відповідний потік.

`ord(c)` – код символу.

`pow(x, y, [r])` -  $(x ** y) \% r$

`range([start = 0], stop, [step = 1])` – арифметична прогресія від `start` до `stop` з кроком `step`.

`repr(obj)` – представлення об'єкта.

`print([object, ...], *, sep = "", end = '\n', file = s)`

`set([object])` – створює множину.

`slice([start = 0], stop, [step = 1])` – об'єкт зрізу від `start` до `stop` з кроком `step`.

`str([object], [кодування], [помилки])` – строкове представлення об'єкту.

Використовує метод `__str__`.

`tuple(obj)` – перетворення до кортежу.

Нижче наведено перелік функцій, які діють на масиви:

`Sqrt()` – обчислити квадратний корінь з кожного елемента. Еквівалентно `arr**0.5`

`Square()`– обчислити квадрат кожного елемента. Еквівалентно `arr ** 2`

`Exp()` – обчислити експоненту `e` кожного елемента.

`Log()`, `log10()`, `log2()`, `loglp()` – натуральний (за основою  $e$ ), десятковий, двійковий логарифм і функція  $\log(1 + x)$  відповідно.

`Sign()` – обчислити знак кожного елемента: 1 (для додатних чисел), 0 (для нуля) або -1 (для від'ємних чисел).

`Ceil()` – обчислити для кожного елемента найменше ціле число, що не менше його.

`Floor()` – обчислити для кожного елемента найбільше ціле число, не більше його.

`Rint()` – округлити елементи до найближчого цілого зі збереженням `dtype`.

`Modf()` – повернути дробові і цілі частини масиву у вигляді окремих масивів

`Isnan()` – повернути логічний масив, який показує, які значення є NaN (не числом).

`Isfinite()`, `isinf()` – повернути логічний масив, який показує, які елементи є кінцевими (НЕ inf і не NaN) або нескінченними відповідно.

`Cos()`, `cosh()`, `sin()`, `tan()`, `tanh()`, `sinh()` – звичайні і гіперболічні тригонометричні функції.

`arccos()`, `arccosh()`, `arcsinh()`, `arctan()`, `arcsin()`, `arctanh()` – зворотні тригонометричні функції.

`logical_not()` – обчислити значення істинності `not x` для кожного елемента. Еквівалентно – `arr`.

`add()` – скласти відповідні елементи масивів.

`subtract()` – відняти елементи другого масиву з відповідних елементів першого.

`multiply()` – перемножити відповідні елементи масивів.

`divide()`, `floor_divide()` – ділення і ділення з відкиданням залишку.

Найбільш вживані функції з модуля `numpy.linalg`, який підключається до програми на Python, якщо вказати його назву:

`diag()` – повертає діагональні елементи квадратної матриці у вигляді одновимірною масиву або перетворює одновимірний масив в квадратну матрицю, в якій всі елементи, крім що знаходяться на головній діагоналі, дорівнюють нулю.

`dot()` – обчислює добуток матриць.

`trace()` – Обчислює слід матриці – суму діагональних елементів.

`det()` – Обчислює визначник матриці.

`Eig()` – обчислює власні значення і власні вектори квадратної матриці.

`inv()` – Обчислює зворотну матрицю.

`pinv()` – Обчислює псевдообернену матрицю Мура-Пенроуза для квадратної матриці.

qr() – Обчислює QR-розкладання.  
svd() – Обчислює сингулярне розкладання (SVD).  
solve() – Вирішує лінійну систему  $Ax = B$ , де  $A$  – квадратна матриця.  
lstsq() – Обчислює рішення рівняння  $y = xb$  за методом найменших квадратів.

#### 2.2.4. Правила запису формул у Пайтоні

Економічні формули у більшості випадків складаються з чотирьох дій арифметики, але їх складність полягає у тому, що всі ці операції мають виконуватися водночас.

Наприклад, візьмемо формулу прибутку на одну акцію

$$ПОА = \frac{\text{Чистий прибуток} - \text{привілейовані дивіденди}}{\text{Середнє число звичайних акцій}}$$

Для її реалізації у Пайтоні потрібно визначити числові значення усіх змінних, а потім створити формулу. Варто визначити, що дії множення та ділення виконуються у першу чергу, віднімання і додавання – у другу.

Якщо ми бажаємо змінити цей порядок, потрібно використовувати дужки.

```
1 PI=45687 # Чистий прибуток
2 PD= 36752 # Привілейовані дивіденди
3 NA=1023 # Середнє число звичайних акцій
4 Ion1A=(PI - PD) / NA
5 print(Ion1A)
| 8.734115347018573
```

Якби ми не поставили дужок, а написали  $\text{Ion1A} = \text{PI} - \text{PD} / \text{NA}$ , то Пайтон спочатку б поділив  $\text{PD} / \text{NA}$ , а вже потім відняв результат від  $\text{PI}$ , що привело б до помилки.

Тип дужок має значення. У Пайтоні при написанні формул можна використовувати тільки круглі дужки.

Якщо формула є складнішою, наприклад, формула для середньозваженої вартості капіталу,

$$BK = \frac{\sum_{i=1}^n BK_i S_i}{\sum_{i=1}^n S_i}$$

то дужки треба додати і у знаменник.

```
File Edit Format Run Options Window Help
1 BK1=10
2 BK2=20
3 S1=10200
4 S2=58300
5 BK= (BK1*S1+BK2*S2) / (S1+S2)
6 print(BK)
7
| 18.51094890510949
```

### 2.3. Масиви даних та операції з ними

До цього ми вивчали, як діють змінні різних типів у Пайтоні. Як робити з ним різні операції. Та в економіці у більшості випадків ми працюємо з масивами однотипних даних. Наприклад, список співробітників з ФІО, посадою, окладом, тощо.

Цей список представляє собою двовимірний масив даних, який, хотілося б, обробляти як одна змінна.

ПІБ	Посада	Оклад, грн
Фрідман Семен Якович	директор	15000
Дмитренко Оксана Семенівна	головбух	12000
Чепель Василь Михайлович	шофер	10000

Далі розглянемо які можливості для створення і обробки масивів даних надає нам Пайтон.

### 2.3.1. Списки (list)

Перший тип масивів даних – це списки. Списки в Python – це впорядковані змінювані колекції об'єктів довільних типів. Тобто, в кожному списку може бути і текст і числа.

Щоб використовувати списки, їх потрібно створити. Створити список можна декількома способами. Наприклад, можна обробити будь-який об'єкт (наприклад, рядок) вбудованою функцією `list`:

```
>>> list(' ім'я списку')
['список']
```

Список можна створити і за допомогою літералу:

```
>>> s = [] # Порожній список
>>> l = ['s', 'p', ['isok'], 2]
>>> s
[]
>>> l
['S', 'p', ['isok'], 2]
```

Як видно з прикладу, список може містити будь-яку кількість будь-яких об'єктів (в тому числі і вкладені списки), чи не містити нічого.

І ще один спосіб створити список – це генератори списків. Генератор списків – це спосіб побудувати новий список, застосовуючи вираз до кожного елементу послідовності. Генератори списків дуже схожі на цикл `for`.

```
>>> c = [c * 3 for c in 'Дніпро']
>>> c
['ДДД', 'ннн', 'ііі', 'ппп', 'ррр', 'ооо']
```

Тут текст повторюється по буквах слова.

Можлива і більш складна конструкція генератора списків:

```
>>> c = [c * 3 for c in 'Дніпро' if c != 'Д']
>>> c
['ннн', 'ііі', 'ппп', 'ррр', 'ооо']
>>> c = [c + d for c in 'list' if c != 'l' for d in 'spam' if d != 'A']
>>> c
['ls', 'lp', 'la', 'lm', 'is', 'ip', 'ia', 'im', 'ss', 'sp', 'sa',
'sm', 'ts', 'tp', 'ta', 'tm']
```

Тут заповнення списку ведеться з урахуванням логічного оператора, що виключає певні символи.

Але в складних випадках краще користуватися звичайним циклом for для генерації або введення з клавіатури списків.

Всі наведені вище приклади стосуються одномірних масивів даних, але приклад зі списком співробітників показує, що для економістів більш актуальним є двовимірний масив.

Розглянемо основні поняття таких масивів даних.

Ось двовимірний масив, в якому кожен рядок – це одновимірний масив, обведений червоним. Нижче подано опис такого рядка. Щоб описати двовимірний масив, потрібно скористатися подвійними квадратними дужками, ось так

1	2	3	12
23	5	13	4
8	6	7	11

a[1] = [23, 5, 13, 4]

```
a = [[1, 2, 3, 12], [23, 45, 13, 4], [8, 6, 7, 11]]
```

Цей опис показує, що кожен рядок даних записується через кому, береться у квадратні дужки і цей блок відділяється також комою від іншого блоку. Весь список блоку також береться у квадратні дужки. Ось приклад для списку співробітників.

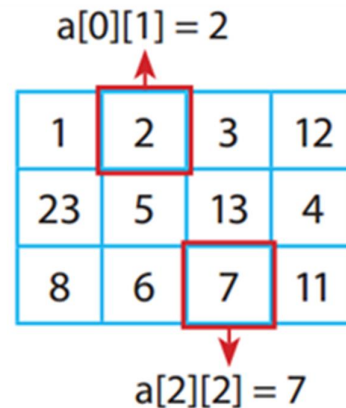


```

>>> emp=[['Фрідман Семен Якович', 'директор', 15000], ['Дмитренко
Оксана Семенівна', 'головбух', 12000], ['Чепель Василь Михайл
ович', 'шофер', 10000]]
>>> emp
[['Фрідман Семен Якович', 'директор', 15000], ['Дмитренко
Оксана Семенівна', 'головбух', 12000], ['Чепель Василь Мих
айлович', 'шофер', 10000]]

```

Щоб отримати якесь значення з масиву, потрібно вказати номер даного у ньому. Нумерація починається з нуля. Для одновимірних масивів це просто запис типу `a[2]`. Для двовимірних масивів треба вказати спочатку номер рядка, а потім номер даного у цьому рядку. На рисунку видно, як утворюється відповідна нумерація. Наприклад, якщо нам потрібно ПІБ головбуха, то треба вказати, що це рядок номер 1, а дані номер 0.



```

>>> emp[1][0]
'Дмитренко Оксана Семенівна'

```

З цими даними можна робити ті самі операції, що й зі змінними: текстові дані можна склеювати або вирізати, числові дані можна піддавати арифметичним діям.

```

>>> emp[1][0][0:8]
'Дмитренк'
>>> emp[0][2]+emp[1][2]+emp[2][2]
37000

```

Тут ми вирізали з ПІБ головбуха тільки прізвище та знайшли суму всіх окладів співробітників.

Для списків доступні основні вбудовані функції, а також методи списків. Зауважимо, що замість слова `list`, при використанні тієї чи іншої функції, треба писати ім'я конкретного списку.

Метод	Що робить
<code>list.append(x)</code>	Додає елемент у кінець списку
<code>list.extend(L)</code>	Розширяє список <code>list</code> , додаючи в кінець всі елементи списку <code>L</code>
<code>list.insert(i, x)</code>	Вставляє на $i$ -ий елемент значення $x$
<code>list.remove(x)</code>	Видаляє перший елемент у списку, що має значення $x$
<code>list.pop([i])</code>	Видаляє $i$ -ий елемент і повертає його. Якщо індекс не вказано, видаляється останній елемент
<code>list.index(x, [start [, end]])</code>	Повертає положення першого елемента від <code>start</code> до <code>end</code> зі значенням $x$
<code>list.count(x)</code>	Повертає кількість елементів зі значенням $x$
<code>list.sort(key = функція, reverse=True або False)</code>	Сортує список на основі функції
<code>list.reverse()</code>	Розгортає список
<code>list.copy()</code>	Поверхнева копія списку
<code>list.clear()</code>	Очищає список
<code>len(list)</code>	Повертає кількість елементів масиву

Потрібно відзначити, що методи, які застосовуються до списків, змінюють сам список, а тому результат виконання не потрібно іще раз записувати у цей список. Наприклад:

```
>>> l = [5, 2, 3, 9, 7]
>>> l.sort()
>>> l
[2, 3, 5, 7, 9]
```

Для двовимірного масиву, якщо не вказано ключ сортування, операція виконується по першому стовпцю.

```
>>> emp.sort()
>>> emp
[['Дмитренко Оксана Семенівна', 'головбух', 12000], ['Фрідман Семен Якович', 'директор', 15000], ['Чепель Василь Михайлович', 'шофер', 10000]]
>>>
```

Операції з додавання даних з одного списку до іншого через функцію `extend` виконуються так, неначе це одновимірний масив.

```
>>> L=['Шевченко Борис Петрович', 'експедитор', 1350 ]
>>> emp.extend(L)
>>> emp
[['Дмитренко Оксана Семенівна', 'головбух', 12000], ['Фрідман Семен Якович', 'директор', 15000], ['Чепель Василь Михайлович', 'шофер', 10000], 'Шевченко Борис Петрович', 'експедитор', 1350]
```

Тобто, Шевченко має адресу [3], експедитор – [4].

```
>>> emp[3]
'Шевченко Борис Петрович'
>>> emp[5]
1350
>>> emp[4]
'експедитор'
```

Цей приклад показує, що більшість функцій операцій з масивами орієнтовані на одновимірні масиви.

А якщо ми хотіли створити ще один рядок таблиці співробітників, треба застосувати функцію `insert`, але і ця функція виконує операцію вставлення тільки для одновимірного масиву.

```
>>> emp.insert(3, 'Шевченко Борис Петрович')
>>> emp
[['Дмитренко Оксана Семенівна', 'головбух', 12000], ['Фрідман Семен Якович', 'директор', 15000], ['Чепель Василь Михайлович', 'шофер', 10000], 'Шевченко Борис Петрович', 'Шевченко Борис Петрович', 'експедитор', 1350]
```

Висновком може бути твердження, що двовимірні масиви потрібно створювати в окремому файлі, а потім вводити з нього у програму Пайтон.

### 2.3.2. Індекси і зрізи

Як і в текстових змінних, в масивах взяття за індексом означає, вказати номер рядка та стовпця у масиві. Якщо такого елемента не існує виникає помилка *IndexError*.. На прикладі видно, що спроба звернутися до масиву `emp` за адресою `[1][3]`, викликає повідомлення про помилку бо такого елемента не існує.

```
>>> emp
[['Фрідман Семен Якович', 'директор', 15000], ['Дмитренко
Оксана Семенівна', 'головбух', 12000], ['Чепель Василь Мих
айлович', 'шофер', 10000]]
>>> emp[1][3]
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    emp[1][3]
IndexError: list index out of range
```

В Python також підтримуються від'ємні індекси, при цьому нумерація йде з кінця. Якщо індекс виходить за існуючі межі, виникає помилка, наприклад:

```
>>> a=[2, 4, 1, 6, 9]
>>> a[-2]
6
>>> a[-8]
Traceback (most recent call last):
  File "<pyshell#41>", line 1, in <module>
    a[-8]
IndexError: list index out of range
```

В Python, крім індексів, існують ще й зрізи:

Командою **Назва масиву [START: STOP: STEP]** можна витягти зріз від номера `START`, до `STOP` (не включаючи його), з кроком `STEP`. За замовчуванням `START = 0`, `STOP =` довжині об'єкта, `STEP = 1`. Відповідно, якісь (а можливо, і всі) параметри можуть бути опущені.

```

>>> a=[2,4,1,6,9]
>>> a[2:]
      [1, 6, 9]
>>> a[::2]
      [2, 1, 9]
.....

```

Останній приклад показує, що вибирався весь масив з кроком 2.

Також всі ці параметри можуть бути і від'ємними:

```

>>> a=[2,4,1,6,9]
>>> a[-3]
      1
>>> a[:-2]
      [2, 4, 1]
.....

```

В останньому прикладі вибиралися елементи масиву з кінця [-2] до початку. Якщо діапазон значень виявиться за межами об'єкта, то масив, що показується, буде пустим.

```

>>> a[10:20]
      []
.....

```

Також за допомогою зрізів можна не тільки отримувати елементи, але і додавати і видаляти елементи.

```

>>> a=[2,4,1,6,9]
>>> a[1:3]=['A','b','c']
>>> a
      [2, 'A', 'b', 'c', 6, 9]
>>> del a[:-3]
>>> a
      ['c', 6, 9]
.....

```

### 2.3.3. Кортежі (tuple)

Кортеж, по суті – це незмінний список. Зазвичай кортежі мають кращу продуктивність, ніж списки, оскільки не потрібно виконувати операції вставки або видалення елементів.

Створюємо порожній кортеж:

```
>>> a = tuple () # За допомогою вбудованої функції
tuple ()
>>> a
()
>>> a = () # За допомогою літералу кортежу
>>> a
()
>>>
```

Створюємо кортеж з одного елемента:

```
>>> a = ('s')
>>> a
's'
```

Але в цьому випадку вийшов рядок. Як отримати кортеж? Наступним чином:

```
>>> a = ('s',)
>>> a
('S',)
```

Вся справа у комі після символу *s*. Самі по собі дужки нічого не значать, точніше, означають те, що всередині них знаходиться одна інструкція, яка може бути відділена пробілами, перенесенням рядків та ін. Кортеж можна створити і так:

```
>>> a = 's',
>>> a
('S',)
```

Але все-таки не захоплюйтеся, і ставте дужки, тим більше, що бувають випадки, коли дужки необхідні.

Ну і створити кортеж з об'єкту можна за допомогою все тієї ж функції tuple()

```
>>> a = tuple('hello, world!')
>>> a
('h', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd', '!')
```

Найбільш практичним є створення списку, а потім його заміна на кортеж. У прикладі, список emp перетворюється на кортеж b, яким вже можна користуватися як довідником, що дозволить бути упевненим, що цей масив даних ніхто не зіпсує.

```
>>> emp=[['Фрідман Семен Якович', 'директор', 15000], ['Дмитренко
Оксана Семенівна', 'головбух', 12000], ['Чепель Василь Михайл
ович', 'шофер', 10000]]
>>> b=tuple(emp)
>>> b
(['Фрідман Семен Якович', 'директор', 15000], ['Дмитренко
Оксана Семенівна', 'головбух', 12000], ['Чепель Василь Мих
айлович', 'шофер', 10000])
```

І хоча кортежі створюються з круглими дужками, доступ до двовимірного кортежу здійснюється через адреси, вказані у квадратних дужках.

```
>>> b[0][0]
'Фрідман Семен Якович'
>>> b[0][2]+b[1][2]+b[2][2]
37000
```

Для одновимірного кортежу те сам. Спроба задати адресу елемента викликає помилку.

```
>>> a=[2, 4, 1, 6, 9]
>>> c=tuple(a)
>>> c(0)
Traceback (most recent call last):
  File "<pyshell#71>", line 1, in <module>
    c(0)
TypeError: 'tuple' object is not callable
>>> c[0]
2
```



Навіщо потрібні кортежі, якщо є списки? Ось приклади з поясненнями:

1. Захист від дурня. Тобто кортеж захищений від змін, як навмисних (що погано), так і випадкових (що добре).

```
# Кортеж
my_tuple = (1, 2, 3)
print(my_tuple[0]) # Виведе: 1
# Не можна змінити значення кортежу
# my_tuple[0] = 5 # Помилка!
```

```
# Список
my_list = [1, 2, 3]
print(my_list[0]) # Виведе: 1
# Можна змінити значення списку
my_list[0] = 5
print(my_list) # Виведе: [5, 2, 3]
```

2. Менший розмір.

3. Можливість використовувати кортежі в якості ключів словника:

З кортежами працювати приблизно так само, як і зі списками.

Всі операції над списками, що не змінюють список (додавання, множення на число, методи `index()` і `count()` і деякі інші операції). Можна також по-різному змінювати елементи місцями і так далі.

### 2.3.4. Словники (dict) і робота з ними. Методи словників

Словники в Python – це неупорядковані колекції довільних об'єктів з доступом по ключу. Їх іноді ще називають асоціативними масивами або хеш-таблицями. На відміну від списків, які є впорядкованими послідовностями елементів довільного типу, елементи словника - це неупорядковані послідовності пар {ключ: значення}. Обидві пари можуть бути даними різних типів. Зверніть увагу, що для словників використовуються фігурні дужки. Іноді словники

називають асоціативними масивами, іноді відображеннями (мається на увазі відображення безлічі ключів словника на безліч його значень). Як і списки, словники мають змінну довжину, довільну вкладеність і можуть зберігати значення довільних типів.

Щоб працювати зі словником, його потрібно створити. Створити його можна кількома способами. По перше, за допомогою літералу:

```
>>> d = {}
>>> d{}
>>> d = { 'dict': 1, 'dictionary': 2}
>>> d
{'Dict': 1, 'dictionary': 2}
```

По-друге, за допомогою функції dict:

```
>>> d = dict (short = 'dict', long = 'dictionary')
>>> d
{'Short': 'dict', 'long': 'dictionary'}
>>> d = dict([(1, 1), (2, 4)])
>>> d
{1: 1, 2: 4}
```

По-третє, за допомогою методу fromkeys:

```
>>> d = dict.fromkeys(['a', 'b'])
>>> d
{'A': None, 'b': None}
>>> d = dict.fromkeys(['a', 'b'], 100)
>>> d
{'A': 100, 'b': 100}
```

По-четверте, за допомогою генераторів словників, які дуже схожі на генератори списків.

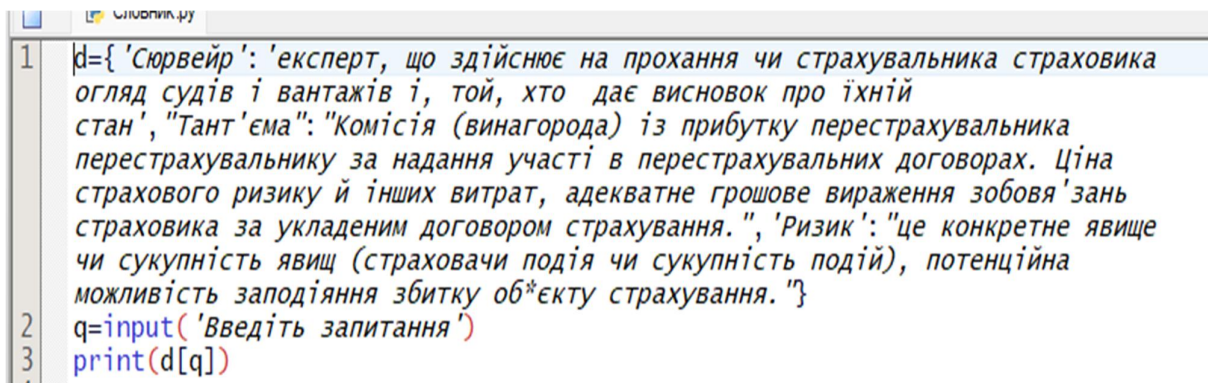
```
>>> d = {a: a ** 2 for a in range (7)}
>>> d
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

Тепер спробуємо додати записів в словник і витягти значення ключів:

```
>>> d = {1: 2, 2: 4, 3: 9}
>>> d [1]
2
>>> d [4] = 4 ** 2
>>> d
{1: 2, 2: 4, 3: 9, 4: 16}
>>> d ['1']
```

Traceback (most recent call last): File "", line 1, in d ['1'] KeyError: '1'

Ось короткий фрагмент програми, в якому вводиться слово запиту, і видається пояснення цього терміну зі словника.



```
1 d={'Сюрвейр': 'експерт, що здійснює на прохання чи страхувальника страховика огляд судів і вантажів і, той, хто дає висновок про їхній стан', 'Тант'єма': 'Комісія (винагорода) із прибутку перестраховальника перестраховальнику за надання участі в перестраховальних договорах. Ціна страхового ризику й інших витрат, адекватне грошове вираження зобов'язань страховика за укладеним договором страхування.', 'Ризик': 'це конкретне явище чи сукупність явищ (страховачи подія чи сукупність подій), потенційна можливість заподіяння збитку об'єкту страхування.'}
2 q=input('Введіть запитання')
3 print(d[q])
```

```
Введіть запитанняРизик
це конкретне явище чи сукупність явищ (страховачи подія чи сукупність подій), потенційна можливість заподіяння збитку об'єкту страхування.
```

Як видно з прикладів, присвоєння по новому ключу розширює словник, присвоєння за існуючим ключем перезаписує його, а спроба вилучення неіснуючого ключа породжує виключення. Для уникнення виключення є спеціальний метод або можна перехоплювати виняток.

Методи словників описані наступними функціями:

`dict.clear()` – очищає словник.

`dict.copy()` – повертає копію словника.

`classmethod dict.fromkeys(seq [, value])` – створює словник з ключами з `seq` і значенням `value` (за замовчуванням `None`).

`dict.get(key [, default])` – повертає значення ключа, але якщо його немає, не видає виняток, а повертає `default` (за замовчуванням `None`).

`dict.items()` – повертає пари (ключ, значення).

`dict.keys()` – повертає ключі в словнику.

```
>>> d.keys()
dict_keys(['Сюрвейр', 'Тант'єма', 'Ризик'])
```

`dict.pop(key [, default])` – видаляє ключ і повертає значення. Якщо ключа немає, повертає `default` (за замовчуванням видає виняток).

`dict.popitem()` – видаляє і повертає пару (ключ, значення). Якщо словник порожній, видає виняток `KeyError`. Пам'ятайте, що словники неупорядковані.

```
>>> d.popitem()
('Ризик', 'це конкретне явище чи сукупність явищ (страхова чи подія чи сукупність подій), потенційна можливість заподіяння збитку об'єкту страхування.')
```

`dict.setdefault(key [, default])` – повертає значення ключа, але якщо його немає, не видає виняток, а створює ключ з значенням `default` (за замовчуванням `None`).

`dict.update ([other])` – оновлює словник, додаючи пари (ключ, значення) з `other`.

`dict.values ()` – повертає значення в словнику.

Ще декілька прикладів роботи зі словниками.

**При переборі значень словників**, ключі та відповідні значення можуть отримуватися за допомогою методу `iteritems()`:

```
>>> knights = {'Галлагад': 'Чистий', 'Робін': 'Хоробрий'}
```

```
>>> for k, v in knights.iteritems():
```

```
...     print k, v
```

```
...
```

```
Галлагад Чистий
```

```
Робін Хоробрий
```

Для словників визначена операція злиття, яка реалізована в методі *update*:

```
phone1 = { 'Dan' : '7926765431' , 'John' : '74881234567' }
```

```
phone2 = { 'Daniel' : '7926765431' , 'John' : '74775566221' }
```

```
phone1.update ( phone2 )
```

```
print ( phone1 )
```

```
>>> { 'Daniel' : '79267654 31' , 'John' : '74775566221' , 'Dan': '7926765431' }
```

Як бачимо з цього прикладу, при злитті дані, що повторюються ( для імені John), зливаються в один елемент.

Інший корисний метод при роботі зі словниками – метод *get (key, default)*. Сенс його в наступному: при зверненні до значення неіснуючого ключа можна передбачити значення за замовчуванням, яке можна використовувати.

Для попереднього прикладу замінено оператор виводу на

```
print (d.get (q) )
```

Тоді при введенні запиту з неіснуючим ключем, отримали не збій, а слово None.

```
Введіть запитання>vt  
None
```

## Контрольні запитання

1. Які операції можна виконати з рядками (строками) у Python?
2. Як виконати конкатенацію рядків у Python, і який оператор для цього використовується?
3. Як визначити довжину рядка у Python, і яку функцію для цього використовується?
4. Як можна виконати вилучення підрядки (срізу) з рядка у Python?
5. Як перетворити рядок на верхній регістр (великі літери) або на нижній регістр (маленькі літери) у Python?
6. Як виконати пошук та заміну певного тексту у рядку за допомогою методів у Python?
7. Які основні математичні операції можна виконати з числами у Python?
8. Як перетворити рядок у число у Python, і як це впливає на операції з цим числом?
9. Як виконати операції ділення та отримати результат у форматі цілого числа (цілочислового ділення) у Python?
10. Як виконати піднесення числа до степеня у Python, і який оператор для цього використовується?
11. Як виконати обчислення остачі від ділення двох чисел (операція залишку) у Python?
12. Як округлити число до певної кількості знаків після коми у Python?
13. Що таке масив даних у Python, і як ви визначаєте його?
14. Які основні типи масивів даних доступні у Python?
15. Як створити масив даних у Python за допомогою вбудованих типів даних, таких як список (list) чи кортеж (tuple)?
16. Як виконати доступ до елементів масиву даних у Python за допомогою індексів?
17. Як виконати зміну значення певного елемента масиву даних у Python?

18. Як визначити довжину масиву даних у Python, і який метод для цього використовується?
19. Як виконати злиття (об'єднання) двох масивів даних у Python?
20. Як виконати додавання нового елемента до кінця масиву даних у Python?
21. Як видалити певний елемент з масиву даних у Python?
22. Як ви використовуєте цикл для ітерації через елементи масиву даних у Python?

*Вивчення можливостей оперування з числами, текстом, та масивами цих даних дозволить студентові перейти до вивчення основних операторів мови Пайтон.*



# Розділ 3.

## ОСНОВНІ ОПЕРАТОРИ МОВИ ПАЙТОН

*Вивчивши матеріали цього розділу студенти зможуть писати програми, створювати підпрограми та GUI-модулі для роботи програми без інтерпретатора.*

«Скільки вже можна товкти воду в ступі» – скажете ви. – «Коли вже будемо програмувати?». От якраз і почнемо. Всі описані тут оператори дозволять вам створювати повноцінні програми. Окремі елементи програм, що пояснюють той чи інший оператор, стануть вам у пригоді, спочатку для виконання навчальних завдань, а вже потім і виробничих. Починаймо!

З першим оператором ви вже знайомі з самого початку цього курсу – це оператор присвоювання. Він використовує знак «дорівнює», але його зміст зовсім інший. Це не порівняння лівої частини з правою.

Оператор присвоювання означає, що результат розрахунку, що міститься у правій частині, буде перенесено у комірку, що вказана у лівій частині. Наприклад

```
>>> name='Іван Петренко'  
>>> result=28+25
```

Тут в комірку name вміщається текст, а у комірку result – сума двох чисел. Коміркою, куди кладеться результат розрахунку, також може бути елемент масиву даних.

```

>>> a=[1, 2]
>>> a[0]='Іван Петренко'
>>> a[1]=28+25
>>> a
['Іван Петренко', 53]

```

Існують цікаві модифікації цього оператора, яких немає в інших мовах програмування:

1. Поміняти місцями значення двох змінних, яким не обов'язково мати однаковий тип. Раніше це робилося так  $a = a + 3$ . Тобто, до попереднього значення числа в комірці  $a$  додається число 3 і результат записується у комірку  $a$ .

```

>>> w='Іван Петренко'
>>> q=20
>>> w, q=q, w
>>> print(w, q)
20 Іван Петренко

```

Цей оператор діє і для елементів масиву.

```

>>> a=[1, 2]
>>> a[0], a[1]=a[1], a[0]
>>> a
[2, 1]

```

2. Додати або відняти у попереднього значення даних у комірці іще якісь дані. Для елементів масиву цей оператор теж діє. Для тексту діє тільки варіант «+=». У прикладі до існуючого тексту додається зріз тексту з 0 по 5 символ.

```

>>> q=20
>>> q+=10
>>> q
30
>>> q-=18
>>> q
12

>>> a=[1,2]
>>> a[0]+=30
>>> a[0]
31
-----
>>> a[1]-=2
>>> a[1]
0

>>> w='Іван Петренко'
>>> w+=w[:5]
>>> w
'Іван ПетренкоІван '

```

### 3.1. Логічний оператор IF

Як ви пам'ятаєте, програма – це текст, який заставляє комп'ютер виконувати різні обчислювальні операції. Виконання цих операцій іде так само, як ми читаємо текст – зверху до низу – монотонно. А як бути, коли в розрахунку потрібно обрати певну формулу, причому цей вибір залежить від даних, визначених раніше. Наприклад, при розрахунку оплати за електроенергію при двозонній системі оплати, коли нічні та денні часи споживання тарифікуються різними сумами. Отже, необхідно у програмі мати розгалуження розрахунків. Цим і займається умовний оператор `if`.

Умовна інструкція *if-elif-else* (її ще іноді називають оператором розгалуження) – основний інструмент вибору в *Python*. Простіше кажучи, вона вибирає, яку дію слід виконати, в залежності від значення змінних в момент перевірки умови.

Спочатку записується частина `if` з умовним виразом, далі можуть слідувати **одна або більше** необов'язкових частин `elif`, і, нарешті, необов'язкова частина `else`.

Для оператора `if` виконання умови означає, що вона має логічний тип «Істина», невиконання – тип «Брехня». Операції порівняння застосовуються до структур даних та повертають *True* або *False*. (Істина чи Брехня).

Взагалі в логічних виразах оператора if діє простий принцип:

- Будь-яке число, не рівне 0, або непорожній об'єкт – істина.
- Числа, рівні 0, порожні об'єкти і значення None – брехня

Загальна форма запису умовної інструкції if виглядає наступним чином:

if умова1:

оператор, що виконується при виконанні умови 1

elif умова 2:

оператор, що виконується при виконанні умови 2

else:

оператор, що виконується при невиконанні умов 1 та 2

Всі знаки логічних виразів, що використовуються в операторі if, завжди складаються з двох символів, написаних без пробілу.

Ось перелік символів, які дозволяють скласти всі необхідні логічні умови

Символ	Значення символу	Приклад
==	Дорівнює	a == b
>=	Більше, або дорівнює	a >= b
<=	Менше, або дорівнює	a <= b
!=	Не дорівнює	a != b
and	Істина, якщо обидва значення X і Y істинні.	X and Y
or	Істина, якщо хоча б одне зі значень X або Y істинно.	X or Y
not	Істина, якщо X хибне.	not X

Давайте розробимо програму, яка б використала декілька elif. Нехай програма вітається з людьми різних імен, використовуючи ключний відмінок іменників. Для цього використовувався зріз кінця тексту, бо необхідно або

додати, або змінити останню букву імені. Ось що вийшло. Як бачите, не для всіх імен вдалося зробити вірний кличний відмінок.

```
1 n=input("Ваше ім'я ")
2 if n=='Ігор' or n=='Семен':
3     print('Привіт ', n+'е')
4 elif n=='Петро':
5     print('Де гроші ', n[:3]+'е?')
6 elif n=='Бармалей':
7     print('Я тебе боюсь, ', n[:6]+'ею!')
8 else:
9     a=len(n)
10    print('Звідкіля ви, ', n[:a-1]+'е?')
11
```

Ваше ім'я Ігор  
Привіт Ігоре

Ваше ім'я Семен  
Привіт Семене

Ваше ім'я Бармалей  
Я тебе боюсь, Бармалею!

Ваше ім'я  
Звідкіля ви, е?

Ваше ім'я Навуходоносор  
Звідкіля ви, Навуходоносое?

Тепер розробимо довідник бухгалтера на початок березня 2024 року.

```
1 d=int(input('Введіть день березня 2024 року'))
2 if d==1:
3     print('Останній день подання\нзвіту про виробництво продукції
4     тваринництва\нзвіту про вартість природного газу, який постачається
5     споживачам за I півріччя 2024 року \нзвіту про вартість електроенергії, яка
6     постачається споживачам за I півріччя 2024 року')
7 elif d==4:
8     print('Останній день подання\нзвіту про товарооборот торгової мережі за
9     лютий 2024 року \нзвіту про ціни виробників промислової продукції за лютий
10    2024 року')
11 elif d==5:
12    print('Останній день подання\нробстеження інноваційної діяльності
13    промислового підприємства за 2023 рік\н Звіту про здійснення наукових
14    досліджень і розробок за 2023 рік')
15 elif d==6:
16    print('Останній день подання\нзвіту про використання та запаси палива за
17    лютий 2024 року')
18 elif d==7:
19    print('Останній день подання\нзвіту про реалізацію продукції сільського
20    господарства за лютий 2024 року \нзвіту з праці за лютий 2024 року\нзвіту
21    про заборгованість з оплати праці за лютий 2024 року')
22 elif d==8:
23    print('Останній день сплати\нчастини чистого прибутку (доходу)
24    державними унітарними підприємствами та їх об'єднаннями за результатами
25    фінансово-господарської діяльності за 2023 рік\нПодатку на прибуток
26    підприємства за 2023 рік\нПодатку на додану вартість особою-нерезидентом,
27    зареєстрованою як платник ПДВ')
28 else:
29    print('Сьогодні ви відпочиваєте')
```

Оскільки дані типу текст мають бути в один рядок, а редактор Пайтона не переносить рядки так, як це роблять інші текстові редактори.

Тому при наборі довгих текстів повідомлень, варто це робити в редакторі

Блокнот. Результат роботи подано нижче.

```
Введіть день березня 2024 року7
Останній день подання
Звіту про реалізацію продукції сільського господарства за л
ютий 2024 року
Звіту з праці за лютий 2024 року
Звіту про заборгованість з оплати праці за лютий 2024 року
-----
Введіть день березня 2024 року6
Останній день подання
Звіту про використання та запаси палива за лютий 2024 року
```

Тримісний вираз if / else працює за наступною інструкцією:

```
if X:
    A = Y
else:
    A = Z
```

Хоч ця інструкція досить коротка, але, тим не менше, займає цілих 4 рядки.

Спеціально для таких випадків і було придумано вираз if / else:

```
A = Y if X else Z
```

У даній інструкції інтерпретатор виконає вираз Y, якщо X істинно, в іншому випадку виконається вираз Z.

Скористаємося цією конструкцією для визначення тарифу на оплату електроенергії на конкретний час доби за схемою: день, з 07:00 до 23:00 - 2,64 грн/кВт-год; ніч, з 23:00 до 07:00 - 1,32 грн/кВт-год.

```
1 hh=str(input('Введіть поточний час доби\n за 24-схемою типу 12:38\n ---> '))
2 t1='2,64 грн/кВт-год'
3 t2='1,32 грн/кВт-год'
4 h1=hh[:1]
5 h2=hh[3:]
6 tarif = t1 if ((int(h1)>= 7 and int(h2) >=0)or(int(h1)<= 23 and int(h2)
<=59)) else t2
7 print('У цей момент діє тариф ',tarif)
```



Перша версія програми малі тільки 2 рядки, але транслятор показував помилку, тому прийшлося виділення часу і хвилин з тестової змінної та переведення їх у цілочисловий формат винести в окремі змінні. Ось результат виконання робочої версії.

```
Введіть поточний час доби
за 24-схемою типу 12:38
---> 14:42
У цей момент діє тариф    2,64 грн/кВт-год
|
```

## 3.2. Оператори циклу

Інколи в програмі потрібно повторити декілька разів одну і ту ж дію. Наприклад, щоб знайти суму всіх чисел масиву, треба до першого числа додати друге, третє і т.д. Тобто, потрібно повторити операцію складання стільки разів, скільки чисел у масиві. Для того, що не писати багато разів складання, треба скористатися операторами циклу. Тоді ви напишете операцію складання один раз, а оператор циклу повторить її необхідну кількість разів.

### 3.2.1. WHILE

*While* – один з найбільш універсальних циклів в Python, тому досить повільний. Виконує тіло циклу до тих пір, поки умова циклу істинна. Наприклад,

```
i = 5
while i < 10:
    print(i)
    i = i + 2
5
7
9
11
```



При виконанні циклу `while` спочатку перевіряється умова. Якщо вона помилкова, то виконання циклу припиняється і керування передається на наступну інструкцію після тіла циклу `while`. Якщо умова істина, то виконується інструкція, після чого умова перевіряється знову і знову виконується інструкція. Так триває до тих пір, поки умова буде істинно. Як тільки умова стане помилково, робота циклу завершиться і управління передається наступній інструкції після циклу. Це єдиний оператор, після якого можуть йти декілька вкладених операторів.

Наприклад, наступний фрагмент програми надрукує на екран квадрати всіх цілих чисел від 1 до 10.

```
i = 1
while i <= 10:
    print(i ** 2)
    i += 1
```

У цьому прикладі змінна `i` всередині циклу змінюється від 1 до 10. Така змінна, значення якої змінюється з кожним новим проходом циклу, називається лічильником. Зауважимо, що після виконання цього фрагмента значення змінної `i` дорівнюватиме 11, оскільки саме при `i == 11` умова `i <= 10` вперше перестане виконуватися.

Ось ще один приклад використання циклу `while` для визначення кількості цифр натурального числа `n`:

```
n = int(input())
length = 0
while n > 0:
    n //= 10 # Це еквівалентно визначенню цілої частини від ділення на 10
    length += 1
print(length)
```

У цьому циклі ми відкидаємо по одній цифрі числа, починаючи з кінця, що еквівалентно цілочисловому діленню на 10 (`n //= 10`), при цьому в змінній `length` визначаємо, скільки разів це було зроблено.

У мові Пітон є й інший спосіб вирішення цього завдання: `length = len(str(n))`.

Тут Пайтон спочатку переводить число у текст, а потім визначає кількість цифр у числі.

Після тіла циклу можна написати слово `else`: і після нього блок операцій, який буде виконаний один раз після закінчення циклу, коли перевіряється умова стане невірною. Тут `while` виступає аналогом `for`, тому `else` пишеться на одному рівні з `while`.

```
i = 1
while i <= 10:
    print(i)
    i += 1
else:
    print('Цикл закінчено, i =', i)
```

Здавалося б, ніякого сенсу в цьому немає, адже цю ж інструкцію можна просто написати після закінчення циклу. СENS з'являється тільки разом з інструкцією `break`. Якщо під час виконання Пайтон зустрічає інструкцію `break` всередині циклу, то він відразу ж припиняє виконання цього циклу і виходить з нього. При цьому гілка `else` виконуватися не буде. Зрозуміло, інструкцію `break` має сенс викликати тільки всередині інструкції `if`, тобто вона повинна виконуватися тільки при виконанні якоїсь особливої умови.

Наведемо приклад програми, яка зчитує числа до тих пір, поки не зустріне від'ємне число. При появі від'ємного числа програма завершується. У першому варіанті послідовність чисел завершується числом 0 (при зчитуванні якого треба зупинитися).

```
1 # Цикл WHILE
2 print('Перевірка на від'ємне число. При появі програма спиняється')
3 a=10
4 while a>0 :
5     a = int(input('Введіть довільне число, вказавши знак: '))
6     if a < 0:
7         print('Зустрілось від'ємне число', a)
8         break
9 else:
0     print('Жодного від'ємного числа не зустрілося')
```

```
Перевірка на від'ємне число. При появі програма спиняється
Введіть довільне число, вказавши знак: +4
Введіть довільне число, вказавши знак: +30
Введіть довільне число, вказавши знак: -7
Зустрілось від'ємне число -7
```

### 3.2.2. FOR

Для розуміння відміни цього циклу від `while`, розгляньте програму, що виконує те саме, що й остання з п. 3.2.1.

```
n = int(input())
for i in range(n):
    a = int(input())
    if a < 0:
        print('Зустрілось від'ємне число', a)
        break
else:
    print('Жодного від'ємного числа не зустрілося')
```

Тут необхідно ввести кількість разів (`n`), коли цикл повторить визначення, чи є введене число (`a`) від'ємним? І тільки `break` припиняє цей цикл.

Синтаксис циклу має включати в себе лічильник кількості разів його виконання. У прикладі це змінна `i`, але можна задати будь яке ім'я, яке б не змінювалося б усередині циклу.

Цикл `for` вже трішки складніший, трохи менше універсальний, але виконується набагато швидше циклу `while`. Цей цикл проходиться по будь-якому об'єкту, що піддається повтору, або як кажуть ітерації (рядку, списку, ...), і під час кожного проходу виконує тіло циклу.

```
for i in 'hello world': print (i * 2, end = " ")
hheelllloo wwoorrlldd
```

Оператор *continue* починає наступний прохід циклу, минаючи тіло циклу, що залишилося (for або while) і виконання циклу продовжується з наступної ітерації.

```
>>> for i in 'hello world':
    if i == 'o': continue
    print (i * 2, end = ")
hheellll wwrrlldd
```

Оператор *break* достроково перериває цикл.

```
for i in 'hello world':
    if i == 'o':
        break
    print (i * 2, end = ")
hheellll
```

Слово *else*, застосоване в циклі for або while, перевіряє, чи був проведений вихід з циклу інструкцією *break*, або ж "природним" чином. Блок інструкцій всередині *else* виконається тільки в тому випадку, якщо вихід з циклу стався без допомоги *break*.

```
for i in 'hello world':
    if i == 'a':
        break
    else:
        print ( 'Букви а в рядку немає')
        Букви а в рядку немає
```

З прикладів добре видно, що результати роботи програми з'являються у тому ж вікні, що й сам текст програми.

Інша інструкція управління циклом – **continue** (продовження циклу). Якщо ця інструкція зустрічається десь посередині циклу, то пропускаються всі інструкції, що залишилися до кінця циклу.

В операторі for існує поняття внутрішнього циклу або якогось іншого оператора. Для цього цей оператор потрібно розмістити зі зсувом управо,

натиснувши кнопку TAB. В стандартних налаштуваннях зміщення виконується на три позиції.

Якщо інструкції **break** і **continue** містяться всередині декількох вкладених циклів, то вони впливають лише на виконання самого внутрішнього циклу. Наведемо приклад, який це демонструє:

```
for i in range(3):
    for j in range(5):
        if j > i:
            break
        print(i, j)
```

А ось приклад, коли вводиться матриця  $a$ , для якої розраховується сума елементів по рядках, починаючи з другої колонки. Ця сума множиться на квадрат номеру колонки і записується в матрицю  $q$ .

```
File Edit Format Run Options Window Help
1 a=[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
2 q=[]
3 c=0
4 for i in range(3):
5     for j in range(2):
6         for k in range(2):
7             if k<1:
8                 continue
9             else:
10                c+=k**2*a[i][j]
11        q.append(c)
12 print(q)
```

```
| [3, 12, 27]
```

Оскільки посилання на масив  $q$  знаходиться під циклом для  $i$ , то номер елементу масиву вставляється автоматично.

### 3.3. Оператори вводу та виводу даних

Розрахунки, які виконує програма, ведеться за даними, що їх надає користувач. Так само, користувачу потрібно отримати результати цих розрахунків. Тому й потрібні оператори вводу та виводу даних.

З найпростішим варіантом виведення ви вже знайомі – це оператор `print`.

З попередніх прикладів ви підсвідомо вже засвоїли, що `print` може вивести будь яку послідовність символів або цифр, якщо в дужках вказати їх через кому.

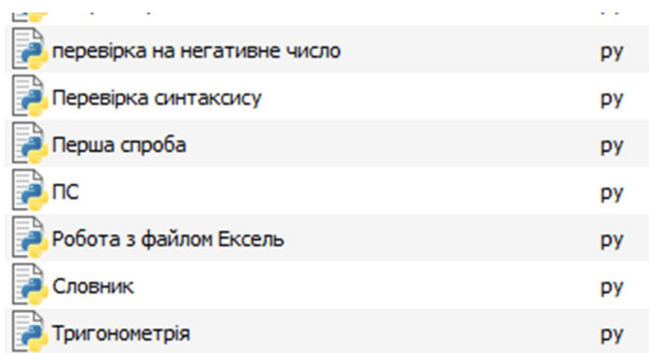
Наприклад, `print("Результати розрахунку", a)`.

Ось його повна структура з якої ми вже вивчили тільки `value` та `\n`. Але для економістів цього цілком достатньо, що вирішувати всі необхідні задачі.

```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

А куди цей текст і число потраплять ви теж знаєте – в стрічку інтерпретатора IDLE. А що буде, коли програма не відкрита в IDLE?

Виявляється Пайтон продовжує слідувати за всіма файлами, що мають розширення `*.py`, підключаючи до імені таких файлів у файл-менеджері свою іконку, ось так.



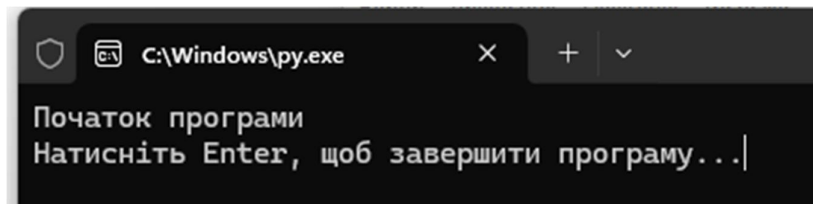
Працюючи з файлами Word, Excel та іншими, ми вже знаємо, що варто клацнути по ним, як відповідна програма ввімкнеться і ми отримаємо можливість з нею працювати. З Пайтон так само, але програма буде виконуватися у вікні Термінал, там де ми інсталиювали бібліотеки. Після виконання програми, це вікно негайно закриється і ми не встигнемо прочитати результат розрахунків.

Щоб цьому запобігти, в кінці програми потрібно вставити команду `exit_text = input("Натисніть Enter, щоб завершити програму...")`

До речі, використання команди `exit()`, припиняє роботу програми негайно. Цей оператор може бути корисною заміною команди `break`, яка діє тільки для циклів, а не для всієї програми.

Замість трьох крапок можна вставити назву програми.

Тоді ми отримаємо вікно Терміналу, яке не закриється у буде чекати на натискання кнопки Enter.



Якщо нам потрібно вивести масив даних у вигляді таблиці, треба запам'ятати, що робота оператора `print` виконується по рядках таблиці.

Для прикладу, візьмемо масив зі списком співробітників з п.2.3 і виведемо його як таблицю із заголовком і лініями граф ось такого виду.

ПІБ	Посада	Оклад, грн
Фрідман Семен Якович	директор	15000
Дмитренко Оксана Семенівна	головбух	12000
Чепель Василь Михайлович	шофер	10000

Перше, що ми повинні зробити, це вирівняти кількість символів у ПІБ та Посаді, інакше графи таблиці будуть нерівними. Для цього потрібно тільки додати зайві пробіли у текстові змінні.

Горизонтальні лінії таблиці можна робити символом «мінус», а вертикальні – символом «|».

У заголовку таблиці теж потрібно врахувати ширину найбільшої змінної, щоб графи заголовків збігалися з графами даних, що будуть виводитися.

Ось як виглядає така програма.

```

1 emp=[[ 'Фрідман Семен Якович      ', 'директор', 15000],[ 'Дмитренко
  Оксана Семенівна', 'головбух', 12000],[ 'Чепель Василь Михайлович
  ', 'шофер ',10000]]
2 # Найбільша довжина ПІБ - 26 символів
3 # Найбільша довжина Посади - 8 символів
4 # тому для коротких рядків додано пробіли
5 print('=====')
6 print('I          ПІБ                I Посада I Оклад, грн I')
7 print('=====')
8 for i in range(3):
9     print('I ', emp[i][0], 'I', emp[i][1], 'I ', emp[i][2], ' I
10 ')
11 print('=====')
exit_text=input('Якщо ви вже ознайомилися з таблицею, натисніть
Enter')

```

```

=====
I          ПІБ                I Посада I Оклад, грн I
=====
I Фрідман Семен Якович      I директор I   15000 I
I Дмитренко Оксана Семенівна I головбух I   12000 I
I Чепель Василь Михайлович  I шофер   I   10000 I
=====
Якщо ви вже ознайомилися з таблицею, натисніть Enter

```

З цієї програми ви також можете побачити, як друкувати двовимірний масив даних.

Введення даних з клавіатури виконується оператором **input**, який має таку структуру.

Змінна = **input**('Підказка для оператора')

Тобто, результат роботи цього оператора неодмінно має бути присвоєний якійсь змінній, а підказка слугує нагадуванням оператору про те, що і як потрібно набирати на клавіатурі.

За одну дію оператора **input** можна ввести тільки одну змінну. Як же ввести масив? По одному числу, наприклад.

```

a=[]
for i in range(4):
    a.append(input())
print(a)

```



При відкриванні масиву, він містить тільки одну комірку, але команда **append** додає по одній новій комірці. І ось що вийшло.

```
12
'Семен'
12.56e+12
'УраУра'
['12', "'Семен'", '12.56e+12', "'УраУра'"]
```

А вийшло так, що все, що ми вводили, має текстовий формат! Отже, щоб скористатися комірками з числами, ми повинні перетворити їх із тесту на число, наприклад отак `a[0]=int(a[0])` або `a[2]=float(a[2])`. Ми ж пам'ятаємо, що нумерація елементів масиву починається з нуля?

Якщо наш масив даних складається з однотипних елементів, то варто одразу перетворити дані, що вводяться, у потрібний формат, наприклад

**`a.append((float(input())))`**

Будь хто з економістів неодмінно скаже, що вище було описано роботу з даними, розташованими в оперативній пам'яті комп'ютера. Але ж, скажуть вони, робота економіста полягає не тільки в обрахунках, а і у збереженні результатів цих розрахунків. Та ще й так, щоб можна було б їх передивитися декілька разів, не запускаючи програму повторно. Тобто, економісти хочуть, аби отримані дані були розміщені у файлі. І така можливість існує.

Перш, ніж працювати з файлом, його треба відкрити. З цим чудово впорається вбудована функція **open** (тут `f` – допоміжна змінна):

**`f = open ('ім'я файлу.txt', 'Режим відкриття')`**

У функції **open** багато параметрів, але нам тільки важливі три аргументи: перший, це ім'я файлу. Шлях до файлу може бути відносним або абсолютним. Якщо шлях не сказано, то програма буде шукати цей файл у тій же директорії,

де знаходиться сама. Другий аргумент, це режим, в якому ми будемо відкривати файл. Ім'я файлу та шлях до нього мають бути записані тільки латиницею.

Режим	Позначення
'r'	Відкривання для читання (є значенням за замовчуванням).
'w'	Відкривання для запису, зміст файлу видаляється, якщо файлу не існує, тоді створюється новий.
'x'	Відкривання для запису, якщо файлу не існує, то – збій.
'a'	Відкривання для дозапису, інформація додається у кінець файлу.
'b'	Відкривання для перегляду у двійковому режимі.
't'	Відкривання для перегляду в текстовому режимі (є значенням за замовчуванням).
'+'	Відкривання для читання й запису

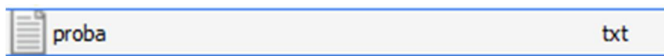
Режими можуть бути об'єднані, тобто, наприклад, 'rb' – читання в двійковому режимі. За замовчуванням режим дорівнює 'rt'.

З урахуванням потреб економістів, з усіх режимів підходить 'r', 'a', 'w'.

Відкрили ми файл, а тепер ми хочемо прочитати з нього інформацію. Для цього є кілька способів, але великого інтересу заслуговують лише два з них.

Перший – метод `read`, – читає весь файл цілком, якщо був викликаний без аргументів, і *n* символів, якщо був викликаний з аргументом (цілим числом *n*).

Для прикладу створимо тестовий файл



, в який запишемо одне слово

'Пістунов', редактором Блокнот. Тепер прочитаємо цей текст, спочатку перший символ, а потім усі.

```
1 with open('proba.txt') as file:
2     content = file.read(1)
3     print(content)
4 with open('proba.txt') as file:
5     content = file.read()
6     print(content)
```

```
П
Пістунов
```

Конструкція `with ...as` зручна тим, що вона одразу закриває файл після читання. Якщо використовувався оператор типу `file = open`, то після закінчення програми обов'язково необхідно поставити оператор закриття файлу типу

### **`file.close()`**

Тут ім'я файлу – `file`.

Конструкція `with ... as` використовується для повернення виконання блока інструкцій менеджером контексту.

Синтаксис конструкції `with ... as`:

```
"with" expression ["as" target] ("," expression ["as" target])* ":" suite
```

Тепер по порядку того, що відбувається при виконанні даного блоку:

1. Виконується вираз `expression` в конструкції `with ... as`.
2. Завантажується спеціальний метод `__exit__` для подальшого використання.
3. Виконується метод `__enter__`. Якщо конструкція включає в себе слово `as`, то методом `__enter__` значення, що повертається, записується в змінну, яку ми обрали для вмісту файлу в оперативній пам'яті.
4. Виконується дія переміщення даних файлу у змінну.
5. Викликається метод `__exit__`, причому не має значення, чи виконаний вміст блоку, який у структурі позначено як `Suite`, чи сталося вимкнення. В цей метод передаються параметри вимкнення, якщо воно відбулося, або у всіх аргументах значення `Non`, якщо не було виключень.

Якщо в конструкції `with - as` було декілька виразів, то це еквівалентно декільком вкладеним конструкціям:

```
with A() as a, B() as b:  
    suite
```

Еквівалентно:

```
with A() as a:  
    with B() as b:  
        suite
```

Для чого застосовується конструкція `with - as`? Для гарантування того, що критичні функції виконуються в будь-якому випадку. Самий розповсюджений приклад використання цієї конструкції – відкриття файлів. зазвичай відкриття файлів здійснюється за допомогою функції `open`, але конструкція `with - as`, як правило, є більш зручною і гарантує закриття файлу в будь-якому випадку.

Тепер запишемо у файл три рядки тесту, ось так

```
1 Пістунов
2 Ігор
3 Миколайович
```

Спробуємо прочитати його по рядках, скориставшись циклом `for`. Прочитано і видано на консоль.

```
1 with open('proba.txt') as file:
2     for line in file:
3         print(line)
```

Пістунов  
Ігор  
Миколайович

Тут `line` – допоміжна змінна, в яку у циклі попадає весь рядок тексту з файлу.

Тепер розглянемо запис в файл. Спробуємо записати в файл ту саму таблицю зі списком співробітників. Для цього файл треба відкрити на запис.

Відкриємо файл на запис і замість оператора `print`, використаємо оператор `f.write`. Тут `f` – допоміжна змінна. Зверніть увагу, що при виведенні даних прийшлося перетворити їх на один текстовий рядок, бо оператор `write` списку змінних не сприймає, на відміну від оператора `print`.

```
1 emp=[['Фрідман Семен Якович', 'директор', 15000], ['Дмитренко Оксана Семенівна', 'головбух', 12000], ['чепель Василь Михайлович', 'шофер', 10000]]
2 # Найбільша довжина ПІБ - 26 символів
3 # Найбільша довжина Посади - 8 символів
4 # тому для коротних рядків додано пробіли
5
6 f=open('tabl.txt', 'w')
7
8 f.write('=====\n')
9 f.write('I          ПІБ          I Посада I Оклад, грн I\n')
10 f.write('=====\n')
11 for i in range(3):
12     f.write('I '+ str(emp[i][0])+' I '+ str(emp[i][1])+' I '+str(
emp[i][2])+' I \n')
13 f.write('=====\n')
14 f.close()
15 exit_text=input('Якщо ви вже ознайомилися з таблицею, натисніть Enter')
```

Тепер з'явився файл tabl.txt з наступним вмістом.

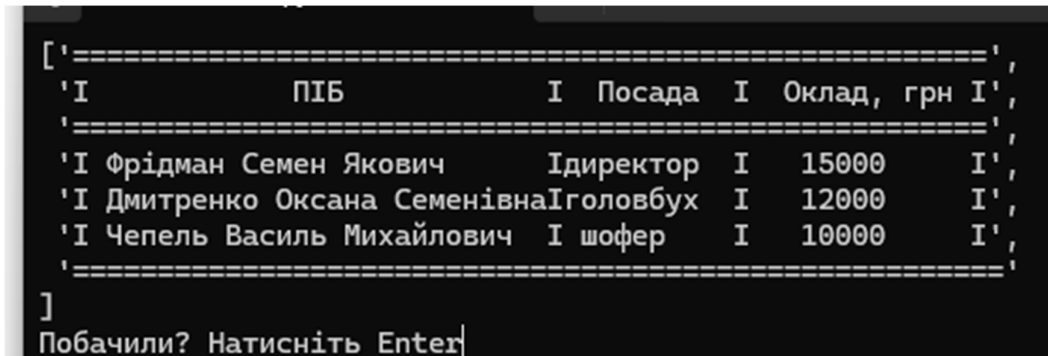
```
Файл  Правка  Вигляд  Plugins  Кодування  Довідка
=====
I      ПІБ      I  Посада  I  Оклад, грн I
=====
I Фрідман Семен Якович      Iдиректор  I  15000  I
I Дмитренко Оксана СеменівнаIголовбух  I  12000  I
I Чепель Василь Михайлович I шофер    I  10000  I
=====
```

Після закінчення роботи з файлом, якщо не був застосований метод **with...as**, його обов'язково потрібно закрити за допомогою методу **close**.

Тепер спробуємо прочитати цей файл і вивести його на екран.

Відкриємо файл на читання і прочитаємо рядки.

```
1 f = open ( 'tabl.txt', 'r')
2 print( [line.strip () for line in f])
3 end_progr=input('Побачили? Натисніть Enter')
```



```
[ '=====',
'I      ПІБ      I  Посада  I  Оклад, грн I',
'=====',
'I Фрідман Семен Якович      Iдиректор  I  15000  I',
'I Дмитренко Оксана СеменівнаIголовбух  I  12000  I',
'I Чепель Василь Михайлович I шофер    I  10000  I',
'=====',
]
Побачили? Натисніть Enter
```

Отже, зчитування файлу ведеться по рядках тексту і з тестового формату може бути перетворене у числові формати. Достатньо тільки знати, з якої позиції знаходяться ті числа.

Наприклад, у файлі tabl.txt зарплата директора знаходиться на 44-48 символах четвертого рядку.

Отже, необхідно відкрити файл і весь його вміст покласти у допоміжну змінну, перевірити, чи є у ньому 4 рядки, чи є у рядку 48 символів.

Якщо все вірно, витягнути необхідні символи, якщо ні, подати повідомлення про невірні номери рядків та символів.

```

1  # Відкриття файлу в режимі читання
2  with open('tab1.txt', 'r') as file:
3      # Читання всіх рядків у файлі
4      lines = file.readlines()
5
6      # Перевірка, чи є в файлі принаймні 4 рядки
7      if len(lines) >= 4:
8          # Отримання четвертого рядка
9          line_4 = lines[3]
10
11         # Перевірка, чи має рядок принаймні 48 символів
12         if len(line_4) >= 48:
13             # Витяг символів з 44 по 48 (Python індексується
14             з 0)
15             characters = line_4[43:48]
16             print(f"Символи з 44 по 48 у четвертому рядку:
17             {characters}")
18         else:
19             print("четвертий рядок не має 48 символів.")
20     else:
21         print("у файлі немає 4 рядків.")
22 end_progr=input('Побачили? Натисніть Enter ')

```

Символи з 44 по 48 у четвертому рядку: 15000  
Побачили? Натисніть Enter

Недоліком роботи з файлом у базових можливостях Python є те, що файл може бути тільки текстового формату.

Більше можливостей зі вводу та виводу ви узнаете з розділу 4.

### 3.4. Підпрограми-функції

При розрахунку економічних показників ми часто зустрічаємося з тим що деякі формули або який порядок розрахунку використовуються не один раз. Для зменшення обсягу програми використовуються підпрограми-функції.

Функція у програмуванні – це окрема програма, що робить розрахунки на підставі даних, які їй передають як аргументи. Після проведення розрахунків, функція повертає результат, так, що типовим використанням функції є

**A = Fun(Arg1, Arg2, Arg3)**

Тут Fun означає ім'я підпрограми функції, а Arg – це аргументи, що містять дані, необхідні для розрахунків. Після виконання роботи підпрограми змінна **a** отримує результат.

Отже, функція в Python – це об'єкт, який приймає аргументи і повертає значення. Зазвичай функція визначається за допомогою інструкції `def`.

Визначимо найпростішу функцію:

```
def add (x, y):  
    return x + y
```

Інструкція `return` каже, що потрібно повернути якесь значення. В нашому випадку функція повертає суму `x` та `y`. Зверніть увагу на те, що після оператора `def` стоїть двокрапка, тобто, всі оператори, які виконуються в межах функції мають бути зсунуті праворуч, щоб позначити те, що вони є внутрішніми для цієї функції. Це стосується і оператора `return`.

Ще однією особливістю використання функцій в Python є вимога розташування функцій перед основними операторами, що їх використовують, наприклад

```
1 def a (f) :  
2     f+=1  
3     return f  
4 z=2  
5 z=z+a (z)  
6 print (z)
```

Тут спочатку визначається функція `a(f)`, що закінчується оператором `return`. Потім ідуть основні оператори. В результаті ми отримаємо число 5.

Функція може бути будь-якої складності і повертати будь-які об'єкти (списки, кортежі, і навіть функції!).

У наступному прикладі формується функція **myfunc** у середині функції **newfunc**. Потім у **newfunc** передається число 100 і водночас у операторі



присвоювання визначається нове ім'я функції – **new**. Потім у цю функцію передається число 200 і в результаті отримуємо число 300.

```
1 def newfunc (n):
2     def myfunc (x):
3         return x + n
4     return myfunc
5 new = newfunc (100) # new - це нове ім'я функції
6 print( new (200))
```

Функція може і не закінчуватися інструкцією **return**, при цьому функція поверне значення None:

```
>>> def func (): pass
>>> print (func ())
None
>>>
```

Але такої конструкції варто уникати.

Функція може приймати будь-яку кількість аргументів чи не приймати їх зовсім. Також поширені функції з довільним числом аргументів, функції з позиційними і іменованими аргументами, обов'язковими і необов'язковими.

```
1 def func(a, b, c = 2): # c - необов'язковий аргумент
2     return a + b + c
3 print("Введено значення тільки обов'язкових аргументів і ось
4     результат ",func (1, 2))
5     # a = 1, b = 2, c = 2 (за замовчуванням)
6 print("Введено значення всіх аргументів і змінено необов'язковий
7     аргумент і ось результат ",func (1, 2, 3))
8     # a = 1, b = 2, c = 3
9 print("Введено значення тільки обов'язкових аргументів в іншому
10    форматі введення і ось результат ", func (a = 1, b = 3))
11    # a = 1, b = 3, c = 2
```

Введено значення тільки обов'язкових аргументів і ось результат 5

Введено значення всіх аргументів і змінено необов'язковий аргумент і ось результат 6

Введено значення тільки обов'язкових аргументів в іншому форматі введення і ось результат 6



Функція також може приймати змінну кількість позиційних аргументів, тоді перед ім'ям ставиться \*:

```
1 def func(* args):
2     return args
3 print(func(1, 2, 3, 'abc'))
4 print(func ())
5 print(func(1, 'Петренко'))

(1, 2, 3, 'abc')
()
(1, 'Петренко')
```

Як видно з прикладу, args – це кортеж з усіх переданих аргументів функції, і зі змінною можна працювати так само, як і з кортежем.

Функція може приймати і довільне число іменованих аргументів, тоді перед ім'ям ставиться \*\*:

Приклад аргументів функції:

```
1 def func(** kwargs):
2     return kwargs
3 print(func(a = 1, b = 2, c = 3))
4 print(func ())
5 print(func(a = 'python'))

{'a': 1, 'b': 2, 'c': 3}
{}
{'a': 'python'}
```

В змінній kwargs у нас зберігається словник, з яким ми, знову-таки, можемо робити все, що нам заманеться.

Анонімні функції можуть містити лише один вислів, але виконуються вони швидше. Анонімні функції створюються за допомогою інструкції **lambda**.

Крім цього, їх не обов'язково привласнювати змінній, як робили ми інструкцією **def func ()**.

У наступному прикладі створюється нова функція з іменем **proba\_funk**, що виконує дії з чотирма аргументами, серед яких **t** – необов'язковий аргумент, якому спочатку присвоюється число 2. Ця функція використовується двічі з різними формами завдання аргументів. Причому, у другому випадку не вказано значення **t**, тому функція використовує його наперед задану величину.

```
1|proba_funk=lambda x,y,z,t=2: (x+y)**z/t
2|print(proba_funk(23,12,3,53))
3|print(proba_funk(x=23,y=12,z=3))
|
| 808.9622641509434
| 21437.5
```

Оператор **lambda** для створення функції, на відміну від звичайної, не потрібно інструкція **return**, а в іншому, поводитьься так само, як і стандартна функція:

```
>>> func = lambda * args: args
>>> func(1, 2, 3, 4)
(1, 2, 3, 4)
```

І тепер можемо створити повноцінну функцію визначення чистого приведенного прибутку (NPV), в яку передаються масиви даних доходів (In), витрат (Lo) та значення норми дисконту Et. Функція викликається два рази, причому імена масивів відрізняються від імен в самій функції . Також різною є розмір масивів.

```

1  # Розрахунок чистого приведеного прибутку, визначного на початок
   проекту
2
3  # Підпрограма NPV
4  def NPV(In,Lo,Et):
5      npv=0
6      for t in range(len(In)):
7          npv+=(In[t]-Lo[t]) / (1+Et)**t
8      return npv
9
10 # Основна програма
11 # Перше звертання до функції
12 Lo1=[37.5, 65.25, 40.5, 61.5, 53.25, 40.5, 54.0, 48.0]
13 In1= [136.3, 71.05, 121.8, 76.85, 107.3, 124.7, 71.05,
14      123.25]
15 Et1=0.25
16 print(NPV(In1,Lo1,Et1))
17
18 # Друге звертання до функції з іншими іменами масивів і даних
19 Doh=[56055.6,65804.4,77990.4,40620.0,40620.0,64179.6,61742.4,73928.
20     4,53618.4,78802.8,46306.8,63367.2,69054.0,47931.6,51993.6,39807.6]
21 Vytr=[42563.2,35318.4,28073.6,42563.2,30790.4,34865.6,33507.2,32148
22     .8,23998.4,40752.0,21734.4,42563.2,28526.4,34412.8,22640.0,36224.0]
23 NDisk = 0.15
24 print(NPV(Doh,Vytr,NDisk))
25 z=input()

```

```

233.31135999999998
169982.08543733263

```

### 3.5. Створення GUI додатку

Вивчивши таку кількість різних операторів Пайтона, стає необхідним створювати власний інтерфейс програми, а не користуватися інтерфейсом інтерпретатора IDLE або Термінал.

Хочеться, аби програма була схожа на інші програми, щоб вона мала своє вікно у вигляді прямокутника певного забарвлення, віконця, куди вводяться дані, віконця, де виводиться результат, кнопки, тощо.

Для цього у Пайтон є GUI додаток – це графічний інтерфейс користувача (Graphical User Interface), який дозволяє користувачам взаємодіяти з програмою за допомогою візуальних елементів, таких як вікна, кнопки, текстові поля, замість текстових команд. Tkinter є стандартною бібліотекою Python для створення GUI. Вона використовується для розробки додатків, які можуть

працювати на різних операційних системах, забезпечуючи кросплатформену сумісність.

Tkinter надає різні елементи управління, такі як **buttons, labels, text boxes**, загалом все, що використовується у програмі з графічним інтерфейсом. Ці елементи зазвичай у IT світі називають віджетами.

Python надає різні варіанти розробки GUI. Найважливіші з них:

- **Tkinter** - це інтерфейс Python для TK GUI Toolkit, поставляється з Python.
- **wxPython** - це open-source Python інтерфейс для wxWindows
- **JPython** є портом Python для Java, який дає скриптам Python прямий доступ до бібліотек класів Java на локальній машині.

### Віджети Tkinter

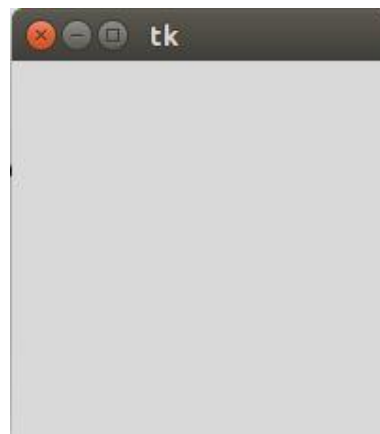
Tkinter – GUI бібліотека для Python. Python в поєднанні з Tkinter забезпечує швидкий та простий спосіб у створенні додатків з графічним інтерфейсом. Створення додатків з графічним інтерфейсом є простим завданням, якщо використовувати Tkinter. Все, що вам потрібно, це виконати наступні дії:

- Імпортувати модуль Tkinter.
- Створити головне вікно вашого GUI додатку.
- Додати віджети до GUI додатку.
- Ввести цикл обробки події.

Приклад:

```
from tkinter import *  
top=Tk()  
#Код для віджетів  
top.mainloop()
```

Ось що вийшло:



Як бачите, ми просто створили вікно програми з іменем tk..

Модуль Tkinter на даний момент забезпечує підтримку 15 віджетів:

- **Button** відображає кнопку у вашому додатку;

Приклад використання віджета Button у модулі Tkinter для створення кнопки, яка закриває програму

```
1 # Створення головного вікна
2 root = tk.Tk()
3 root.title('Приклад кнопки')
4
5 # Функція, яка буде викликатися при натисканні на кнопку
6 def close_window():
7     root.destroy()
8
9 # Створення кнопки, яка викликає функцію close_window
10 button = tk.Button(root, text='Закрити', command=close_window)
11
12 # Розміщення кнопки у головному вікні
13 button.pack()
14
15 # Запуск головного циклу обробки подій Tkinter
16 root.mainloop()
```

У цьому коді створюється головне вікно з заголовком “Приклад кнопки”. Кнопка з текстом “Закрити” розміщується у вікні, і при її натисканні викликається функція `close_window`, яка закриває вікно.

- **Canvas** даний віджет використовується для малювання;

Наприклад, створимо вікно з Canvas, на якому ми малюємо лінію, прямокутник та овал. Ви можете змінити параметри методів `create_line`, `create_rectangle` та `create_oval`, щоб налаштувати форми за своїм бажанням.

```

1 import tkinter as tk
2
3 # Створення головного вікна
4 root = tk.Tk()
5 root.title('Canvas Example')
6
7 # Створення об'єкта Canvas
8 canvas = tk.Canvas(root, width=400, height=300)
9 canvas.pack()
10
11 # Малювання лінії
12 canvas.create_line(25, 25, 200, 25)
13
14 # Малювання прямокутника
15 canvas.create_rectangle(50, 50, 150, 100, fill="blue")
16
17 # Малювання овалу
18 canvas.create_oval(50, 150, 150, 200, fill="red")
19
20 # Запуск головного циклу Tkinter
21 root.mainloop()

```

- **Checkbutton** використовується для відображення варіантів вибору, як прапорцями, можна вибирати одразу декілька варіантів;

Ось приклад використання віджета Checkbutton модуля Tkinter, який дозволяє користувачу вибрати або скасувати вибір опції.

```

1 import tkinter as tk
2
3 def toggle():
4     if var.get():
5         print("Checkbutton вибрано.")
6     else:
7         print("Checkbutton не вибрано.")
8
9 # Створення головного вікна
10 root = tk.Tk()
11 root.title('Checkbutton Example')
12
13 # Створення змінної типу IntVar для зберігання стану checkbutton
14 var = tk.IntVar()
15
16 # Створення Checkbutton
17 check = tk.Checkbutton(root, text='Виберіть', variable=var, command=toggle)
18 check.pack()
19
20 # Запуск головного циклу обробки подій Tkinter
21 root.mainloop()

```



У цьому коді створюється вікно з Checkbutton, який має текст “Виберіть”. Коли користувач вибирає Checkbutton, стан змінної var змінюється, і викликається функція toggle, яка друкує в консоль стан вибору.

- **Entry** віджет однорядкового текстового поля;

Приклад використання віджета Entry модуля Tkinter, який дозволяє користувачу ввести рядок тексту/ У цьому коді створюється вікно з полем вводу Entry і кнопкою “Submit”. Коли користувач натискає кнопку, функція retrieve\_input викликається і виводить введений текст у консоль.

```
1 import tkinter as tk
2
3 def retrieve_input():
4     input_value = entry.get()
5     print(input_value)
6
7 # Створення головного вікна
8 root = tk.Tk()
9 root.title('Entry Widget Example')
10
11 # Створення віджета Entry
12 entry = tk.Entry(root)
13 entry.pack()
14
15 # Створення кнопки, яка виведе введений текст у консоль
16 submit_button = tk.Button(root, text='Submit', command=retrieve_input)
17 submit_button.pack()
18
19 # Запуск головного циклу Tkinter
20 root.mainloop()
```

- **Frame** використовується як віджет контейнер для організації інших віджетів;

Віджет Frame модуля Tkinter, дозволяє організувати інші віджети в рамках вікна. Тут створюється головне вікно з рамкою сірого кольору, в яку додано кнопку з текстом “Натисни мене”. Коли користувач натискає на кнопку, викликається функція on\_click, яка друкує повідомлення в консоль.

```
1 import tkinter as tk
2 # Функція для демонстрації, що робить кнопка
3 def on_click():
4     print("Кнопка натиснута")
5 # Створення головного вікна
6 root = tk.Tk()
7 root.title('Frame Example')
8 # Створення рамки
9 frame = tk.Frame(root, width=200, height=100, bg='grey')
10 frame.pack()
11 # Додавання кнопки до рамки
12 button = tk.Button(frame, text='Натисни мене', command=on_click)
13 button.pack()
14 # Запуск головного циклу Tkinter
15 root.mainloop()
```



- **Label** використовується для забезпечення однорядкових заголовків для інших віджетів, він також може містити зображення;

Віджет Label модуля Tkinter дозволяє відображати текст у вікні програми. У цьому коді створюється головне вікно з назвою "Label Example". Віджет Label з текстом "Це приклад віджета Label" розміщується у вікні. Коли програма запущена, вона відображає цей текст у вікні1.

```
1 import tkinter as tk
2 # Створення головного вікна
3 root = tk.Tk()
4 root.title('Label Example')
5 # Створення віджета Label
6 label = tk.Label(root, text='Це приклад віджета Label')
7 label.pack()
8 # Запуск головного циклу Tkinter
9 root.mainloop()
```

- **Listbox** використовується для забезпечення списку опцій для користувача;

У цьому коді створюється головне вікно з Listbox, до якого додано чотири елементи. Коли користувач вибирає елемент у Listbox, викликається функція on\_select, яка оновлює текст віджета Label на текст вибраного елемента1.

```
1 import tkinter as tk
2 def on_select(event):
3     # Отримання індексу вибраного елемента
4     index = listbox.curselection()
5     # Отримання тексту вибраного елемента
6     label.config(text=listbox.get(index))
7 # Створення головного вікна
8 root = tk.Tk()
9 root.title('Listbox Example')
10 # Створення віджета Listbox
11 listbox = tk.Listbox(root)
12 listbox.pack()
13 # Додавання елементів до Listbox
14 listbox.insert(1, "Пункт 1")
15 listbox.insert(2, "Пункт 2")
16 listbox.insert(3, "Пункт 3")
17 listbox.insert(4, "Пункт 4")
18 # Створення віджета Label для відображення вибраного елемента
19 label = tk.Label(root, text="")
20 label.pack()
21 # Прив'язка події вибору елемента у Listbox до функції on_select
22 listbox.bind('<<ListboxSelect>>', on_select)
23 # Запуск головного циклу Tkinter
24 root.mainloop()
```

- **Menubutton** забезпечує меню у вашому додатку;

У цьому прикладі створюється головне вікно з Menubutton, який має текст “Опції”. При натисканні на Menubutton відкривається меню з трьома опціями. Коли користувач вибирає опцію, викликається функція on\_option\_select, яка друкує вибрану опцію в консоль.

```
1 import tkinter as tk
2 def on_option_select(value):
3     print(f"Вибрано опцію: {value}")
4     # Створення головного вікна
5     root = tk.Tk()
6     root.title('Menubutton Example')
7     # Створення віджета Menubutton
8     menubutton = tk.Menubutton(root, text='Опції', relief=tk.RAISED)
9     menubutton.pack()
10    # Створення меню, яке буде пов'язане з Menubutton
11    menu = tk.Menu(menubutton, tearoff=0)
12    menubutton['menu'] = menu
13    # Додавання опцій до меню
14    options = ['Опція 1', 'Опція 2', 'Опція 3']
15    for option in options:
16        menu.add_command(label=option, command=lambda value=option:
17            on_option_select(value))
18    # Запуск головного циклу Tkinter
19    root.mainloop()
```

- **Menu** використовується для забезпечення різних команд для користувача. Ці команди містяться всередині Menubutton;

У цьому прикладі створюється головне вікно з меню, яке має підменю “File” з пунктами “New”, “Open...”, та “Exit”. Пункт “Exit” використовує метод root.quit для закриття програми. Інші пункти викликають функцію on\_menu\_click, яка друкує повідомлення в консоль.

```

1 import tkinter as tk
2 # Функція, яка буде викликатися при виборі пункту меню
3 def on_menu_click():
4     print("Пункт меню вибрано")
5 # Створення головного вікна
6 root = tk.Tk()
7 root.title('Menu Example')
8 # Створення об'єкта меню
9 menu = tk.Menu(root)
10 root.config(menu=menu)
11 # Створення підменю 'File'
12 file_menu = tk.Menu(menu, tearoff=0)
13 menu.add_cascade(label='File', menu=file_menu)
14 # Додавання пунктів до підменю 'File'
15 file_menu.add_command(label='New', command=on_menu_click)
16 file_menu.add_command(label='Open...', command=on_menu_click)
17 file_menu.add_separator()
18 file_menu.add_command(label='Exit', command=root.quit)
19 # Запуск головного циклу Tkinter
20 root.mainloop()

```

- **Message** використовується для відображення багаторядкових текстових полів для прийняття значення від користувача;

Віджет Message з текстом, який може бути розміщений на декількох рядках, розміщується у вікні. Коли програма запущена, вона відображає цей текст у вікні.

```

1 import tkinter as tk
2 # Створення головного вікна
3 root = tk.Tk()
4 root.title('Message widget Example')
5 # Створення віджета Message
6 message = tk.Message(root, text="Це приклад віджета Message, який
7 може відображати текст на декількох рядках.")
8 message.pack()
9 # Запуск головного циклу Tkinter
10 root.mainloop()

```

- **Radiobutton** використовується для відображення ряду варіантів, серед яких користувач може вибрати тільки один;

У цьому коді створюється головне вікно з трьома радіокнопками, кожна з яких представляє різні опції. Коли користувач вибирає одну з радіокнопок, викликається функція `selected_option`, яка друкує в консоль вибрану опцію.



```

1 | import tkinter as tk
2 | # Функція, яка буде викликатися при виборі радіокнопки
3 | def selected_option():
4 |     print(f"Вибрано опцію: {var.get()}")
5 |     # Створення головного вікна
6 |     root = tk.Tk()
7 |     root.title('Radiobutton Example')
8 |     # Створення змінної для зберігання вибору
9 |     var = tk.StringVar()
10 |    # Створення радіокнопок
11 |    r1 = tk.Radiobutton(root, text='Опція 1', variable=var, value='1',
12 |                       command=selected_option)
13 |    r1.pack(anchor=tk.W)
14 |    r2 = tk.Radiobutton(root, text='Опція 2', variable=var, value='2',
15 |                       command=selected_option)
16 |    r2.pack(anchor=tk.W)
17 |    r3 = tk.Radiobutton(root, text='Опція 3', variable=var, value='3',
18 |                       command=selected_option)
19 |    r3.pack(anchor=tk.W)
20 |    # Запуск головного циклу Tkinter
21 |    root.mainloop()

```

- **Scale** використовується щоб забезпечити слайдер;

У прикладі, що наведено нижче, створюється головне вікно з горизонтальним повзунком Scale, який має діапазон значень від 0 до 100. Коли користувач переміщує повзунок, викликається функція `on_scale_change`, яка виводить вибране значення в консоль.

```

1 | import tkinter as tk
2 | # Функція, яка буде викликатися при зміні значення повзунка
3 | def on_scale_change(value):
4 |     print(f"Вибране значення: {value}")
5 |     # Створення головного вікна
6 |     root = tk.Tk()
7 |     root.title('Scale Widget Example')
8 |     # Створення віджета Scale
9 |     scale = tk.Scale(root, from_=0, to=100, orient='horizontal',
10 |                    command=on_scale_change)
11 |     scale.pack()
12 |     # Запуск головного циклу Tkinter
13 |     root.mainloop()

```

- **Text** використовується для відображення тексту в декілька рядків, тобто створювати багаторядковий текстовий редактор;

У цьому коді створюється головне вікно з Text віджетом, де користувач може ввести текст. Кнопка “Отримати текст” дозволяє отримати весь текст, введений у Text віджет, і вивести його в консоль.

```

1 | import tkinter as tk
2 | def retrieve_text():
3 |     # Отримання тексту з Text widget
4 |     input_text = text_widget.get("1.0", tk.END)
5 |     print(input_text)
6 |     # Створення головного вікна
7 |     root = tk.Tk()
8 |     root.title('Text Widget Example')
9 |     # Створення віджета Text
10 | text_widget = tk.Text(root, height=10, width=40)
11 | text_widget.pack()
12 | # Створення кнопки для отримання тексту з Text widget
13 | button = tk.Button(root, text='Отримати текст',
14 |                    command=retrieve_text)
15 | button.pack()
16 | # Запуск головного циклу Tkinter
17 | root.mainloop()

```

- **Toplevel** використовується для забезпечення контейнера в окремому вікні, тобто дозволяє створити нове вікно поверх основного вікна програми;

У цьому коді ми створюємо основне вікно з кнопкою “Відкрити Нове Вікно”. Коли користувач натискає на кнопку, викликається функція `create_new_window`, яка створює нове вікно з назвою “Нове Вікно” і містить текстовий напис.

```

1 | import tkinter as tk
2 | # Функція для створення нового вікна
3 | def create_new_window():
4 |     new_window = tk.Toplevel()
5 |     new_window.title('Нове Вікно')
6 |     tk.Label(new_window, text='Це нове вікно').pack()
7 |     # Створення головного вікна
8 |     root = tk.Tk()
9 |     root.title('Головне Вікно')
10 | # Кнопка, яка відкриє нове вікно
11 | tk.Button(root, text='Відкрити Нове Вікно',
12 |           command=create_new_window).pack()
13 | # Запуск головного циклу Tkinter
14 | root.mainloop()

```

**Spinbox** це різновид стандартного віджету `Entry`, який може бути використаний для вибору фіксованої кількості значень, а отже дозволяє користувачу вибрати числове значення з певного діапазону за допомогою повзунка;

У цьому коді створюється головне вікно з Spinbox, який має діапазон значень від 0 до 10. Коли користувач змінює значення за допомогою повзунка, викликається функція on\_value\_change, яка виводить вибране значення в консоль

```
1 | import tkinter as tk
2 | # Функція, яка буде викликатися при зміні значення повзунка
3 | def on_value_change(value):
4 |     print(f"Вибране значення: {value}")
5 |     # Створення головного вікна
6 |     root = tk.Tk()
7 |     root.title('Spinbox Example')
8 |     # Створення віджета Spinbox
9 |     spinbox = tk.Spinbox(root, from_=0, to=10,
10 |        command=on_value_change)
11 |     spinbox.pack()
12 |     # Запуск головного циклу Tkinter
13 |     root.mainloop()
```

- **Panedwindow** являє собою контейнер віджет, який має містити будь-яку кількість панелей, які можуть бути розміщені горизонтально, або вертикально;

У цьому коді створюється головне вікно з двома панелями: лівою і правою. Користувач може переміщати сепаратор між панелями для зміни їх розмірів.

```
1 | import tkinter as tk
2 | from tkinter import ttk
3 | # Створення головного вікна
4 | root = tk.Tk()
5 | root.title('Panedwindow Example')
6 | # Створення Panedwindow віджета
7 | panedwindow = ttk.Panedwindow(root, orient=tk.HORIZONTAL)
8 | panedwindow.pack(fill=tk.BOTH, expand=True)
9 | # Створення лівої панелі
10 | left_pane = ttk.Frame(panedwindow, width=100, height=300,
11 |    relief=tk.SUNKEN)
12 | panedwindow.add(left_pane)
13 | # Створення правої панелі
14 | right_pane = ttk.Frame(panedwindow, width=200, height=400,
15 |    relief=tk.RAISED)
16 | panedwindow.add(right_pane)
17 | # Запуск головного циклу Tkinter
18 | root.mainloop()
```

- **Labelframe** простий контейнер віджет. Його використовують як роздільник або контейнер для складних віконних макетів.

У цьому коді створюється головне вікно з Labelframe, який має назву “Це Labelframe”. В середині Labelframe розміщено віджет Label з текстом “Віджет



всередині Labelframe”. Коли програма запущена, вона відображає Labelframe з вмістом у вікні.

```
1 | import tkinter as tk
2 | from tkinter import ttk
3 | # Створення головного вікна
4 | root = tk.Tk()
5 | root.title('Labelframe Example')
6 | # Створення Labelframe віджета
7 | labelframe = ttk.Labelframe(root, text='Це Labelframe')
8 | labelframe.pack(fill="both", expand="yes")
9 | # Додавання віджета всередину Labelframe
10 | inside_label = ttk.Label(labelframe, text="Віджет
    всередині Labelframe")
11 | inside_label.pack()
12 | # Запуск головного циклу Tkinter
13 | root.mainloop()
```

### Geometry management

Всі віджети Tkinter мають доступ до певних методів управління "геометрією", які мають мету розміщення віджетів по всій області батьківського віджету. Для цього Tkinter надає наступні класи: pack, grid та place.

- `**Pack()**`- організує віджети в блоках перед розміщенням їх на батьківському віджеті
- `**Grid()**`- організує віджети в табличній структурі в батьківському віджеті.
- `**Place()**`- організує віджети, розміщуючи їх в певному положенні на батьківському віджеті.

```
1 | import tkinter as tk
2 | root = tk.Tk()
3 | root.title('Place Example')
4 | # Віджети будуть розміщені за абсолютними координатами
5 | tk.Label(root, text='Лівий верхній кут').place(x=0, y=0)
6 | tk.Label(root, text='Правий нижній кут').place(relx=1.0,
    rely=1.0, anchor='se')
7 | root.mainloop()
```

Кожен з цих методів управління геометрією має свої особливості та найкраще підходить для різних сценаріїв. pack простий у використанні для лінійного розміщення, grid ідеально підходить для створення подібного до таблиці інтерфейсу, а place дає вам повний контроль над точним розташуванням віджетів.



У цьому пункті немає прикладів роботи віджетів. Студентам пропонується самостійно перевірити, як вони працюють згідно наданих прикладів.

Наостанок пропонуємо вам розглянути роботу програми яка реалізує всі ці віджети, що її склала студентка гр. 051-14-1 Марія Малахова для програмної реалізації економіко-математичної моделі у дипломній роботі на тему «Розробка інформаційної системи для оптимізації та прискорення обчислення кошторису для діяльності будівельного підприємства “МТК Україна інжиніринг”» за посиланням

[https://myngu-my.sharepoint.com/:v:/g/personal/pistunov\\_i\\_m\\_nmu\\_one/EQxlmv8wjYtJh6s9K9AuXgBercPBOBpPxs0\\_1aWjWG\\_Q?nav=eyJyZWZlcnJhbEluZm8iOmsicmVmZXJyYWxBcHAIoiJPbmVEcm12ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=9hUONb](https://myngu-my.sharepoint.com/:v:/g/personal/pistunov_i_m_nmu_one/EQxlmv8wjYtJh6s9K9AuXgBercPBOBpPxs0_1aWjWG_Q?nav=eyJyZWZlcnJhbEluZm8iOmsicmVmZXJyYWxBcHAIoiJPbmVEcm12ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=9hUONb)

Загалом, програма містить 421 команду.

### 3.6. Компіляція текстів програм у \*.exe файл

Компіляція програми на Python під Windows може знадобитися, коли програмою буде користуватися людина, у якої на комп'ютері не встановлено Python. Недоліком даного способу є те, що крім самої програми, в папці буде присутній багато файлів, необхідних для її запуску на машинах без Python. Проте, компіляція в .exe дозволяє запускати програму так, як звикли користувачі Windows.

Для цього є кілька способів. Спочатку розглянемо найбільш простий – за допомогою бібліотеки pyinstaller

Дамо в командному рядку Терміналу дві команди:

**pip install wheel**

**pip install pyinstaller**

На екрані з'явиться протокол процесу встановлення цих програм.

```

C:\Users\VM\AppData\Local\Programs\Python\Python36\Scripts\pip.exe
Collecting pyinstaller
  Downloading PyInstaller-3.3.1.tar.gz (3.5MB)
    100% |#####| 3.5MB 341kB/s
Requirement already satisfied (use --upgrade to upgrade): setuptools in c:\users\vm\appdata\local\programs\python\python36\lib\site-packages (from pyinstaller)
Collecting pefile>=2017.8.1 (from pyinstaller)
  Downloading pefile-2017.11.5.tar.gz (61kB)
    100% |#####| 71kB 3.6MB/s
Collecting macholib>=1.8 (from pyinstaller)
  Downloading macholib-1.9-py2.py3-none-any.whl (40kB)
    100% |#####| 40kB 4.1MB/s
Collecting future (from pyinstaller)
  Downloading future-0.16.0.tar.gz (824kB)
    100% |#####| 829kB 1.1MB/s
Collecting pypiwin32 (from pyinstaller)
  Downloading pypiwin32-220-cp36-none-win_amd64.whl (9.0MB)
    99% |#####| 9.0MB 10.2MB/s eta 0:00:01

```

Після установки можна скомпілювати програму, яка поки що знаходиться у файлі з розширенням \*.ру. Створимо, наприклад, на диску C: \ якусь папку з простим ім'ям. Нехай, це буде 1. Помістимо в цю папку скрипт, який ми хочемо скомпілювати – наприклад, 01.ру. Тепер зайдемо в командний рядок і дамо команду:

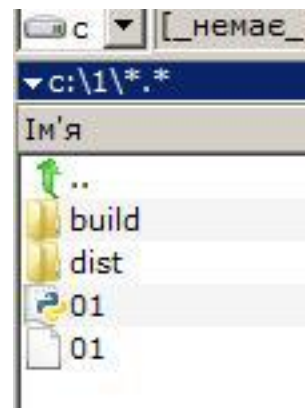
**pyinstaller --onedir --onefile --name=01 "C:\1\01.py"**

Тут слово name означає те ім'я, яке матиме файл з розширенням exe, а текст в лапках означає повний шлях до файлу та ім'я файлу з розширенням ру, який необхідно скомпілювати. Якщо ця команда не спрацює, необхідно вказати повний шлях до цієї команди ось так:

C:\Users\VM\AppData\Local\Programs\Python\Python36\Scripts\pyinstaller --onedir --onefile --name=01 "C:\1\01.py"

Після закінчення роботи pyinstaller в теці, де була тільки одна програма з'явиться декілька нових тек. Файл з прикладу 01.exe буде знаходитися у теці **dist**.

Інколи pyinstaller не справляється з компіляцією складних скриптів. Тому розглянемо альтернативний спосіб компіляції Python скриптів за допомогою cx\_freeze. Для цього отримуємо



потрібну версію програми за адресою [http://www.lfd.uci.edu/~gohlke/pythonlibs/#cx\\_freeze](http://www.lfd.uci.edu/~gohlke/pythonlibs/#cx_freeze) .

Щоб почати отримання файлу, досить тільки клацнути на необхідну версію. Варто також звернути увагу на розрядність вашої операційної системи: 32 чи 64 біта?

**cx\_Freeze**, a set of scripts and modules for freezing Python scripts into executable.

[cx\\_Freeze-5.1.1-cp27-cp27m-win32.whl](#)

[cx\\_Freeze-5.1.1-cp27-cp27m-win\\_amd64.whl](#)

[cx\\_Freeze-5.1.1-cp34-cp34m-win32.whl](#)

[cx\\_Freeze-5.1.1-cp34-cp34m-win\\_amd64.whl](#)

[cx\\_Freeze-5.1.1-cp35-cp35m-win32.whl](#)

[cx\\_Freeze-5.1.1-cp35-cp35m-win\\_amd64.whl](#)

[cx\\_Freeze-5.1.1-cp36-cp36m-win32.whl](#)

[cx\\_Freeze-5.1.1-cp36-cp36m-win\\_amd64.whl](#)

[cx\\_Freeze-5.1.1-cp37-cp37m-win32.whl](#)

[cx\\_Freeze-5.1.1-cp37-cp37m-win\\_amd64.whl](#)

Перед установкою програми cx\_Freeze, необхідно встановити програму wheel через Термінал, тобто набрати команду `pip install wheel`.

Для початку установки варто скопіювати скачаний файл в корінь того логічного диска, де містяться ваші програми, наприклад D, щоб легко прописати шлях до нього. Далі пройдіть в командний рядок Windows, і наберіть команду `pip install D:\ cx_Freeze-5.1.1-cp37-cp37m-win32.whl` замість імені файлу з розширенням whl потрібно підставити ваш варіант. Ця команда встановить cx\_Freeze на ваш комп'ютер. Тепер в каталозі з програмою яку потрібно скомпілювати створіть Блокнотом текстовий файл "setup.py", з таким змістом

```
from cx_Freeze import setup, Executable
setup (
    name = "Freelance",
```

```
version = "1.0",  
description = "Freelance Parser",  
executables = [Executable ( "01.py")] )
```

Тут замість 01.py потрібно вставити ім'я вашого файлу на Python який ви компілюєте. Перейшовши в командному рядку в каталог з вашим скриптом дайте команду `python.exe setup.py build`.

Після виконання програми з'явиться новий каталог `build`, в якому, серед купи необхідних для роботи бібліотек з'явиться файл `01.exe`.

### 3.7. Обробка помилок у програмі

При розробці програми ми часто зустрічаємося з помилками, що їх робить оператор, який експлуатує програму. Для того, щоб робота програми не переривалася, а заставила оператора ввести правильні дані, у Python винятки (exceptions) можна обробляти за допомогою конструкції `try-except`. Ця конструкція дозволяє вам визначити блок коду, в якому може виникнути виняток, і виконати альтернативний код в разі його виникнення, щоб програма не перервалася.

Ось загальний синтаксис конструкції `try-except`:

```
try:  
    # Блок коду, в якому може виникнути виняток  
    # Цей код буде виконаний  
except ExceptionType:  
    # Блок коду, який буде виконаний, якщо виникає виняток типу ExceptionType  
    # Цей код буде виконаний тільки у випадку виникнення винятку
```

Ви можете використовувати цю конструкцію для обробки будь-якого винятку, що може виникнути під час виконання вашої програми. Наприклад:

```

a=int(input('введіть знаменник дробі'))
try:
    # Спробуйте виконати код, який може викликати помилку
    a=int(input('введіть знаменник дробу, що не дорівнює нулю '))
    x = 1 / a # Помилка: ділення на нуль
    print(x)
except ZeroDivisionError:
    # Обробіть помилку ділення на нуль
    print('ділення на нуль неможливе')

```

введіть знаменник дробі0  
введіть знаменник дробу, що не дорівнює нулю 0  
ділення на нуль неможливе

У цьому прикладі, якщо в коді в блоку try виникне помилка ділення на нуль, програма не буде перервана, а замість цього буде виведено повідомлення про помилку.

Важливо пам'ятати, що обробка винятків повинна бути якомога конкретнішою. Намагайтеся визначити тип винятку, який може виникнути в певному місці коду, і обробляти його окремо.

У конструкції try-ехсерт у Python можна вказати різні типи винятків для обробки. Ось кілька типів обробки помилок, які можна використовувати у блоках ехсерт:

Ехсертіон: Це базовий клас для всіх винятків у Python. Використання цього класу в блоках ехсерт сприймає будь-який тип винятка. Проте використовувати його повинно бути обережно, оскільки він може зловити навіть системні винятки, які зазвичай не потрібно обробляти. Вище наведено таку конструкцію.

Конкретні типи винятків: Ви можете вказати конкретний тип винятка, який ви очікуєте. Це дозволяє точно визначити, які типи помилок будуть оброблені.

```

try:
    # Код, який може викликати виняток
except ValueError:
    # Обробка помилки типу ValueError
except ZeroDivisionError:
    # Обробка помилки типу ZeroDivisionError

```

Можливість обробки кількох типів винятків одночасно:

```
try:
    # Код, який може викликати виняток
except (ValueError, TypeError):
    # Обробка помилок типів ValueError та TypeError
```

Обробка винятків без їх зазначення:

```
try:
    # Код, який може викликати виняток
except:
    # Обробка будь-якого винятка
```

Важливо правильно вибирати типи винятків для обробки, щоб забезпечити коректну обробку помилок і уникнути непередбачуваних результатів.

### Контрольні запитання

1. Які основні правила присвоювання значень змінним у Python?
2. Що таке змінна і як ви можете створити змінну у Python?
3. Як виконати множинне присвоювання значень у Python?
4. Як використовувати значення однієї змінної для присвоювання іншій змінній у Python?
5. Як видалити значення змінної у Python за допомогою оператора присвоювання?
6. Як ви зчитуєте введені дані з клавіатури у Python?
7. Які функції використовуються для зчитування рядків з клавіатури та перетворення їх на числа у Python?
8. Як вивести значення змінної на екран у Python?

9. Як вивести відформатовані рядки з даними, використовуючи оператор виведення у Python?
10. Як використовувати команду print() для виведення кількох значень на екран?
11. Що таке оператор if у Python, і для чого він використовується?
12. Як виглядає синтаксис оператора if у Python?
13. Як виконується перевірка умови в операторі if, і які можливі результати?
14. Як використовувати умовні оператори elif та else в разі, якщо перша умова не виконується?
15. Як ви можете використовувати логічні оператори (and, or, not) в операторах if для складних умов?
16. Що таке оператор for у Python, і для чого він використовується?
17. Як виглядає синтаксис оператора for у Python?
18. Як використовувати оператор for для ітерації (перебору) елементів списку (list) у Python?
19. Як ви можете використовувати оператор for для ітерації через символи рядка (string) у Python?
20. Як використовувати оператор for для виконання певних дій певне число разів?
21. Що таке оператор while у Python, і для чого він використовується?
22. Як виглядає синтаксис оператора while у Python?
23. Як виконується перевірка умови в операторі while, і які можливі результати?
24. Як уникнути безкінечного циклу (лупу) при використанні оператора while у Python?
25. Як використовувати оператор while для повторення певних дій до досягнення певної умови?
26. Що таке бібліотека для розробки GUI у Python і які вона найпоширеніші?
27. Яка різниця між бібліотеками Tkinter, PyQt та PyGTK?
28. Як створити вікно використовуючи Tkinter?



- 29.Що таке віджети і які вони бувають в Tkinter?
- 30.Як додати текст або зображення на віджет Tkinter?
- 31.Як зробити взаємодіючі кнопки в Tkinter?
- 32.Як додати меню до програми Tkinter?
- 33.Як ви організуєте розміщення віджетів у вікні Tkinter?
- 34.Як зберегти та завантажити дані у програмі Tkinter?
- 35.Як зробити графічний інтерфейс для введення даних з допомогою Tkinter?
- 36.Як додати анімацію або переходи між екранами у програмі Tkinter?
- 37.Як перевірити введені дані на коректність у програмі Tkinter?
- 38.Як додати обробку подій до віджетів у Tkinter?
- 39.Як можна покращити вигляд вашого додатку Tkinter?
- 40.Як створити власний віджет у Tkinter?

*Студенти, що добре засвоїли матеріали розділу, навчилися використовувати оператори введення та виведення, умовні оператори, оператори циклу, створювати підпрограми та GUI-додатки.*

# Розділ 4.

## БІБЛІОТЕКИ ПАЙТОН

*У цьому розділі студенти вивчать можливості найбільш популярних бібліотек Пайтон, без яких ця мова не була б на третьому місці по частоті вживання.*

Бібліотеки Пайтон почали зростати по міру набуття популярності цієї мови. В них програмісти вмщали наперед запрограмовані методи та формули, що часто використовуються в різних науках. Для економічних потреб в них теж знайшлося місце. Тому всі розрахунки зі статистики, економетрики, прогнозування, побудови графіків різних типів вже запрограмовані, варто тільки знайти потрібну бібліотеку, звернутися до потрібної функції та використати. Програми стають простішими, швидкість їх розробки збільшується.

У розділі 1 подано порядок пошуку та інсталяції потрібних бібліотек. Але варто пам'ятати, що використання бібліотек ускладняється їх розмірами. Як знайти потрібну функцію, як вона працює? Для цього кожна бібліотека має розширений довідник, що містить не тільки опис функцій, але й приклади їх використання.

Ось список ресурсів, де ви можете знайти описи бібліотек Python:

- Unite.AI – <https://unite.ai/uk/10-%D0%BD%D0%B0%D0%B9%D0%BA%D1%80%D0%B0%D1%89%D0%B8%D1%85-%D0%B1%D1%96%D0%B1%D0%BB%D1%96%D0%BE%D1%82%D0%B5%D0%BA-Python-%D0%B4%D0%BB%D1%8F->

<https://habr.com/ru/companies/edison/articles/474622/> на Хабрі ви знайдете статтю про 34 open source бібліотеки Python.

У цих статтях подано короткий опис бібліотек.

А щоб з подробицями вивчити якусь бібліотеку, досить у пошуковій машині ввести її назву, наприклад NumPy, як вам відкриється сторінка з повним її описом, але англійською мовою.

The screenshot shows the NumPy website. At the top left is the 'NumPy' logo with a 3D cube icon. To its right is the tagline 'The fundamental package for scientific computing with Python' and a button that says 'LATEST RELEASE: NUMPY 1.26. VIEW ALL RELEASES'. Below this is a dark blue banner with the text 'NumPy 1.26.0 released' and the date '2023-09-16'. Underneath the banner is a grid of six feature cards:

- Powerful N-dimensional arrays**: Fast and versatile, the NumPy vectorization, indexing, and broadcasting concepts are the de-facto standards of array computing today.
- Numerical computing tools**: NumPy offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more.
- Open source**: Distributed under a liberal [BSD license](#), NumPy is developed and maintained [publicly on GitHub](#) by a vibrant, responsive, and diverse [community](#).
- Interoperable**: NumPy supports a wide range of hardware and computing platforms, and plays well with distributed, GPU, and sparse array libraries.
- Performant**: The core of NumPy is well-optimized C code. Enjoy the flexibility of Python with the speed of compiled code.
- Easy to use**: NumPy's high level syntax makes it accessible and productive for programmers from any background or experience level.

В сучасних умовах можна цим не перейматися, адже більшість браузерів мають вбудований перекладач.

Ми не дарма звернулися спочатку саме до цієї бібліотеки, бо NumPy об'єднує у собі більшість популярних бібліотек, розроблених раніше не тільки на Пайтоні, але й на Фортрані та Сі. Причому, всі ці бібліотеки абсолютно безплатні. В подальшому викладенні будуть описані тільки бібліотеки вільного доступу.

Якщо бібліотека вже встановлена (інстальована) на вашому комп'ютері, її функції ще не доступні для використання. Доступ до функцій бібліотеки здійснюється тільки після того, як буде написана команда, наприклад для NumPy

```
import numpy as np
```

Зверніть увагу на додаток **as**. Цей додаток дозволяє використовувати бібліотеку зі спрощеним ім'ям. А для чого нам це спрощене ім'я?

Виявляється, що для використання функції з конкретної бібліотеки потрібно спочатку вказати ім'я бібліотеки, поставити крапку, а вже потім писати ім'я конкретної функції.

Ми з таким вже стикалися, коли вивчали вбудовану в Пайтон бібліотеку `math`. Тоді ми писали звертання, наприклад до косинуса як `math.cos()`. Якщо водночас до програми підключено декілька бібліотек, постійне написання довгого імені є незручним, простіше використовувати наперед задане скорочення такого імені, наприклад, звертання до функції `arange` бібліотеки NumPy записується таким чином

```
x = np.arange(15, dtype=np.int64).reshape(3, 5)
```

Оскільки бібліотек багато, у цьому розділі будуть описані ті їх функції, які необхідні економістам, а не конкретні бібліотеки повністю.

Якщо якісь функції, вам потрібні, не будуть описані, ви завжди можете їх знайти самостійно. У пригоді вам стане і пошукові можливості браузерів і штучний інтелект, наприклад ChatGPT або Copilot. Останній є додатком до Windows 11 – практично, «дзеркалом» ChatGPT.

Наприклад

**You:** Яка функція якої бібліотеки знаходить середнє вибірки?

**ChatGPT**

**ChatGPT**

У бібліотеці NumPy, яка є однією з найпопулярніших бібліотек для обробки числових даних у Python, є функція для знаходження середнього

значення вибірки. Ця функція називається `numpy.mean()`. Вона приймає один аргумент - масив даних (вектор або матрицю), і повертає середнє арифметичне значення цієї вибірки.

Ось приклад використання `numpy.mean()` для знаходження середнього значення вибірки:

```
import numpy as np

# Приклад вибірки
data = [1, 2, 3, 4, 5]

# Знаходження середнього значення вибірки
mean_value = np.mean(data)

print("Середнє значення вибірки:", mean_value)
```

Цей код виведе середнє значення вибірки `data`. У даному випадку, якщо ви виконаєте цей код, результат буде:

```
Середнє значення вибірки: 3.0
```

де 3.0 - це середнє арифметичне значення вибірки `[1, 2, 3, 4, 5]`.

У розділі всі наведені програми друкуються в текстовому форматі, для можливості прямого використання.

#### 4.1. Економічні та фінансові розрахунки

А чи існують чисто економічні розрахунки у Пайтон? Насправді, чисто економічних, таких як от розрахунок фінансових коефіцієнтів, чи балансова звітність у Пайтоні не існує. Для цього є спеціалізовані програми. Але всі бібліотеки Пайтону дозволяють робити розрахунки, які необхідні при плануванні бізнесової діяльності, аналізу тенденцій ринку, прогнозуванню та оптимізації. Ось деякі з цих бібліотек:

**NumPy:**

Функції: Має багато вбудованих функцій для математичних розрахунків, таких як обчислення середнього, медіани, стандартного відхилення, коваріації тощо.

Економічні застосування: Може використовуватися для обробки і аналізу економічних даних, в тому числі для побудови фінансових моделей.

### **Pandas:**

Функції: Має потужні інструменти для роботи з даними, включаючи зчитування, запис, фільтрацію, об'єднання, групування і агрегацію даних.

Економічні застосування: Зазвичай використовується для аналізу фінансових даних, побудови таблиць для фінансових звітів та іншого аналізу.

### **SciPy:**

Функції: Має багато модулів для числового аналізу, включаючи модуль `scipy.stats` для статистичного аналізу.

Економічні застосування: Використовується для статистичного аналізу економічних даних, включаючи тести гіпотез, аналіз варіації тощо.

### **StatsModels:**

Функції: Має модуль для статистичного моделювання, включаючи моделі регресії, аналіз часових рядів, а також інструменти для проведення економетричного аналізу.

Економічні застосування: Використовується для побудови різноманітних моделей економічних процесів, прогнозування макроекономічних показників тощо.

### **Matplotlib:**

Функції: Має модуль для створення графіків і діаграм для візуалізації даних, включаючи фінансові графіки, такі як лінійні графіки, стовпчикові графіки, графіки свічок тощо.

Економічні застосування: Використовується для візуалізації фінансових даних, а також для побудови графіків результатів аналізу даних.

### **scikit-learn:**

Функції: Має багато алгоритмів машинного навчання для класифікації, регресії, кластеризації, а також інструменти для оцінювання моделей.

Економічні застосування: Використовується для побудови прогностичних залежностей.

## 4.2. Операції вводу виводу з файлами Excel та виведення на друк

Для роботи з Excel файлами з Python варто використати бібліотеки: xlrd, xlwt для файлів типу \*.xls або openpyxl для файлів типу \*.xlsx.

Для прикладу будемо використовувати готовий файл Excel з якого ми спочатку завантажуюмо дані і з першої клітинки, а потім запишемо їх в другу. Треба також мати на увазі, що всі ці бібліотеки працюють тільки з такими файлами, що мають імена, написані латиницею. Те саме стосується й повного шляху до потрібного файлу Excel – всі теки і підтеки потрібно іменувати теж латиницею, інакше, програма не працюватиме, а видаватиме помилку.

Для початку завантажимо потрібні бібліотеки і відкриємо файл xls на читання і виберемо потрібний лист з даними:

```
import xlrd, xlwt
# Відкриваємо файл
rb = xlrd.open_workbook ( '../ ArticleScripts / ExcelPython / xl.xls',
formatting_info = True)
# Вибираємо активний лист
sheet = rb.sheet_by_index (0)
```

Тепер давайте подивимося, як завантажити значення з потрібних клітинок:

```
# Отримуємо значення першої клітинки A1
val = sheet.row_values (0) [0]
# Клітинки в Excel нумеруються так: номер рядку в круглих дужках,
номер стовпця – у квадратних.
```



```
# Отримуємо список значень з усіх записів
vals = [sheet.row_values (rownum) for rownum in range (sheet.nrows)]
```

Як видно читання даних не складає труднощів. Тепер запишемо їх в інший файл. Для цього створюємо новий Excel файл з новою робочою книгою:

```
wb = xlwt.Workbook ()
ws = wb.add_sheet ( 'Test')
```

Запишемо в новий файл отримані раніше дані і збережемо зміни:

```
# В A1 записуємо значення з комірки A1 з іншого файлу
ws.write (0, 0, val [0])
```

# В стовпець B запишемо нашу послідовність зі стовпця A вихідного файлу

```
i = 0
for rec in vals:
ws.write (i, 1, rec [0])
    i = + i
```

# Зберігаємо робочу книгу

```
wb.save ( '../ ArticleScripts / ExcelPython / xl_rec.xls')
```

З прикладу вище видно, що бібліотека xlrd відповідає за читання даних, а xlwt – за запис, тому немає можливості внесення змін в уже створену книгу без її копіювання в нову. Крім цього зазначені бібліотеки працюють тільки з файлами формату xls (Excel 2003) і у них немає підтримки нового форматуxlsx (версії Excel 2007 і вище).

Щоб успішно працювати з форматом `xlsx`, знадобиться бібліотека `openpyxl`. Для демонстрації її роботи виконаємо дії, які були показані для попередніх бібліотек.

Для початку завантажимо бібліотеку і виберемо потрібну книгу і робочий лист:

```
import openpyxl
wb = openpyxl.load_workbook(filename =
'./ArticleScripts/ExcelPython/openpyxl.xlsx')
sheet = wb ['test']
```

Як видно з вищенаведеного лістингу зробити це не складно. Тепер подивимося як можна завантажити дані:

```
# Зчитуємо значення певної клітинки
val = sheet ['A1']. value

# Зчитуємо заданий діапазон
vals = [v [0].value for v in sheet.range ('A1: A2')]
```

Відмінність від попередніх бібліотек в тому, що `openpyxl` дає можливість звертатися до клітинок і їх груп через їх імена, такі ж, як і в самому Excel що досить зручно і зрозуміло при читанні програми.

Тепер подивимося як нам зробити запис і зберегти дані:

```
# Записуємо значення в певну комірку
sheet ['B1'] = val

# Записуємо послідовність
i = 0
for rec in vals:
    sheet.cell (row = i, column = 2).value = rec
```

```
i = + 1
```

```
# Зберігаємо дані
```

```
wb.save ( '../ ArticleScripts / ExcelPython / openpyxl.xlsx')
```

У випадку, якщо один лист електронної таблиці містить одну таблицю, можна скористатися простішим кодом, наприклад:

```
import xlrd
xls_file = pd.ExcelFile('table_D1_transp.xls')
table = xls_file.parse('Sobol_tr')
print(table)
```

Тут спочатку відкриваються потрібна бібліотека «xlrd», потім відкривається файл «table\_D1\_transp.xls». Далі, в масив «table» вносяться дані з листа, що має ім'я «Sobol\_tr». Але цей прийом діє тільки для електронних таблиць версії «\*.xls».

Ось ще один приклад роботи з файлом Excel:

```
# Завантажте книгу Excel
wb = Workbook("E:\Lib_Py\Cost_of_AU.xl")

# Отримати колекцію робочих аркушів
worksheet = wb.getWorksheets().get(0)

# Надрукувати назву аркуша
print("Worksheet: " + str(worksheet.getName()))

# Отримати кількість рядків і стовпців
rows = worksheet.getCells().getMaxDataRow()
cols = worksxsheet.getCells().getMaxDataColumn()

# Перебирайте ряди
for i in range(rows):

    # Перегляньте кожен стовпець у вибраному рядку
    for j in range(cols):
        # надрукувати значення клітинки
        print(worksheet.getCells().get(i, j).getValue(), end = " | ")

# Розрив рядка друку
print("\n")
```

Щоб вивести дані на принтер, їх треба спершу передати на принтер, використовуючи відповідні функції для роботи з принтером у вашій операційній системі. У Python можна використовувати різні бібліотеки для взаємодії з принтером, залежно від вашої потреби та операційної системи.

Наприклад, у Windows можна використовувати модуль win32print для взаємодії з принтером. Однак, цей модуль не входить до стандартної бібліотеки Python і його треба буде встановити окремо.

Ось приклад використання модуля win32print для виведення даних на принтер у Windows:

```
import win32print
import win32ui
import win32con

def print_text(text):
    printer_name = win32print.GetDefaultPrinter()
    hprinter = win32print.OpenPrinter(printer_name)
    try:
        hdc = win32ui.CreateDC()
        hdc.CreatePrinterDC(printer_name)
        hdc.StartDoc('Python Document')
        hdc.StartPage()
        hdc.TextOut(100, 100, text)
        hdc.EndPage()
        hdc.EndDoc()
    finally:
        win32print.ClosePrinter(hprinter)

# Викликаємо функцію для друку тексту
print_text("Це текст, який буде надрукований на принтері.")
```

Будь ласка, зауважте, що перед використанням цього коду вам потрібно буде встановити модуль win32print. Будь ласка, зауважте, що перед використанням цього коду вам потрібно буде встановити модуль ruwin32, який містить модуль win32print., який містить модуль win32print.

### 4.3. Зчитування даних зі сторінок і Інтернеті

Для Python розроблено різні бібліотеки, які дозволяють здійснювати отримання даних з Інтернету. Однією з найпопулярніших бібліотек для цього є requests. Інші варіанти включають urllib, BeautifulSoup, Scrapy та інші.

Отримання даних з веб-сайту може бути виконано за допомогою HTTP-запитів. Ось простий приклад використання бібліотеки requests для отримання HTML-сторінки з веб-сайту.

```
1 import requests
2
3 # Здійснюємо GET-запит на веб-сайт
4 response = requests.get('https://www.example.com')
5
6 # Виводимо статус-код відповіді (200 - успішно)
7 print(response.status_code)
8
9 # Виводимо HTML-код сторінки
10 print(response.text)
```

Зверніть увагу, що для взаємодії з веб-сайтом, вам може бути потрібно дослідити його структуру та протоколи, які він використовує. Також, слід враховувати політику веб-сайту та робити запити відповідно до їхніх правил.

Python може розрізняти тексти та таблиці на сторінці HTML за допомогою бібліотеки BeautifulSoup, яка дозволяє парсити HTML-код і витягувати різні елементи, такі як тексти, таблиці, посилання і т.д.

Ось приклад того, як ви можете використовувати BeautifulSoup, щоб отримати тексти та таблиці зі сторінки HTML:

```
from bs4 import BeautifulSoup
import requests

# Здійснюємо GET-запит на веб-сайт
response = requests.get('https://www.example.com')

# Парсимо HTML-код з використанням BeautifulSoup
soup = BeautifulSoup(response.text, 'html.parser')

# Отримуємо всі тексти зі сторінки
texts = soup.find_all(text=True)

# Виводимо всі тексти
for text in texts:
```

```
print(text)

# Отримуємо всі таблиці зі сторінки
tables = soup.find_all('table')

# Виводимо всі таблиці
for table in tables:
    print(table)
```

У цьому прикладі ми використовуємо requests для здійснення GET-запиту на веб-сайт, а потім використовуємо BeautifulSoup для парсингу HTML-коду сторінки. Ми витягуємо всі тексти за допомогою soup.find\_all(text=True) та виводимо їх. Потім ми витягуємо всі таблиці за допомогою soup.find\_all('table') та виводимо їх.

Парсинг – це процес аналізу структурованих даних для екстракції корисної інформації. У контексті веб-розробки парсинг відноситься до аналізу HTML-коду веб-сторінок для видобування певних даних, таких як тексти, посилання, таблиці, зображення та інші елементи.

Зазвичай для парсингу веб-сторінок використовуються спеціальні бібліотеки, такі як BeautifulSoup, lxml, Scrapy та інші. Ці бібліотеки дозволяють здійснювати різноманітні операції парсингу, такі як знаходження елементів за тегами, класами або ідентифікаторами, отримання атрибутів елементів, вилучення тексту, знаходження посилань і багато іншого.

Парсинг використовується для різних цілей, таких як створення пошукових систем, збір даних для аналізу, створення агрегаторів новин, моніторинг цін на товари в Інтернеті та багато іншого.

Якщо при виконанні попередньої програми виникає повідомлення про помилку, можна використати таку зміну.

```
from bs4 import BeautifulSoup
import requests

# Здійснюємо GET-запит на веб-сайт
response = requests.get('https://www.example.com')

# Парсимо HTML-код з використанням BeautifulSoup
soup = BeautifulSoup(response.text, 'html.parser')

# Отримуємо всі тексти зі сторінки
texts = soup.find_all(string=True)
```

```
# Виводимо всі тексти
for text in texts:
    print(text)
```

Спробуємо допомогою Пайтона зчитати зі сторінки НБУ таблицю з інформацією про інфляцію і перенести її у двомірний масив, а потім побудувати графіки залежності інфляції від місяців та річної інфляції від номера року.

Ось ця адреса <https://index.minfin.com.ua/ua/economy/index/inflation/>

Щоб зчитати таблицю з веб-сторінки та перетворити її на двовимірний масив, можна використати бібліотеки pandas для зчитування даних та matplotlib для побудови графіків. Ось як це можна зробити:

```
import pandas as pd
import matplotlib.pyplot as plt

# Зчитуємо дані з таблиці
url = 'https://index.minfin.com.ua/ua/economy/index/inflation/'
tables = pd.read_html(url)

# Вибираємо потрібну таблицю
table = tables[1] # Вибір другої таблиці на сторінці (індексація з нуля)

# Відображаємо перші кілька рядків таблиці
print(table.head())

# Побудова графіків залежності інфляції від місяців та річної інфляції від номера року
plt.figure(figsize=(10, 5))

# Графік залежності інфляції від місяців
plt.subplot(1, 2, 1)
plt.plot(table['Місяць'], table['Інфляція'], marker='o', linestyle='-')
plt.title('Інфляція за місяцями')
plt.xlabel('Місяць')
plt.ylabel('Інфляція')

# Графік річної інфляції від номера року
plt.subplot(1, 2, 2)
plt.plot(table['Рік'], table['Інфляція'], marker='o', linestyle='-')
plt.title('Річна інфляція')
plt.xlabel('Рік')
plt.ylabel('Інфляція')

plt.tight_layout() # Розташування графіків
plt.show()
```

У цьому коді ми використовуємо pd.read\_html(url) для зчитування всіх таблиць з веб-сторінки, а потім обираємо потрібну таблицю (tables[1]). Потім ми



будуємо два графіки: один показує залежність інфляції від місяців, а інший - річну інфляцію від номера року. Ось робоча програма:

```
import requests
import pandas as pd
import matplotlib.pyplot as plt
from bs4 import BeautifulSoup

# Виконуємо GET-запит на веб-сторінку
url = 'https://index.minfin.com.ua/ua/economy/index/inflation/'
response = requests.get(url)

# Перевіряємо, чи запит успішний
if response.status_code == 200:
    # Парсимо HTML-код сторінки
    soup = BeautifulSoup(response.text, 'html.parser')

    # Знаходимо всі таблиці на сторінці
    tables = soup.find_all('table')

    # Шукаємо таблицю з інфляцією (індекс споживчих цін)
    target_table = None
    for table in tables:
        if 'споживчих цін' in str(table): # Шукаємо таблицю, яка містить текст "Інфляція"
            target_table = table
            break

    # Якщо знайшли потрібну таблицю
    if target_table:
        # Зчитуємо таблицю за допомогою pandas
        df = pd.read_html(str(target_table))[0] # Зчитуємо першу таблицю зі списку
        print(df) # Відображаємо таблицю

        # Побудова графіків залежності інфляції за місяць січень та річної інфляції від номера року
        plt.figure(figsize=(10, 5))

        # Графік залежності інфляції за місяць січень від року
        plt.subplot(1, 2, 1)
        plt.plot(df.iloc[:, 0], df.iloc[:, 1], marker='o', linestyle='-')
        plt.title('Інфляція за січень по роках')
        plt.xlabel('Роки')
        plt.ylabel('січень')

        # Графік річної інфляції від номера року
        plt.subplot(1, 2, 2)
        plt.plot(df.iloc[:, 0], df.iloc[:, 12], marker='o', linestyle='-')
        plt.title('Річна інфляція')
        plt.xlabel('Роки')
        plt.ylabel('Інфляція')

        plt.tight_layout() # Розташування графіків
        plt.show()

    else:
        print("Таблиця з інфляцією не знайдена на сторінці.")
else:
    print("Помилка при отриманні вмісту сторінки:", response.status_code)
```

Функція `plt.subplot()` використовується для створення сітки графіків, в якій кожен графік розташовується у власному рядку та стовпці. Аргументи `plt.subplot()` вказують розмір сітки та позицію конкретного графіка в цій сітці.

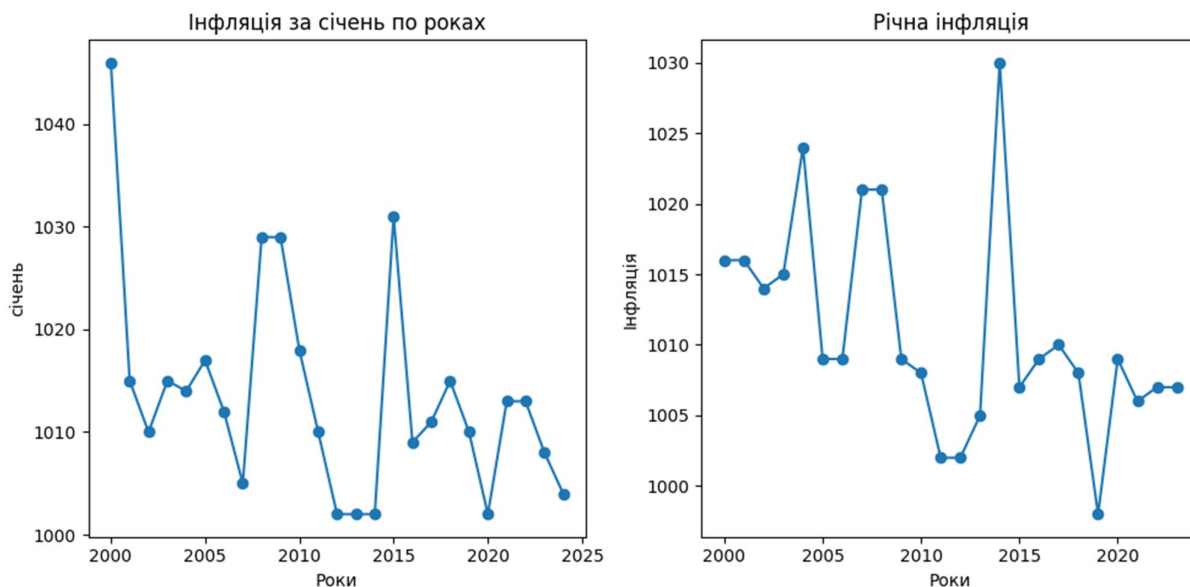
Отже, перший аргумент 1 вказує на кількість рядків у сітці графіків. У тут лише один рядок; другий аргумент 2 вказує на кількість стовпців у сітці графіків. Тут два стовпці; третій аргумент 1 вказує на позицію поточного графіка у сітці, рахуючи зліва направо та зверху вниз, тобто у першому випадку це перший графік у першому рядку.

Отже, `plt.subplot(1, 2, 1)` створює сітку графіків з одним рядком та двома стовпцями і встановлює поточний графік у першому стовпці цього рядка.

Дані з таблиці можна вибирати за назвою колонок, але треба бути уважним, тому, що Пайтон чітко розрізняє регістр напису і слово «Січень» відрізняє від слова «січень». Тому при виборі даних за назвами колонок, варто уважно дотримуватись того регістру напису, який використовується на сторінці.

Щоб не плутатися з назвами колонок, ми оператором `plt.plot(df.iloc[:, 0], df.iloc[:, 1], marker='o', linestyle='-')` вибираємо перший рядок таблиці як аргумент для осі X (номер року), а другий рядок, як аргумент для осі Y (інфляція за січень).

	Unnamed: 0	січень	лютий	березень	...	жовтень	листопад	грудень	За рік
0	2000	1046	1033	1020.0	...	1014.0	1004.0	1016.0	1258
1	2001	1015	1006	1006.0	...	1002.0	1005.0	1016.0	1061
2	2002	1010	986	993.0	...	1007.0	1007.0	1014.0	994
3	2003	1015	1011	1011.0	...	1013.0	1019.0	1015.0	1082
4	2004	1014	1004	1004.0	...	1022.0	1016.0	1024.0	1123
5	2005	1017	1010	1016.0	...	1009.0	1012.0	1009.0	1103
6	2006	1012	1018	997.0	...	1026.0	1018.0	1009.0	1116
7	2007	1005	1006	1002.0	...	1029.0	1022.0	1021.0	1166
21	2021	1013	1010	1017.0	...	1009.0	1008.0	1006.0	1100
22	2022	1013	1016	1045.0	...	1025.0	1007.0	1007.0	1266
23	2023	1008	1007	1015.0	...	1008.0	1005.0	1007.0	1051
24	2024	1004	1003	NaN	...	NaN	NaN	NaN	1007



Як видно з таблиць, надрукованих програмою, частина таблиці не заповнена, але для побудови графіків засобами matplotlib не це стало на заваді, і вони вийшли як треба.

Інший прийом зчитування даних з сайту пропонує така конструкція програми, в якій всі дані з необхідної таблиці одразу копіюються у файл \*.xls, а вже потім всі інші розрахунки ведуться, вибираючи дані з цього файлу.

```
import requests
import xlrd
import numpy as np
from statistics import mean
import matplotlib.pyplot as plt

#ссылка на таблицю на сайті
url1 = 'https://ukrstat.gov.ua/operativ/operativ2020/zf/zf_reg/zf_reg19_eu.xls'
#скачати файл за посиланням
def download(url=''):
    try:
        res = requests.get(url=url1)
        with open('tab.xls', 'wb') as file:
            file.write(res.content)
        return "завантажено"
    except Exception as _ex:
        return "напевно ви ввели посилання з похибкою"
```

Багато сайтів показують дані у вигляді HTML-таблиць, зручних для перегляду в браузері, але не пропонують їх в таких форматах, як JSON або XML. Так, наприклад, йде справа з даними про біржових опціонах на сайті Yahoo!

Finance. Опціон – це похідний фінансовий інструмент (дериватив), який дає право купувати (опціон на придбання, або колл-опціон) або продавати (опціон на продаж, або пут-опціон) акції компанії за деякою ціною (ціною виконання) в проміжку часу між поточним моментом і деяким фіксованим моментом в майбутньому (кінцевою датою). Колл- і пут-опціони торгуються з різними цінами виконання і кінцевими датами; ці дані можна знайти в таблицях на сайті Yahoo! Finance.

Для початку вирішіть, з якої URL-адреси ви хочете завантажувати дані, потім відкрийте його за допомогою засобів з бібліотеки urllib2 і розберіть потік, користуючись xml:

```
from lxml.html import parse
from urllib2 import urlopen. .
parsed = parse(urlopen('http://finance.yahoo.com/q/op?s=AAPL+Options'))
doc = parsed.getroot()
```

Маючи цей об'єкт, ми можемо вибрати всі HTML-теги зазначеного типу, наприклад, теги table, всередині яких знаходяться дані, що нас цікавлять. Для прикладу отримаємо список усіх активних посилань в документі, вони представляються в HTML тегом A. Викличемо метод findall кореневого елемента документа, передавши йому вираз XPath (це мова, на якому записуються «запити» до документу):

```
In [906]: links = doc.findall('.// a')
In [907]: links [15: 20] Out [907]:
[<Element a at 0x6c488f0>,
<Element a at 0x6c48950>,
<Element a at 0x6c489b0>,
<Element a at 0x6c48a10>,
<Element a at 0x6c48a70>]
```

Але це об'єкти, що представляють HTML-елементи; щоб отримати URL і текст посилання, нам потрібно скористатися методом get елемента (для отримання URL) або методом text\_content (для отримання тексту):

```
In [908]: lnk = links [28]
In [909]: lnk
Out [909]: <Element a at 0x6c48dd0>
In [910]: lnk.get ( 'href')
Out [910]: 'http://biz.yahoo.com/special.html'
In [911]: lnk.text_content ()
Out [911]: 'Special Editions'
```

Таким чином, отримання усіх активних посилань в документі зводиться до включення за списком:

```
In (912): urls = [lnk.get ( 'href') for lnk in doc.findall ( './ a')]
In (913): urls [-10:]
Out [913]:
[ 'Http://info.yahoo.com/privacy/us/yahoo/finance/details.html',
'Http://info.yahoo.com/relevantads/',
'Http://docs.yahoo.com/info/terms/',
'Http://docs.yahoo.com/info/copyright/copyright.html',
'http://help.yahoo.com/l/us/yahoo/finance/forms\_index.html', 'http:
// help .yahoo.eom / l / us / yahoo / finance / quotes / fitadelay.html
', 'http://help.yahoo.com/l/us/yahoo/finance/quotes/fitadelay.html ',
'Http://www.capitaliq.com',
'Http://-www.csidata.com',
'Http: //w^.morningstar.com/']
```

Що стосується відшукування потрібних таблиць в документі, то це робиться методом проб і помилок; на деяких сайтах рішення цього завдання спрощується, тому що таблиця має атрибут id. Я знайшов, які таблиці містять дані про колл- і пут-опціони:

```
tables = doc.findall ( './ table') calls = tables [9] puts = tables [13]
У кожній таблиці є рядок-заголовок, а за нею йдуть рядки з даними:
In [915]: rows = calls.findall ( './ tr')
```

Для всіх рядків, включаючи заголовок, ми хочемо отримати текст з кожного осередку; в разі заголовка осередками є елементи TR, а для рядків даних – елементи TD:

```
def _unpack (row, kind = 'td'):
```

```
elts = row.findall ( './% s% kind) return [val.text_content () for val
in elts]
```

Таким чином, отримуємо:

```
In [917]: _unpack (rows [0], kind = 'th')
Out [917]: [ 'Strike', 'Syi ^ iol', 'Last', 'Chg', 'Bid', 'Ask', 'Vol',
'OpenInt'
In [918]: _unpack (rows [1], kind = 'td')
Out [918]:
[ '295.00',
'AAPL12 0818C002 95 000',
'310.40',
'0.00',
'289.80',
'290.80',
'1',
'169']
```

Тепер для перетворення даних в об'єкт DataFrame залишилося об'єднати всі описані кроки разом. Оскільки числові дані як і раніше записані у вигляді рядків, можливо, буде потрібно перетворити деякі, але не всі стовпці в формат з плаваючою точкою. Це можна зробити і вручну, але, на щастя, в бібліотеці pandas є клас TextParser, який використовується для здійснення read\_csv і іншими функціями розбору для автоматичного перетворення типів:

```
from pandas.io.parsers import TextParser
def parse_options_data (table): rows = table.findall ( './ tr') header
= _unpack (rows [0], kind = 'th') data = [_unpack (r) for r in rows
[1:]] return TextParser (data, names = header) .get_chunk ()
```

Нарешті, викликаємо цю функцію розбору для табличних об'єктів lxml і отримуємо результат у вигляді DataFrame:

```
In [920]: call_data = parse_options_data (calls)
In [921]: put_data = parse_options_data (puts)
In [922]: call_data [: 10]
Out[922]:
Strike Symbol Last Chg Bid Ask Vol Open Int
0 295 AAPL120818C00295000 310.40 o.o 289.80 290.80 1 169
1 300 AAPL120818C00300000 277.10 1.7 284.80 285.60 2 478
2 305 AAPL120818C00305000 300.97 o.o 279.80 280.80 10 316
3 310 AAPL120818C00310000 267.05 o.o 274.80 275.65 6 239
4 315 AAPL120818C00315000 296.54 o.o 269.80 270.80 22 88
```

5 320 AAPL120818C00320000 291.63 o.o 264.80 265.80 96 173  
 6 325 AAPL120818C00325000 261.34 o.o 259.80 260.80 N/A 108  
 7 330 AAPL120818C00330000 230.25 o.o 254.80 255.80 N/A 21  
 8 335 AAPL120818C00335000 266.03 o.o 249.80 250.65 4 46  
**9 340 AAPL120818C00340000 272.58 o.o 244.80 245.80 4 30**

XML (розширювана мова розмітки) – ще один популярний формат представлення структурованих даних, що підтримує ієрархічно вкладені дані, забезпечені метаданими.

Продемонструємо альтернативний інтерфейс, зручний для роботи з XML-даними, – lxml .objectify.

Управління міського транспорту Нью-Йорка (MTA) публікує тимчасові ряди з даними про роботу автобусів і електричок (<http://j/www.mta.info/developers/download.html>). Ми зараз розглянемо дані про якість обслуговування, зберігається у вигляді XML-файлів. Для кожної автобусної та залізничної компанії існує свій файл (наприклад, Performance\_MNR. Xml для компанії MetroNorth Railroad), що містить дані за один місяць в вигляді послідовності таких XML-повідомлень

```
<INDICATOR>
<INDICATOR_SEQ> 3 73 889 </ INDICATOR_SEQ>
<PARENT_SEQ> </ PARENT_SEQ>
<AGENCY_NAME> Metro-North Railroad </ AGENCY_NAME>
<INDICATOR_NAME> Escalator Availability </
INDICATOR_NAME> <DESCRIPTION> Percent of the time that
escalators are operational systemwide. The availability rate is based
on physical observations performed the morning of regular business
days only. This is a new indicator the agency began reporting in
2009. </ DESCRIPTION>
<PERIOD_YEAR> 2011 </ PERIOD_YEAR>
<PERIOD_MONTH> 12 </ PERIOD_MONTH>
<CATEGORY> Service Indicators </ CATEGORY>
<FREQUENCY> M </ FREQUENCY>
<DESIRED_CHANGE> U </ DESIRED_CHANGE>
<INDICATOR_UNIT>% </ INDICATOR_UNIT>
<DECIMAL_PLACES> 1 </ DECIMAL_PLACES>
<YTD_TARGET> 97.00 </ YTD_TARGET>
<YTD_ACTUAL> </ YTD_ACTUAL>
<MONTHLY_TARGET> 97.00 </ MONTHLY_TARGET>
```



```
<MONTHLY_ACTUAL> </ MONTHLY_ACTUAL>
</ INDICATOR>
```

Використовуючи lxml. obj ecti fy, ми розбираємо файл і отримуємо посилання на кореневий вузол XML-документа від методу getroot:

```
from lxml import objectify
path = 'Performance_MNR.xml'
parsed = objectify.parse (open (path))
root = parsed.getroot ()
```

Властивість root. INDICATOR повертає генератор, послідовно віддає всі елементи <INDICATOR>. Для кожного запису ми заповнюємо словник імен тегів (наприклад, YTD\_ACTUAL) значеннями даних (деякі теги пропускаються):

```
data = []
skip_fields = [ 'PARENT_SEQ', 'INDICATOR_SEQ',
'DESIRED_CHANGE', 'DECIMAL_PLACES' ]
for elt in root.INDICATOR:
    el_data = { }
    for child in elt.getchildren ():
        if child.tag in skip_fields:
            continue
        el_data [child.tag] = child.pyval
    data.append (el_data)
```

Нарешті, перетворимо цей список словників в об'єкт DataFrame:

```
In [927]: perf = DataFrame (data)
In [928]: perf
Out [928]:
Empty DataFrame
Columns: array ([], dtype = int64)
Index: array ([], dtype = int64)
```

XML-документи можуть бути набагато складніше, ніж в цьому прикладі. Зокрема, в кожному елементі можуть бути метадані. Розглянемо тег гіперпосилання в форматі HTML, який є окремим випадком XML:

```
from StringIO import StringIO
tag = '<a href = "http: // ^www.google.com"> Google </a>'
root = objectify.parse (StringIO (tag)). getroot ()
```

Тепер ми можемо звернутися до будь-якого атрибуту тега (наприклад, href) або до тексту посилання:

```
In [930]: root
```

```
Out [930]: <Element a at 0x88bd4b0>
In [931]: root.get ( 'href')
Out [931]: 'http://^www.google.com'
In [932]: root.text
Out [932]: 'Google'
```

У pandas є також підтримка для читання табличних даних в форматі Excel 2003 (і більш пізніх версією) за допомогою класу `ExcelFile`. На внутрішньому рівні `ExcelFile` користується пакетами `xlrd` і `openpyxl`, тому їх потрібно попередньо встановити. Для роботи з `ExcelFile` створіть його екземпляр, передавши конструктору шлях до файлу з розширенням `xls` або `xlsx`:

```
xls_file = pd.ExcelFile ( 'data.xls')
Прочитати дані з робочого листа в об'єкт DataFrame дозволяє
метод parse:
table = xls_file.parse ( 'Sheet1')
Взаємодія з HTML і Web AP /
```

Багато сайтів надають відкритий API для отримання даних в форматі JSON або якомусь іншому. Отримати доступ до таких API з Python можна різними способами; рекомендується простий пакет `requests` (<http://docs.python-requests.org>). Для пошуку за словами «python pandas» в Твіттері ми можемо відправити такий HTTP-запит GET:

```
In [944]: import requests
In [945]: url = 'http://search.twitter.com/search.json?q=python%20pandas'
In [946]: resp = requests.get (url)
In [947]: resp
Out [947]: <Response [200]>
```

У об'єкта `Response` має атрибут `text`, в якому зберігається вміст відповіді на запит GET. Багато API в веб повертають JSON-рядок, яку слід завантажити в об'єкт Python:

```
In [948]: import json
In [949]: data = json.loads (resp.text)
In [950]: data.keys ()
Out [950]:
```

```
[U'next_page ', u'completed_in', u'max_id_str ', u'since_id_str',  
u'refresh_url ', u'results', u 'since_id', u'results_per_page ', u'query',  
u'max_id ', u'page ']
```

Поле відповіді results містить список твітів, кожен з яких представлений таким словником Python:

```
{U'created_at ': u'Mon, 25 Jun 2012 17:50:33 +0000', u'from_user  
' : u'wesmckinn', u'from_user_id ': 115494880, u'from_user_id_str':  
u'115494880 ', u'from_user_name ': u'Wes McKinney •, u'geo':  
None,  
u'id ': 217313849177686018,  
u'id_str ': u'217313849177686018',  
u'iso_language_code ': u'pt',  
u'metadata ': {u'result_type': u'recent '},  
u'source ': u' <a href="http://twitter.com/"> web </a> ',  
u'text ': u'Lunchtime pandas-fu http://t.co/SI70xZZQ #pydata',  
u'to_user ': None,  
u'to_user_id ': 0,  
u'to_user_id_str ': u'O',  
u'to_user_name ': None}
```

Далі ми можемо побудувати список полів твіту, що нас цікавлять та передати його конструктору DataFrame:

```
In [951]: tweet_fields = [ 'created_at', 'from_user', 'id', 'text']  
In [952]: tweets = DataFrame (data [ 'results'], columns =  
tweet_fields)  
In [953]: tweets Out [953]:  
<Class 'pandas.core.frame.DataFrame'>  
Int64Index: 15 entries, Про to 14 Data columns:  
created_at 15 non-null values from_user 15 non-null values  
id 15 non-null values  
text 15 non-null values  
dtypes: int64 (1), object (3)
```

Тепер в кожному рядку DataFrame знаходяться дані, витягнуті з одного твіту:

```
Out [121]: created_at from_user id  
In [121]: tweets.ix [7]  
Thu, 23 Jul 2012 9:54:00 +0000  
deblike
```

## 4.4. Графіка

### 4.4.1. TURTLE

Бібліотека turtle – це розширення мови Пітон, що дозволяє малювати на екрані нескладні малюнки.

Для цього потрібно підключити бібліотеку turtle. Програму, яка використовує цей графічний модуль треба починати командою

```
import turtle turtle.reset ()
```

А для того, щоб затримати графічне вікно на екрані, необхідно закінчувати всі програми, які використовують модуль turtle командою:

```
turtle._root.mainloop ()
```

У Пайтоні, як і в інших мовах програмування, необхідно використовувати коментарі. Вони теж позначаються знаком #, коментарі можна писати не тільки в окремому рядку, а й в тому ж рядку, правіше команди.

Пронумерувати рядки можна клавішею F11. Команди для малювання пишуться англійською мовою. Ось необхідний набір команд.

<b>forward (a)</b>	<b>уперед на a кроків</b>
<b>backward (a)</b>	<b>назад на a кроків</b>
<b>left(β)</b>	<b>ліворуч на β градусів</b>
<b>right (β)</b>	<b>праворуч на β градусів</b>

<b>circle (r)</b>	Намалювати окружність радіусом <b>r</b> , центр ліворуч - <b>r&gt;0</b> , праворуч - <b>r&lt;0</b>
<b>circle(r,β)</b>	Намалювати дугу радіуса <b>r</b> та градусної міри <b>β</b>
<b>goto (x,y)</b>	"перейти" в точку з координатами <b>(x,y)</b>
<b>down()</b>	перо опусти
<b>up()</b>	перо підніми
<b>width(a)</b>	нова ширина пера
<b>color(s)</b>	колір пера, "red" - червоний, "green" - зелений, "blue" - синій, "white" - білий, "black" - чорний, "yellow" - жовтий, "pink" - рожевий
<b>fill(f)</b>	Зафарбовує замкнуту область, перед роботою дайте команду <i>turtle.fill(1)</i> , а як закінчите розфарбування – <i>turtle.fill(0)</i> .
<b>reset()</b>	Відміна налаштувань
<b>clear()</b>	очистка екрану
<b>write(s)</b>	вивід тексту (=напиши)

На малюнку нижче показано приклад застосування цих команд.

```

#!/usr/bin/python
#-*- coding: utf-8 -*-
import turtle           # Підключаємо модуль turtle
turtle.reset()         # Приводим черепашку в початкове
                        # положення
turtle.down()          # Опускаємо перо (початок
                        # рисования)
turtle.forward(20)     # Проползти 20 пікселів вперед
turtle.left(90)        # Поворот вліво на 90 градусів
turtle.forward(20)     # Рисуємо другу сторону квадрата
turtle.left(90)        # Рисуємо третью сторону квадрата
turtle.forward(20)    # Рисуємо четверту сторону квадрата
turtle.up()            # Підняти перо (закінчити рисовать)
turtle.forward(100)    # Відвести черепашку від рисунка в
                        # сторону
turtle._root.mainloop() # Задержати вікно на екрані

```

Строка: 1 Столбец: 1 ВСТАВКА ОБЫЧНЫЙ turt\_1

```

cd '/home/shinjitsu/Documents/Python'
thon 'u/Documents/Py
[shinjitsu@localhost Python]$ █

```

Треба відзначити, що на екрані ми побачимо квадрат.

Малювання простих, заповнених усередині, форм можна також робити такими прийомами. Для прикладу, намалюємо прапор України та стилізований червоний тризуб по центру.

```
# Створення вікна та налаштування
wn = turtle.Screen()
wn.setup(width=600, height=400)
wn.bgcolor("white")

# Створення черепашки
t = turtle.Turtle()
t.speed(0)
t.up()
t.goto(-300, 200)
t.down()
t.width(3)

# Малює синій прямокутник угорі
t.color("blue")
t.begin_fill()
for _ in range(2):
    t.forward(600)
    t.right(90)
    t.forward(150)
    t.right(90)
t.end_fill()

# Малює жовтий прямокутник нижче
t.up()
t.goto(-300, 50)
t.down()
t.color("yellow")
t.begin_fill()
for _ in range(2):
    t.forward(600)
    t.right(90)
    t.forward(150)
    t.right(90)
t.end_fill()

# Малює червоний стилізований тризуб у центрі
# Перший зуб
t.up()
t.goto(-50, 85)
t.down()
t.color("red")
t.begin_fill()
for _ in range(2):
    t.forward(20)
    t.right(90)
    t.forward(80)
    t.right(90)
t.end_fill()

# Другий зуб
```

```

t.up()
t.goto(-10, 85)
t.down()
t.color("red")
t.begin_fill()
for _ in range(2):
    t.forward(20)
    t.right(90)
    t.forward(80)
    t.right(90)
t.end_fill()

# Третій зуб
t.up()
t.goto(25, 85)
t.down()
t.color("red")
t.begin_fill()
for _ in range(2):
    t.forward(20)
    t.right(90)
    t.forward(80)
    t.right(90)
t.end_fill()

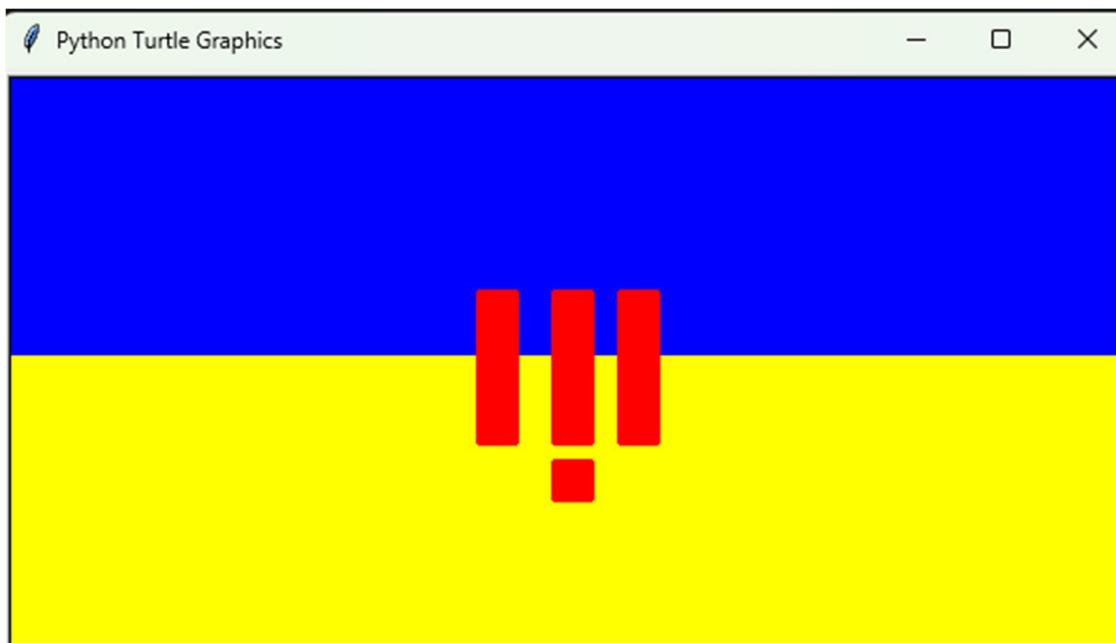
# Півострів Крим унизу
t.up()
t.goto(-10, -5)
t.down()
t.color("red")
t.begin_fill()
for _ in range(2):
    t.forward(20)
    t.right(90)
    t.forward(20)
    t.right(90)
t.end_fill()

# Малювання трикутника тимчасово припинено
#t.up()
#t.goto(0, 100)
#t.down()
#t.color("red")
#t.begin_fill()
#t.goto(150, -50)
#t.goto(-150, -50)
#t.goto(0, 100)
#t.end_fill()

# Приховати черепашку та вікно
t.hideturtle()
wn.mainloop()

```





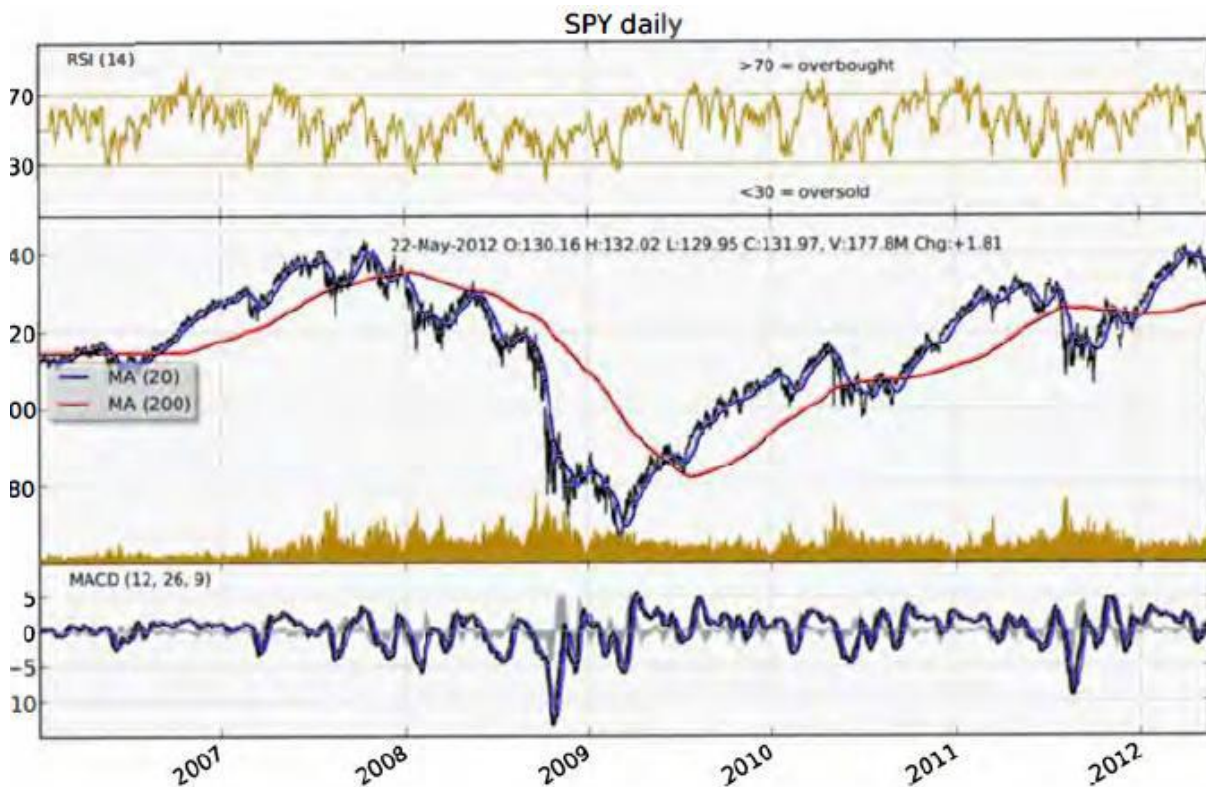
У програмі спочатку визначається розмір полотна, а потім по черзі зображають синій та жовтий трикутники, що утворюють прапор. Потім малюються 4 прямокутники різних розмірів.

Нагадуємо, що координати початку малюнку відраховуються від центра і вимірюються у пікселях.

#### 4.4.2. MATPLOTLIB

Побудова графіків, а також статична або інтерактивна візуалізація – одне з найважливіших завдань аналізу даних. Вони можуть бути частиною процесу дослідження, наприклад, застосовуватися для виявлення викидів, ухвали необхідних перетворень даних або пошуку ідей для побудови моделей.

Matplotlib – це пакет для побудови графіків (головним чином, двовимірних) поліграфічної якості. При використанні в поєднанні з якою-небудь бібліотекою ГП (наприклад, всередині IPython), matplotlib набуває інтерактивні можливості: панорамування, масштабування і інші. Цей пакет підтримує різноманітні системи ГП у всіх операційних системах, а також вміє експортувати графічні дані у всіх векторних і растрових форматах: PDF, SVG, JPG, PNG, BMP, GIF і т. д.



Для matplotlib є цілий ряд додаткових бібліотек, наприклад mplot3d для побудови тривимірних графіків і basemap для побудови карт і проєкцій. Для опрацювання наведених в цьому пункті прикладів коду, потрібно завантажити IPython в режимі ruylab (ipython --ruylab) або включити інтеграцію з циклом обробки подій ГП за допомогою функції `gui`.

Взаємодіяти з matplotlib можна декількома способами. Самий поширеним способом – запустити IPython в режимі ruylab за допомогою команди `ipython --ruylab`. В результаті IPython конфігується для підтримки обраної системи ГП (Tk, wxPython, PyQt, платформний ГП OS X, GTK). Для більшості користувачів мається на увазі за замовчуванням системи ГП. У режимі ruylab в IPython також імпортується багато модулів і функцій, щоб інтерфейс був більше схожий на MATLAB. Переконайтеся, що все працює, можна, побудувавши простий графік:

```
plot(np.arange(10))
```

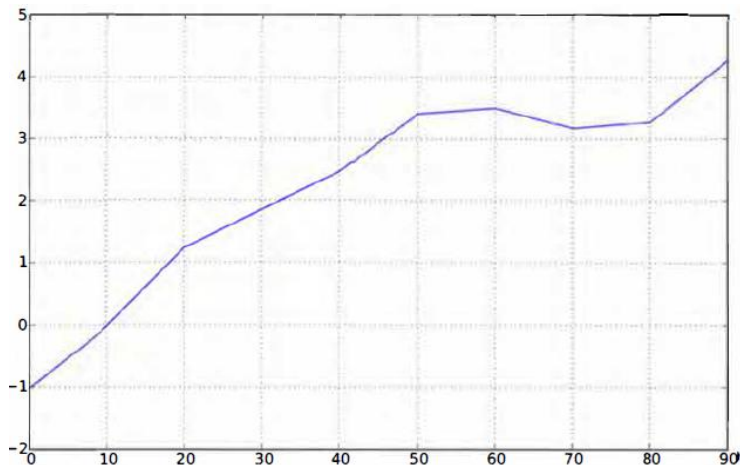
Якщо все налаштовано правильно, то з'явиться нове вікно з лінійним графіком. Його можна закрити мишею або ввівши команду `close ()`. Всі функції `matplotlib API`, зокрема `plot` і `close`, знаходяться в модулі `matplotlib.pyplot`, при імпорті якого зазвичай дотримуються наступної угоди:

```
import matplotlib.pyplot as plt
```

Як ви могли переконатися, `matplotlib` – бібліотека досить низького рівня. Графік в ній складається з базових компонентів: спосіб відображення даних (тип графіка: лінійний графік, стовбчаста діаграма, коробчаста діаграма, діаграма розсіювання, контурний графік і т. д.), Пояснювальний напис, назва, мітки рисок і інші анотації. Почасти так зроблено тому, що в багатьох випадках дані, необхідні для побудови повного графіка, розкидані по різних об'єктах. У бібліотеці `pandas` у нас вже є мітки рядків, мітки колонок і, можливо, інформація про угруповання. Це означає, що багато графіків, для побудови яких засобами `matplotlib` довелося б писати багато коду, в `pandas` можуть бути побудовані за допомогою одного-двох коротких команд. Тому в `pandas` є багато високорівневих методів побудови графіків для стандартних типів візуалізації, в яких використовується інформація про внутрішньої організації об'єктів `DataFrame`.

У об'єктів `Series` і `DataFrame` є метод `plot`, який вміє будувати графіки різних типів. За замовчуванням він будує лінійні графіки:

```
In [55]: s = Series
(np.random.randn
(10) .cumsum (),
index = np.arange
(0, 100, 10)) In
[56]: s.plot ()
```



Індекс об'єкту Series передається matplotlib для нанесення рисок на вісь X, але це можна відключити, задавши параметр `use_index = False`. Ризики і діапазон значень на осі X можна налаштувати за допомогою параметрів `xticks` і `xlim`, а на осі Y – за допомогою параметрів `yticks` і `ylim`. Повний перелік параметрів методу `plot` наведено нижче:

`label` – Мітка для пояснювальної написи на графіку.

`ax` – Об'єкт підграфіка matplotlib, всередині якого будувати графік. Якщо параметр не заданий, то використовується активний підграфік.

`Style` – Рядок стилю, наприклад `'ko--'`, що передається у matplotlib.

`alpha` – Рівень непрозорості графіка (число від 0 до 1).

`kind` – Може приймати значення `'line'`, `'bar'` `barh` ; «kde».

`logy` – Використовувати логарифмічний масштаб по осі Y.

`use_index` – Брати мітки рисок з індексу об'єкта.

`rot` – Кут повороту міток рисок (від 0 до 360).

`xticks` – Значення рисок на осі X.

`yticks` – Значення рисок на осі Y.

`xlim` – Межі по осі X (наприклад, `[0, 10]`)

`ylim` – Межі по осі Y.

`grid` – Відображати координатну сітку (за замовчуванням включено).

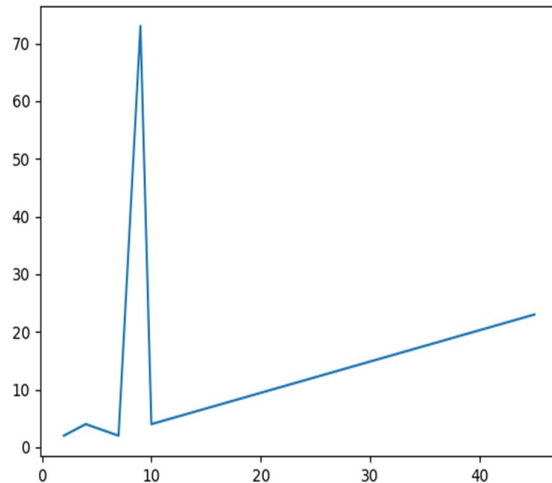
За допомогою matplotlib ви можете створити різні види графіків у Python. Найпростіший графік - це лінійний графік. Приклад, побудови:

```

import matplotlib.pyplot as plt

x=[2,4,7,9,10,45]
y=[2,4,2,73,4,23]
plt.plot(x,y)
plt.show()

```



## 4.5. Статистичний аналіз

### 4.5.1. Описова статистика. Кореляція

Бібліотека NumPy містить численні функції для обчислення описової статистики. Ось перелік деяких з найбільш популярних функцій:

`numpy.mean()`: Обчислює середнє значення елементів масиву.

`numpy.median()`: Знаходить медіану елементів масиву.

`numpy.var()`: Розраховує дисперсію (середньоквадратичне відхилення) елементів масиву.

`numpy.std()`: Обчислює стандартне відхилення елементів масиву.

`numpy.min()`: Знаходить мінімальне значення в масиві.

`numpy.max()`: Знаходить максимальне значення в масиві.

`numpy.percentile()`: Обчислює перцентиль (квантиль) для заданого відсоткового значення.

`numpy.histogram()`: Будує гістограму даних, розподілених за певними інтервалами.

`numpy.corrcoef()`: Розраховує кореляцію між двома масивами даних.

`numpy.cov()`: Обчислює коваріацію між двома масивами даних.

`numpy.unique()`: Повертає унікальні значення з масиву.

`numpy.percentile()`: Знаходить квантиль для заданого відсоткового значення.

`numpy.nanmean()`: Обчислює середнє значення масиву, ігноруючи значення NaN.

`numpy.nanmedian()`: Знаходить медіану масиву, ігноруючи значення NaN.

`numpy.nanstd()`: Обчислює стандартне відхилення масиву, ігноруючи значення NaN.

`numpy.nanvar()`: Обчислює дисперсію масиву, ігноруючи значення NaN.

Ось приклад використання можливостей NumPy

```
import numpy as np

a = [37.5, 65.25, 40.5, 61.5, 53.25, 40.5, 54.0, 48.0]
b = [136.3, 71.05, 121.8, 76.85, 107.3, 124.7, 71.05, 123.25]

print('Параметри вибірок   a   b')
print('Середнє : ', np.mean(a), ' ', np.mean(b))
print('Сума : ', np.sum(a), ' ', np.sum(b))
print('Мах : ', np.max(a), ' ', np.max(b))
print('Мін : ', np.min(a), ' ', np.min(b))
print('Кількість : ', len(a), ' ', len(b))
print('Медіана : ', np.median(a), ' ', np.median(b))
print('Стандартне відхилення : ', np.std(a), ' ', np.std(b))
print('Дисперсія : ', np.var(a), ' ', np.var(b))
print('Кореляція між двома масивами даних', np.corrcoef(a, b))
print('Коваріація між двома масивами даних', np.cov(a, b))

b = input('Побачили? Натисніть Enter')
```

Останній оператор дозволяє виконувати програми не вмикаючи IDLE. Тоді вся робота програми з'явиться у Терміналі і буде демонструватися доти, поки не натиснеш Enter/

А ось результат роботи програми

```

--
Параметри вибірок      a      b
Середнє : 50.0625      104.0375
Сума : 400.5      832.3
Мах : 65.25      136.3
Мін : 37.5      71.05
Кількість : 8      8
Медіана : 50.625      114.55
Стандартне відхилення : 9.566175764118073      25.195705859332463
Дисперсія : 91.51171875      634.8235937500002
Кореляція між двома масивами даних [[ 1.      -0.90436419]
[-0.90436419  1.      ]]
Коваріація між двома масивами даних [[ 104.58482143 -249.11517857
]
[-249.11517857  725.51267857]]
Побачили? Натисніть Enter

```

#### 4.5.2. Закони розподілу. Статистичні критерії

У бібліотеці SciPy, яка є широко використовуваною бібліотекою для наукових обчислень у Python, реалізовані різноманітні статистичні розподіли, включаючи ті, які вас цікавлять. Ось декілька з них:

##### **Нормальний розподіл (нормальний розподіл):**

Імплементований як `scipy.stats.norm`.

Має параметри: середнє значення (`loc`) та стандартне відхилення (`scale`).

Може бути використаний для генерації випадкових величин, обчислення функцій щільності, функцій розподілу та іншого.

##### **Хі-квадрат (розподіл хі-квадрат):**

Імплементований як `scipy.stats.chi2`.

Має параметр `df`, що відповідає ступеням свободи.

Використовується для роботи з тестами незалежності, а також для побудови довірчих інтервалів та іншого.

##### **Розподіл Стьюдента (розподіл Стьюдента-t):**

Імплементований як `scipy.stats.t`.

Має параметр `df`, що відповідає ступеням свободи.

Зазвичай використовується для обробки вибірових середніх.



## Ось приклад застосування функцій

```
1 from scipy.stats import norm, chi2, t
2
3 # Нормальний розподіл з середнім значенням 0 і стандартним відхиленням 1
4 normal_dist = norm(loc=0, scale=1)
5
6 # Xi-квадрат розподіл з 5 ступенями свободи
7 chi2_dist = chi2(df=5)
8
9 # Розподіл Стьюдента з 10 ступенями свободи
10 t_dist = t(df=10)
11
```

У Python для розподілу Фішера (F-розподілу) можна скористатися бібліотекою SciPy. Функція для роботи з розподілом Фішера називається `f`, і вона доступна в `scipy.stats`. Ось приклад використання:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import f

# Параметри розподілу Фішера
dfn = 5 # ступені свободи для чисельника
dfd = 10 # ступені свободи для знаменника

# Генеруємо випадкові дані з розподілу Фішера
data = f.rvs(dfn, dfd, size=1000)

# Побудова гистограми випадкових значень
plt.hist(data, bins=30, density=True, alpha=0.6, color='g')

# Побудова функції щільності ймовірності
x = np.linspace(f.ppf(0.01, dfn, dfd), f.ppf(0.99, dfn, dfd), 100)
plt.plot(x, f.pdf(x, dfn, dfd), 'r-', lw=2, label='f pdf')

plt.title('Розподіл Фішера')
plt.xlabel('Значення')
plt.ylabel('Щільність ймовірності')
plt.legend()
plt.grid(True)
plt.show()
```

У цьому прикладі ми спочатку задаємо параметри розподілу Фішера (`dfn` - ступені свободи для чисельника, `dfd` - ступені свободи для знаменника). Потім ми генеруємо випадкові дані з цього розподілу за допомогою `f.rvs()`. Після цього ми побудуємо гистограму цих випадкових значень та функцію щільності ймовірності розподілу Фішера за допомогою `f.pdf()`.

Критерій Пірсона (також відомий як тест  $\chi^2$ -квадрат) використовується для оцінки того, наскільки добре емпіричний розподіл даних відповідає теоретичному розподілу. Цей тест може бути застосований для перевірки гіпотези про те, що деяка вибірка дійсно походить з тієї ж самої генеральної сукупності, що і інші вибірки.

Ось приклад програми Python, яка демонструє використання критерію Пірсона для порівняння двох вибірок та оцінки їх подібності.

```
import numpy as np
from scipy.stats import chi2_contingency

# Приклад двох вибірок (можна ввести реальні дані)
sample1 = [37.5,65.25,40.5,61.5,53.25,40.5,54.0,48.0]
sample2 = [36.3,71.05,121.8,76.85,107.3,124.7,71.05,123.25]

# Проведення тесту  $\chi^2$ -квадрат для порівняння двох вибірок
stat, p_value, _, _ = chi2_contingency([sample1, sample2])

# Виведення результатів тесту
print("Статистика тесту  $\chi^2$ -квадрат:", stat)
print("p-значення:", p_value)

# Інтерпретація результатів
alpha = 0.05 # Задамо рівень значущості
if p_value > alpha:
    print("Не вдається відхилити нульову гіпотезу: середні вибірок близькі")
else:
    print("Вдається відхилити нульову гіпотезу: середні вибірок різні")
```

```
Статистика тесту  $\chi^2$ -квадрат: 45.95446786688513
p-значення: 8.922136802531447e-08
Вдається відхилити нульову гіпотезу: середні вибірок різні
```

У цьому прикладі ми використовуємо `scipy.stats.chi2_contingency` для обчислення тесту  $\chi^2$ -квадрат. Результати тесту (статистика та p-значення) дозволяють нам прийняти чи відхилити нульову гіпотезу про рівність середніх значень вибірок. Якщо p-значення менше за заданий рівень значущості (зазвичай 0.05), ми відхиляємо нульову гіпотезу і припускаємо, що середні значення вибірок різні.

Для порівняння дисперсій вибірок можна використовувати тест Лівеня (тест рівності дисперсій). Цей тест дозволяє визначити, чи можна припустити, що дисперсії двох або більше груп даних рівні.

Ось приклад програми Python, яка демонструє використання тесту Лівеня для порівняння дисперсій двох вибірок та оцінки їх подібності:

```
from scipy.stats import levene

# Приклад двох вибірок (можна ввести реальні дані)
sample1 = [37.5,65.25,40.5,61.5,53.25,40.5,54.0,48.0]
sample2 = [36.3,71.05,121.8,76.85,107.3,124.7,71.05,123.258]

# Проведення тесту Лівеня для порівняння дисперсій двох вибірок
stat, p_value = levene(sample1, sample2)

# Виведення результатів тесту
print("Статистика тесту Лівеня:", stat)
print("p-значення:", p_value)

# Інтерпретація результатів
alpha = 0.05 # рівень значущості
if p_value > alpha:
    print("Не вдається відхилити нульову гіпотезу: дисперсії вибірок близькі")
else:
    print("Вдається відхилити нульову гіпотезу: дисперсії вибірок різні")
```

```
Статистика тесту Лівеня: 14.944359346189376
p-значення: 0.0017133730600639688
Вдається відхилити нульову гіпотезу: дисперсії вибірок різні
```

У цьому прикладі ми використовуємо `scipy.stats.levene` для обчислення тесту Лівеня. Результати тесту (статистика та p-значення) дозволяють нам прийняти чи відхилити нульову гіпотезу про рівність дисперсій вибірок. Якщо p-значення менше за заданий рівень.

## 4.6. Регресійний аналіз. Економетрика

Нехай у нас є три екзогенні параметри, ми шукаємо множинну лінійну регресію. Нижче наведено приклад коду Python для розрахунку коефіцієнтів регресії та визначення значення R-квадрат. Для цього ми можемо використати бібліотеку `statsmodels`.

Щоб визначити t-статистику значимості кожного з коефіцієнтів регресії, можна скористатися результатами, отриманими від моделі множинної лінійної

регресії. Зазвичай ці результати містяться у властивостях моделі після її підгонки до даних.

```
1 import numpy as np
2 import statsmodels.api as sm
3
4 # Згенеруємо випадкові дані
5 np.random.seed(0)
6 X = np.random.randn(100, 3) # Три екзогенні параметри
7 y = 2 * X[:,0] + 3 * X[:,1] - 5 * X[:,2] +
8     np.random.randn(100) # Лінійна комбінація з шумом
9
10 # Додамо стовпець з одиницями до матриці X для регресії
11 X = sm.add_constant(X)
12
13 # Побудова моделі множинної лінійної регресії
14 model = sm.OLS(y, X).fit()
15
16 # Виведення результатів
17 print("Коефіцієнти регресії:", model.params)
18 print("R-квадрат:", model.rsquared)
19 print("t-статистика значимості кожного коефіцієнта:",
20       model.tvalues)
```

```
Коефіцієнти регресії: [-0.18204639  1.9579429   2.9494035  -4.9585557 ]
R-квадрат: 0.9749868770668852
t-статистика значимості кожного коефіцієнта: [ -1.88616728  21.0580908   30.5429
9794 -50.85607635]
```

Результат розрахунку означає, що ми отримали економіко-математичну модель виду  $Y = 2.94 - 0.18X_1 + 1.95X_2 - 4.95X_3$ , тут навмисно опущена повна мантиса коефіцієнтів. Перший коефіцієнт – це вільний член регресії, наступні – коефіцієнти при екзогенних змінних у порядку їх розташування у таблиці.

У цьому прикладі ми: Згенерували випадкові дані для трьох екзогенних змінних  $X$  та відповідну залежну змінну  $Y$ ; Додали стовпець з одиницями до матриці  $X$  для регресії (це обов'язковий атрибут); Побудували модель множинної лінійної регресії за допомогою `statsmodels.OLS`; Вивели коефіцієнти регресії та значення R-квадрат.

Також у цьому прикладі `t_stats` містить t-статистику значимості для кожного коефіцієнта регресії. Ці значення можна використовувати для визначення значущості кожного коефіцієнта: якщо абсолютне значення t-статистики велике, то це свідчить про те, що коефіцієнт є значущим. Зазвичай для визначення значущості використовують критичне значення t-статистики для

заданого рівня значущості (зазвичай 0.05). Якщо абсолютне значення t-статистики більше за критичне значення, то коефіцієнт є значущим.

Бібліотека statsmodels також надає інші статистичні показники та діагностику моделі, яку можна використовувати для аналізу регресійних моделей.

Зверніть увагу, що звертання до елементів масиву NumPy відрізняється від звертання до елементів списку, як це прийнято у синтаксисі Пайтона.

Для того щоб мати можливість звертатися до елементів масиву NumPy по рядках та стовпцях, вам потрібно переконатися, що X – це масив NumPy, а не звичайний Python список.

Ось приклад, як можна перетворити список на масив NumPy:

```
1 import numpy as np
2
3 # Ваш список
4 a = [[34, 54, 56], [34, 23, 56], [98, 45, 89]]
5
6 # Перетворення списку на масив NumPy
7 a = np.array(a)
8
9 # Тепер ви можете звертатися до елементів по рядках та
10 # стовпцях
11 print(a[2, 0]) # Виведе елемент у третьому рядку та
12 # першому стовпці (індекси починаються з 0)
```

Результат – 98. Це третє число першої колонки таблиці.

Якщо ми використаємо вкладений список, тоді звернення до елементів може виглядати так:

```
a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
print(a[2][0]) # Виведе елемент у третьому рядку та першому стовпці
(індекси починаються з 0), тобто 3.
```

В обох випадках ми отримаємо значення елемента у вказаному рядку та стовпці. Перший підхід з масивом NumPy більш ефективний, особливо якщо вам потрібно працювати з великими об'ємами даних.

Але економетричні моделі бувають і нелінійні.



Нехай у нас є масив з трьох екзогенних змінних, але ми хочемо додати до них нелінійні ефекти цих змінних, щоб отримати нелінійну модель. Наприклад  $X_1^2$ ,  $1/X_2$ ,  $X_3/(X_1+X_2)$ , тощо.

```
import numpy as np
import statsmodels.api as sm
# Створимо масив даних, розташовуючи дані по рядках таблиці
# яка складається з трьох стовпців, які подають значення
# змінних X1, X2, X3, та семи рядків, які визначають нові
# значення змінних
y=[20.47,16.98,15.12,13.95,19.30,19.77,12.79]
x=[[65.00,84.48,603.00],[51.00,93.72,603.00],[58.00,104.28,636.50],[54.00,97.68,623.10],
[52.00,93.72,489.10],[82.00,81.84,469.00],[81.00,88.44,482.40]]
# Перетворимо списки у масиви NumPy
y=np.array(y)
x=np.array(x)
# Додамо нелінійні перетворення X
# Створюємо 4-й стовпець з квадратом від X1
x=np.insert(x,3,x[:,0]**2,axis=1)
# Створюємо 5-й стовпець зі зворотнім значенням X2
x=np.insert(x,4,1/x[:,1],axis=1)
# Створюємо 6-й стовпець з перетворенням усіх трьох X
x=np.insert(x,5,x[:,2]/(x[:,0]+x[:,1]),axis=1)
print(x)
# Додамо стовпець з одиницями до матриці X для регресії
x = sm.add_constant(x)
# Побудова моделі множинної лінійної регресії
model = sm.OLS(y, x).fit()
# Виведення результатів
print("Коефіцієнти регресії:", model.params)
print("R-квадрат:", model.rsquared)
print("t-статистика значимості кожного коефіцієнта:", model.tvalues)
```

А ось і результат розрахунку:

```
[ [6.50000000e+01 8.44800000e+01 6.03000000e+02 4.22500000e+03
1.18371212e-02 4.03398448e+00]
[5.10000000e+01 9.37200000e+01 6.03000000e+02 2.60100000e+03
1.06700811e-02 4.16666667e+00]
[5.80000000e+01 1.04280000e+02 6.36500000e+02 3.36400000e+03
9.58956655e-03 3.92223318e+00]
[5.40000000e+01 9.76800000e+01 6.23100000e+02 2.91600000e+03
1.02375102e-02 4.10799051e+00]
[5.20000000e+01 9.37200000e+01 4.89100000e+02 2.70400000e+03
1.06700811e-02 3.35643700e+00]
[8.20000000e+01 8.18400000e+01 4.69000000e+02 6.72400000e+03
1.22189638e-02 2.86254883e+00]
[8.10000000e+01 8.84400000e+01 4.82400000e+02 6.56100000e+03
1.13071009e-02 2.84702550e+00]]
Коефіцієнти регресії: [-1.53755421e+03 4.89923388e+00 8.3585
1355e+00 -6.55590508e-01
-2.53193030e-02 5.61038120e+04 9.21258598e+01]
R-квадрат: 1.0
t-статистика значимості кожного коефіцієнта: [-0. 0. 0. -0.
-0. 0. 0.]
```

Додавання стовпців ведеться оператором виду

**`x=np.insert(x,3,x[:,0]**2,axis=1)`**

в якому  $x$  – це масив, звідкіля беруться і куди вкладаються нові дані, 3 – номер нового стовпця,  $x[:,0]$  – означає, що весь стовпець номер 0 треба звести у квадрат,  $axis=1$  – обов’язковий атрибут. Параметр  $axis$  визначає, вздовж якої осі відбувається певна операція. В цьому випадку  $axis=1$  означає, що вставка має відбуватися вздовж осі стовпців.

Тоді, отримана формула має вигляд

$$y = -153 + 4,9X_1 + 8,35X_2 - 0,65X_3 - 0,025X_1^2 + \frac{5610}{X_2} + \frac{92X_3}{X_1 + X_2}$$

Параметр  $R^2$  доводить, що якість апроксимації ідеальна.

#### **4.7. Прогнозування часових рядів**

Визначення можливих майбутніх значень економічних показників, записаних послідовно у різні моменти часу, є важливим елементом роботи економістів. Попри велику кількість різних методів, найбільшу популярність зажив метод Бокса-Дженкінса, що враховує не тільки попередні значення показника, але і значення поточного середнього.

Метод Бокса-Дженкінса, також відомий як метод авторегресії зі змінними ковзними середніми (ARIMA), реалізований у бібліотеці `statsmodels` у Python.

У `statsmodels` для моделювання ARIMA імплементований клас ARIMA, який можна використовувати для побудови ARIMA моделі та прогнозування майбутніх значень часового ряду. Важливим є визначення параметрів  $p$  та  $q$ , які впливають на якість прогнозування.



Рішення, які значення задати для  $p$ ,  $d$  і  $q$ , є не простим і вимагає експерименту з різними моделями. Проте, є наступні практичні рекомендації, узагальнюючі обидва графіка:

1. Задати параметр  $p = 1$ , якщо АКФ експоненціально убуває, а ЧАКФ різко виділяє значення на лагу 1, немає кореляції на інших лагах.

2. Задати параметр  $p = 2$ , якщо АКФ має форму синусоїди або експоненціально убуває. ЧАКФ повинна різко виділятися на значеннях лагів 1 і 2.

3. Задати параметр  $q = 1$ , якщо АКФ різко виділяє значення на лагу 1, немає кореляції на інших лагах. ЧАКФ експоненціально убуває.

4. Задати параметр  $q = 2$ , якщо АКФ різко виділяється на лагу 1 і 2, немає кореляції на інших. ЧАКФ має форму синусоїди або експоненціально убуває.

5. Поставити  $p = 1$  і  $q = 1$ , якщо АКФ експоненціально убуває з лага 1, ЧАКФ експоненціально убуває з лага 1.

6. Значення  $d$  обирається згідно таких міркувань: скільки попередніх значень різниць між попереднім та наступним значення параметра потрібно взяти.

Ці міркування говорять про те, що розрахунок прогнозу варто виконати декілька разів з різними значеннями  $p$ ,  $d$  і  $q$ .

Тут АКФ означає автокореляційну функцію, а ЧАКФ – часткову автокореляційну функцію.

У Python для розрахунку та візуалізації автокореляційної функції (ACF) і часткової автокореляційної функції (PACF) можна використовувати бібліотеку `statsmodels`.

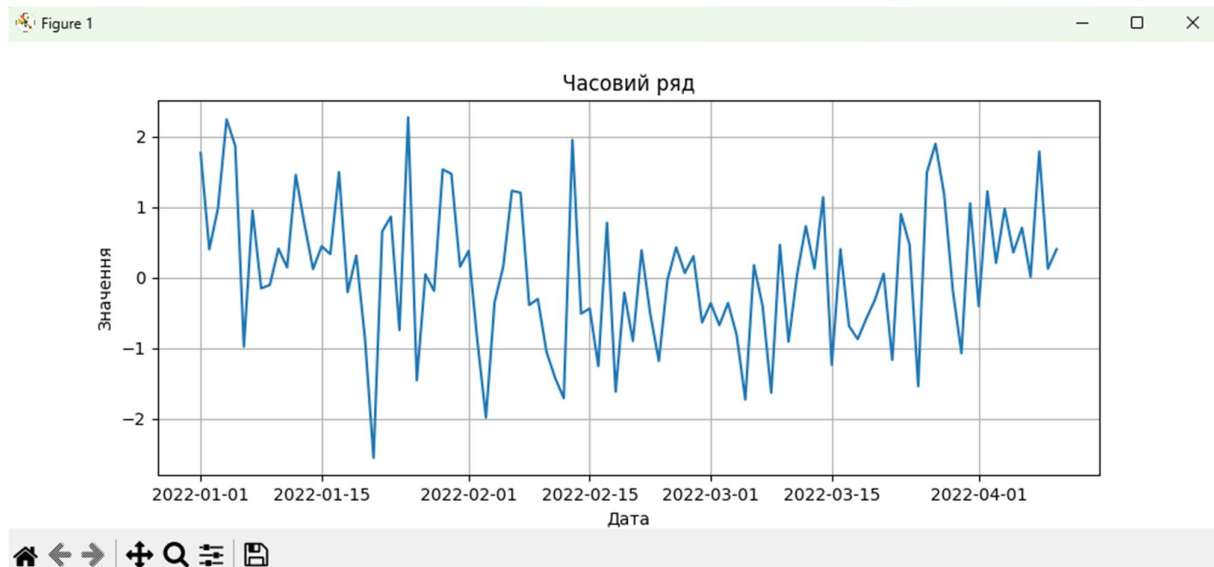
У цьому прикладі ми спочатку генеруємо випадковий часовий ряд, побудований на 100 випадкових значеннях. Потім ми виводимо цей часовий ряд на графік. Після цього ми розраховуємо та виводимо на графіках автокореляційну функцію (ACF) та часткову автокореляційну функцію (PACF) з використанням функцій `plot_acf()` та `plot_pacf()` відповідно.

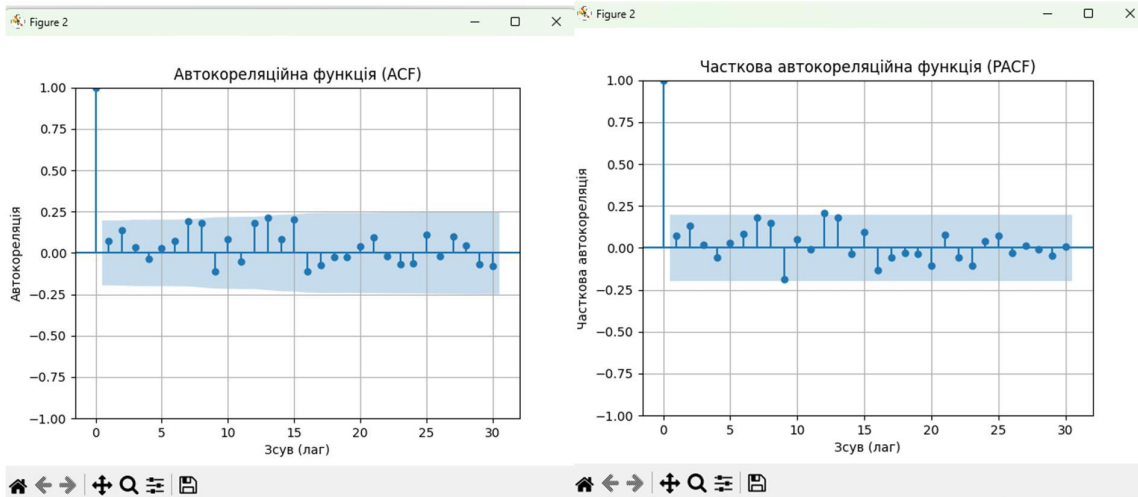
```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
# Згенеруємо випадковий часовий ряд
np.random.seed(0)
data = np.random.randn(100)
time_index = pd.date_range('2022-01-01', periods=100)
# Виведення часового ряду на графіку
plt.figure(figsize=(10, 4))
plt.plot(time_index, data)
plt.title('Часовий ряд')
plt.xlabel('Дата')
plt.ylabel('Значення')
plt.grid(True)
plt.show()
# Розрахунок та виведення автокореляційної функції
plt.figure(figsize=(10, 4))
plot_acf(data, lags=30)
plt.title('Автокореляційна функція (ACF)')
plt.xlabel('Зсув (лаг)')
plt.ylabel('Автокореляція')
plt.grid(True)
plt.show()
# Розрахунок та виведення часткової автокореляційної функції
plt.figure(figsize=(10, 4))
plot_pacf(data, lags=30)
plt.title('Часткова автокореляційна функція (PACF)')
plt.xlabel('Зсув (лаг)')
plt.ylabel('Часткова автокореляція')
plt.grid(True)
plt.show()

```

Далі показані графіки, що їх видає програма. З них видно, що для генератора випадкових чисел, розподілених рівномірно, дані не дають характерних типів графіків, про які сказано вище.





Ось приклад використання ARIMA для моделювання та прогнозування часового ряду для  $p = 1$ ,  $d=1$  і  $q = 1$ .

У цьому прикладі ми використовуємо бібліотеку statsmodels для моделювання ARIMA за допомогою класу ARIMA. Потім ми викликаємо метод forecast() для прогнозування майбутніх значень, а також для розрахунку довірчого інтервалу з вказаною імовірністю  $\alpha=0.05$ . Але спочатку отримуємо часовий ряд, потім побудуємо ARIMA модель з параметрами (1, 1, 1) та прогнозуємо 3 майбутніх значення для довірчої ймовірності 0,95, що відповідає параметру  $\alpha=0.05$ .

```
import numpy as np
from statsmodels.tsa.arima.model import ARIMA
```

```
# Заданий масив
```

```
x = [72, 68, 63, 74, 66, 51, 86, 61, 72, 78, 77, 91, 91, 87, 95, 48, 61, 82]
```

```
# Побудова ARIMA моделі
```

```
model = ARIMA(x, order=(1, 1, 1)) # Параметри p, d, q
results = model.fit()
```

```
# Прогнозування майбутніх значень
```

```
forecast = results.forecast(steps=3) # Прогноз на 3 кроки вперед
```

```
# Розрахунок довірчого інтервалу
```

```
conf_int = results.get_forecast(steps=3).conf_int(alpha=0.05) # Довірчий інтервал з alpha=0.05
```

```
# Виведення прогнозу та довірчого інтервалу
```

```
print("Прогноз:", forecast)
print("Довірчий інтервал:", conf_int)
```

```
Прогноз: [75.24386394 74.10085369 73.907478 ]
Довірчий інтервал: [[ 47.76263188 102.72509599]
 [ 45.90884591 102.29286146]
 [ 45.60905164 102.20590437]]
```

#### 4.8. Оптимізація

Знайдення найкращого значення економіко-математичної моделі, змінюючи економічні параметри, що в неї входять, називається оптимізацією.

Найбільш повно розробленими і вживаними на практиці моделями, що дозволяють оптимізувати управлінські рішення, є статистичні моделі. Вони дають можливість робити вибір сукупності чисел  $x_i$  (змінних в рівняннях), що забезпечують екстремум деякої функції  $Z$  (цільова функція або показник якості ухваленого рішення) при обмеженнях, визначуваних умовами роботи системи. Математично модель записується так: обчислити вектор, що обертає в максимум або мінімум  $\bar{X} = x_1, x_2, x_3, \dots, x_j, \dots, x_n$ , цільову функцію

$$z = f(\bar{X}) = f(x_1, x_2, x_3, \dots, x_j, \dots, x_n) \quad (1)$$

за умов

$$\left. \begin{aligned} q_i(x_1, x_2, x_3, \dots, x_j, \dots, x_n) &= 0 \quad (i = 1, 2, 3, \dots, m_1); \\ q_i(x_1, x_2, x_3, \dots, x_j, \dots, x_n) &\leq 0 \quad (i = m_1 + 1, m_1 + 2, \dots, m); \\ x_j &\geq 0 \quad (j = 1, 2, 3, \dots, n). \end{aligned} \right\} \quad (2)$$

Вирази типу (2) називаються обмеженнями, які задають область рішень цільової функції (1). Саме цільова функція, почасти, і є моделлю соціально-економічної системи. Варто відмітити, що вид функцій  $f$  та  $q_i$  може бути як лінійний так і нелінійний. Для знайдення оптимуму моделей цього типу застосовуються числові методи.

Загалом, ці методи вже запрограмовані в різних бібліотеках, опис яких є на сайті <https://wiki.python.org/moin/PythonForOperationsResearch>.

Дамо описи деяких бібліотек:

- АРМ Python – це безкоштовна програма оптимізації через веб-сервіс. Задача нелінійного програмування надсилається на сервер АРМонитор і результати повертаються до локального скрипту Python. Автоматичне завантаження веб-інтерфейсу допомагає візуалізувати рішення, зокрема проблеми динамічної оптимізації, що включають диференціальні та алгебраїчні рівняння. Типовими розв'язками є АРОРТ, ВРОРТ та ІРОРТ. Попередньо налаштовані режими включають оптимізацію, оцінку параметрів, динамічне моделювання та нелінійний контроль.

- Соорг - Програмний продукт Соорг інтегрує різні пакети, пов'язані з оптимізацією Python.

- СVOХРТ – це безкоштовний пакет програм для опуклої оптимізації на основі мови програмування Python. Він може бути використаний з інтерактивним інтерпретатором Python, у командному рядку шляхом виконання сценаріїв Python або інтегрований в інше програмне забезпечення через модулі розширення Python. Його основна мета полягає в тому, щоб зробити розробку програмного забезпечення для опуклих додатків оптимізації простими, будуючи велику стандартну бібліотеку Python та сильні сторони Python як мову програмування високого рівня.

- OpenOpt (ліцензія: BSD) містить з'єднання з десятками розв'язувачів та має деякі власні Python-написані, наприклад нелінійний розв'язувач з визначеною точністю: `interalg`, графічний вихід збіжності та деяка більш чисельна оптимізація "ПОВИННІ БУТИ". Також OpenOpt може вирішити проблеми з функціями `FuncDesigner` з автоматичною диференціацією, які зазвичай працюють швидше і дають більш точні результати, ніж апроксимація похідних від звичайних відмінностей.

- PuLP – це оптимізація методом лінійного програмування, написаний на python. PuLP може генерувати файли MPS або LP і викликати GLPK, COIN CLP / CBC, CPLEX та GUROBI для вирішення лінійних проблем.

- Puomo – Пакет оптимізації Python для моделювання об'єктів (Puomo) – це інструмент з відкритим кодом для моделювання додатків для оптимізації в Python. Puomo може бути використаний для визначення символічних проблем, створення конкретних випадків проблеми та вирішення цих випадків з використанням стандартних розв'язувачів. Puomo надає можливість, яка зазвичай пов'язана з мовами алгебраїчного моделювання, такими як AMPL, AIMMS та GAMS, але об'єкти моделювання Puomo вбудовані в повнофункціональну мову програмування високого рівня з насиченим набором допоміжних бібліотек. Puomo використовує можливість бібліотеки програмного забезпечення Coorg, яка інтегрує пакети Python для визначення оптимізаторів, моделювання додатків для оптимізації та керування обчислювальними експериментами.

- `scipy.optimize` – деякі розв'язувачі, написані або пов'язані розробниками SciPy.

- `pyOpt` – це пакет для формулювання та вирішення нелінійних обмежених оптимізаційних завдань у ефективній, багаторазовій і портативній формі (ліцензія: LGPL).

Зверніть увагу на те, що слова «ліцензія» в опису, означають, що використання цих пакетів можливо тільки з дозволу конкретної фірми. А отже, може бути платним.

Для прикладу розберемо детальніше використання функції `solve` з пакету `puLP`, який є безплатним. Для цього вирішимо наступну оптимізаційну задачу.

Витрати на рекламу в місяць не повинні перевищувати 10 000 грошових одиниць (ГО). Хвилина радіореклами коштує 5 ГО, а телереклами 90 ГО Фірма має намір використовувати радіорекламу в три рази частіше ніж телерекламу. Практика показує, що 1 хвилина телереклами забезпечує обсяг продажів в 30 разів більший ніж 1 хвилина радіореклами. Знайти оптимальні витрати на кожен вид реклами.

Позначимо як  $x_1$  – витрати на телерекламу,  $x_2$  – витрати на радіорекламу.

Тоді, цільова функція, що максимізує ефективність від реклами прийме

вигляд  $30x_1 + x_2 \rightarrow \max$

а обмеження:

– за обсягом витрат:  $90x_1 + 5x_2 \leq 10\,000$

– за часом реклами:  $x_2 = 3x_1$

– за типом даних  $x_1, x_2 \geq 0$ .

На наступному тексті покажемо роботу функції solve. Коментарі в програмі пояснюють особливості її використання.

```
from pulp import *
import time
start = time.time() # Визначаємо час на початок оптимального розрахунку
# Вмикаємо програму знайдення максимально можливого значення pulp.
problem = LpProblem('Максимізація_прибутку', LpMaximize)
# Встановлюємо відповідність змінних у програмі їх умовним позначенням
# в нашому випадку вони однакові, але можна було написати в лапках
# замість x1 - витрати на радіорекламу
# замість x2 - витрати на телерекламу
# параметр LowBound встановлює нижню межу значень змінних
x1 = pulp.LpVariable("x1", lowBound=0)
x2 = pulp.LpVariable("x2", lowBound=0)
# Цільова функція
problem += 30*x1 + x2, '-->max' #"Цільова функція*"
# Текст в лапках після формули та коми обов'язковий

# Обмеження. Цифра в лапках після формули та коми показують номер
# обмеження
problem += 90*x1 + 5*x2 <= 10000, '1' #"Перше обмеження"
```



```

problem += x2 == 3*x1, '2' #"Друге обмеження"
# запускаємо процес вирішення оптимальної лінійної задачі
problem.solve ()
print ("Результат оптимального розрахунку: ")
# Друкуємо оптимальні значення змінних
for variable in problem.variables():
    print (variable.name, "=", variable.varValue)

print ("Прибуток:")
print (value(problem.objective))
stop = time.time ()
print ("Час на вирішення оптимізаційної задачі:")

# Друкуємо різницю в часі між початком розрахунку та його закінченням
print (stop - start)

```

Результат розрахунку показано нижче

```

RESTART: D:\02 Навчальний процес\00 Еле:
оптиміз2.py
Результат оптимального розрахунку:
x1 = 95.238095
x2 = 285.71429
Прибуток:
3142.85714
Час на вирішення оптимізаційної задачі:
0.09360027313232422

```

Розглянемо тепер транспортну задачу, представлену в наступній таблиці. Як видно з неї, це задача відкритого типу з перевищенням запитів на товар.

		Споживачі			
		1	2	3	
Запаси\ Запити на постачання		74	40	36	
Постачал ьники	1	20	7	3	6
	2	45	4	8	2
	3	30	1	5	9

Введемо умовні позначення для обсягів поставки від першого постачальника до 3-х споживачів, як  $x_{11}$ ,  $x_{12}$ ,  $x_{13}$ . Відповідно для другого та третього ці позначення будуть  $x_{21}$ ,  $x_{22}$ ,  $x_{23}$ ,  $x_{31}$ ,  $x_{32}$ ,  $x_{33}$ .

Тоді вартість перевезення всіх товарів буде знайдена як

$$F(x) \rightarrow \min$$

Обмеження для постачальників означають, що весь товар має бути вивезений зі складів:

$$x_{11} + x_{12} + x_{13} = 20;$$

$$x_{21} + x_{22} + x_{23} = 45;$$

$$x_{31} + x_{32} + x_{33} = 30.$$

Обмеження для споживачів, навпаки, визначають, що запит на товари може виконаний не повністю:

$$x_{11} + x_{21} + x_{31} \leq 74;$$

$$x_{12} + x_{22} + x_{32} \leq 40;$$

$$x_{13} + x_{23} + x_{33} \leq 36.$$

Варто не забувати, що функція solve вирішує тільки задачі максимізації, тому цільову функцію у випадку, коли нам потрібно мінімальне значення критерію оптимізації, потрібно записувати зі знаком мінус.

Нижче на рисунку показано програму, що заходить оптимальне рішення.

```

from pulp import *
import time
start = time.time()

x11 = pulp.LpVariable("x11", lowBound=0)
x12 = pulp.LpVariable("x12", lowBound=0)
x13 = pulp.LpVariable("x13", lowBound=0)
x21 = pulp.LpVariable("x21", lowBound=0)
x22 = pulp.LpVariable("x22", lowBound=0)
x23 = pulp.LpVariable("x23", lowBound=0)
x31 = pulp.LpVariable("x31", lowBound=0)
x32 = pulp.LpVariable("x32", lowBound=0)
x33 = pulp.LpVariable("x33", lowBound=0)

problem = pulp.LpProblem('0', pulp.LpMaximize)

problem += - (7*x11 + 3*x12 + 6*x13 + 4*x21 + 8*x22 + 2*x23 + 1*x31 + 5*x32 + 9*

problem +=x11 + x12 + x13 == 20, '1'
problem +=x21 + x22 + x23 == 45, '2'
problem +=x31 + x32 + x33 == 30, '3'

problem +=x11 + x21 + x31 <= 74, '4'
problem +=x12 + x22 + x32 <= 40, '5'
problem +=x13 + x23 + x33 <= 36, '6'

problem.solve()

print ("Чисельні значення змінних")
for variable in problem.variables():
    print (variable.name, "=", variable.varValue)
print ("Оптимальна вартість перевезень:")
print (abs(value(problem.objective)))
stop = time.time()
print ("Час на вирішення оптимізаційної задачі")
print(stop - start)

```

Результат оптимального розрахунку показав, що деякі замовлення не будуть виконані

```

RESTART: D:\02 Навчальний процес\00 Електрс
Оптиміз 1.py
Чисельні значення змінних
x11 = 0.0
x12 = 20.0
x13 = 0.0
x21 = 9.0
x22 = 0.0
x23 = 36.0
x31 = 30.0
x32 = 0.0
x33 = 0.0
Оптимальна вартість перевезень:
198.0
Час на вирішення оптимізаційної задачі
0.171600341796875

```

Вирішимо іще одну задачу. Підприємство випускає три виду продукції А, Б і С. Для виробництва цієї продукції потрібні такі ресурси, як матеріали, праця робочих та інженерів, а прибуток для кожного виду теж різниться. Для продукції А ці параметри мають значення 2, 2, 2; Для продукції В – 1, 0.45, 2; Для продукції С – 2, 0.4, 6. Прибуток від реалізації А, В, С становить відповідно 90, 65, 270.

Праця робочих обмежена зверху числом 100, витрати на матеріали – числом 80, витрати на працю інженерів – 260. Знайти оптимальний план випуску продукції для максимального прибутку.

```
from pulp import *

# Створення об'єкту задачі лінійного програмування для максимізації прибутку
prob = LpProblem("Maximize Profit", LpMaximize)

# Оголошення змінних рішення
A = LpVariable("Quantity A", lowBound=0, cat='Integer')
B = LpVariable("Quantity B", lowBound=0, cat='Integer')
C = LpVariable("Quantity C", lowBound=0, cat='Integer')

# Оголошення обмежень
prob += 90 * A + 65 * B + 270 * C # Функція прибутку, що максимізується
prob += 2 * A + B * 1 + C * 2 <= 80 # Обмеження на матеріали
prob += 02 * A + B * 0.45 + C * 0.4 <= 100 # Обмеження на працю робочих
prob += 2 * A + B * 2 + C * 6 <= 260 # Обмеження на працю інженерів

# Вирішення задачі
prob.solve()

# Виведення результатів
print("Optimal Production Plan:")
print("Quantity A:", value(A))
print("Quantity B:", value(B))
print("Quantity C:", value(C))
print("Maximized Profit:", value(prob.objective))
```

```
Optimal Production Plan:
Quantity A: 0.0
Quantity B: 0.0
Quantity C: 40.0
Maximized Profit: 10800.0
```

## 4.9. Теорія ігор

Теорія ігор – це галузь математики і економіки, що вивчає прийняття рішень в умовах конфлікту та конкуренції. Вона досліджує поведінку учасників, які намагаються максимізувати свої вигоди у ситуаціях обмежених ресурсів та конкурентного середовища.

Основні поняття теорії ігор включають наступні.

Гра: Формально визначений процес, в якому учасники (гравці) приймають рішення на основі власних обмежених можливостей з метою досягнення своїх цілей.

Гравець: Учасник гри, який приймає рішення.

Стратегія: План дій гравця, що визначає, як він буде діяти у будь-якій можливій ситуації гри.

Пай: Результат гри для кожного гравця в залежності від його вибраних стратегій та стратегій інших гравців.

Рівновага: Ситуація, у якій жоден гравець не може поліпшити своє положення, змінивши свою стратегію, при умові, що інші гравці залишаються при своїх поточних стратегіях.

Розв'язок: Результат, який виникає після застосування рішень всіма гравцями у грі.

Теорія ігор використовується для аналізу широкого спектру ситуацій, від політичних конфліктів до стратегій у бізнесі та управлінні ресурсами. Вона надає універсальні інструменти для розуміння взаємодії між учасниками та прийняття рішень в умовах невизначеності та конфлікту.

У Python існують бібліотеки, які дозволяють працювати з теорією ігор та вирішувати різні задачі, такі як знаходження сідлових точок, аналіз стратегій, моделювання ігор тощо. Ось деякі з них:

**Gambit:** Це бібліотека Python для розв'язання, аналізу та моделювання різних ігор. Вона має широкий спектр функцій, таких як пошук стратегій Неша, знаходження сідлових точок та інші.

**Nashpy:** Ця бібліотека призначена для роботи з рівновагою Неша у стратегічних іграх. Вона дозволяє знаходити рівноваги Неша для двох та багатьох гравців у різних типах ігор.

**Axelrod-Python:** Ця бібліотека призначена для моделювання стратегій у багатьох іграх, зокрема, кооперативних іграх. Вона містить різні вбудовані стратегії, які можна аналізувати та порівнювати.

**Sympy:** Це потужна бібліотека символьних обчислень у Python. Хоча вона не спеціалізується на теорії ігор, вона може бути використана для аналізу ігор та розв'язання різних математичних проблем, пов'язаних з цією галуззю.

Задача знаходження сідлової точки гри полягає у визначенні комбінації стратегій, яка гарантує найкращий результат для одного гравця, незалежно від стратегії, яку обирає інший гравець. Для матричних ігор це означає знаходження максимального мінімуму (або мінімального максимуму) у матриці виграшів гравців.

У Python можна знайти сідлову точку гри, використовуючи бібліотеку NumPy. Ось як це можна зробити для вказаної матриці гри A:

```
import numpy as np

# Задаємо матрицю гри
A = np.array([[10, 8, 11],
              [12, 14, 15],
              [8, 7, 9]])

# Знаходимо мінімальні значення в кожному стовпчику
min_in_columns = np.min(A, axis=0)

# Знаходимо максимальне значення серед мінімумів
max_of_mins = np.max(min_in_columns)

# Знаходимо індекси стовпців, де мінімальне значення дорівнює максимальному з
# мінімумів
saddle_points_indices = np.where(min_in_columns == max_of_mins)[0]

# Якщо знайдено сідлову точку
if len(saddle_points_indices) > 0:
    # Виводимо сідлові точки
    for column_index in saddle_points_indices:
        row_index = np.argmin(A[:, column_index])
        saddle_point_value = A[row_index, column_index]
        print(f"Сідлова точка знаходиться на позиції ({row_index + 1}, {column_index + 1}) зі значенням {saddle_point_value}.")
else:
    print("Сідлової точки не знайдено.")
```

Сідлова точка знаходиться на позиції (3, 3) зі значенням 9.

Цей код спочатку знаходить мінімальні значення в кожному стовпчику матриці гри, потім визначає максимальне серед цих мінімумів. Нарешті, він

знаходить стовпці, де мінімальне значення дорівнює максимальному з мінімумів, і виводить координати цих сідлових точок.

У випадку, коли відсутня сідлова точка, потрібно знайти частоти застосування стратегій. Для цього можна використовувати різні методи аналізу гри. Одним із способів є аналіз рівноваги Неша, який можна використовувати для визначення оптимальних стратегій для гравців у грі.

Принцип рівноваги Неша полягає у виборі оптимальних стратегій для кожного гравця, враховуючи вибір іншого гравця, таким чином, що жоден з гравців не має мотиву змінити свою стратегію, враховуючи вибір іншого.

Цей код знайде всі рівноваги Неша в грі і виведе їх стратегії для кожного гравця. Частота застосування стратегій визначається через кількість рівноважних точок, у яких кожен гравець використовує певну стратегію.

```
import numpy as np

# Задаємо матрицю гри
A = np.array([[11, 9, 10],
              [14, 13, 8],
              [10, 8, 7]])

# Знаходимо рівновагу Неша за допомогою бібліотеки Nashpy
import nashpy as nash

game = nash.Game(A)
equilibria = game.support_enumeration()

# Виводимо результати рівноваги Неша
for eq in equilibria:
    print("Рівновага Неша:")
    print("Гравець 1:", eq[0])
    print("Гравець 2:", eq[1])
    print()
    print("Рівновага Неша:
Гравець 1: [0.83333333 0.16666667 0.
Гравець 2: [-0.          0.33333333 0.66666667]
```

Стратегію Неша можна застосовувати і до біматричних ігор, коли кожен гравець має свою власну матрицю виграшів. Ось приклад використання бібліотеки Nashpy для пошуку рівноваги Неша у біматричній грі:



```

import numpy as np
import nashpy as nash

# Задаємо матриці гри для обох гравців
A = np.array([[11, 9, 10],
              [14, 13, 8],
              [10, 8, 7]])

B = np.array([[101, 29, 100],
              [104, 130, 84],
              [105, 87, 97]])

# Створюємо об'єкт гри з використанням матриць гри кожного гравця
game = nash.Game(A, B)

# Знаходимо рівновагу Неша
equilibria = game.support_enumeration()

# Виводимо результати рівноваги Неша
for eq in equilibria:
    print("Рівновага Неша:")
    print("Гравець 1:", eq[0])
    print("Гравець 2:", eq[1])
    print()

```

```

Рівновага Неша:
Гравець 1: [0. 1. 0.]
Гравець 2: [0. 1. 0.]

```

У цьому прикладі ми задаємо дві матриці гри A і B, одна для кожного гравця, і знаходимо всі рівноваги Неша в цій біматричній грі. Кожна рівновага Неша пов'язана зі стратегіями обох гравців, які утворюють рівновагу.

Бібліотека Nashpy підтримує аналіз рівноваги Неша для ігор з більшою кількістю гравців. Ви можете використовувати цю бібліотеку для аналізу біматричних ігор з будь-якою кількістю гравців, включаючи 3, 4 або більше.

Ось приклад застосування для біматричної гри з трьома гравцями:

```

import numpy as np
import nashpy as nash

# Задаємо матриці гри для трьох гравців
A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

```

```

B = np.array([[9, 8, 7],
              [6, 5, 4],
              [3, 2, 1]])
C = np.array([[11, 12, 13],
              [14, 15, 16],
              [17, 18, 19]])

# Створюємо об'єкт гри з використанням матриць гри кожного з трьох
гравців
game = nash.Game(A, B, C)
# Знаходимо рівновагу Неша
equilibria = game.support_enumeration()

# Виводимо результати рівноваги Неша
for eq in equilibria:
    print("Рівновага Неша:")
    print("Гравець 1:", eq[0])
    print("Гравець 2:", eq[1])
    print("Гравець 3:", eq[2])
    print()

```

У цьому прикладі ми задаємо три матриці гри A, B і C для трьох гравців і знаходимо всі рівноваги Неша в цій грі. Кожна рівновага Неша пов'язана зі стратегіями кожного з гравців, які утворюють рівновагу.

#### **4.10. Кластеризація, класифікація**

В економіці, з метою кращого управління соціально-економічними об'єктами, досить часто виникає потреба в угрупованні об'єктів в однорідні групи. Наприклад, банки об'єднані у такі групи: системні (найбільші), середні і

малі. Віднесення їх до певної категорії відбувається за критеріями валюти балансу, власного капіталу та кількості клієнтів.

Але у більшості випадків, для різних підприємств, фірм, організацій таке угруповання провести складно. Для вирішення такої задачі існують спеціальні методи, які називаються кластеризацією.

У прямому стосунку до задачі кластеризації стоїть задача класифікації. Її сутність полягає у наступному.

Нехай ми маємо декілька груп (кластерів) об'єктів. І коли з'явиться новий об'єкт, до якого кластеру його треба віднести? Ця задача вирішується, коли ми маємо інші, вже наперед розкласифіковані об'єкти.

Найбільш популярним на наш час є метод *k*-середніх (*k*-means), який базується на такому принципі: об'єкти групуються у кластери таким чином, щоб середнє координати центру одного кластера знаходилося від середнього координати центру другого кластера на відстані більшій, аніж відстань між центром кластера і об'єктами що до нього входять.

Нехай нам відомі 5 об'єктів, позначені як *Обі*, що характеризуються п'ятьма показниками, позначеними як *Покі*.

	Пок1	Пок2	Пок3	Пок4	Пок5
Об1	19,889503	66	106,2	30,71	269
Об2	26,519337	58	113,3	22,94	237,1
Об3	16,022099	74	56,64	24,05	273,6
Об4	20,165746	86	101,5	27,75	232,6

Для проведення кластеризації можна використати бібліотеку *scikit-learn* у Python. Ось приклад коду для кластеризації 4 об'єктів, що характеризуються 5 показниками

У цьому прикладі ми використовуємо метод кластеризації *k*-means з бібліотеки *scikit-learn*. Після виконання кластеризації ми отримуємо мітки кластерів для кожного об'єкта, а також центри кластерів. Ви можете змінювати кількість кластерів, змінюючи параметр *n\_clusters* у класі *KMeans*.

У наданому коді є можливість отримати мітки кластерів для кожного об'єкта за допомогою атрибуту `labels_` об'єкта `KMeans`. Цей атрибут містить мітки кластерів для кожного з навчальних прикладів.

Також цей код виведе індекс об'єкта разом з його міткою кластера. Об'єкт з індексом 1 відноситься до кластера з міткою 0, об'єкт з індексом 2 відноситься до кластера з міткою 1, і так далі.

Зверніть увагу, що для роботи цього алгоритму необхідно інсталиувати бібліотеку командою `pip install scikit-learn.`

```
from sklearn.cluster import KMeans
```

```
# Задання об'єктів (приклад)
```

```
data = [  
    [19.889503,66.0,106.2,30.71,269],  
    [26.519337,58.0,113.3,22.94,237.1],  
    [16.022099,74.0,56.64,24.05,273.6],  
    [20.165746,86.0,101.5,27.75,232.6]
```

```
]
```

```
# Виконання кластеризації
```

```
kmeans = KMeans(n_clusters=3) # Вказуємо кількість кластерів  
kmeans.fit(data) # Проведення кластеризації
```

```
# Виведення результатів кластеризації
```

```
labels = kmeans.labels_ # Отримання міток кластерів  
centroids = kmeans.cluster_centers_ # Отримання центрів кластерів
```

```
# Виведення міток кластерів та центрів кластерів
```

```
print("Мітки кластерів:", labels)  
print("Центри кластерів:", centroids)
```

```
# Виведення міток кластерів для кожного об'єкта
```

```
for i, label in enumerate(labels):  
    print("Об'єкт {} відноситься до кластера {}".format(i+1, label))  
Мітки кластерів: [1 0 2 0]  
Центри кластерів: [[ 23.3425415  72.          107.4          25.345          234.85          ]  
 [ 19.889503  66.          106.2          30.71          269.          ]  
 [ 16.022099  74.          56.64          24.05          273.6          ]]  
Об'єкт 1 відноситься до кластера 1  
Об'єкт 2 відноситься до кластера 0  
Об'єкт 3 відноситься до кластера 2  
Об'єкт 4 відноситься до кластера 0
```

Щоб визначити кластери для нових об'єктів, тобто, вирішити задачу класифікації, ми формуємо масив, у якому містяться вже кластеризовані дані та такі, що їх потрібно віднести до певних класів.

У прикладі ці дані розміщено у останніх двох рядках масиву даних. А щоб вивести всю інформацію про навчальні дані та тестові (тобто, дані, які потрібно класифікувати) включаючи номери кластерів для кожного об'єкту, можна використати наступний код. Тут пропонується розбити дані на три кластери. Рядки 5 та 6 – це нові об'єкти.

```

1 | import numpy as np
2 | from sklearn.cluster import KMeans
3 |
4 | # Припустимо, що у нас є масив даних, де останні два рядки
5 |   - це нові об'єкти
6 | data = np.array([
7 |     [19.889503,66.0,106.2,30.71,269],
8 |     [26.519337,58.0,113.3,22.94,237.1],
9 |     [16.022099,74.0,56.64,24.05,273.6],
10 |    [20.165746,86.0,101.5,27.75,232.6],
11 |    [25.625,61,23.75,68.15,372.93],
12 |    [24.6875,86,20.625,107.3,336.84]
13 | ])
14 | # Розділення даних на навчальний і тестовий набори
15 | X_train = data[:-2]
16 | X_new = data[-2:]
17 |
18 | # Проведення кластеризації
19 | kmeans = KMeans(n_clusters=3)
20 | kmeans.fit(X_train)
21 |
22 | # Визначення номерів кластерів для нових об'єктів
23 | predicted_clusters = kmeans.predict(X_new)
24 |
25 | # Виведення інформації про навчальні дані та тексти,
26 |   включаючи номери кластерів
27 | for i, (row, cluster) in enumerate(zip(data,
28 |   np.concatenate((kmeans.labels_, predicted_clusters)))):
29 |     print("Рядок {}: Дані: {}, Номер кластера:
30 |     {}".format(i+1, row, cluster))

```

```

Рядок 1: Дані: [ 19.889503  66.          106.2          30.71
269.          ], Номер кластера: 0
Рядок 2: Дані: [ 26.519337  58.          113.3          22.94
237.1         ], Номер кластера: 1
Рядок 3: Дані: [ 16.022099  74.           56.64          24.05
273.6         ], Номер кластера: 2
Рядок 4: Дані: [ 20.165746  86.           101.5          27.75
232.6         ], Номер кластера: 1
Рядок 5: Дані: [ 25.625  61.          23.75  68.15  372.93 ],
Номер кластера: 2
Рядок 6: Дані: [ 24.6875  86.          20.625  107.3  336.84
], Номер кластера: 2

```

## 4.11. Нейронні сітки

Нейронні сітки – це сітки, що складаються зі зв'язаних між собою простих елементів – формальних нейронів. Ядром використовуваних представлень є ідея про те, що нейрони можна моделювати досить простими формулами, а вся складність процесу моделювання визначається зв'язками між нейронами. Кожен зв'язок представляється як зовсім простий елемент, що служить для передачі сигналу. Для опису кожного нейрону використовується проста і одна й та сама функція. Вони імітують нейронну мережу в живих організмах.

### 4.11.1. Регресійні моделі

Для створення математичної моделі за допомогою нейронних мереж у Python можна скористатися бібліотекою TensorFlow або PyTorch. Давайте розглянемо приклад застосування нейронних мереж для вирішення цієї задачі за допомогою TensorFlow.

Спочатку, встановимо TensorFlow:

```
pip install tensorflow
```

Створимо за допомогою нейронних сіток регресійну модель. за начальними масивами даних  $X$ , що складаються з 3-х колонок і 10 рядків, та  $Y$  – значення ендогенної змінної. Необхідно отримати математичну модель, яка б дозволила вирахувати значення  $Y$  для 3-х нових значень  $X$ .

У цьому коді ми створюємо нейронну мережу з трьома шарами Dense (повнозв'язними) за допомогою бібліотеки TensorFlow. Ми навчаємо модель на навчальних даних  $X_{train}$  та  $Y_{train}$ , використовуючи середньоквадратичну помилку як функцію втрат. Після навчання ми можемо використовувати модель для передбачення значень  $Y$  для нових значень  $X$ , використовуючи метод `predict`.

```

import numpy as np
import tensorflow as tf
from tensorflow.keras import Input, Model
from tensorflow.keras.layers import Dense

# Навчальні дані у форматі списку
X_train = [[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]]
Y_train = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

# Перетворення списків у масиви NumPy
X_train = np.array(X_train)
Y_train = np.array(Y_train)

# Створення вхідного шару
inputs = Input(shape=(3,))

# Створення моделі
dense1 = Dense(10, activation='relu')(inputs)
dense2 = Dense(10, activation='relu')(dense1)
output = Dense(1)(dense2)

model = Model(inputs=inputs, outputs=output)

# Компіляція моделі
model.compile(optimizer='adam', loss='mse')

# Навчання моделі
model.fit(X_train, Y_train, epochs=1000, verbose=0)

# Передбачення Y для нових значень X
X_new = [[31, 32, 33], [34, 35, 36], [37, 38, 39]]
X_new = np.array(X_new) # Перетворення у формат NumPy
predictions = model.predict(X_new)

print("Predictions Y for new X values:", predictions.flatten())

```

```

| [1m1/10[0m [32m _____ [0m [37m [0m [1m0s
| [0m 34ms/step [0m [1m1/10[0m [32m _____
| _____ [0m [37m [0m [1m0s [0m 56ms/step
| Predictions for new X values: [111.704704 122.13899 132.57327 ]

```

Програма спочатку друкує протокол роботи з навчання нейронної сітки, а вже потім – результат розрахунку нових значень  $Y$  за новими значеннями  $X$ .

Ось пояснення можливих параметрів для функцій **tf.keras.layers.Dense**:

**units:** Це ціле число, яке вказує кількість нейронів (одиниць) у шарі. В першому прикладі вище, 10 вказує на те, що у шарі є 10 нейронів.

**activation:** Це рядок або функція активації, яка використовується для активації виходів нейронів. Деякі з популярних функцій активації включають



'relu' (линійне ректифіковане одинице), 'sigmoid' (сигмоїд), 'tanh' (гіперболічний тангенс) тощо.

**input\_shape:** Це кортеж, що вказує форму вхідних даних. У першому Dense-шарі це використовується для вказівки форми вхідних даних. В даному випадку (3,) вказує на те, що кожен вхід має 3 ознаки.

У другому Dense-шарі input\_shape не вказується, оскільки вихідні дані з першого Dense-шару використовуються як вхід для другого Dense-шару.

В останньому Dense-шарі units=1, оскільки ми хочемо отримати одне значення на виході моделі.

Ці параметри можна налаштовувати залежно від архітектури вашої нейронної мережі і особливостей задачі.

Розглянемо також інші функції програми:

**Model(inputs=inputs, outputs=output):** Цей рядок визначає модель, використовуючи функціональний API у бібліотеці TensorFlow / Keras. Параметр inputs вказує на вхідні дані моделі, тобто форму вхідних даних. outputs вказує на вихід моделі, тобто її результат. Це може бути один або кілька виходів, залежно від архітектури вашої моделі.

**model.compile(optimizer='adam', loss='mse'):** Цей рядок компілює модель з параметрами, необхідними для її навчання. optimizer визначає метод оптимізації, який використовується для оновлення ваг моделі під час навчання. 'adam' - це алгоритм оптимізації Adam, який часто використовується для навчання нейронних мереж. loss визначає функцію втрат, яка використовується для оцінки різниці між прогнозованими значеннями моделі та фактичними значеннями. 'mse' означає середньоквадратичну помилку, яка часто використовується для задач регресії.

Отже, ці параметри дозволяють визначити архітектуру моделі, скомпілювати її для навчання та визначити методи оптимізації та функцію втрат для навчання моделі.

## 4.11.2. Кластеризація. Класифікація

Кластеризація за допомогою нейронних мереж часто використовується для навчання без учителя, тобто без наявності цільової змінної. Для цього використовуються такі методи, як автокодування або методи глибокого навчання, наприклад, варіаційні автокодувачі.

Одним з таких методів є автокодувач. Він навчається відтворювати вхідні дані на виході, а потім використовується для знаходження низькорозмірних представлень даних. За допомогою автокодувача можна здійснити кластеризацію, обчисливши відстані між вхідними даними і їх відтвореннями.

За допомогою бібліотеки TensorFlow створюється модель автокодувача з використанням бібліотеки TensorFlow. Модель навчається на нормалізованих даних і використовується для отримання низькорозмірного представлення даних. Отримані представлення даних можуть бути використані для кластеризації об'єктів.

Такий підхід є незручним для економістів, тому для вирішення задачі кластеризації зі вказанням номерів кластерів можна скористатися нейронною мережею, яка буде класифікувати об'єкти в кластери. Ось як це можна зробити з використанням бібліотеки TensorFlow:

Замість методу `predict_classes`, тут використано `np.argmax` для отримання індексу класу з найбільшою ймовірністю для кожного об'єкта. Це уникне помилки `AttributeError`.

В наведеному прикладі нейронна сітка віднесла всі об'єкти до одного класу.

```
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
```

```
# Дані
data = np.array([
    [19.889503, 66.0, 106.2, 30.71, 269],
    [26.519337, 58.0, 113.3, 22.94, 237.1],
    [16.022099, 74.0, 56.64, 24.05, 273.6],
```



Потім використаємо навчену модель для класифікації нових об'єктів data\_new.

Для визначення кластерів нових об'єктів, ми можемо використати вихідні значення нейронної мережі та знайти найближчий об'єкт у тренувальному наборі.

```
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from scipy.spatial.distance import cdist

# Дані
data = np.array([
    [19.889503, 66.0, 106.2, 30.71, 269],
    [26.519337, 58.0, 113.3, 22.94, 237.1],
    [16.022099, 74.0, 56.64, 24.05, 273.6],
    [20.165746, 86.0, 101.5, 27.75, 232.6],
    [25.625, 61, 23.75, 68.15, 372.93],
    [24.6875, 86, 20.625, 107.3, 336.84]
])

data_new = np.array([
    [79.889503, 66.0, 106.2, 130.71, 269],
    [6.519337, 58.0, 113.3, 22.94, 237.1]
])

# Нормалізація даних
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)
scaled_data_new = scaler.transform(data_new)

# Модель нейронної мережі
inputs = tf.keras.Input(shape=(data.shape[1],))
x = tf.keras.layers.Dense(64, activation='relu')(inputs)
x = tf.keras.layers.Dense(32, activation='relu')(x)
x = tf.keras.layers.Dense(16, activation='relu')(x)
x = tf.keras.layers.Dense(8, activation='relu')(x)
outputs = tf.keras.layers.Dense(3, activation='softmax')(x)

model = tf.keras.Model(inputs=inputs, outputs=outputs)

# Компіляція та навчання моделі
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model.fit(scaled_data, np.zeros(len(scaled_data)), epochs=100, verbose=0) # Зразки міток
не використовуються

# Класифікація нових об'єктів
predictions_new = np.argmax(model.predict(scaled_data_new), axis=-1)

# Класифікація тренувальних об'єктів
predictions_train = np.argmax(model.predict(scaled_data), axis=-1)

# Вивід результатів
print("Номери кластерів для навчальних об'єктів:")
```



Отже, атрибут `np.argmax(model.predict(scaled_data_new), axis=-1)` повертає індекси класів, які модель вважає найбільш ймовірними для кожного з переданих нових даних.

#### 4.12. Імітаційне моделювання

Імітаційне моделювання – це процес побудови моделі реального світу для аналізу його роботи та передбачення поведінки в різних сценаріях. У Python для імітаційного моделювання доступно кілька бібліотек, серед яких найбільш популярні:

**SimPy:** SimPy є однією з найпопулярніших бібліотек для імітаційного моделювання в Python. Вона пропонує потужні засоби для моделювання подійно-орієнтованих систем.

**DEAP (Distributed Evolutionary Algorithms in Python):** DEAP – це бібліотека для розробки та експериментів з алгоритмами еволюційного обчислення, такими як генетичні алгоритми.

**Salabim:** Salabim є бібліотекою, яка пропонує простий інтерфейс для створення імітаційних моделей за допомогою агентно-орієнтованого підходу.

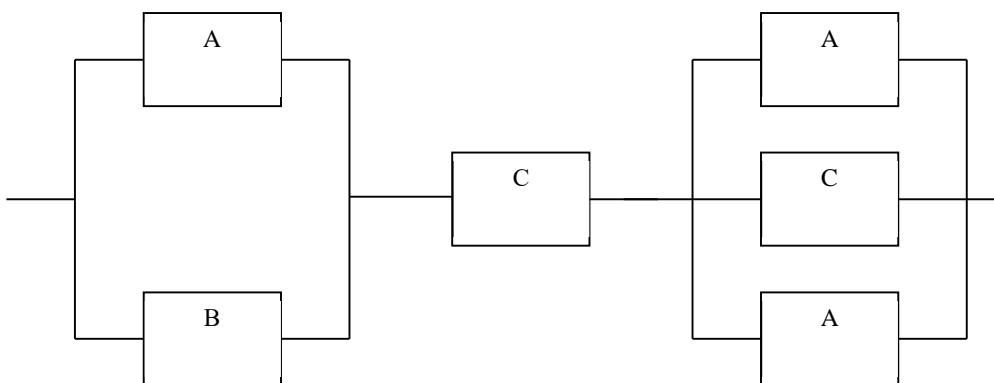
**PyCX:** PyCX – це бібліотека для комп'ютерного моделювання та візуалізації поведінки складних систем.

Ці бібліотеки мають свої власні набори функцій для розрахунків та моделювання, зокрема для створення агентів, подій, сценаріїв і т.д. Які саме функції вам потрібно використовувати, залежить від вашого конкретного застосування і моделі.

Для економістів важливим є імітаційне моделювання соціально-економічних систем, які характеризуються набором формул, що описують поведінку системи у певному взаємному зв'язку. На вхід такої системи поступають сигнали з наперед визначеним законом розподілу.

Реакція системи обробляється статистичними методами, що й визначає її ефективність роботи.

Для прикладу, візьмемо соціально-економічну систему, що описується наступною блок-схемою.



Моделюємо бізнес-процес узгодження рішення у плановому відділі за певний час. Співробітники розглядають інформацію у вказаній на схемі послідовності, але їх характеристики мають тільки три відмінності і позначені на схемах буквами А, В, С. Паралельне розташування співробітників означає, що інформацію може розглядати один із двох.

Кожна буква означає, тип продуктивності співробітника, що коливається за нормальним законом з характеристиками  $Mx$  – середня продуктивність,  $b_x$  – стандартне відхилення продуктивності. Продуктивністю є ймовірність того, що подана пропозиція буде розглянута цим співробітником.

У наступні таблиці подано числові значення цих параметрів для трьох типів співробітників.

$M_{pC}$	$b_{pC}$	$M_{pA}$	$b_{pA}$	$M_{pB}$	$b_{pB}$
0,826	0,067	0,88	0,083	0,77	0,06



Розглянути ситуацію, коли до планового відділу протягом місяця надійде 200 пропозицій. Визначити статистичні характеристики результативності роботи відділу. Тобто, скільки в середньому пропозицій буде розглянуто і яка дисперсія (або стандарт) обсягу розглянутих справ.

Імовірність проходження такої справи через відділ буде знайдена як

$$P = [1-(1-PA)*(1-PB)]*PC*[1-(1-PA)(1-PB)(1-PC)],$$

а кількість справ, що буде розглянуто в середньому  $200*P$ . Ось програма

```
import numpy as np

# Задані середні та стандартні відхилення для кожної ймовірності
MpA, MpB, MpC = 0.88, 0.77, 0.826
std_pA, std_pB, std_pC = 0.083, 0.06, 0.067

# Кількість спроб
num_trials = 200

# Генерація випадкових значень для ймовірностей
# PA, PB, PC та P - це масиви значень ймовірностей
PA = np.random.normal(MpA, std_pA, num_trials)
PB = np.random.normal(MpB, std_pB, num_trials)
PC = np.random.normal(MpC, std_pC, num_trials)

# Обчислення ймовірності за заданою формулою
P = (1 - (1 - PA) * (1 - PB)) * PC * (1 - (1 - PA) * (1 - PB) * (1 - PC))

# Середнє значення та стандартне відхилення ймовірності, помножені на кількість спроб
mean_P = np.mean(P) * num_trials
std_P = np.std(P) * num_trials

# Виведення результатів
print("Середня ймовірність:", np.mean(P))
print("Стандарт ймовірності:", np.std(P))
print("Кількістьб прав, що надійшла до відділу", num_trials)
print("Середня кількість справ, що пройдуть через відділ:", round(mean_P, 0))
print("Стандарт цієї кількості:", std_P)
```

```
Середня ймовірність: 0.8029585686479729
Стандарт ймовірності: 0.06625870572283324
Кількістьб прав, що надійшла до відділу 200
Середня кількість справ, що пройдуть через відділ: 161.0
Стандарт цієї кількості: 13.251741144566648
```

Результат не радує, адже через відділ реально пройде тільки 80% справ. Але цей приклад показує, наскільки легко імітаційне моделювання може бути реалізоване простими засобами бібліотеки NumPy.

Простий запис генерації ймовірностей може бути ускладнений умовним оператором. Наприклад, як можна переписати формулу генерації імовірності  $P$ , якщо вираз  $(1 - (1 - PA) * (1 - PB))$  повинен змінюватися в залежності від значення  $PA$ . Якщо  $PA > MP_A + 6PA$ , то він повинен дорівнювати  $PB$ ?

Тоді цей оператор повинен виглядати отак

```
P = np.where(PA > MP_A + 6 * std_pA, PB * PC * (1 - (1 - PA) * (1 - PB) * (1 - PC)),  
1 - (1 - PA) * (1 - PB) * PC * (1 - (1 - PA) * (1 - PB) * (1 - PC)))
```

можете вставити ще одну функцію `np.where` всередині першої функції `np.where`, для реалізації додаткової логіки вибору значень в залежності від умов.

```
import numpy as np  
# Приклад умов та значень  
PA = np.array([0.9, 0.6, 0.3])  
PB = np.array([0.8, 0.5, 0.2])  
PC = np.array([0.7, 0.4, 0.1])  
MP_A, MP_B, MP_C = 0.5, 0.4, 0.3  
std_pA, std_pB, std_pC = 0.1, 0.1, 0.1  
# Формула з використанням вкладеного np.where  
P = np.where(PA > MP_A + 6 * std_pA, PB, np.where(PB > MP_B + 6 * std_pB,  
PC, 1 - (1 - PA) * (1 - PB) * PC * (1 - (1 - PA) * (1 - PB) * (1 - PC))))  
print(P)
```

У цьому прикладі використано дві вкладені функції `np.where`. Перша функція `np.where` перевіряє умову  $PA > MP_A + 6 * std_pA$ . Якщо ця умова виконується, вона повертає значення  $PB$ , в іншому випадку вона переходить до другої функції `np.where`. Друга функція `np.where` перевіряє умову  $PB > MP_B + 6 * std_pB$ . Якщо ця умова виконується, вона повертає значення  $PC$ , інакше вона обчислює значення за зазначеною формулою.

### 4.13. Нечіткі множини

Існують описові моделі на звичайній мові, які дозволяють зрозуміти якісні характеристики соціально-економічної системи. Але такі моделі не дозволяють отримати потрібні для їх вивчення, прогнозування і керування числові характеристики. В такому випадку варто застосувати синтез нечітких моделей, який базується на уявленні групи експертів про функціональну діяльність системи.

Нечіткі моделі базуються на поняттях нечітких множин, які представляють собою множину можливих значень нечіткої величини у формі функції приналежності виду  $A = \{x/\mu_A(x)>0\}$ .

Хай  $E$  – універсальна множина,  $x$  – елемент  $E$ , а  $R$  – певна властивість. Звичайна (чітка) підмножина  $A$  універсальної множини  $E$ , елементи якої задовольняють властивість  $R$ , визначається як множина впорядкованої пари  $A = \{\mu_A(x)/x\}$ , де  $\mu_A(x)$  – характеристична функція, що приймає значення 1, коли  $x$  задовольняє властивості  $R$ , і 0 – в іншому випадку.

Для отримання чіткого висновки з нечіткої моделі соціально-економічної системи і системи управління нею, потрібно мати їх нечіткий опис у вигляді функцій приналежності, описаних у п. 4.7. Тоді процедура отримання чіткого результату з нечітких моделей на підставі чіткого значення вхідних факторів полягає у наступному.

Хай в базі правил є  $m$  правил вигляду:

$R_1$ : ЯКЩО  $x_1$  це  $A_{11}$  . І .  $x_n$  це  $A_{1n}$ , ТО  $y$  це  $B_1$ .

$R_i$ : ЯКЩО  $x_1$  це  $A_{i1}$  . І .  $x_n$  це  $A_{in}$ , ТО  $y$  це  $B_i$ .

· · · · ·

$R_m$ : ЯКЩО  $x_1$  це  $A_{m1}$  . І .  $x_n$  це  $A_{mn}$ , ТО  $y$  це  $B_m$ .

де  $x_k$ , ( $k = 1..n$ ) – вхідні змінні;  $y$  – вихідна змінна;  $A_{ik}$  – задані нечіткі множини з функціями приналежності для  $x_k$ ,  $B_i$  – задані нечіткі множини з функціями приналежності для  $y$ .

Результатом нечіткого висновку є чітке значення змінної  $y^*$  на підставі заданих чітких значень  $x_k, k=1..n$

Наприклад, ось нечіткі правила керування автомобілем.

X1 – відстань до передньої машини, X2 – швидкість вашого авто.

V – міра натискання акселератора.

Якщо X1 = **БЛИЗЬКО** і X2 = **ШВИДКО** ТО **ПРИГАЛЬМУЙ**

Якщо X1 = **БЛИЗЬКО** і X2 = **НОРМАЛЬНО** ТО **ПРИГАЛЬМУЙ**

Якщо X1 = **БЛИЗЬКО** і X2 = **ПОВІЛЬНО** ТО **ТАК І ЇДЬ**

Якщо X1 = **НОРМАЛЬНО** і X2 = **ШВИДКО** ТО **ПРИГАЛЬМУЙ**

Якщо X1 = **НОРМАЛЬНО** і X2 = **НОРМАЛЬНО** ТО **ТАК І ЇДЬ**

Якщо X1 = **НОРМАЛЬНО** і X2 = **ПОВІЛЬНО** ТО **ТАК І ЇДЬ**

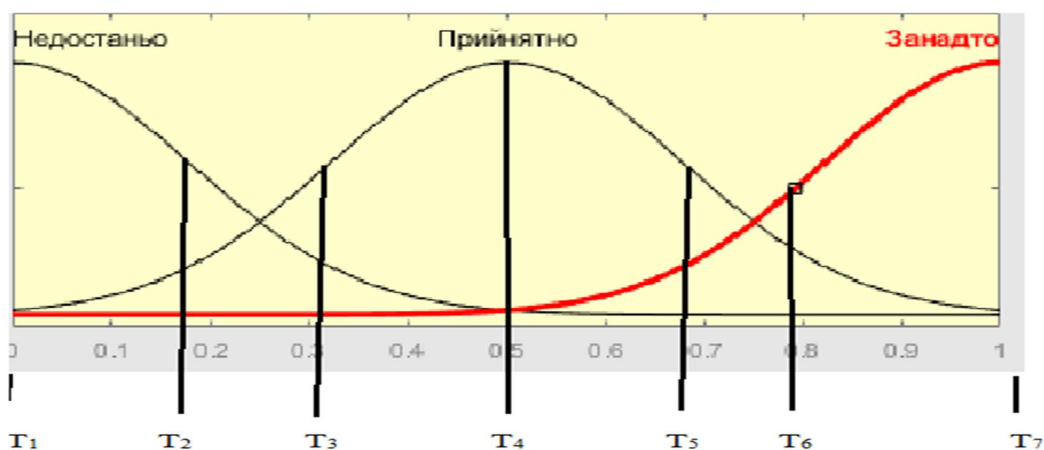
Якщо X1 = **ДАЛЕКО** і X2 = **ШВИДКО** ТО **ТАК І ЇДЬ**

Якщо X1 = **ДАЛЕКО** і X2 = **НОРМАЛЬНО** ТО **ТАК І ЇДЬ**

Якщо X1 = **ДАЛЕКО** і X2 = **ПОВІЛЬНО** ТО **ПРИТОПИ**

Кожна з цих нечітких змінних позначена червоним кольором і має три варіанти: мало, середнє і дуже. Визначення групою експертів кожного з варіантів нечіткої змінної утворює множину точок, яку можна описати набором графіків.

Наприклад



Щоб у Пайтоні задати нечітку змінну, що характеризується кривими трикутної форми, форми нормального закону та форми логістиної криво, потрібно скористатися бібліотекою scikit-fuzzy, яка надає засоби для роботи з

нечіткою логікою. Нижче наведений приклад створення нечіткої змінної, що характеризується кривими трикутної форми, форми нормального закону та форми логістичної кривої.

Функція `plt.subplot()` визначає сітку підграфіків для розташування декількох графіків на одному зображенні. Цифри, передані як аргументи у функцію `plt.subplot()`, визначають розміри сітки та позицію кожного підграфіка у цій сітці.

У вищенаведеному прикладі перший аргумент 3 вказує на кількість рядків у сітці, другий аргумент 1 вказує на кількість стовпців у сітці, третій аргумент 3 вказує на номер поточного підграфіка у цій сітці.

Отже, `plt.subplot(3, 1, 3)` створює підграфік, який займає останню позицію у рядку, на який складається зі стовпців, який в цьому випадку є єдиним, а також займає останню позицію серед усіх рядків.

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

# Створення універсальної множини (простору значень) для змінної X
x = np.arange(0, 11, 1)

# Створення нечіткої трикутної функції приналежності
triangular_membership = fuzz.trimf(x, [3, 6, 9])

# Створення нечіткої функції приналежності з формою нормального закону
normal_membership = fuzz.gaussmf(x, np.mean(x), np.std(x))

# Створення нечіткої функції приналежності з формою логістичної кривої
logistic_membership = fuzz.sigmf(x, np.mean(x), 1)

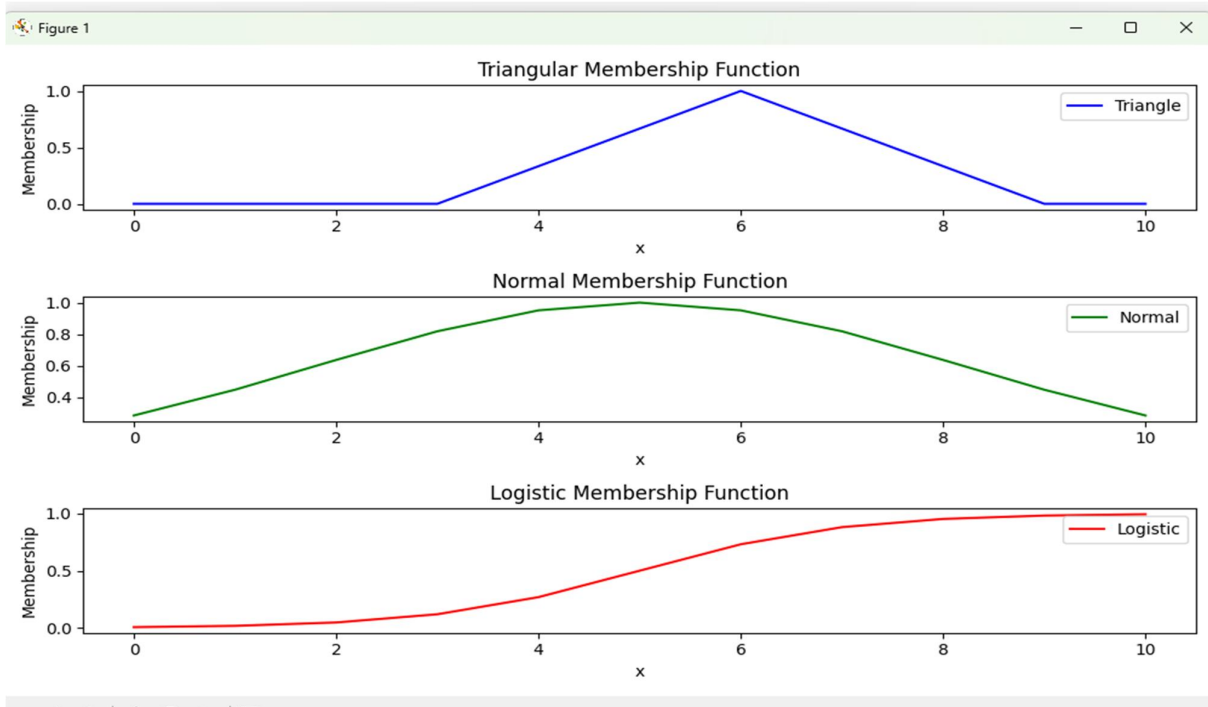
# Візуалізація нечітких функцій приналежності
plt.figure(figsize=(10, 6))

plt.subplot(3, 1, 1)
plt.plot(x, triangular_membership, 'b', label='Triangle')
plt.title('Triangular Membership Function')
plt.xlabel('x')
plt.ylabel('Membership')
plt.legend()

plt.subplot(3, 1, 2)
plt.plot(x, normal_membership, 'g', label='Normal')
plt.title('Normal Membership Function')
plt.xlabel('x')
plt.ylabel('Membership')
plt.legend()
```

```
plt.subplot(3, 1, 3)
plt.plot(x, logistic_membership, 'r', label='Logistic')
plt.title('Logistic Membership Function')
plt.xlabel('x')
plt.ylabel('Membership')
plt.legend()
```

```
plt.tight_layout()
plt.show()
```



Аргументи `np.arange(0, 11, 1)` визначають масив значень від 0 до 10 з кроком 1. Конкретно:

Перший аргумент 0 – це початкове значення (включаючи).

Другий аргумент 11 – це кінцеве значення (не включаючи).

Третій аргумент 1 – це крок (відстань між послідовними значеннями).

Таким чином, `np.arange(0, 11, 1)` генерує масив чисел від 0 до 10 включно з кроком 1.

Тепер створимо програму, що реалізує 9 нечітких правил для двох вхідних змінних  $X_1$  та  $X_2$  і вихідної змінної  $B$ .

```
import numpy as np
```

```

import skfuzzy as fuzz
from skfuzzy import control as ctrl

# Визначення нечітких змінних
X1 = ctrl.Antecedent(np.arange(0, 11, 0.1), 'X1')
X2 = ctrl.Antecedent(np.arange(0, 11, 0.1), 'X2')
B = ctrl.Consequent(np.arange(0, 11, 0.1), 'B')

# Визначення нечітких функцій приналежності
X1['triangular'] = fuzz.trimf(X1.universe, [0, 3, 6])
X1['normal']      =      fuzz.gaussmf(X1.universe,      np.mean(X1.universe),
np.std(X1.universe))
X1['logistic'] = fuzz.sigmf(X1.universe, 9, 1)
X2['triangular1'] = fuzz.trimf(X2.universe, [-3, 0, 6])
X2['triangular2'] = fuzz.trimf(X2.universe, [3, 6, 9])
X2['triangular3'] = fuzz.trimf(X2.universe, [6, 9, 11])
B['normal1'] = fuzz.gaussmf(B.universe, 0, 3.0)
B['normal2'] = fuzz.gaussmf(B.universe, 6, 3.0)
B['normal3'] = fuzz.gaussmf(B.universe, 9, 3.0)

# Визначення вирішальних правил
rule1 = ctrl.Rule(X1['triangular'] & X2['triangular1'], B['normal1'])
rule1 = ctrl.Rule(X1['normal'] & X2['triangular1'], B['normal1'])
rule1 = ctrl.Rule(X1['logistic'] & X2['triangular1'], B['normal1'])
rule1 = ctrl.Rule(X1['triangular'] & X2['triangular2'], B['normal2'])
rule1 = ctrl.Rule(X1['normal'] & X2['triangular2'], B['normal2'])
rule1 = ctrl.Rule(X1['logistic'] & X2['triangular2'], B['normal2'])
rule1 = ctrl.Rule(X1['triangular'] & X2['triangular3'], B['normal3'])
rule1 = ctrl.Rule(X1['normal'] & X2['triangular3'], B['normal3'])
rule1 = ctrl.Rule(X1['logistic'] & X2['triangular3'], B['normal3'])

```

```
# Створення нечіткої системи
system = ctrl.ControlSystem([rule1])
simulator = ctrl.ControlSystemSimulation(system)

# Визначення вхідних значень
simulator.input['X1'] = 3.2
simulator.input['X2'] = 7.4

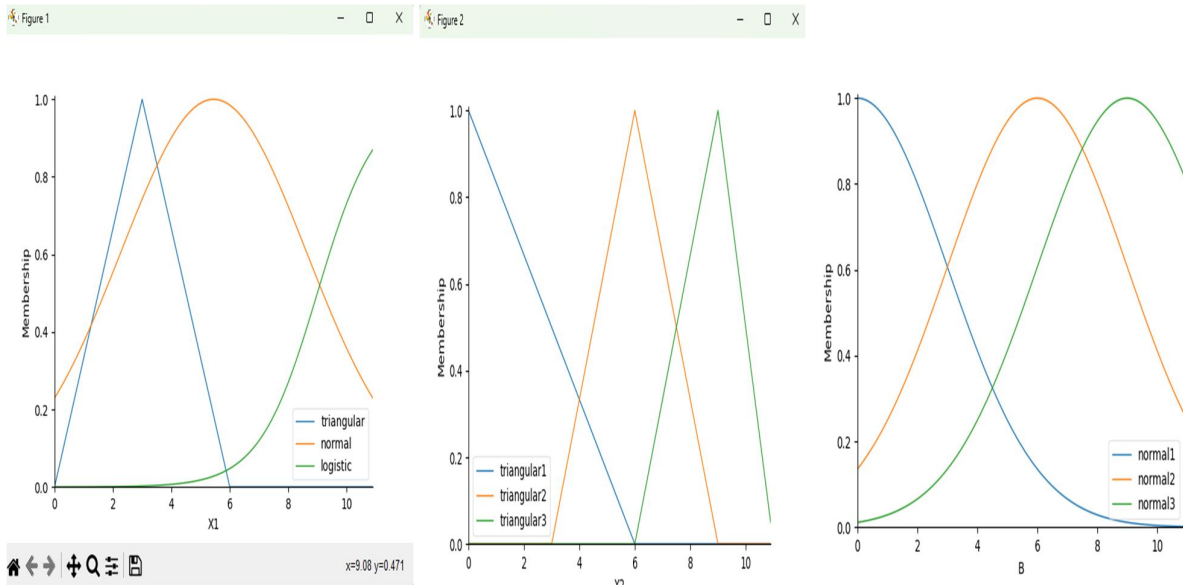
# Виконання нечіткої інференції
simulator.compute()

# Отримання значення B
result_B = simulator.output['B']
print('Точне значення виходу нечіткої моделі',result_B)

# Візуалізація системи та результатів
X1.view()
X2.view()
B.view()
|Точне значення виходу нечіткої моделі 5.4500000000000006
```

А ось і рафіки, що описують кожен з нечітких змінних.





## Контрольні запитання

1. Що таке бібліотека Pandas у Python і для чого вона використовується?
2. Як ви можете імпортувати бібліотеку Pandas у свою програму?
3. Як створити DataFrame у бібліотеці Pandas, і які основні структури даних вона надає?
4. Як зчитати дані з файлу у форматі CSV або Excel у DataFrame за допомогою Pandas?
5. Як вивести перші кілька рядків DataFrame, останні кілька рядків чи випадкові записи?
6. Як виконати фільтрацію, сортування та групування даних у DataFrame за допомогою Pandas?
7. Як виконати об'єднання (merge) та злиття (concatenation) двох DataFrame за певною умовою?
8. Як обробляти відсутні значення (NaN) у DataFrame за допомогою Pandas?
9. Як виконати обчислення агрегованих статистик, таких як середнє значення, медіана, мінімум, максимум тощо, у DataFrame?

10. Як вивести результати аналізу даних за допомогою графіків та діаграм у бібліотеці Pandas?
11. Що таке бібліотека Turtle у Python і для чого вона використовується?
12. Як ви можете імпортувати бібліотеку Turtle у свою програму?
13. Як створити вікно для малювання (скриптове вікно) за допомогою бібліотеки Turtle?
14. Як створити об'єкт черепашки (turtle) та якими командами керувати її рухом?
15. Як намалювати різні фігури, такі як квадрати, кола, трикутники тощо, за допомогою бібліотеки Turtle?
16. Як налаштувати колір, товщину та інші властивості ліній та заливки за допомогою бібліотеки Turtle?
17. Як зберегти малюнок у форматі зображення (наприклад, PNG) за допомогою бібліотеки Turtle?
18. Як використовувати цикли та умовні конструкції для малювання складних малюнків за допомогою бібліотеки Turtle?
19. Як намалювати текст та інші додаткові елементи на малюнку за допомогою бібліотеки Turtle?
20. Як використовувати координати для керування рухом черепашки та малювання у бібліотеці Turtle?
21. Що таке бібліотека Matplotlib у Python і для чого вона використовується?
22. Як ви можете імпортувати бібліотеку Matplotlib у свою програму?
23. Як створити графік ліній у Matplotlib, використовуючи дані зі списків у Python?
24. Як змінити властивості графіка, такі як колір, тип лінії, маркери точок тощо, у Matplotlib?
25. Як створити графік розсіювання (scatter plot) у Matplotlib та відобразити дані на ньому?
26. Як відобразити гістограму (histogram) за допомогою бібліотеки Matplotlib?

27. Як створити діаграму (pie chart) у Matplotlib і відобразити відсоткове співвідношення кожної категорії?
28. Як додати підписи до вісей, заголовки та легенду до графіка у Matplotlib?
29. Як зберегти графік у форматі зображення (наприклад, PNG або JPG) за допомогою бібліотеки Matplotlib?
30. Як використовувати підграфіки (subplots) для відображення декількох графіків на одному полотні у Matplotlib?
31. Що таке бібліотека PuLP у Python і для чого вона використовується?
32. Як ви можете імпортувати бібліотеку PuLP у свою програму?
33. Як створити задачу оптимізації (linear programming problem) за допомогою бібліотеки PuLP?
34. Як додати обмеження до задачі оптимізації у бібліотеці PuLP?
35. Як задати цільову функцію (objective function) для оптимізації у бібліотеці PuLP?
36. Як вирішити задачу оптимізації за допомогою методу оптимізації у бібліотеці PuLP?
37. Як отримати значення змінних, які максимізують або мінімізують цільову функцію у PuLP?
38. Як використовувати різні методи оптимізації, такі як симплекс-метод, симуляційне відпалювання тощо, у бібліотеці PuLP?
39. Як вирішити задачу оптимізації з неперервними або цілочисловими змінними у PuLP?
40. Як обробити неможливість знайти оптимальний розв'язок задачі оптимізації у PuLP?

*Вивчивши матеріали цього розділу студенти опанують методи використання бібліотек Пайтона, які повністю замінюють такі програми як Excel, Statistica та MatLab.*



# Розділ 5.



## ІНДИВІДУАЛЬНІ ЗАВДАННЯ 1

*Це завдання для початкового рівня вивчення мови Пайтон.*

### Методичні вказівки до виконання практичних завдань

Навчальним планом з дисципліни "Мова програмування Python" передбачено виконання практичних робіт.

Перед початком програмування необхідно вивчити відповідний розділ теоретичного матеріалу. При виконанні практичної роботи студент повинний дотримувати таких правил:

1. Готова програма розміщується у папці з вашим прізвищем на «хмарі».
2. Програми мають містити інформацію про автора, номер групи та дату виконання.
3. Кожен скрипт (оператор, команда) необхідно супроводжувати поясненнями у вигляді тексту чи формул.
4. Якщо практична робота після перевірки не зарахована, треба виправити помилки згідно з зауваженнями викладача. Доопрацьована практична робота надається для повторної перевірки разом з першим варіантом.
5. Обов'язкова наявність коментарів з інформацією про розробника та умовами завдання перед написанням коду.
6. Програма повинна містити діалог із оператором, команду `input` у кінці тексту, перевірку на помилкові дії оператора.
6. Студент, що не виконав практичні роботи, до іспиту не допускається.

7. Оцінювання програм виконується за національною шкалою з наступним переведення оцінок у 100-балний формат.

### **Індивідуальне завдання №1. Банківська позика**

Потрібно розрахувати вартість кредиту, тобто треба обчислити, скільки доведеться платити в місяць по позиці та скільки всього віддати грошей банку за весь період.

Місячна виплата по позиці обчислюється за такою формулою:

$$m = (s * p * (1 + p)^n) / (12 * ((1 + p)^n - 1)).$$

де:

$m$  – розмір місячної виплати;

$s$  – сума позики (кредиту) (№ залікової книжки);

$p$  – відсоток банку, виражений в частках одиниці (тобто якщо 20%, то буде 0.2)  
(поточний рік-2000).

$n$  – кількість років, на які береться позика (№ по списку у журналі).

Параметри  $s$ ,  $p$  та  $n$  – вводяться користувачем.

Результат:  $m$  – розмір місячної виплати;  $sum$  – загальна сума за весь період.

Округлити неціле число до двох знаків після коми.

### **Індивідуальне завдання №2.**

#### **Логічні оператори, булевий тип даних**

Напишіть програму, що визначає:

- 1) який з двох введених рядків довший,
- 2) чи введено порожній рядок,
- 3) чи рядки однакові,
- 4) яке з двох введених чисел більше,

5) чи буде від'ємною сума введених чисел,  
за допомогою логічних функцій.

Користувачу подається запит, наприклад:

```
Введіть перший рядок: tu
Введіть другий рядок: kl;
Введіть перше число: 6.8
Введіть друге число: 8
-----
Перший рядок більший за другий: True
Числа не рівні: True
Перший рядок більший другого та числа не рівні: True
Сума чисел більше нуля АБО ні один з рядків не пустий: True
```

В Python є прості логічні оператори ( $=, !=, <, >, <=, >=$ ) і складні (*and, or, not*). Всі логічні оператори, за винятком *not*, є бінарними. Це означає, що зліва і праворуч від них повинні стояти вирази. За допомогою логічних операторів ці вирази так чи інакше порівнюються між собою.

Результат логічних операцій має булевий тип даних (вбудований *class 'bool'* в Python), тобто може приймати лише два значення – “істина” та “неправда”. Потрібно бути обережним, порівнюючи між собою різні типи даних, так як це не завжди можливо. Наприклад, не можна порівнювати числа і рядки, але дробові та цілі числа - можна.

У складних логічних виразах потрібно враховувати послідовність операцій. Якщо немає впевненості, яка операція має пріоритет, то краще використовувати дужки.

### Індивідуальне завдання №3.

#### Конкатенація і повторення рядків

В Python над двома рядками можна виконати операцію, що позначається знаком  $+$ . Однак, на відміну від чисел, виконується не складанням (що для рядків в принципі неможливо), а з'єднання, тобто до кінця першого рядка додається другий. По-іншому операція з'єднання рядків називається *конкатенацією*.

Крім того, в Python є операція повторення (мультиплікації) рядків. Вона позначається знаком \* (також як операція множення для чисел). При повторенні рядки з одного боку від знаку \* ставиться рядок, а з іншого число, що позначає кількість повторів. При цьому не важливо, який об'єкт з якого боку знаходиться (зліва від знаку можна писати число, а праворуч – рядок).

В одному вираженні можна поєднувати операції конкатенації і мультиплікація. При цьому більш високий пріоритет у операції повторення рядка.

Напишіть програму, в якій:

- 1) Користувач вводить два рядка та кількість повторів (Прізвище, Ім'я, По-батькові) та шифр групи,
- 2) Виконується конкатенація рядків (Прізвище, Ім'я, По-батькові+ “студент/ка групи”+шифр групи),
- 3) Пошук підрядка в рядку,
- 4) Повтор об'єднаних рядків задану кількість раз, з нового рядка;
- 5) Надрукувати символ “\*” задану кількість раз у вигляді:

```
*  
  
* *  
  
* * *  
  
* * * *
```

б) Надрукувати ініціали ПІБ, наприклад: Михайло Сергійович Нестеренко

```
М      М      СССС      Н      Н  
М М    М М    С          Н      Н  
М  М    М    С          НННННННН  
М      М      С          Н      Н  
М      М      СССС      Н      Н
```

## Індивідуальне завдання № 4.

### Програма "Вгадай число від 1 до N"

Користувач спочатку вводить число загадує число від 1 до N.

Потрібно його відгадати, задавши якомога менше питань, і обмежитися використання оператора розгалуження та операторів циклу.

Щоб користувачеві задати менше питань, треба "розділити" діапазон чисел на дві по можливості рівні частини і визначити, в якій із них знаходиться шукане число.

## Індивідуальне завдання № 5.

### Робота з файлами

Тема: запис/зчитування даних в/з текстовий файл, \*.csv файл.

- Робота з файлами:
  - 1) Створити файл *Resume.txt*
  - 2) Ввести у файл інформацію про претендента: ПІБ, кваліфікацію, телефон.
  - 3) Закрити файл *Resume.txt*
  - 4) Вивести на екран інформацію з файлу *Resume.txt*
  
- Скласти картотеку з файлів зі структурою:
  - 5) Запросити у користувача ввести шлях до файлу (*user\_url*).
  - 6) Запросити у користувача ввести ПІБ (*user\_name*).
  - 7) Запросити у користувача ввести кваліфікацію.
  - 8) Запросити у користувача ввести бажаний розмір заробітної платні.
  - 9) Запросити у користувача ввести контактний телефон.
  - 10) Створити файл з іменем *user\_url* ■ *user\_name.txt* (Наприклад, *Lab8\_Petrenko.txt*)
  - 11) Записати у файл інформацію про претендента.
  - 12) Закрити файл.
  
- Зчитати/вивести на екран інформацію про потрібного претендента за введеним ім'ям:
  - 13) Запросити з екрану ім'я претендента.



- 14) Вивести про нього інформацію.
- Робота з csv файлами:
  - 15) Зчитати дані з файлу Data.csv
  - 16) Вивести на екран перші 10 записів.

УВАГА! Слідкуйте за режимами доступу до файлу, своєчасно закривайте відкриті файли.

### **Індивідуальне завдання № 6.**

**Визначити індекси елементів масиву (списку), значення яких належать заданому діапазону**

Необхідно визначити індекси елементів списку, значення яких не менше заданого мінімуму і не більше заданого максимуму.

Нехай досліджуваний масив (список в Python) заповнюється випадковими числами в діапазоні від 0 до 99 (включно) і складається з 100 елементів.

Далі мінімум і максимум для пошуку значень задається користувачем.

### **Індивідуальне завдання № 7.**

**Малювання з використанням черепашої графіки**

Тема: Використання бібліотеки turtle, що дозволяє малювати на екрані нескладні малюнки.

Обов'язкова наявність коментарів з умовами завдання перед написанням коду.

- 1) З використанням команд бібліотеки turtle намалювати багатокутник з  $N$  кутами, де  $N$  – число від 1 до 9 (остання цифра номеру за списком).

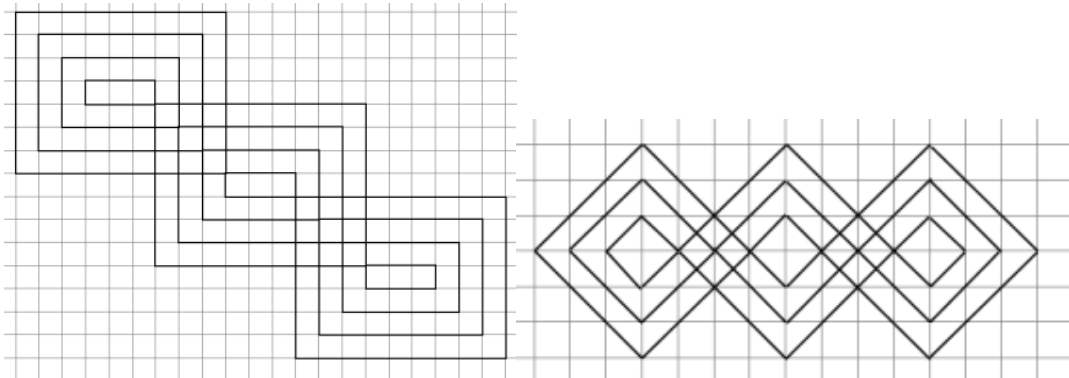
2) намалуйте на екрані рівносторонній трикутник. Намалуйте жовтий рівносторонній трикутник. Намалуйте зафарбований червоний рівносторонній трикутник.

3) намалуйте на екрані квадрат з діагоналями. Намалуйте тільки діагоналі квадрата (два пересічних відрізка). Намалуйте квадрат, сторони якого не паралельні осям координат.

4) намалуйте:

Непарні номери за списком:

Парні номери за списком:



Всі програми, що працюють з черепашкою, повинні починатися з команд

```
import turtle
```

```
turtle.reset ()
```

а закінчуватися рядком

```
turtle._root.mainloop ()
```

### **Індивідуальне завдання № 8.**

#### **Робота з списками в Python**

Масиви в Пітоні називаються списками, тому що вони підтримують ряд додаткових операцій, що не властивих стандартним масивів. Заповніть список

випадковими числами від 0 до  $N$ , де  $N$  – номер за списком у журналі ( $N \geq 5$ ).

Деякі елементи списку можна змінити.

```
import random  
random.random ()
```

Потім:

- 1) Знайдіть найбільший елемент в списку.
- 2) Знайдіть найменший елемент в списку.
- 3) Знайдіть другий за величиною елемент у списку.
- 4) Знайдіть кількість елементів списку, рівних найбільшому.

Створити список студентів групи, відсортувати, додати ще одного студента.

Перетворити в кортеж.

### **Індивідуальне завдання № 9.**

#### **Знаходження оптимального рішення**

І застосування можливостей Python вирішити наступну задачу.

Підприємство випускає три види продукції А, Б і С (табл. 5.1) Для виробництва цієї продукції потрібні такі ресурси, як матеріали, праця робочих та ІТР. Для прийняття рішення оптимального випуску продукції, треба визначити такий план, який максимізує прибуток.

Дані студенти обирають згідно номеру за списком групи з табл. 1.

Таблиця 1

Вхідні данні

Варіант	Види витрат	Продукція			Обмеження за виробничими потужностями
		А	Б	С	
1	Витрати праці робочих на 1 тис. шт., годин	1	2	1	100
	Витрати матеріалів на 1 тис. шт., годин	1	0,4	0,45	75
	Витрати праці ІТР на 1 тис. шт., годин	2	2	7	295
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	95	60	300	
2	Витрати праці робочих на 1 тис. шт., годин	2	1	1	100
	Витрати матеріалів на 1 тис. шт., годин	1	0,35	0,45	75
	Витрати праці ІТР на 1 тис. шт., годин	1	2	7	280
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	95	65	300	
3	Витрати праці робочих на 1 тис. шт., годин	2	1	2	120
	Витрати матеріалів на 1 тис. шт., годин	1	0,4	0,45	75
	Витрати праці ІТР на 1 тис. шт., годин	1	2	6	275
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	90	65	290	
Варіант	Види витрат	Продукція			Обмеження за виробничими потужностями
		А	Б	С	

4	Витрати праці робочих на 1 тис. шт., годин	2	1	2	110
	Витрати матеріалів на 1 тис. шт., годин	1	0,4	0,4	70
	Витрати праці ІТР на 1 тис. шт., годин	1	2	5	260
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	90	65	290	
5	Витрати праці робочих на 1 тис. шт., годин	2	1	2	100
	Витрати матеріалів на 1 тис. шт., годин	2	0,45	0,4	80
	Витрати праці ІТР на 1 тис. шт., годин	2	2	6	260
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	90	65	270	
6	Витрати праці робочих на 1 тис. шт., годин	2	1	2	95
	Витрати матеріалів на 1 тис. шт., годин	2	2	0,5	85
	Витрати праці ІТР на 1 тис. шт., годин	2	2	7	260
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	90	60	265	
7	Витрати праці робочих на 1 тис. шт., годин	1	1	2	95
	Витрати матеріалів на 1 тис. шт., годин	2	3	0,5	120
	Витрати праці ІТР на 1 тис. шт., годин	2	2	7	260
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	90	110	265	
8	Витрати праці робочих на 1 тис. шт., годин	1	1	2	95
	Витрати матеріалів на 1 тис. шт., годин	2	3	0,5	120
	Витрати праці ІТР на 1 тис. шт., годин	3	3	7	310
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	90	125	265	

Варіант	Види витрат	Продукція			Обмеження за виробничими потужностями
		А	Б	С	

9	Витрати праці робочих на 1 тис. шт., годин	1	2	2	105
	Витрати матеріалів на 1 тис. шт., годин	2	3	0,5	120
	Витрати праці ІТР на 1 тис. шт., годин	2	3	6	310
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	190	125	75	
10	Витрати праці робочих на 1 тис. шт., годин	1	2	2	125
	Витрати матеріалів на 1 тис. шт., годин	2	1	1	130
	Витрати праці ІТР на 1 тис. шт., годин	2	1	5	300
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	190	120	170	
11	Витрати праці робочих на 1 тис. шт., годин	3	2	2	120
	Витрати матеріалів на 1 тис. шт., годин	2	3	3	135
	Витрати праці ІТР на 1 тис. шт., годин	2	4	5	300
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	185	125	170	
12	Витрати праці робочих на 1 тис. шт., годин	3	5	4	125
	Витрати матеріалів на 1 тис. шт., годин	4	3	3	130
	Витрати праці ІТР на 1 тис. шт., годин	2	4	5	190
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	180	135	175	
13	Витрати праці робочих на 1 тис. шт., годин	3	5	4	130
	Витрати матеріалів на 1 тис. шт., годин	4	6	7	135
	Витрати праці ІТР на 1 тис. шт., годин	5	4	5	195
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	190	140	180	

Варіант	Види витрат	Продукція			Обмеження за виробничими потужностями
		А	Б	С	
14	Витрати праці робочих на 1 тис. шт., годин	6	5	4	140
	Витрати матеріалів на 1 тис. шт., годин	4	6	7	155
	Витрати праці ІТР на 1 тис. шт., годин	5	7	3	205
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	200	150	210	
15	Витрати праці робочих на 1 тис. шт., годин	6	4	3	145
	Витрати матеріалів на 1 тис. шт., годин	7	6	7	160
	Витрати праці ІТР на 1 тис. шт., годин	5	7	6	210
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	205	155	215	
16	Витрати праці робочих на 1 тис. шт., годин	6	4	4	150
	Витрати матеріалів на 1 тис. шт., годин	7	3	5	165
	Витрати праці ІТР на 1 тис. шт., годин	4	7	6	215
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	225	160	220	
17	Витрати праці робочих на 1 тис. шт., годин	3	4	7	155
	Витрати матеріалів на 1 тис. шт., годин	7	3	5	170
	Витрати праці ІТР на 1 тис. шт., годин	4	5	6	230
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	215	165	210	
18	Витрати праці робочих на 1 тис. шт., годин	2	2	1	90
	Витрати матеріалів на 1 тис. шт., годин	2	3	3	115
	Витрати праці ІТР на 1 тис. шт., годин	1	1	2	300
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	270	120	260	

Варіант	Види витрат	Продукція			Обмеження за виробничими потужностями
		А	Б	С	



19	Витрати праці робочих на 1 тис. шт., годин	2	2	2	100
	Витрати матеріалів на 1 тис. шт., годин	2	3	3	125
	Витрати праці ІТР на 1 тис. шт., годин	3	4	2	250
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	260	180	270	
20	Витрати праці робочих на 1 тис. шт., годин	1	2	2	105
	Витрати матеріалів на 1 тис. шт., годин	2	3	1	135
	Витрати праці ІТР на 1 тис. шт., годин	3	1	2	265
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	255	195	260	
21	Витрати праці робочих на 1 тис. шт., годин	3	2	2	110
	Витрати матеріалів на 1 тис. шт., годин	2	3	1	145
	Витрати праці ІТР на 1 тис. шт., годин	1	1	2	245
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	245	275	255	
22	Витрати праці робочих на 1 тис. шт., годин	3	3	2	115
	Витрати матеріалів на 1 тис. шт., годин	1	2	3	135
	Витрати праці ІТР на 1 тис. шт., годин	2	1	1	180
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	255	285	270	
23	Витрати праці робочих на 1 тис. шт., годин	2	1	2	120
	Витрати матеріалів на 1 тис. шт., годин	1	2	3	130
	Витрати праці ІТР на 1 тис. шт., годин	2	1	1	190
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	245	265	275	

Варіант	Види витрат	Продукція			Обмеження за виробничими потужностями
		А	Б	С	



24	Витрати праці робочих на 1 тис. шт., годин	2	4	1	125
	Витрати матеріалів на 1 тис. шт., годин	4	2	2	135
	Витрати праці ІТР на 1 тис. шт., годин	2	1	3	195
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	255	275	285	
25	Витрати праці робочих на 1 тис. шт., годин	2	4	4	135
	Витрати матеріалів на 1 тис. шт., годин	4	2	2	140
	Витрати праці ІТР на 1 тис. шт., годин	3	3	3	200
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	265	280	285	
26	Витрати праці робочих на 1 тис. шт., годин	5	4	4	155
	Витрати матеріалів на 1 тис. шт., годин	4	5	6	165
	Витрати праці ІТР на 1 тис. шт., годин	3	3	3	210
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	275	290	280	
27	Витрати праці робочих на 1 тис. шт., годин	2	4	4	165
	Витрати матеріалів на 1 тис. шт., годин	4	2	2	170
	Витрати праці ІТР на 1 тис. шт., годин	3	3	3	205
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	255	260	245	
28	Витрати праці робочих на 1 тис. шт., годин	5	4	4	160
	Витрати матеріалів на 1 тис. шт., годин	4	4	5	175
	Витрати праці ІТР на 1 тис. шт., годин	2	3	3	190
	Прибуток на одну тисячу штук кожного виду продукції, тис. грн.	275	250	260	

Рекомендовано використати бібліотеку `ru1p` у такому порядку, як на прикладі:

```

from pulp import # відкриття бібліотеки
import time
start = time.time() # Визначення початку вирішення задачі
problem = pulp.LpProblem('0',pulp.LpMaximize)
# визначення найменувань змінних
x11 = pulp.LpVariable("x11", lowBound=0)
x12 = pulp.LpVariable ("x12", lowBound=0)
x13 = pulp.LpVariable("x13", lowBound=0)
x21 = pulp.LpVariable("x21", lowBound=0)
x22 = pulp.LpVariable("x22", lowBound=0)
x23 = pulp.LpVariable("x23", lowBound=0)
x31 = pulp.LpVariable ("x31", lowBound=0)
x32 = pulp.LpVariable("x32", lowBound=0)
x33 = pulp.LpVariable("x33", lowBound=0)
# скрипт початку максимізації задачі
# цільова функція
problem += - (7*x11 + 3*x12 + 6*x13 + 4*x21 + 8*x22 + 2*x23 + 1*x31 +
5*x32 + 9*x33), '---> MIN'
# обмеження. Після коми – номери обмежень
problem +=x11 + x12 + x13 == 20, '1'
problem +=x21 + x22 + x23 == 45, '2'
problem +=x31 + x32 + x33 == 30, '3'
problem +=x11 + x21 + x31 <= 74, '4'
problem +=x12 + x22 + x32 <= 40, '5'
problem +=x13 + x23 + x33 <= 36, '6'

problem.solve() #

print ("Числові значення змінних")
for variable in problem.variables():

```

```

print (variable.name, "=", variable.varValue)
print ("Оптимальна вартість перевезень:")
print (abs(value(problem.objective)))
stop = time.time() # визначення часу закінчення вирішення задачі
print ("Час на вирішення оптимізаційної задачі", stop - start)

```

## 5.10. Екзаменаційне завдання.

### Прогнозування. Теорія ігор. Нечіткі моделі

Розробити програми що виконують рішення наступних завдань на таких умовах:

1. Згенерувати 100 чисел датчиком випадкових чисел, розподілених за нормальним законом з параметрами  $Mx = N$ ,  $\sigma_x = 0.1N$ . Знайти прогноз на 10 кроків уперед та розрахувати довірчі інтервали для довірчої ймовірності  $p = 0.73 + 0.01N$ , де  $N$  – номер студента за списком групи.
2. Датчиком випадкових чисел, розподілених рівномірно згенерувати 5 двовимірних масивів, які будуть представляти платіжну матрицю кооперативної гри 5 гравців. Діапазон генерації:  $47N - 98N$ . Знайти стратегію Неша для цих гравців.
3. Створити нечітку модель і знайти 3 значення моделі для чітких вхідних параметрів у наступному порядку:
  - a) Студент обирає завдання згідно останньої цифри залікової книжки.
  - b) Самостійно визначає кількість вхідних змінних, але не менше 2-х.
  - c) Вихідна змінна завжди одна.
  - d) Студент самостійно генерує набір нечітких правил у кількості не менше 8.
  - e) Студент створює нечітку модель на мові Пайтон.
  - f) Студент створює таблицю можливих значень для вхідних факторів.

- g) Студент отримує чітке значення вихідного параметру і доповнює ним таблицю вхідних значень.

### *Завдання*

0. Розробити систему нечіткого управління автомобілем.
1. Розробити систему нечіткого керування торгами цінних паперів на біржі.
2. Розробити систему нечіткої зміни цін, при торгівлі в роздріб.
3. Розробити систему нечіткого визначення того, чи є студент успішним.
4. Розробити систему нечіткого визначення якості роботи державної установи.
5. Розробити систему нечіткого виявлення того, що підприємство не сплачує податки.
6. Розробити систему нечіткого керівництва торговою фірмою.
7. Розробити систему нечіткого керівництва промисловим підприємством.
8. Розробити систему нечіткого управління персоналом.
9. Розробити систему нечіткого визначення здоров'я людини.

*Виконавши всі завдання студент отримає досвід першого рівня програмування на мові Пайтон.*

# Розділ 6.

## ІНДИВІДУАЛЬНІ ЗАВДАННЯ 2

*Це завдання для поглибленого рівня вивчення мови Пайтон.*

### Методичні вказівки до виконання практичних завдань

Навчальним планом з дисципліни "Мова програмування Python" передбачено виконання практичних робіт.

Перед початком програмування необхідно вивчити відповідний розділ теоретичного матеріалу. При виконанні практичної роботи студент повинний дотримувати таких правил:

1. Готова програма розміщується у папці з вашим прізвищем на «хмарі».
2. Програми мають містити інформацію про автора, номер групи та дату виконання.
3. Кожен скрипт (оператор, команда) необхідно супроводжувати поясненнями у вигляді тексту чи формул.
4. Якщо практична робота після перевірки не зарахована, треба виправити помилки згідно з зауваженнями викладача. Доопрацьована практична робота надається для повторної перевірки разом з першим варіантом.
5. Обов'язкова наявність коментарів з інформацією про розробника та умовами завдання перед написанням коду.
6. Програма повинна містити діалог із оператором, команду input у кінці тексту, перевірку на помилкові дії оператора.
6. Студент, що не виконав практичні роботи, до іспиту не допускається.
7. Оцінювання програм виконується за національною шкалою з наступним переведенням оцінок у 100-балний формат.

## 6.1. Індивідуальне завдання №1.

### Створення GUI додатку

Тема: Створити програмний додаток для перевірки валідності номеру кредитної/платіжної картки.

Обов'язкова наявність коментарів з умовами завдання перед написанням коду.

1) Тип проекту – Windows Form.

Користувач має ввести 16 цифр номеру картки, за натисканням кнопки «Check» має бути видано повідомлення або «The number 0123456789101112 is correct!!!» або «The number 0123456789101112 is NOT correct!!!». Приклад інтерфейсу програмного додатку наведено на рис. 1.

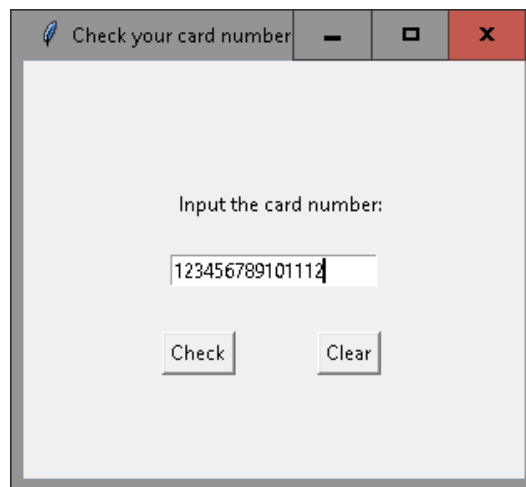


Рис. 1 Головна форма

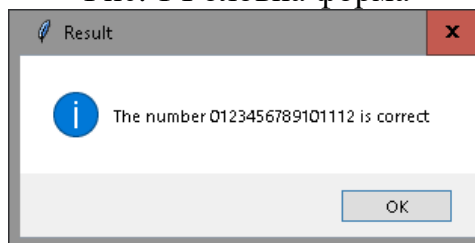


Рис. 2 Результат роботи програми

Перевірка введеного номера здійснюється за алгоритмом Луна ([https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC\\_%D0%9B%D1%83%D0%BD%D0%B0](https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9B%D1%83%D0%BD%D0%B0)).

Також, визначити платіжну систему:

VISA – 4

American Express – 3

MasterCard – 5

Maestro - 3 або 6

JCB International - 3

China UnionPay – 6

<https://metanit.com/python/tutorial/9.2.php>

2) Тип проекту – Windows Form

Запуск виконання завдань здійснювати при натисканні на відповідні кнопки.

Уважно тестувати задачі.

Завдання для парних номерів у списку групи в проекті з іменем ElectroenergiyaВашеПрізвище

Завдання для непарних номерів у списку групи виконати у проекті з іменем ConverterВашеПрізвище,

Практична робота. **Конвертер**

Постановка задачі. Програма **Конвертер** перераховує ціну з доларів в гривні. Демонструє використання компонентів TextBox и Label для введення і відображення числових даних. Форму програми приведено на рис. 1

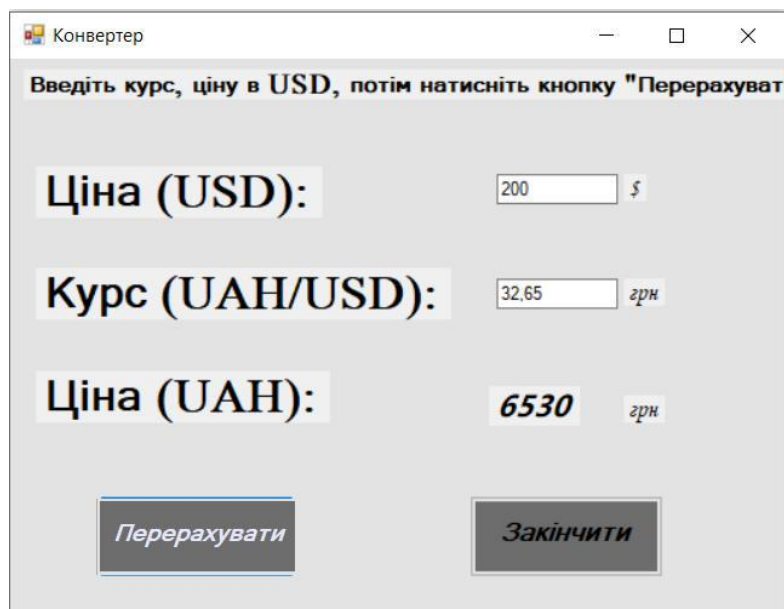


Рис.1. Інтерфейс додатку Конвертер

Програма **Електроенергія** визначає суму, яку потрібно заплатити користування електроенергією виходячи з показань лічильника. Форму програми наведено на рис. 2

Показники лічильника	
Попередні:	1100
Поточні:	1500
Ціна:	7,31 грн/кВт
До сплати:	2924 грн

Buttons: **Розрахувати**, Закрити

Рис.2. Інтерфейс додатку Електроенергія

## 6.2. Індивідуальне завдання №2.

### Робота з комплексом Pyton в Інтернеті

Порядок виконання:

Із сайту Державного управління статистики <http://www.ukrstat.gov.ua/> за допомогою команд Pyton:

1. Вибрати статистичну інформацію згідно номеру студента за списком групи за якнайбільший період. Конкретну таблицю в групі даних обирати довільно.

2. Зберегти цю інформацію на диску у файлі Excel.



3. Розрахувати середнє, дисперсію та стандарт для кожної колонки таблиці.

4. Розрахувати кореляційну матрицю зв'язку усіх колонок таблиці.

5. Представити кореляційну матрицю у вигляді графіку з точками.

№ за списком групи	Статистична інформація
1.	Ринок праці
2.	Освіта
3.	Охорона здоров'я
4.	Доходи та умови життя
5.	Соціальний захист
6.	Населені пункти та житло
7.	Правосуддя та злочинність
8.	Культура
9.	Макроекономічна статистика
10.	Національні рахунки
11.	Економічна діяльність
12.	Діяльність підприємств
13.	Внутрішня торгівля
14.	Капітальні інвестиції
15.	Основні засоби
16.	Сільське, лісове та рибне господарство
17.	Енергетика
18.	Промисловість
19.	Будівництво
20.	Транспорт
21.	Туризм
22.	Реєстр статистичних одиниць
23.	Навколишнє середовище
24.	Державні фінанси, податки та публічний сектор
25.	Зовнішньоекономічна діяльність
26.	Ціни
27.	Наука, технології та інновації
28.	Інформаційне суспільство

### 6.3. Індивідуальне завдання №3.

#### Робота з файлами типу \*CSV

Для виконання роботи цієї частини візьміть файл DATA.CSV, що додається за посиланням [https://myngu-my.sharepoint.com/:x:/g/personal/pistunov\\_i\\_m\\_nmu\\_one/ERey9QLGA3NFpSXavbKQeQgBMnt0SdEWSAqCaC28nnaQGw?e=jpPyGM](https://myngu-my.sharepoint.com/:x:/g/personal/pistunov_i_m_nmu_one/ERey9QLGA3NFpSXavbKQeQgBMnt0SdEWSAqCaC28nnaQGw?e=jpPyGM)

Завдання1. 1.1.Для кожної колонки введених даних файлу Data.csv, розрахувати наведені нижче статистичні параметри.

1.2. Побудувати кореляційну матрицю для всіх даних, включно з датою.

1.3. Переглянути бібліотеки pandas, statistics, statmodels, matplotlibs для визначення найзручнішого способи виконати це завдання.

1.4. Результати розрахунків вивести у файл Data так, щоб не затирати таблиці з початковими даними.

Приклад коду зчитування файлу

```
import csv
with open("probniy.csv")as f:
    reader=csv.reader(f)
    for row in reader:
        print(row)
або
import pandas as pd
# Завантажуємо набір
matx = pd.read_csv("Data.csv")
```

Англійська	Українська
Mean	Середнє
Standard Error	Стандартна помилка
Median	Медіана
Mode	Мода
Standard Deviation	Стандартне відхилення
Sample Variance	Дисперсія вибірки
Kurtosis	Екцес
Skewness	Асиметрія
Range	Розмах вибірки
Minimum	Мінімум
Maximum	Максимум
Sum	Сума
Count	Розмір вибірки

	X1	LnX1	1/X1	X2	LnX2
X1	1				
LnX1	0,99715	1			
1/X1	-0,98902	-0,99733	1		
	0,08785	0,10519			
X2	1	3-0,11991		1	
	0,08397			0,99631	
LnX2	1	0,09953	-0,11266	4	1

Всі інші розрахунки здійснюються за допомогою бібліотеки  
`import numpy as np`

#### 6.4. Індивідуальне завдання №4.

##### Побудова графіків різних типів

##### Порядок виконання:

1. Необхідно зареєструватися на сайті <https://campus.datacamp.com>
2. Обираємо курс **Intermediate Python** за посиланням нижче:

<https://campus.datacamp.com/courses/intermediate-python/matplotlib?ex=1>

Необхідно прослухати першу лекцію та виконати завдання до неї. Для використання функцій для побудови графіків необхідно імпортувати бібліотеку `matplotlib.pyplot`.

Подальша робота вимагає качати з названого у відео сайту інформацію про ВВП країни та термін життя у цій країні.

На відео ви вже бачили, наскільки за останні роки зросло населення світу. Чи буде тенденція продовжуватись? Світовий банк має оцінки населення світу за 1950 - 2100 роки. Роки завантажуються у вашому робочому просторі як список (*list*), що називається *year*, а відповідне населення - як список, що називається *pop*.

Цей курс стосується багатьох концепцій, які ви, можливо, забули, тому, якщо вам коли -небудь знадобиться швидке оновлення, завантажте шпаргалку [Python for data data](#) та тримайте її під рукою!

#### Завдання 2:

1) Виведіть на екран `print()` останній елемент із *year* та *pop* -списків, щоб побачити прогнозовану кількість населення на 2010 рік. Використовуйте дві функції `print()`.

2) Перш ніж почати, вам слід імпортувати `matplotlib.pyplot` як `plt`. `pyplot`-це підпакек `matplotlib`.

3) Використовуйте `plt.plot ()` для побудови лінії. *Year* слід відобразити по горизонтальній осі, *pop* - по вертикальній осі. Не забудьте закінчити з функцією `show()`, щоб фактично відобразити графік.

Подивіться на побудований графік, створений у попередній вправі. Виходячи з графіку, приблизно в якому році на цій планеті буде більше десяти мільярдів людських істот?

Подальші завдання виконуються в одній програмі.

#### Лінійні графіки

#### Завдання 3 Line plot (3):

Дані зібрані у 2007 році. Для вас доступні два списки:

*life\_exp*, який містить тривалість життя для кожної країни та *gdp\_cap*, що містить ВВП на душу населення (тобто на особу) для кожної країни, виражений у доларах США.

ВВП означає Валовий внутрішній продукт. В основному це розмір економіки країни поділений на кількість населення, і ви отримаєте ВВП на душу населення.

Роздрукуйте останній елемент як зі списку *gdp\_cap*, так і зі списку *life\_exp* - інформація по Зімбабве.

Побудуйте лінійну діаграму з *gdp\_cap* на осі *x* та *life\_exp* по осі *y*. Чи має сенс представляти ці дані на лінійному графіку?

Не забудьте закінчити командою *plt.show()*, щоб фактично відобразити графік.

#### Завдання 4 Scatter Plot (1)

Коли у вас є шкала часу уздовж горизонтальної осі, доцільно використовувати лінійний графік. Але в багатьох інших випадках, коли ви намагаєтесь оцінити, чи існує кореляція між двома змінними, наприклад, діаграма розсіювання є кращим вибором. Нижче наведено приклад того, як побудувати діаграму розсіювання.

```
import matplotlib.pyplot as plt
plt.scatter(x,y)
plt.show()
```

Давайте побудуємо графік *gdp\_cap* від *life\_exp*, дані про ВВП та тривалість життя для різних країн у 2007 році.

Змініть вид графіку у попередньому завданні на діаграму розсіювання.

Кореляція стане зрозумілою, якщо відобразити ВВП на душу населення в логарифмічній шкалі. Додайте рядок *plt.xscale('log')*. Завершіть свій скрипт за допомогою *plt.show()*, щоб відобразити графік.

#### Завдання 5 Scatter plot (2)

У попередньому завданні ви побачили, що вищий ВВП зазвичай відповідає більшій тривалості життя. Іншими словами, існує позитивна кореляція. Як ви

вважаєте, чи існує зв'язок між населенням та тривалістю життя країни? Список *life\_exp* з попередньої вправи вже доступний. Крім того, зараз також доступна *pop* - список з переліком відповідних груп населення для країн у 2007 році. Населення представлено в мільйонах осіб.

Почніть: імпортуйте `matplotlib.pyplot` як `plt`.

Побудуйте діаграму розсіювання, де *pop* відображається на горизонтальній осі, а *life\_exp* - на вертикальній осі.

Завершіть сценарій за допомогою `plt.show()`, щоб фактично відобразити графік. Чи існує кореляція?

### Гістограми

Теоретичний матеріал за посиланням:

<https://campus.datacamp.com/courses/intermediate-python/matplotlib?ex=7>

#### Завдання 1 Build a histogram (1)

*life\_exp* - список, що містить дані про тривалість життя для різних країн у 2007 році, доступний у вашій оболонці Python.

Щоб побачити, як розподіляється тривалість життя в різних країнах, створимо гістограму *life\_exp*.

`matplotlib.pyplot` уже доступний як `plt`.

Використовуйте `plt.hist()` для створення гістограми значень з *life\_exp*. Не вказуйте кількість інтервалів; за замовчуванням Python встановить для вас кількість інтервалів = 10.

Додайте `plt.show()`, щоб фактично відобразити гістограму. Чи можете ви сказати, який інтервал містить найбільше спостережень?

#### Завдання 2 Build a histogram (2): bins

У попередній вправі ви не вказали кількість інтервалів. За замовчуванням у цьому випадку Python встановлює кількість інтервалів = 10. Кількість інтервалів є досить важливою. Занадто мало інтервалів занадто спростить реальність і не покаже вам подробиць. Занадто багато інтервалів ускладнить дійсність і не покаже загальної картини.

Щоб контролювати кількість інтервалів для поділу даних, ви можете встановити аргумент *bins*.

Це саме те, що необхідно виконати в цій вправі. Ви будете робити тут два графіка. Код у скрипті вже містить виклики *plt.show()* та *plt.clf()*; *plt.show()* відображає графік; *plt.clf()* очищає, щоб ви могли розпочати заново.

Як і раніше, *life\_exp* доступний, а *matplotlib.pyplot* імпортовано як *plt*.

Побудуйте гістограму *life\_exp* з 5 інтервалами. Чи можете ви сказати, який інтервал містить найбільше спостережень?

Створіть ще одну гістограму *life\_exp*, цього разу з 20 інтервалами. На якій гістограмі представлено дані краще?

### Завдання 3 Build a histogram (3): compare

На відео ви побачили гістограми населення на сьогоднішній день і на майбутнє. Оскільки ми використовували гістограму, було дуже легко провести порівняння.

Давайте зробимо подібне порівняння. Список *life\_exp* містить дані про тривалість життя для різних країн у 2007 році. Ви також маєте доступ до другого списку, *life\_exp1950*, який містить подібні дані за 1950 рік. Чи можете ви побудувати гістограми для обох наборів даних?

Необхідно побудувати 2 графіка. Команди *plt.show()* та *plt.clf()* для гарного рендерингу вже включені. Також для вас імпортується *matplotlib.pyplot*, як *plt*.

Побудуйте гістограму за даними *life\_exp* з 15 інтервалами.

Побудуйте гістограму за даними *life\_exp1950*, також із 15 інтервалами. Чи є велика відмінність між гістограми для даних 2007 і 1950 року?

## **6.5. Індивідуальне завдання №5.**

### **Побудова графіків зі складною формою представлення**

#### Завдання 1 Мітки

Настав час налаштувати власний графік. Це найцікавіша частина, ви побачите, як ваш графік оживає!

Ви збираєтесь працювати над діаграмою розсіювання з даними світового розвитку: ВВП на душу населення на осі *x* (логарифмічна шкала), тривалість життя на осі *y*. Код для побудови цього графіку доступний у скрипті.

```
# Basic scatter plot, log scale
plt.scatter(gdp_cap, life_exp)
plt.xscale('log')

# Strings
xlab = 'GDP per Capita [in USD]'
ylab = 'Life Expectancy [in years]'
title = 'World Development in 2007'

# Add axis labels

# Add title

# After customizing, display the plot
```

На першому кроці, додамо мітки осі та заголовок до графіку. Ви можете зробити це за допомогою функцій *xlabel()*, *ylabel()* та *title()*, доступних у *matplotlib.pyplot*. Цей підпакек уже імпортовано як *plt*.

Рядки *xlab* і *ylab* уже встановлені для вас. Використовуйте ці змінні, щоб встановити мітку осі *x* і *y*.

Назва рядка також кодується для вас. Використовуйте його, щоб додати назву графіку.

Після цих налаштувань завершіть сценарій за допомогою *plt.show()*, щоб фактично відобразити графік.



## Завдання 2 Ticks

Встановлювати у-позначки можна, вказавши два аргументи:

```
plt.yticks ([0,1,2], ["один", "два", "три"])
```

У цьому прикладі позначки, що відповідають числам 0, 1 і 2, будуть замінені на слова «один», «два» та «три» відповідно.

Давайте зробимо подібне для осі *x* вашої світової діаграми розвитку з функцією *xticks()*. Значення позначок 1000, 10000 та 100000 слід замінити на 1k, 10k та 100k. З цією метою для вас уже створено два списки: *tick\_val* та *tick\_lab*.

Використовуйте *tick\_val* і *tick\_lab* як вхідні дані для функції *xticks()*, щоб зробити графік більш читабельним.

Як завжди, після додавання налаштувань відобразіть сюжет за допомогою *plt.show()*.

## Завдання 3 Sizes

Наразі діаграма розсіювання - це просто набір синіх точок, які не відрізняються одна від одної. Давайте змінимо це. Чи не було б добре, якби розмір крапок відповідав даним?

Для цього у вашій робочій області завантажуються список *pop*. Він містить кількість населення кожної країни, виражену в мільйонах. Ви можете бачити, що цей список додається до методу розсіювання, як аргумент *s*, для розміру.

Запустіть скрипт, щоб побачити, як змінюється графік.

Виглядає добре, бо збільшення розміру бульбашок зробить дані виразнішими.

Імпортуйте пакет *numpy* як *np*.

Використовуйте *np.array()*, щоб створити масив *numpy* зі спливаючого списку. Викличте цей масив *Numpy np\_pop*.

Подвойте значення в *np\_pop*, встановивши значення *np\_pop*, що дорівнює *np\_pop \* 2*. Оскільки *np\_pop* - це масив *Numpy*, кожен елемент масиву буде подвоєний.

Змініть аргумент *s* всередині *plt.scatter ()* на *np\_pop* замість *pop*.

#### Завдання 4 Colors

Наступний крок - зробити сюжет більш барвистим! Для цього для вас створено список *col*. Це список із кольором для кожної відповідної країни, залежно від континенту, до складу якого входить країна.

Побудовано словник, який відображає континенти на кольори:

```
dict = {  
    'Asia':'red',  
    'Europe':'green',  
    'Africa':'blue',  
    'Americas':'yellow',  
    'Oceania':'black'  
}
```

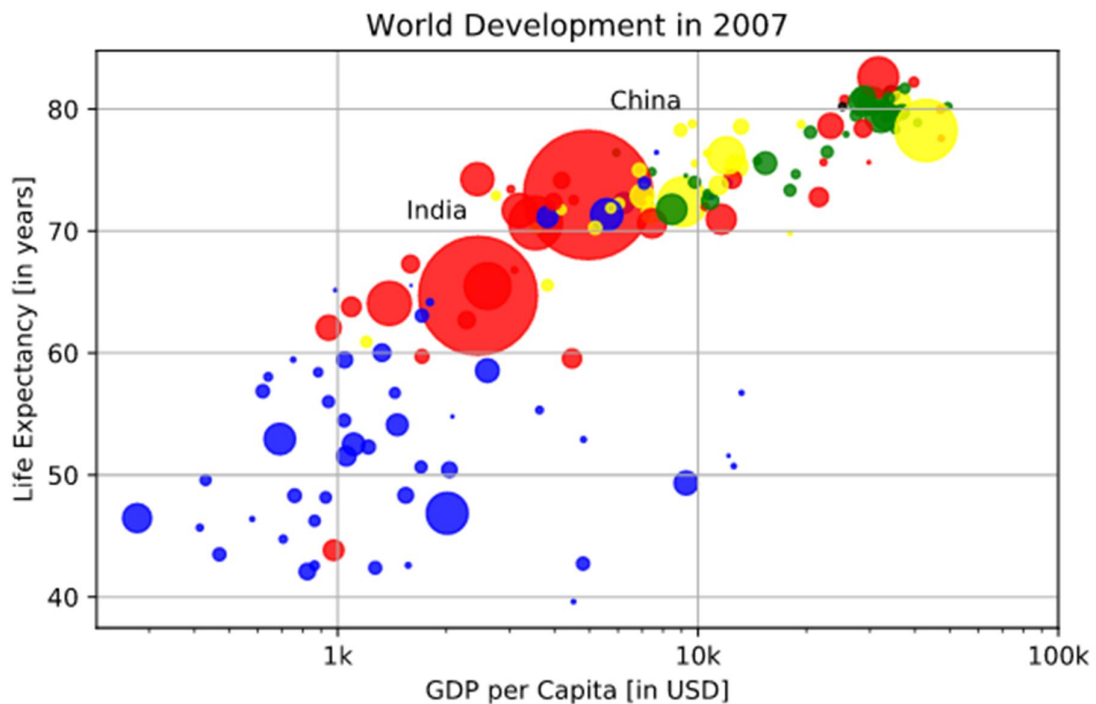
Додайте  $c = col$  до аргументів функції *plt.scatter()*.

Змініть непрозорість бульбашок, встановивши для *alpha* -аргументу значення 0,8 усередині *plt.scatter()*. *Alpha* може бути встановлена від нуля до одиниці, де нуль повністю прозорий, а одиниця зовсім не прозора.

#### Завдання 5 Additional Customizations

Виведіть підписи найбільших даних за допомогою *plt.text()*. Додайте *plt.grid(True)* після *plt.text()*, щоб вивести написи India та China на графік.

```
_plt.text(1550, 71, 'India')  
plt.text(5700, 80, 'China')
```



## 6.6. Індивідуальне завдання №6.

### Побудова графіків різних типів для даних української економіки

У цій роботі студенти повторюють індивідуальні завдання 4 та 5, але для українських даних, взятих з Інтернету у такому порядку:

1. Вибрати статистичну інформацію згідно номеру студента за списком групи за якнайбільший період. Конкретну таблицю в групі даних обирати довільно.

2. Ввести дані з сайту в програму та зберегти цю інформацію на диску у файлі Excel. Наприклад, застосовуючи таку групу команд:

```
import requests
import xlrd
import numpy as np
from statistics import mean
```

```

import matplotlib.pyplot as plt
#ссылка на таблицю на сайті
url1 = 'https://ukrstat.gov.ua/operativ/operativ2020/zf/zf_reg/zf_reg19_eu.xls'
#скачати файл за посиланням
def download(url=""):
    try:
        res = requests.get(url=url1)
        with open('tab.xls', 'wb') as file:
            file.write(res.content)
        return "завантажено"
    except Exception as _ex:
        return "напевно ви ввели ссылку з похибкою"

```

№ за списком групи	Статистична інформація
29.	Ринок праці
30.	Освіта
31.	Охорона здоров'я
32.	Доходи та умови життя
33.	Соціальний захист
34.	Населені пункти та житло
35.	Правосуддя та злочинність
36.	Культура
37.	Макроекономічна статистика
38.	Національні рахунки
39.	Економічна діяльність
40.	Діяльність підприємств
41.	Внутрішня торгівля
42.	Капітальні інвестиції
43.	Основні засоби
44.	Сільське, лісове та рибне господарство
45.	Енергетика
46.	Промисловість
47.	Будівництво
48.	Транспорт
49.	Туризм

№ за списком групи	Статистична інформація
50.	Реєстр статистичних одиниць
51.	Навколишнє середовище
52.	Державні фінанси, податки та публічний сектор
53.	Зовнішньоекономічна діяльність
54.	Ціни
55.	Наука, технології та інновації
56.	Інформаційне суспільство

В якості параметрів терміну життя та ВВП, у третьому та четвертому кварталах.

*Виконавши всі завдання студент отримає досвід поглибленого рівня програмування на мові Пайтон.*

# ВИСНОВКИ

Ось і закінчився курс мови Пайтон для економістів. Чи всі можливості Пайтон описані у посібнику? Ні. Економісти отримали мінімальний рівень знань для ведення розрахунків у напрямку прогнозування, оптимізації, економіко-математичного моделювання, нейронних сіток, нечітких моделей, побудови графіків та графічних фігур, створенню окремих програм, незалежних від інтерпретатора, умінню виконувати операції введення-виведення для текстових файлів та файлів Excel, операції зчитування даних зі сторінок в Інтернеті.

Цей обсяг знань складає малу частину можливостей Пайтона, оскільки завдяки відкритому коду цієї мови над розробкою нових і нових бібліотек працюють тисячі висококваліфікованих програмістів.

Чи стане економіст більш успішним, якщо вивчить Пайтон? Так стане!

Вивчення програмування, зокрема мови програмування Python, може стати цінним додатковим інструментом для економіста і принести значну вигоду у його професійній діяльності. В сучасному світі, де дані відіграють ключову роль у прийнятті рішень, вміння обробляти, аналізувати та візуалізувати дані стає дедалі важливішим для економістів.

Python є однією з найпопулярніших мов програмування у світі, особливо у сферах аналізу даних та наукових досліджень. Вивчення Python дозволяє економістам ефективно обробляти великі обсяги даних, виконувати складний аналіз, будувати прогнози та моделі, а також створювати інтерактивні візуалізації, які допомагають зрозуміти складні економічні залежності.

Одним з ключових аспектів вивчення Python для економіста є можливість автоматизації рутинних завдань. Наприклад, скрипти Python можуть бути використані для автоматичного збору даних з інтернету, обробки і

структурування інформації з різних джерел, що значно заощаджує час і спрощує робочий процес економіста.

Крім того, вивчення Python дозволяє економістам розвивати навички машинного навчання та аналітики, що є дуже цінним у сучасному світі. За допомогою алгоритмів машинного навчання можна робити прогнози економічних показників, виявляти тенденції та закономірності в даних, а також приймати кращі рішення на основі аналізу інформації.

Отже, вивчення Python може значно збільшити успішність економіста, надаючи йому інструменти для ефективної роботи з даними, автоматизації рутинних завдань, аналізу складних економічних явищ та прийняття кращих рішень на основі об'єктивних даних. Інтеграція програмування в їхню професійну діяльність може стати важливим конкурентним перевагою в сучасному світі.

До того ж, ця успішність буде супроводжуватися економією коштів на придбання таких необхідних для сучасного економіста програм як Statistica та MatLab. Тому що всі можливості бібліотек Пайтона надаються на безоплатній основі і до того ж, не мають обмежень по розміру масивів, які необхідно обробити. Тобто, економісти можуть обробляти, робити висновки та приймати рішення за даними, зібраними з усієї діяльності людей на нашій планеті.

# СПИСОК

## ВИКОРИСТАНИХ ДЖЕРЕЛ

1. <https://codeguida.com/post/140>
2. <https://docs.python.org/uk/3/tutorial/index.html>
3. <https://pythonguide.rozh2sch.org.ua/>
4. <https://ela.kpi.ua/bitstream/123456789/25111/1/Python.pdf>
5. <https://w3schoolsua.github.io/python/index.html#gsc.tab=0>
6. Пістунов І.М. Електронна економіка. Том 1. Криптовалюта. Big Data [Електронний ресурс]: Навч. посібник/ І.М. Пістунов, О.П. Антонюк – / М-во освіти і науки України; Нац. Гірн. ун-т. – Д.: НГУ, 2017. – 133 с. Режим доступу: [http://pistunovi.inf.ua/EE\\_KC\\_BD.pdf](http://pistunovi.inf.ua/EE_KC_BD.pdf) (дата звернення: 17.12.2017). – Назва з екрана. Виписка з протоколу №20 Вченої Ради ДВНЗ "НГУ" [http://pistunovi.inf.ua/EE\\_KC\\_BD.pdf](http://pistunovi.inf.ua/EE_KC_BD.pdf)
7. Програмування числових методів мовою Python : підруч. / А. В. Анісімов, А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий ; за ред. А. В. Анісімова : Видавничо-поліграфічний центр "Київський університет" – К, 2014. – 640 с.
8. Пістунов І.М. Нейромережеві технології економіці та фінансах з розрахунками на комп'ютері [Електронний ресурс]: навч. посібн. / І.М. Пістунов, О.П. Антонюк ; Нац. гірн. ун-т. – Електрон. текст. дані. – Д. : НГУ, 2014. – 125 с. Сертифікат ДВНЗ "НГУ" . [http://pistunovi.inf.ua/N\\_M.pdf](http://pistunovi.inf.ua/N_M.pdf)
9. [Python Tkinter - Scale Widget - GeeksforGeeks](https://www.geeksforgeeks.org/python-tkinter-scale-widget/)  
<https://www.geeksforgeeks.org/python-tkinter-scale-widget/>
10. [Using the Tkinter Scale Widget - AskPython](https://www.askpython.com/python-modules/tkinter/tkinter-scale-widget)  
<https://www.askpython.com/python-modules/tkinter/tkinter-scale-widget>



11. Пістунов І.М. Економічна кібернетика [Електронний ресурс]: навч. посіб. / І.М. Пістунов ; Нац. гірн. ун-т. – Електрон. текст. дані. – Видання друге, виправлене й доповнене. – Д. : НГУ, 2014. – 215 с.  
[http://pistunovi.inf.ua/E\\_K.pdf](http://pistunovi.inf.ua/E_K.pdf)
12. Пістунов І.М., Приходченко О.Ю. Теорія ймовірності та математична статистика для економістів: навч. наоч. посіб. Дніпро : НТУ «ДП», 2023. 48с. [http://pistunovi.inf.ua/ТИ\\_ma\\_MC.pptx](http://pistunovi.inf.ua/ТИ_ma_MC.pptx)
13. Глумачний словник зі спеціальності 051 Економіка/ упоряд.: І.М. Пістунов, Д.В. Кабаченко, О.Ю. Приходченко, М.А. Демиденко, О.Ю. Чуріканова, І.Ю. Турчанінова, О.П. Денисенко/під ред. І.М. Пістунова. Дніпро: НТУ «ДП», 2024. 202 с. [http://pistunovi.inf.ua/TS\\_051.pdf](http://pistunovi.inf.ua/TS_051.pdf)
14. Пістунов І.М., Приходченко О.Ю. Економетрика. З розрахунками на Excel: Навч. Посібн. Дніпро: НТУ «ДП», 2024. 221 с.  
<http://pistunovi.inf.ua/ЕkMe.pdf>
15. Пістунов І.М. Штучний інтелект. Прикладні аспекти: посібник. Дніпро: НТУ «ДП», 2025. 246 с. [http://pistunovi.inf.ua/AI\\_AA.pdf](http://pistunovi.inf.ua/AI_AA.pdf)

Навчальне видання

**Пістунів** Ігор Миколайович

# ПАЙТОН ДЛЯ ЕКОНОМІСТІВ

Навчальний посібник

Електронне видання

У редакції автора

Підготовлено у НТУ «Дніпровська політехніка».  
Свідоцтво про внесення до державного реєстру ДК №1842.  
49005, м. Дніпро, просп. Д. Яворницького, 19.