Посібник з програмування на Python

- Розділ 1. Введення в Python
  - Мова програмування Python
  - Встановлення та перша програма на Windows
  - Встановлення та перша програма на Mac OS
  - Встановлення та перша програма на Linux
  - Керування версіями Python на Windows, MacOS та Linux
  - Перша програма в PyCharm
  - Python у Visual Studio
- Розділ 2. Основи Python
  - Введення у написання програм
  - Змінні та типи даних
  - Консольне введення та виведення
  - Арифметичні операції з числами
  - Порозрядні операції з числами
  - Умовні вирази
  - Умовна конструкція if
  - Цикли
  - Функции
  - Параметры функции
  - Оператор return та повернення результату з функції
  - Функція як тип, параметр та результат іншої функції
  - Лямбда-вирази
  - Перетворення типів
  - Область видимості змінних
  - Замикання
  - Декоратори
- Розділ 3. Об'єктно-орієнтоване програмування
  - Класи та об'єкти
  - Інкапсуляція, атрибути та властивості
  - успадкування
  - Перевизначення функціоналу базового класу
  - Атрибути класів та статичні методи
  - Клас об'єкта. Строкове представлення об'єкта
  - Перевантаження операторів
  - Абстрактные классы и методы
- Розділ 4. Обробка помилок та винятків

    - Конструкція try...except...finally
    - except та обробка різних типів винятків
    - Генерація винятків та створення своїх типів винятків
- Розділ 5. Списки, кортежі та словники
    - Список
    - Кортежі
    - Діапазони
    - Словники
    - Безліч
    - List comprehension
    - Упаковка та розпакування
    - Упаковка та розпакування у параметрах функцій
- Розділ 6. Модулі
    - Визначення та підключення модулів
    - Генерація байткоду модулів
    - Модуль random
    - Математичні функції та модуль math
    - Модуль locale
    - Модуль decimal
    - Модуль dataclass. Data-класи
- Розділ 7. Рядки
    - Робота з рядками
    - Основні методи рядків
    - Форматування
- Розділ 8. Pattern matching
    - Конструкція match
    - Кортежі в pattern matching
    - Массивы в pattern matching
    - Словники в pattern matching
    - Класи в pattern matching
    - guards або обмеження шаблонів
    - Установка псевдонимов и паттерн AS
- Розділ 9. Робота з файлами
    - Відкриття та закриття файлів
    - Текстові файли
    - Файли CSV
    - Бинарные файлы
    - Модуль shelve
    - Модуль OS та робота з файловою системою

    - Програма підрахунку слів
    - Запис та читання архівних zip-файлів
- Розділ 10. Робота з датами та часом
    - Модуль datetime
    - Операции с датами

Розділ 1. Введення в Python

Мова програмування Python

Python представляє популярну високорівневу мову програмування, яка призначена для створення додатків різних типів. Це і веб-програми, і ігри, і настільні програми, і робота з базами даних. Досить велике поширення пітон отримав у галузі машинного навчання та досліджень штучного інтелекту.

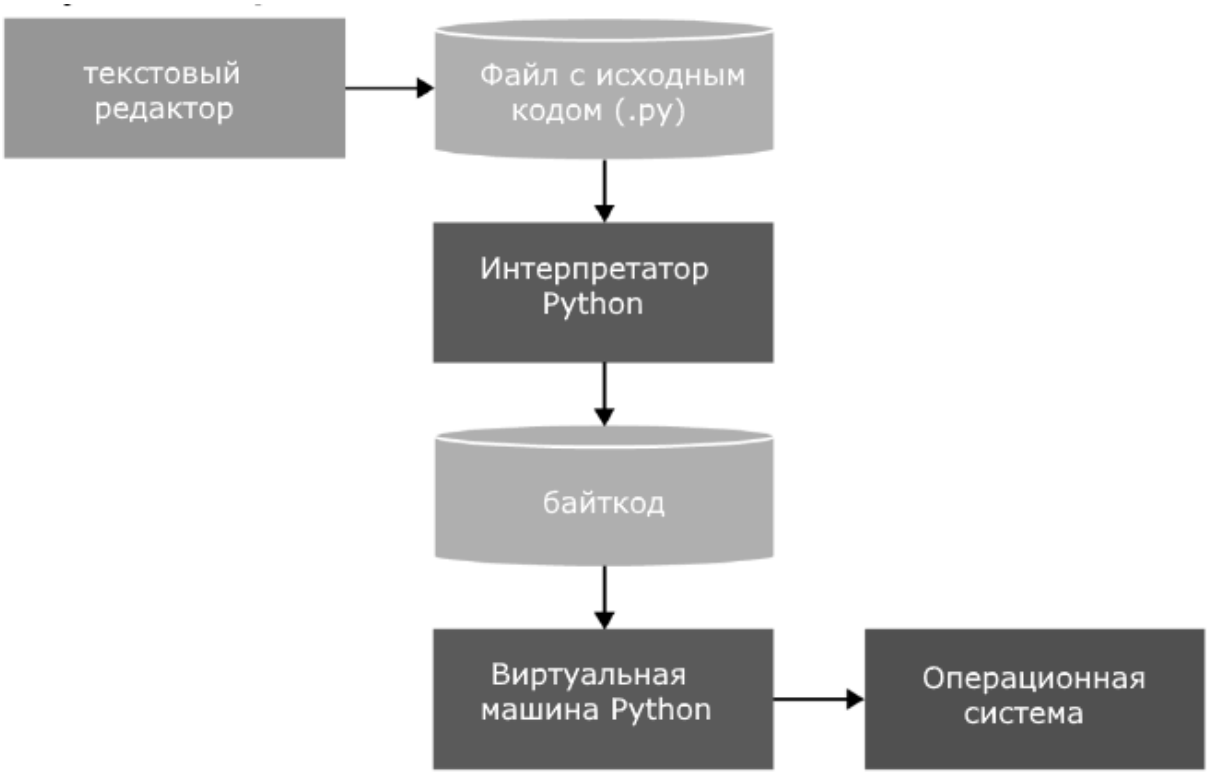
Вперше мова Python була анонсована 1991 року голландським розробником Гвідо Ван Россумом. З того часу ця мова пройшла великий шлях розвитку. У 2000 році було видано версію 2.0, а в 2008 році - версію 3.0. Незважаючи на такі великі проміжки між версіями постійно виходять підверсії. Так, поточною актуальною версією на момент написання цього матеріалу є 3.13 , яка вийшла у жовтні 2024 року.

Основні особливості мови програмування Python:

- Скриптова мова. Код програм визначається як скриптів.
- Підтримка різних парадигм програмування, у тому числі об'єктно-орієнтованої та функціональної парадигм.
- Інтерпретація програм. Для роботи зі скриптами необхідний інтерпретатор, який запускає та виконує скрипт.

  Виконання програми на Python виглядає так. Спочатку ми пишемо в текстовому редакторі скрипт з набором виразів цією мовою програмування. Передаємо цей скрипт виконання інтерпретатору. Інтерпретатор транслює код у проміжний байткод, а потім віртуальна машина переводить отриманий байткод на набір інструкцій, що виконуються операційною системою.

  Тут варто зазначити, що хоча формально трансляція інтерпретатором вихідного коду в байткод і переведення байткоду віртуальною машиною в набір машинних команд представляють два різні процеси, але фактично вони об'єднані в інтерпретаторі.



- Портативність та платформонезалежність. Не має значення, яка у нас операційна система – Windows, Mac OS, Linux, нам достатньо написати скрипт, який запускатиметься на всіх цих ОС за наявності інтерпретатора
- Автоматичне керування пам'яті
- Динамічна типізація

Python - дуже проста програма, вона має короткий і в той же час досить простий і зрозумілий синтаксис. Відповідно його легко вивчати, і власне це одна з причин, через яку він є однією з найпопулярніших мов програмування саме для навчання. Зокрема, у 2014 році він був визнаний найпопулярнішою мовою програмування для навчання у США.

Python також популярний не тільки у сфері навчання, а й у написанні конкретних програм у тому числі комерційного характеру. Немалою мірою тому для цієї мови написано багато бібліотек, які ми можемо використовувати.

Крім того, у цієї мови програмування дуже велике співтовариство програмістів, в інтернеті можна знайти з цієї мови безліч корисних матеріалів, прикладів, отримати кваліфіковану допомогу фахівців.

Пакети та бібліотеки

Інтерпретатор Python супроводжується достатнім функціоналом, який дозволяє створювати програми цією мовою. Проте цього функціоналу може виявитися замало низки завдань. Але через велику спільноту розробників цією мовою по всьому світу також є велика екосистема різних пакетів і бібліотек, які можна використовувати для

різних цілей. У розділі мови Python на сайті METANIT.COM будуть розглянуті деякі з цих бібліотек. Перелічу основні їх.

Для створення графічних програм:

- Tkinter
- PyQt/PySide
- wxPython
- DearPyGui
- EasyGUI

Для створення мобільних додатків:

- Kivy
- Toga

Для створення веб-застосунків:

- Django
- Flask
- FastAPI
- Pylons
- Bottle
- CherryPy
- TurboGears
- Nagare

Для автоматизації процесів:

- Selenium (для тестування веб-додатків)
- Flask
- FastAPI
- Pylons
- Bottle
- CherryPy
- TurboGears
- Nagare
- robotframework
- pywinauto
- Lettuce
- Behave
- Requests

Для роботи з різними типами файлів:

- OpenPyXL (Excel)
- lxml (XML)
- ReportLab/borb (PDF)
- pdfrw / PyPDF2 (PDF)
- Pandas (CSV та Excel)

Для машинного навчання, штучного інтелекту, Data Science:

- Pandas
- SciPy
- PyTorch
- Matplotlib
- Theano
- Tensorflow
- OpenCV
- Scikit-Learn
- Keras
- NumPy

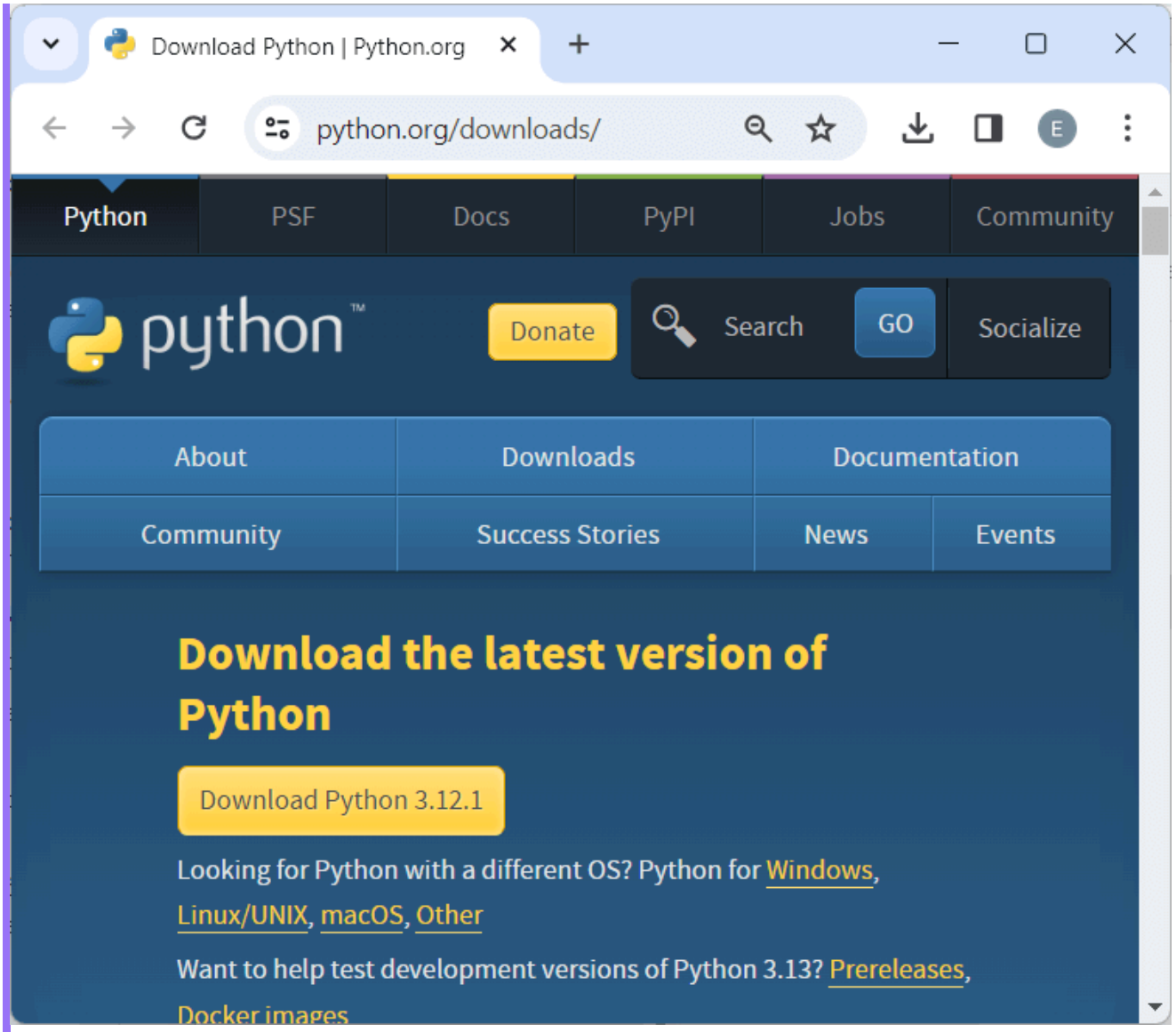
Для візуалізації:

- Matplotlib
- Seaborn
- Plotly
- Bokeh
- Altair
- HoloViews

Встановлення та перша програма на Windows

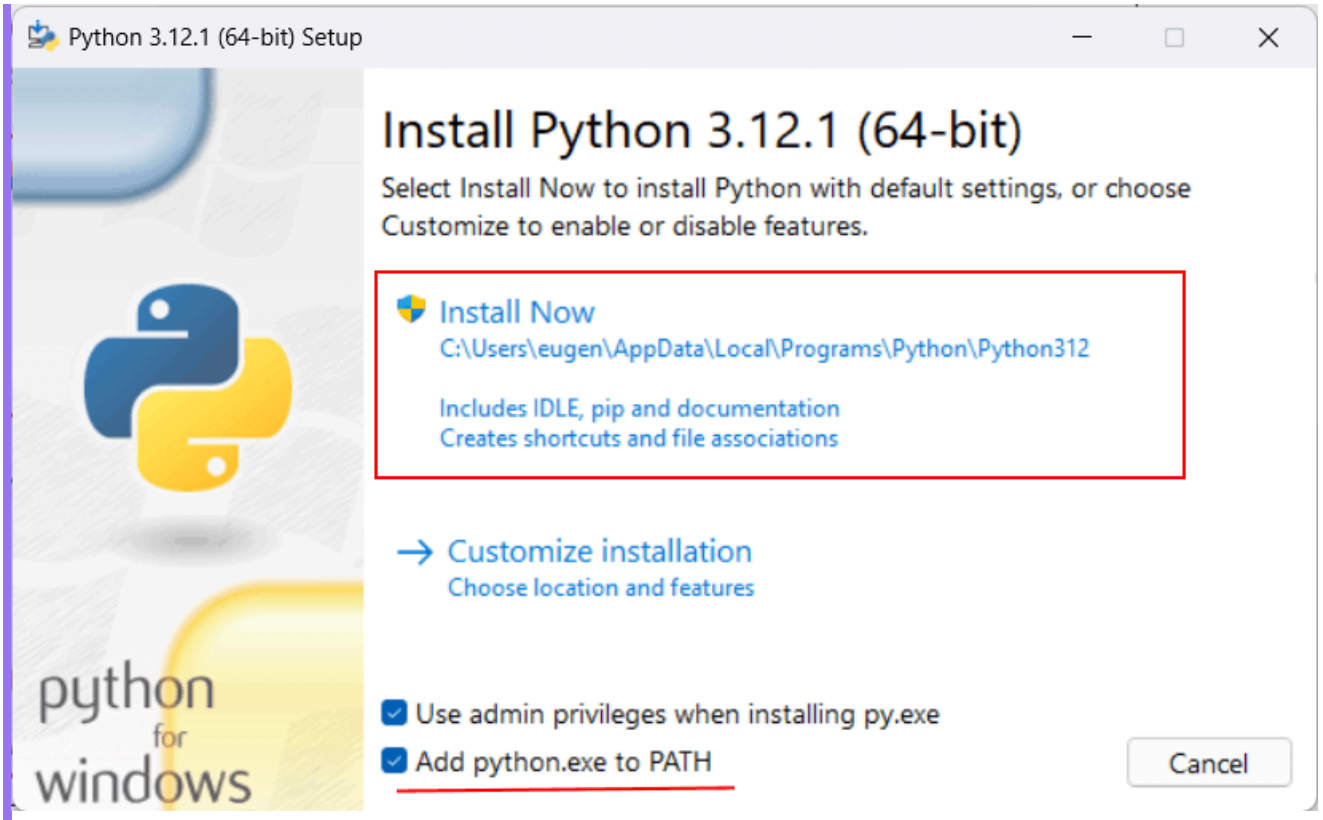
Встановлення

Для створення програм на Python нам знадобиться інтерпретатор. Для його встановлення перейдемо на сторінку https://www.python.org/downloads/ та знайдемо посилання на завантаження останньої версії мови:



Після натискання на кнопку буде завантажено відповідною поточною ОС установщик Python. Слід враховувати, що Windows 7 і попередні версії не підтримуються.

На ОС Windows під час запуску інсталятора запускає вікно майстра установки:



Тут ми можемо задати шлях, яким встановлюватиметься інтерпретатор. Залишимо його за умовчанням, тобто C:\Users[ім'я_користувача]\AppData\Local\Programs\Python\Python312\ .

Крім того, в самому низу відзначимо прапорець "Add Python 3.12 to PATH", щоб додати шлях до інтерпретатора у змінні середовища.

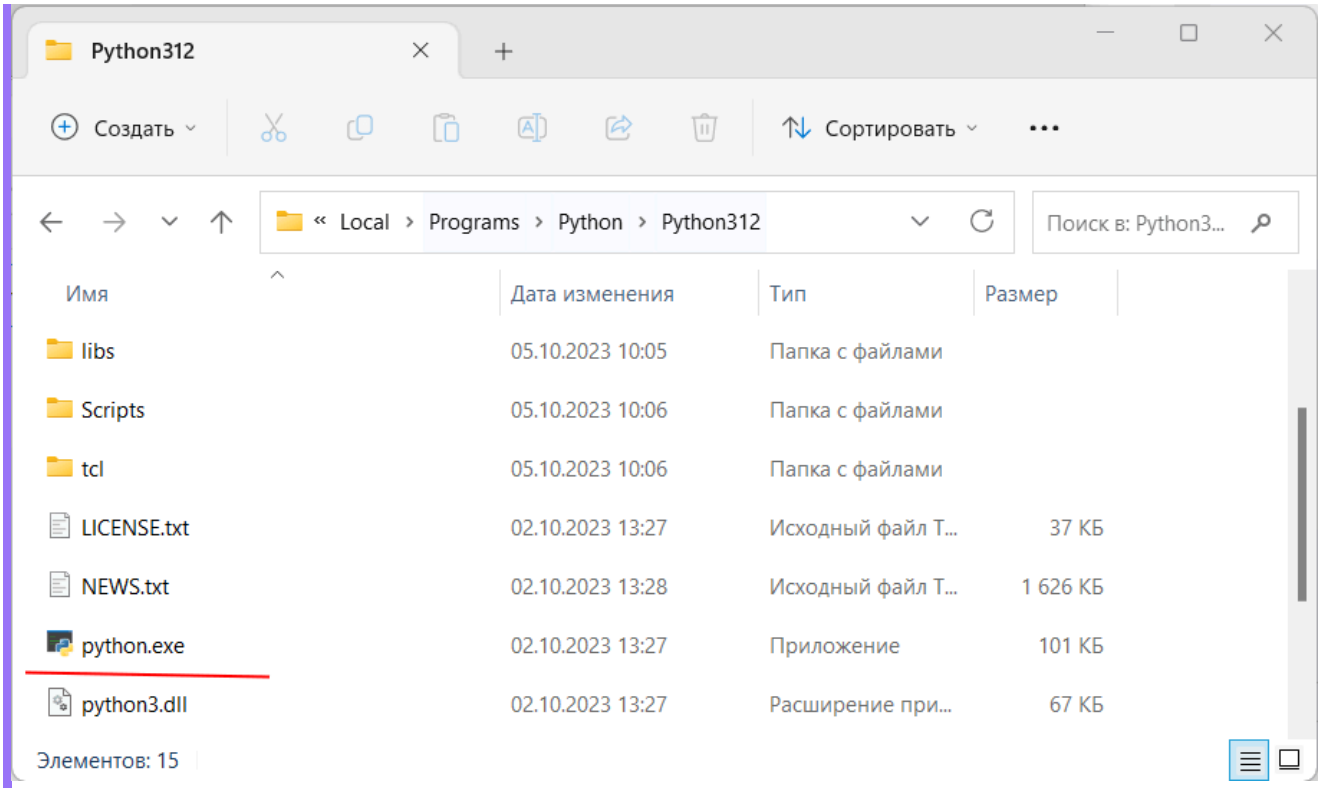
Після цього ми можемо перевірити інсталяцію Python та його версію, запустивши в командному рядку/треміналі команду python --version

```
C:\Users\Nikita>python --version
Python 3.12.1
C:\Users\Nikita>
```

Запуск інтерпретатора

Після встановлення інтерпретатора, як було описано в попередній темі, ми можемо почати створювати програми на Python. Отже, створимо першу просту програму.

Якщо при установці не було змінено адресу, то на Windows Python за замовчуванням встановлюється шляхом C:\Users[ім'я_користувача]\AppData\Local\Programs\Python\Python[номер_версії]\ і представляє файл під назвою python.exe .



Запустимо інтерпретатор і введемо до нього наступний рядок:

```
print("hello world")
```

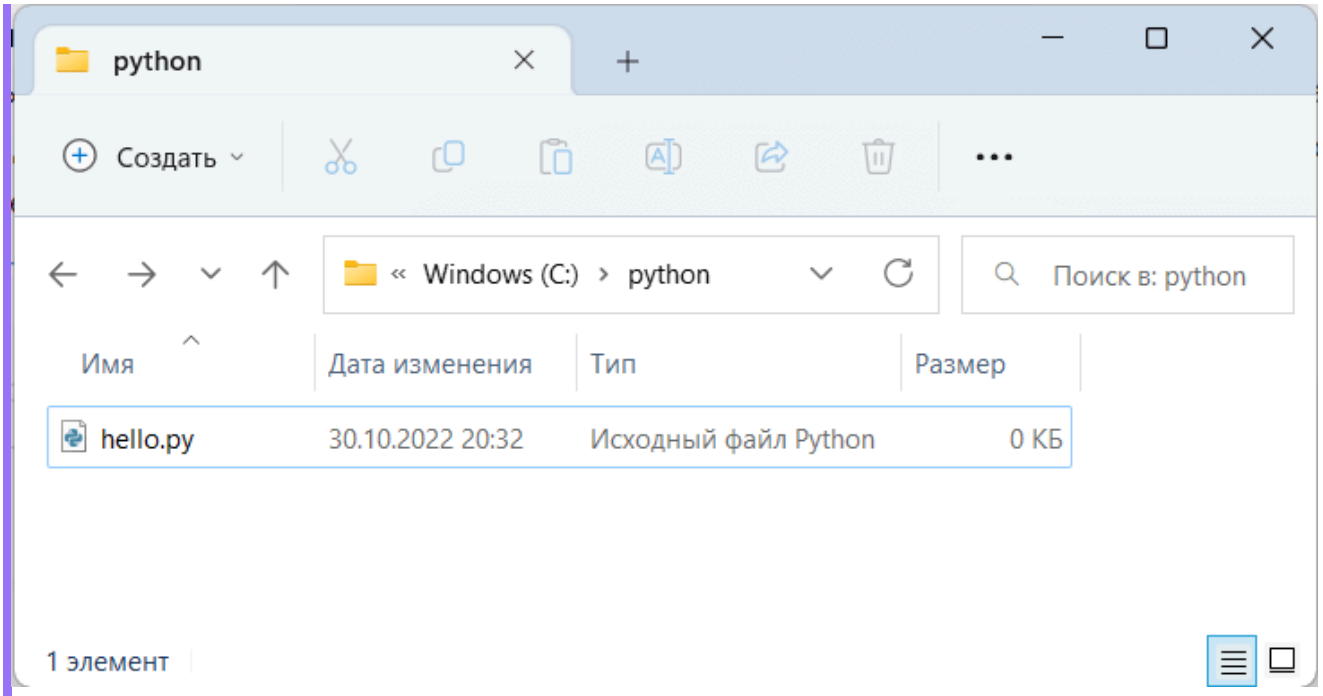
І консоль виведе рядок "hello world":

```
python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64
bit (AMD64)] on win32
Тип "help", "copyright", "кредити" або "license" для більше інформації.
>>> print("hello world")
hello world
>>>
```

Для цієї програми використовувалася функція print() , яка виводить рядок на консоль.

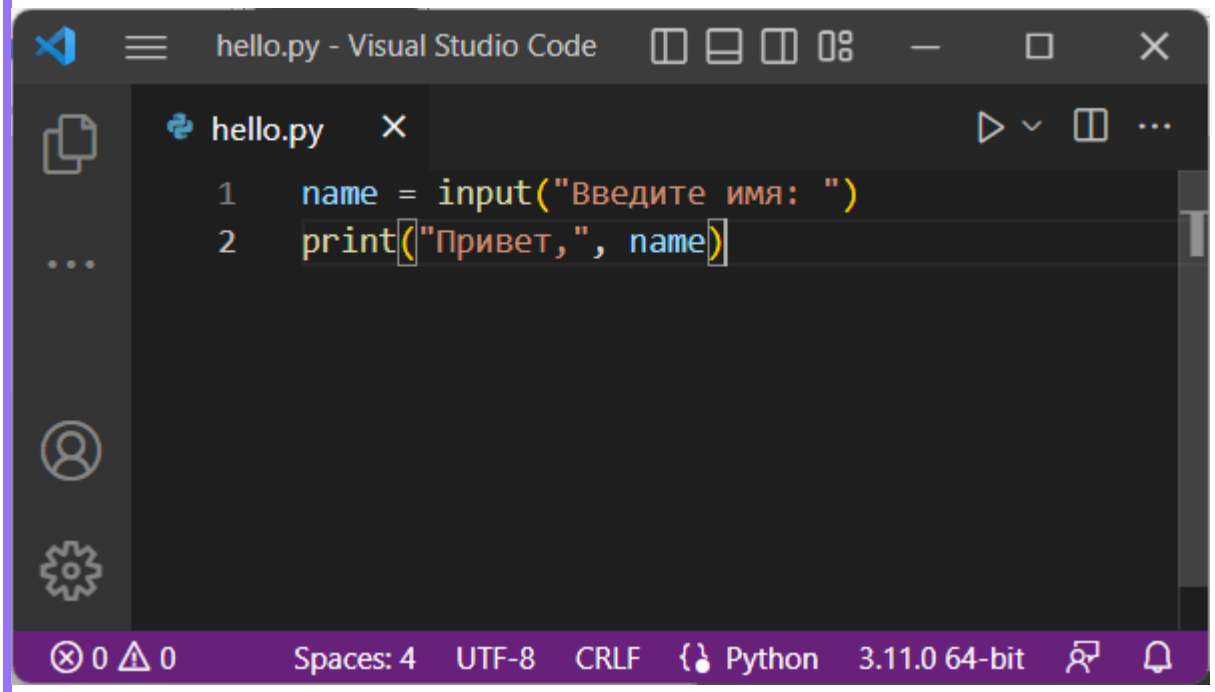
Створення файлу програми

Насправді програми зазвичай визначаються у зовнішніх файлах-скриптах і потім передаються інтерпретатору на виконання. Тому створимо файл програми. Для цього на диску C або в іншому місці файлової системи визначимо для скриптів папку python . А в цій папці створимо новий текстовий файл, який назвемо hello.py . За замовчуванням файли з кодом Python, як правило, мають розширення py .



Відкриємо цей файл у будь-якому текстовому редакторі і додамо до нього наступний код:

```
name = input("Введіть ім'я: ")
print("Привіт", name)
```



Скрипт складається із двох рядків. Перший рядок за допомогою функції input() чекає на введення користувача свого імені. Введене ім'я потім потрапляє до змінної `name` .

Другий рядок за допомогою функції `print()` виводить вітання разом із введеним ім'ям.

Тепер запустимо командний рядок/термінал і за допомогою команди cd перейдемо до папки, де знаходиться файл із вихідним кодом hello.py (наприклад, у моєму випадку це папка C:\python).

```
cd c:\python
```

Далі спочатку введемо повний шлях до інтерпретатора, потім повний шлях до файлу скрипта. Наприклад, у моєму випадку в консоль треба буде вести:

```
C:\Users\Nikita\AppData\Local\Programs\Python\Python312\python.exe hello.py
```

Але якщо при установці була вказана опція "Add Python 3.12 to PATH" , тобто шлях до інтерпретатора Python був доданий до змінних середовищ, то замість повного шляху до інтерпретатора можна просто написати python:

```
python hello.py
```

Або навіть можна скоротити:

```
py hello.py
```

Варіанти з обома способами запуску:

```
Microsoft Windows [Version 10.0.22621.2361]
(c) Корпорація Майкрософт (Microsoft Corporation). Усі права захищені.
C:\Users\Nikita>cd c:\python
c:\python>C:\Users\Nikita\AppData\Local\Programs\Python\Python312\python.ex
e hello.py
Введіть ім'я: Nikita
Привіт, Nikita
c:\python>python hello.py
Введіть назву: Tom
Привіт, Tom
c:\python>py hello.py
Введіть ім'я: Bob
Привіт, Bob
c:\python>
```

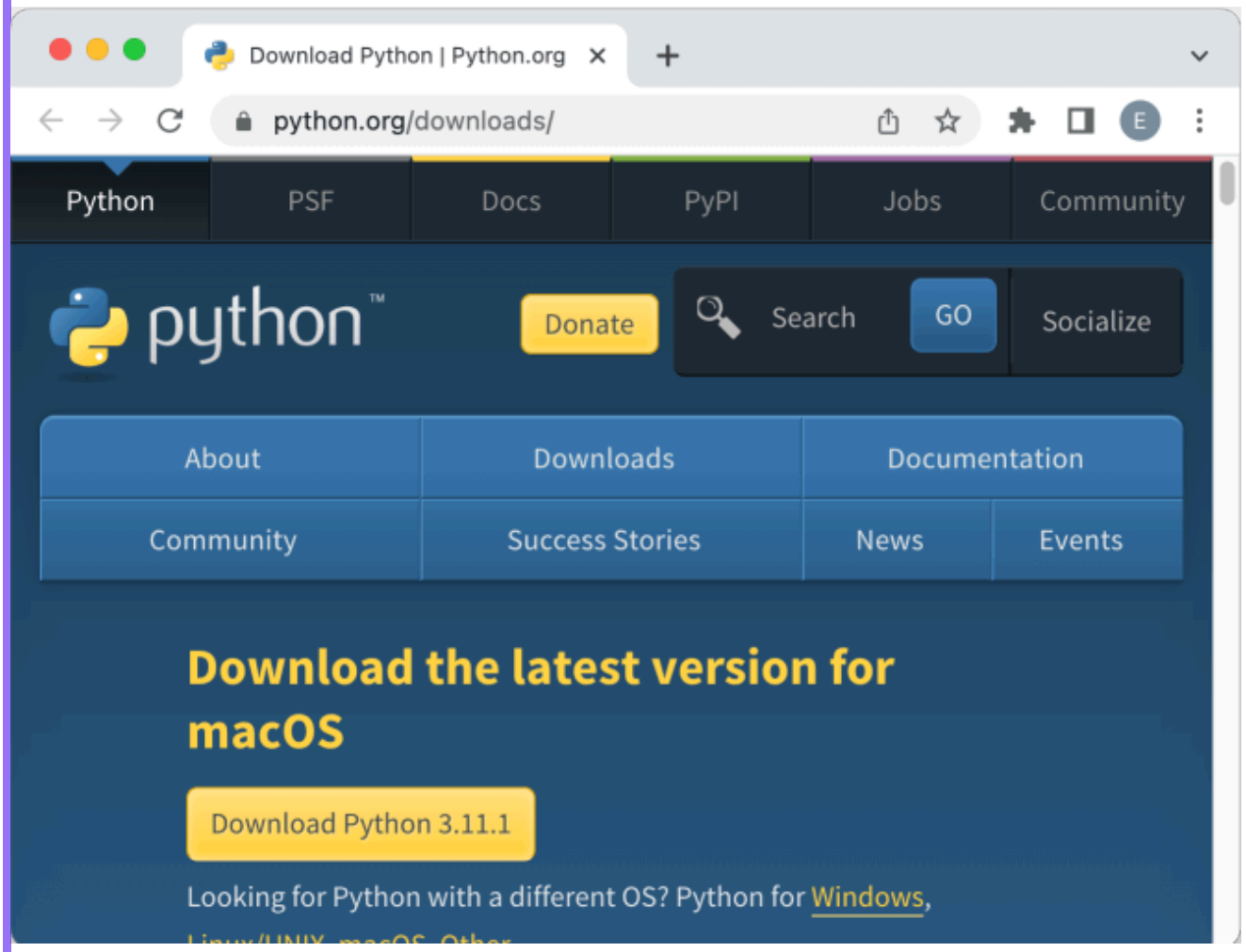
У результаті програма виведе запрошення до введення імені, та був привітання.

Встановлення та перша програма на Mac OS

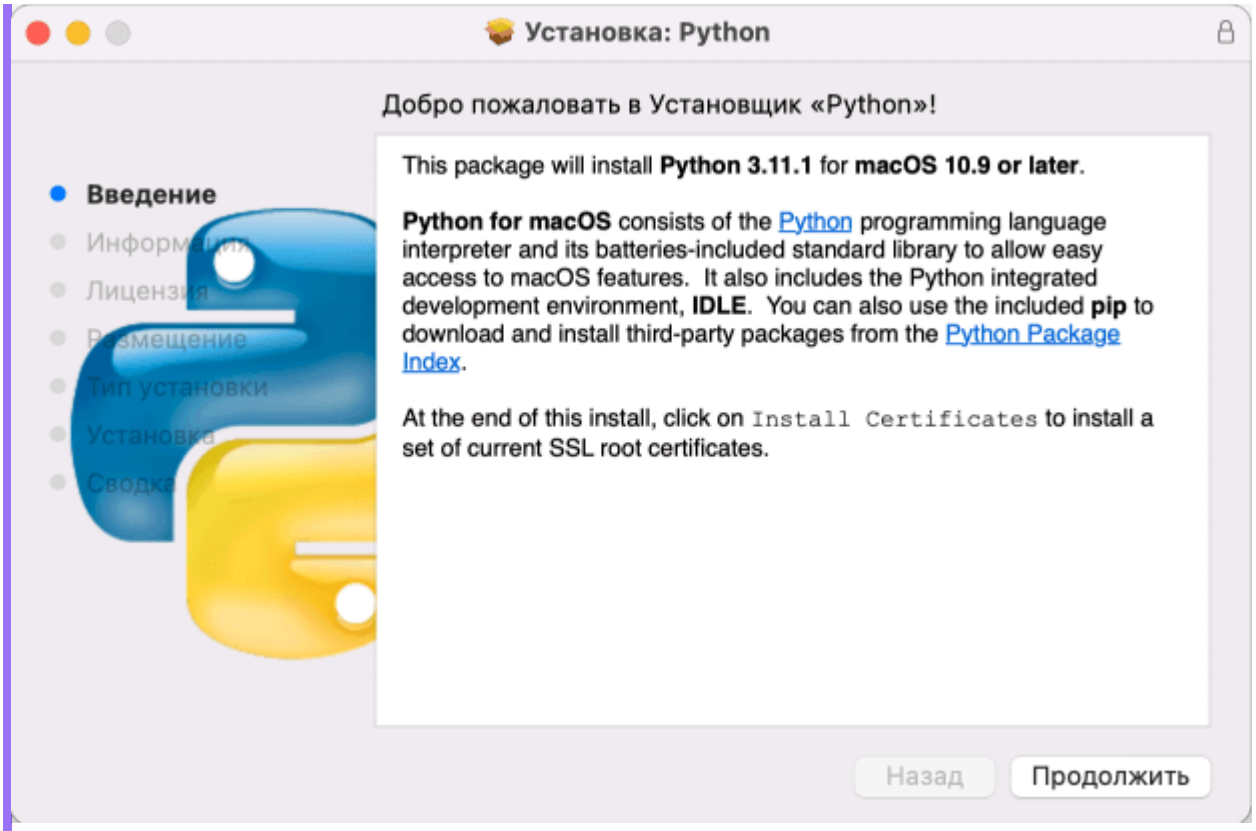
Встановлення

Для створення програм на Python нам знадобиться інтерпретатор. Для його встановлення перейдемо на сторінку

https://www.python.org/downloads/ та знайдемо посилання на завантаження останньої версії мови:

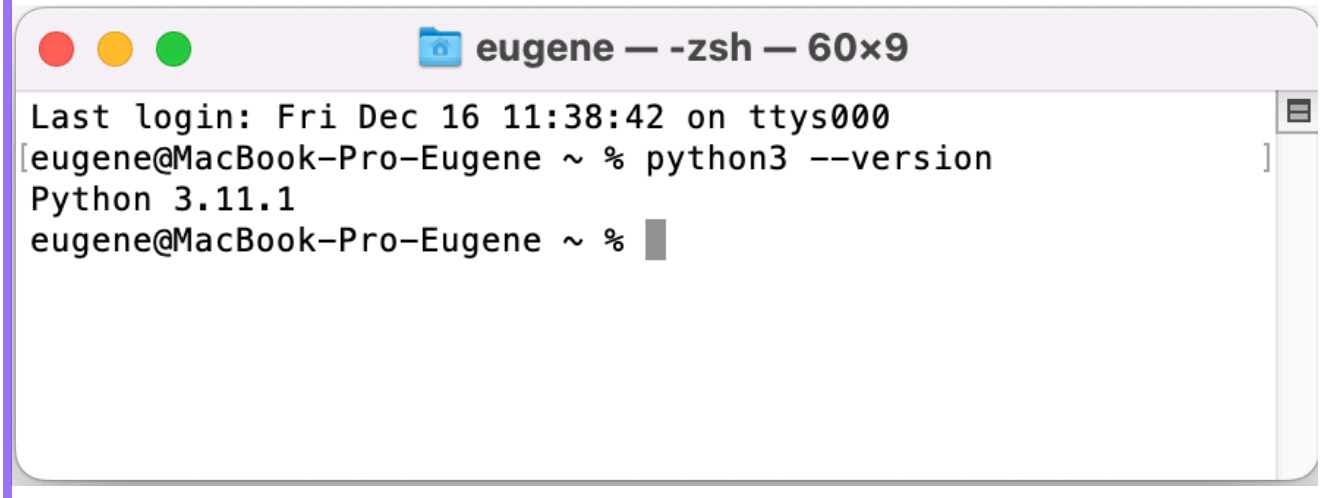


Якщо поточна ОС - Mac OS, за адресою https://www.python.org/downloads/ буде запропоновано завантажити графічний інсталятор для MacOS. Завантажимо, запустимо його та виконаємо покрокову установку:



Для звернення до інтерпретатора Python на MacOS застосовується команда python3 . Наприклад, після встановлення інтерпретатора перевіримо його версію командою

```
python3 --version
```

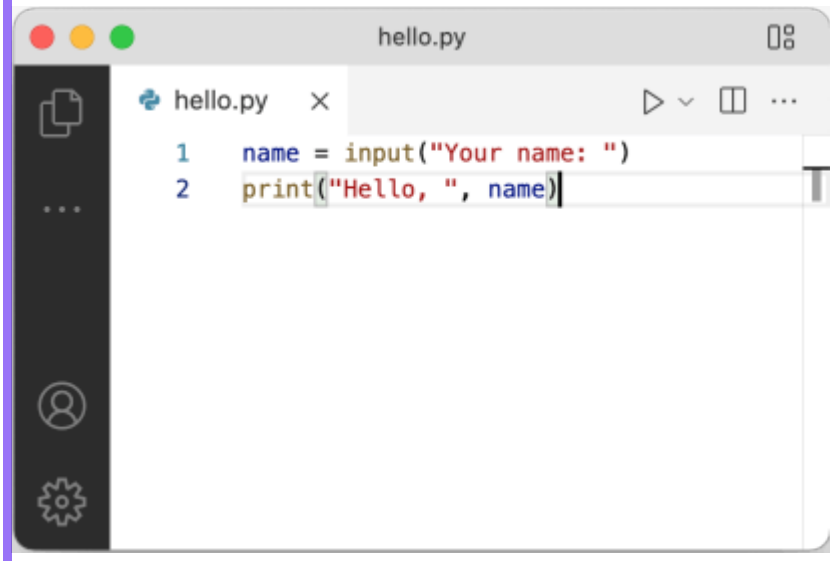


Перша програма

Спочатку визначимо десь на жорсткому диску для скриптів папку python . А в цій папці створимо новий текстовий файл, який назвемо hello.py . За замовчуванням файли з кодом Python, як правило, мають розширення py .

Відкриємо цей файл у будь-якому текстовому редакторі і додамо до нього наступний код:

```
name = input("Your name: ")
print("Hello, ", name)
```



Скрипт складається із двох рядків. Перший рядок за допомогою функції input() чекає на введення користувача свого імені. Введене ім'я потім потрапляє до змінної `name`.

Другий рядок за допомогою функції `print()` виводить вітання разом із введеним ім'ям.

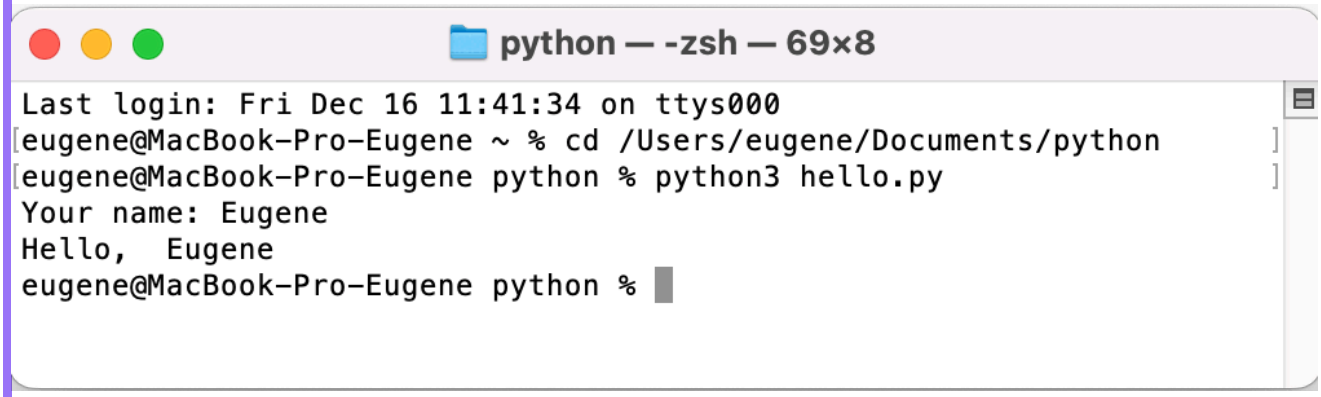
Тепер запустимо командний рядок/термінал і за допомогою команди cd перейдемо до папки, де знаходиться файл із вихідним кодом hello.py (наприклад, у моєму випадку це папка "/Users/Nikita/Documents/python").

```
cd /Users/Nikita/Documents/python
```

Далі для виконання скрипту hello.py передамо його інтерпретатору python:

```
python3 hello.py
```

У результаті програма виведе запрошення до введення імені, та був привітання.

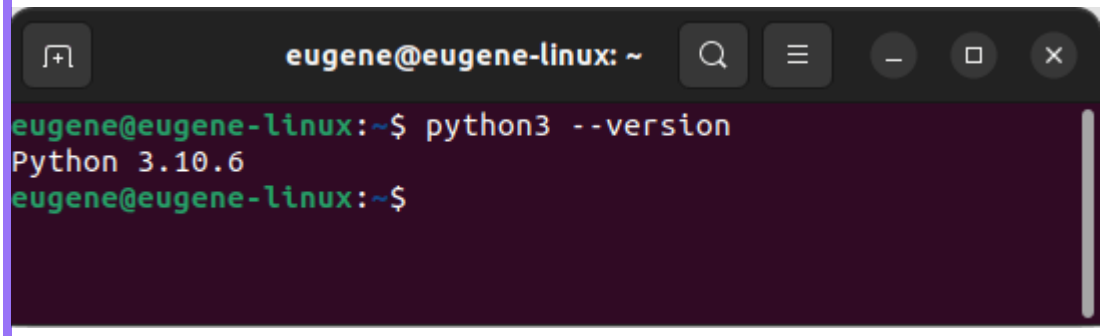


Встановлення та перша програма на Linux

Встановлення

Для створення програм на Python нам знадобиться інтерпретатор. Варто зазначити, що в деяких дистрибутивах Linux (наприклад, в Ubuntu) Python може бути встановлений за умовчанням. Для перевірки версії Python у терміналі потрібно виконати наступну команду

```
python3 --version
```



Якщо Python встановлений, вона відобразить версію інтерпретатора.

Однак навіть якщо Python встановлений, його версія може бути не останньою. Для встановлення останньої доступної версії Python виконаємо таку команду:

```
sudo apt-get update && sudo apt-get install python3
```

Якщо потрібно встановити не останню доступну, а певну версію, то вказується також підверсія Python. Наприклад, встановлення версії Python 3.10:

```
sudo apt-get install python3.10
```

Відповідно, встановлення версії Python 3.11:

```
sudo apt-get install python3.11
```

Запуск інтерпретатора

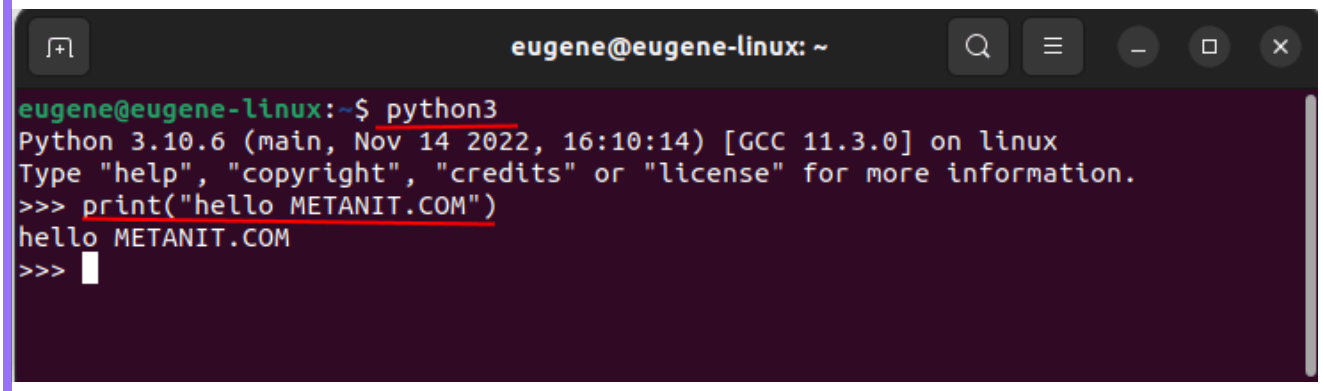
Після встановлення інтерпретатора ми можемо почати створювати програми на Python. Отже, створимо першу просту програму. Для цього введемо в терміналі

```
python3
```

В результаті запускається інтерпретатор Python. Введемо до нього наступний рядок:

```
print("hello METANIT.COM")
```

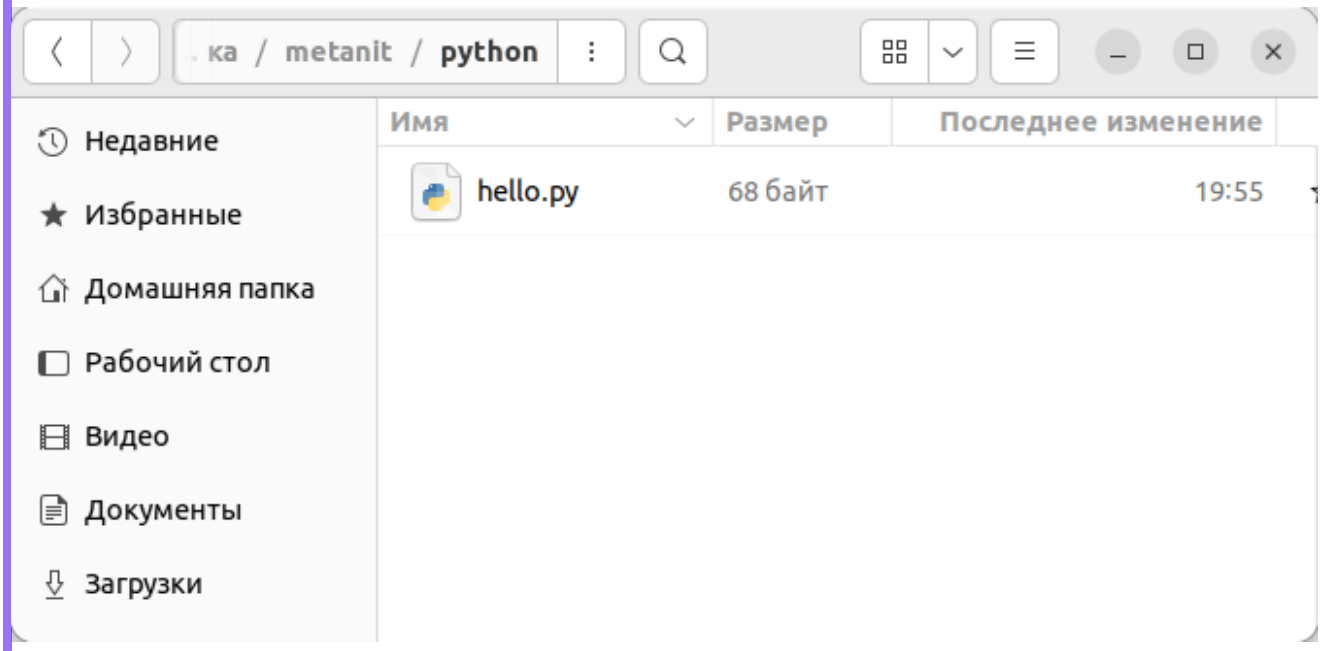
І консоль виведе рядок "hello METANIT.COM":



Для цієї програми використовувалася функція print() , яка виводить рядок на консоль.

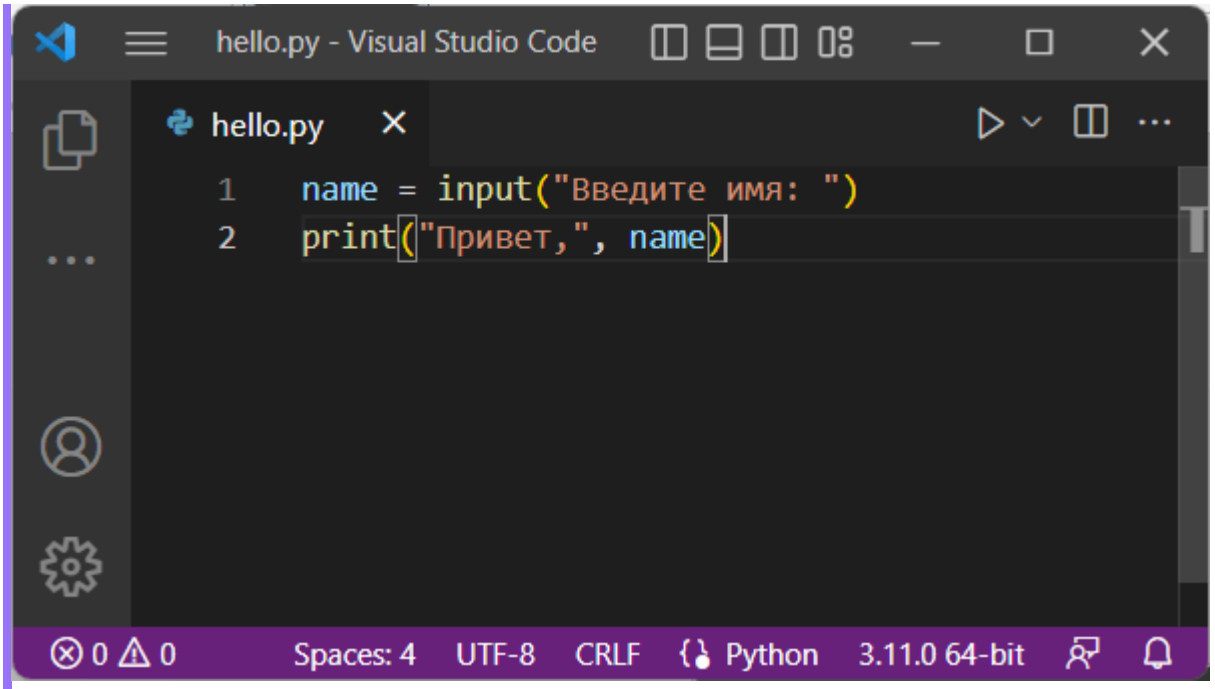
Створення файлу програми

Насправді програми зазвичай визначаються у зовнішніх файлах-скриптах і потім передаються інтерпретатору на виконання. Тому створимо файл програми. Для цього визначимо для скриптів папку python . А в цій папці створимо новий текстовий файл, який назвемо hello.py . За замовчуванням файли з кодом Python, як правило, мають розширення py .



Відкриємо цей файл у будь-якому текстовому редакторі і додамо до нього наступний код:

```
name = input("Введіть ім'я: ")
print("Привіт", name)
```



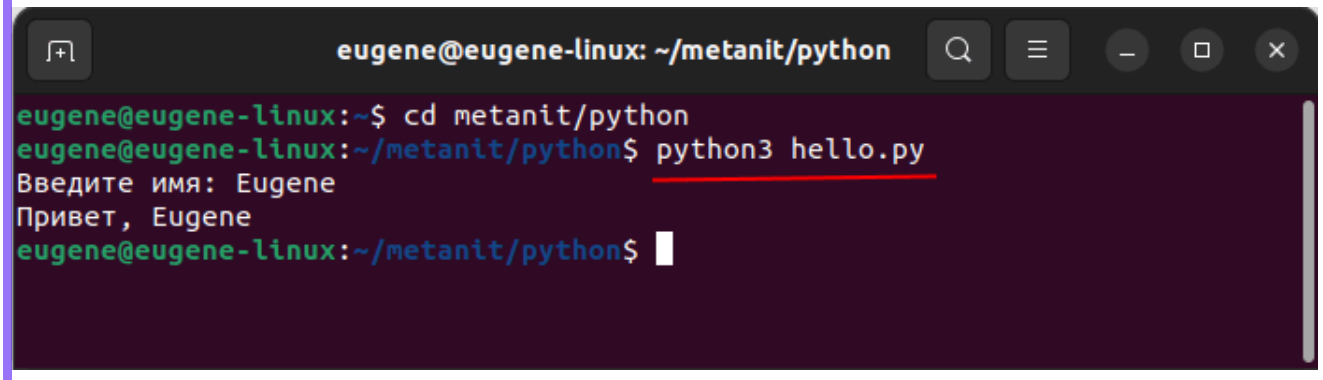
Скрипт складається із двох рядків. Перший рядок за допомогою функції input() чекає на введення користувача свого імені. Введене ім'я потім потрапляє до змінної `name`.

Другий рядок за допомогою функції `print()` виводить вітання разом із введеним ім'ям.

Тепер запустимо термінал та за допомогою команди cd перейдемо до папки, де знаходиться файл з вихідним кодом hello.py (наприклад, у моєму випадку це папка `metanit/python`
в каталозі поточного користувача). І потім виконаємо код `hello.py` за допомогою наступної команди

```
python3 hello.py
```

У результаті програма виведе запрошення до введення імені, та був привітання.



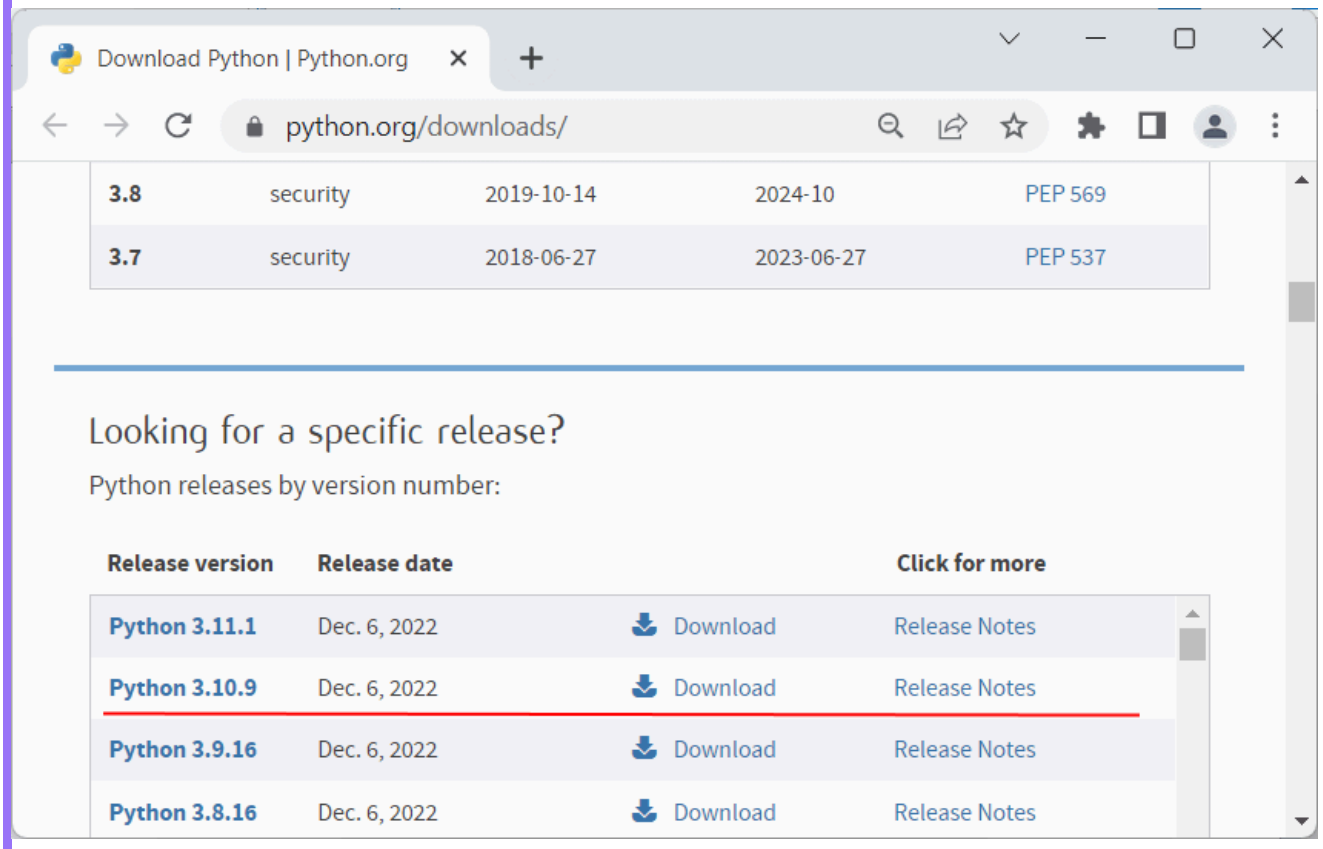
Керування версіями Python на Windows, MacOS та Linux

На одній робочій машині одночасно можна встановити кілька версій Python. Це буває корисно, коли йде робота з деякими зовнішніми бібліотеками, які підтримують різні версії python, або з якихось інших причин нам треба використовувати кілька різних версій. Наприклад, на момент написання статті останньою та актуальною є версія

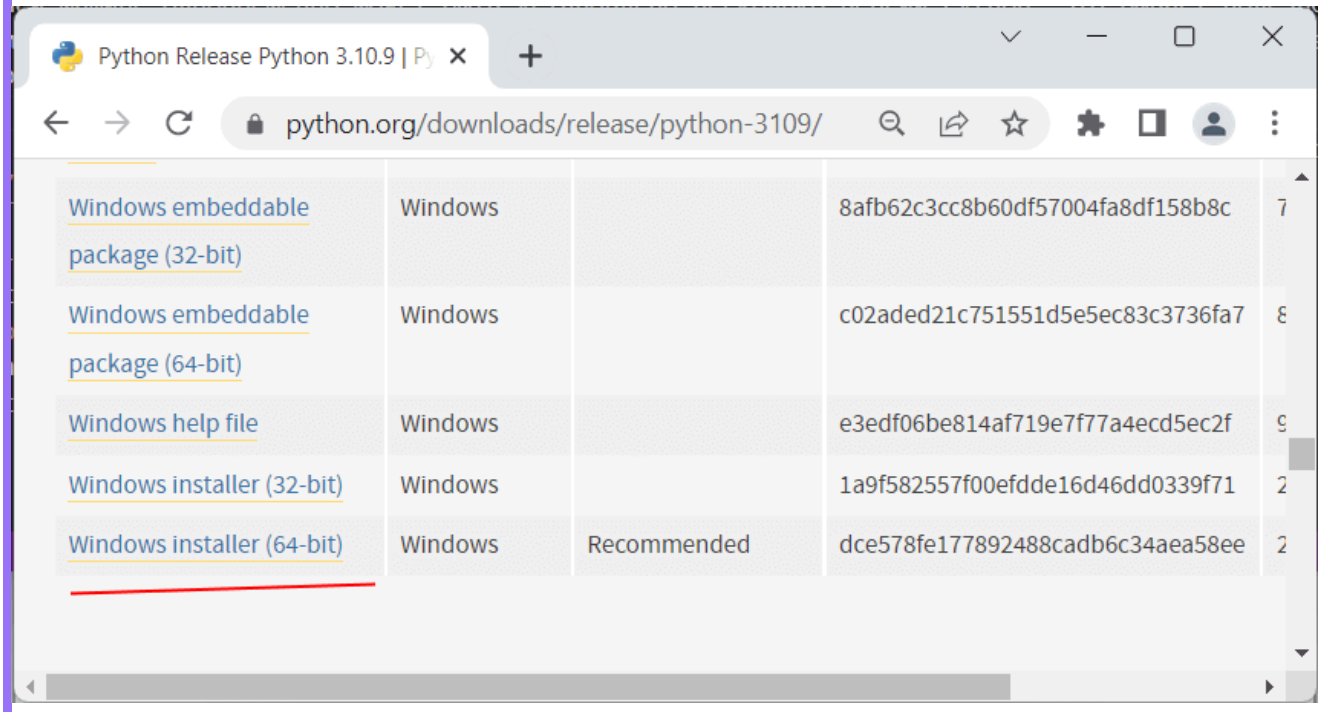
Python 3.11 . Але, скажімо, потрібно також встановити версію 3.10 , як у цьому випадку керувати окремими версіями Python?

Windows

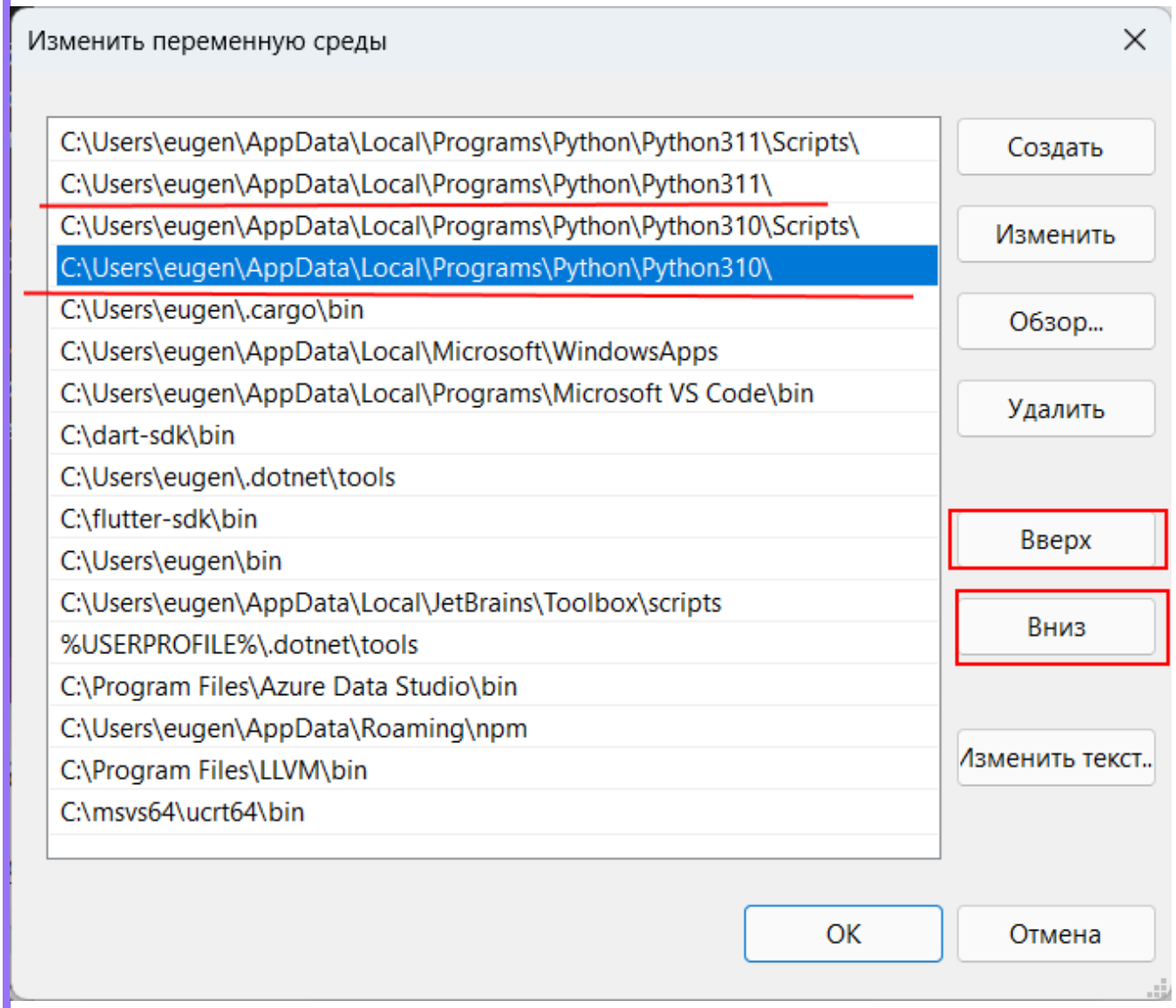
На сторінці завантажень https://www.python.org/downloads/ ми можемо знайти посилання на потрібну версію:



І також завантажити її та встановити:



Щоб при використанні інтерпретатора Python не прописувати до нього весь шлях, додамо при встановленні в змінні середовища. Але тут треба враховувати, що в змінних середовищах може міститися кілька шляхів до різних інтерпретаторів Python:



Та версія Python, яка знаходиться вище, буде стандартною версією. За допомогою кнопки "Вгору" можна потрібну нам версію перемістити на початок, зробивши версією за замовчуванням. Наприклад, у моєму випадку це версія 3.11. Відповідно, якщо я введу в терміналі команду

```
python --version
```

або

```
py --version
```

то консоль відобразить версію 3.11:

```
C:\python>python --version
Python 3.11.0
```

Для звернення до версії 3.10 (та для всіх інших версій) необхідно використовувати вказувати номер версії:

```
C:\python>py -3.10 --version
Python 3.10.9
```

наприклад, виконання скрипту `hello.py` за допомогою версії 3.10:

```
py -3.10 hello.py
```

Так само можна викликати й інші версії Python.

MacOS

На MacOS можна встановити різні версії, наприклад, завантаживши з офіційного сайту пакет інсталятора для певної версії.

Для звернення до певної версії Python на MacOS вказуємо явно підверсію у форматі `python3.[номер_подверсии]`. Наприклад, я маю версію Python 3.10. Перевіримо її версію:

```
python3.10 --version
```

Аналогічно звернення до версії `python3.9` (за умови, якщо вона встановлена)

```
python3.9 --version
```

Наприклад виконання скрипту `hello.py` за допомогою версії python 3.10:

```
python3.10 hello.py
```

Linux

На Linux можна встановити одночасно кілька версій Python. Наприклад, встановлення версій 3.10 та 3.11:

```
sudo apt-get install python3.10
sudo apt-get install python3.11
```

Однією з версій є стандартна версія. І для звернення до неї достатньо прописати python3 , наприклад, перевіримо версію за замовчуванням:

```
python3 --version
```

Для звернення до інших версій треба вказувати підверсію:

```
python3.10 --version
python3.11 --version
```

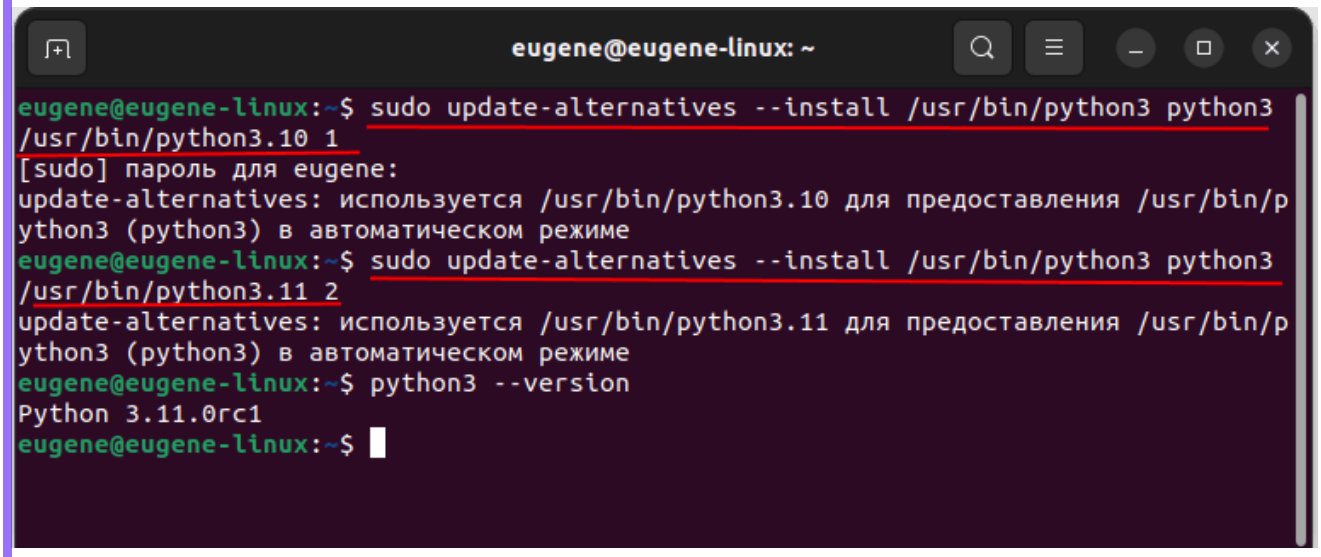
Наприклад, виконання скрипту `hello` за допомогою версії Python 3.10:

```
python3.10 hello.py
```

Але може скластися ситуація, коли нам треба змінити стандартну версію. У цьому випадку використовується команда update-alternatives для зв'язування певної версії Python з командою python3. Наприклад, ми хочемо встановити як версію за промовчанням Python 3.11. У цьому випадку послідовно виконаємо такі команди:

```
sudo update-alternatives --install /usr/bin/python3 python3
/usr/bin/python3.10 1
sudo update-alternatives --install /usr/bin/python3 python3
/usr/bin/python3.11 2
```

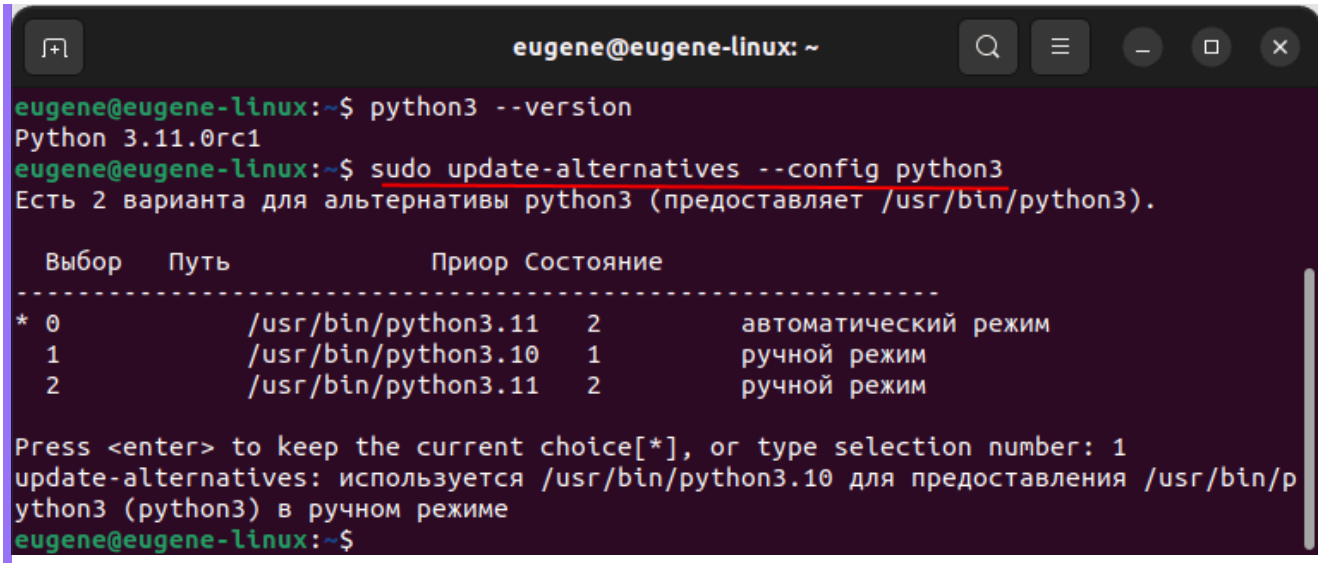
Числа праворуч вказують на пріоритет/стан. Так, для версії 3.11 вказано більший пріоритет, тому при зверненні до використання `python3` буде використовуватися саме версія 3.11 (у моєму випадку це Python 3.11.0rc1)



За допомогою команди

```
sudo update-alternatives --config python3
```

можна змінити стандартну версію



Перша програма в PyCharm

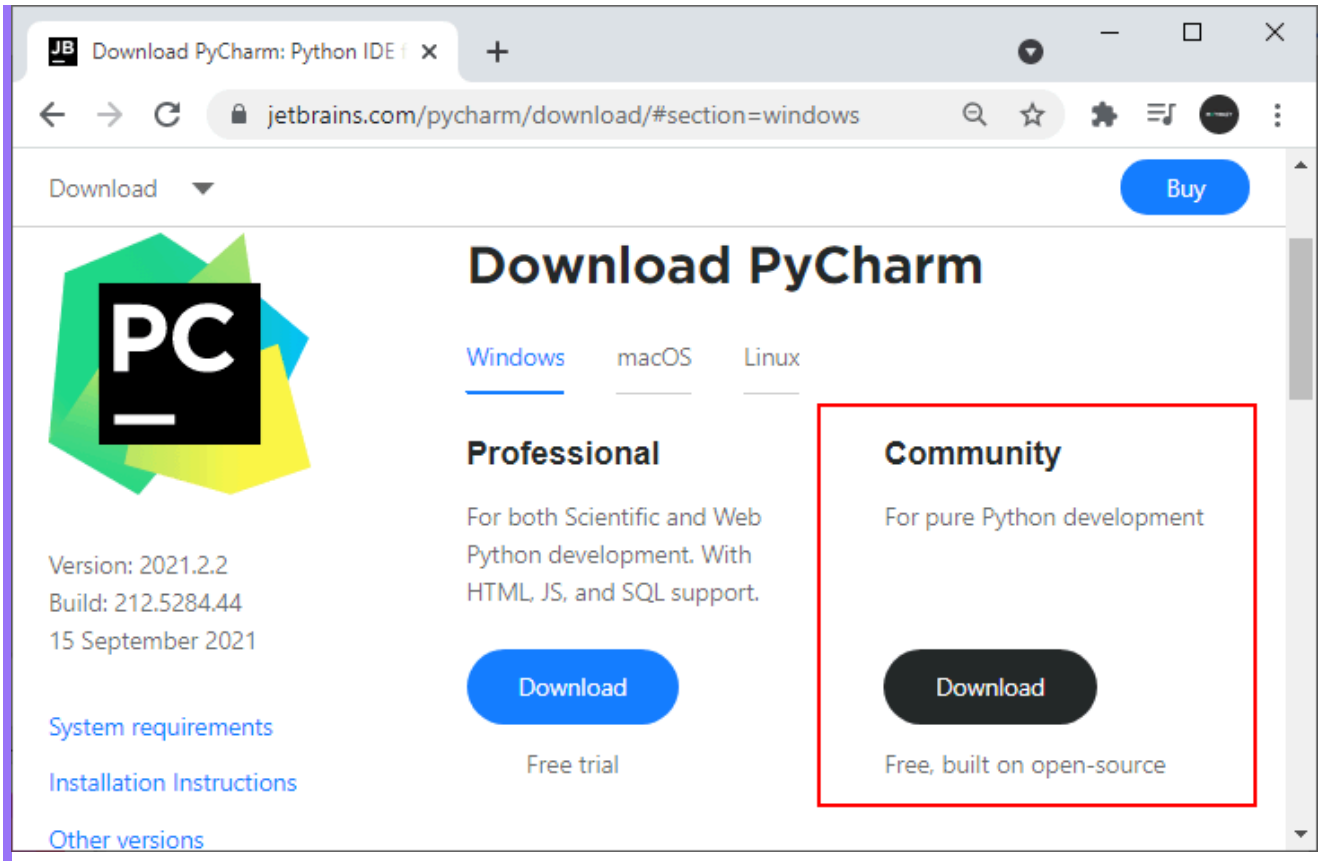
Минулої теми було описано створення найпростішого скрипта мовою Python. Для створення скрипта використовувався текстовий редактор. У моєму випадку це був Notepad. Але є інший спосіб створення програм, який представляє використання різних інтегрованих середовищ розробки або IDE.

IDE надають нам текстовий редактор для набору коду, але на відміну від стандартних текстових редакторів, IDE також забезпечує повноцінне підсвічування синтаксису, автодоповнення або інтелектуальну підказку коду, можливість відразу виконати створений скрипт, а також багато іншого.

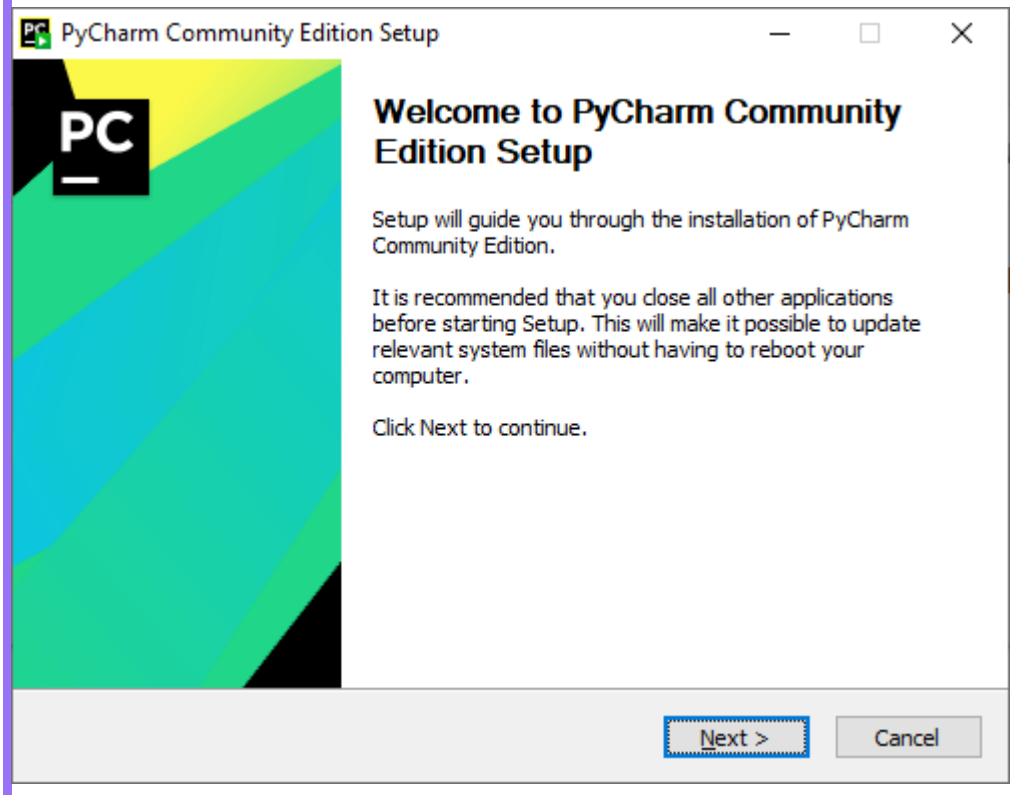
Для Python можна використовувати різні середовища розробки, але однією з найпопулярніших серед них є середовище PyCharm , створене компанією JetBrains. Це середовище динамічно розвивається, постійно оновлюється і доступне найбільш поширених операційних систем - Windows, MacOS, Linux.

Щоправда, вона має одне важливе обмеження. А саме вона доступна у двох основних варіантах: платний випуск Professional та безкоштовний Community. Багато базових можливостей доступні і в безкоштовному випуску Community. У той же час ряд можливостей, наприклад, веб-розробка доступні тільки в платному Professional.

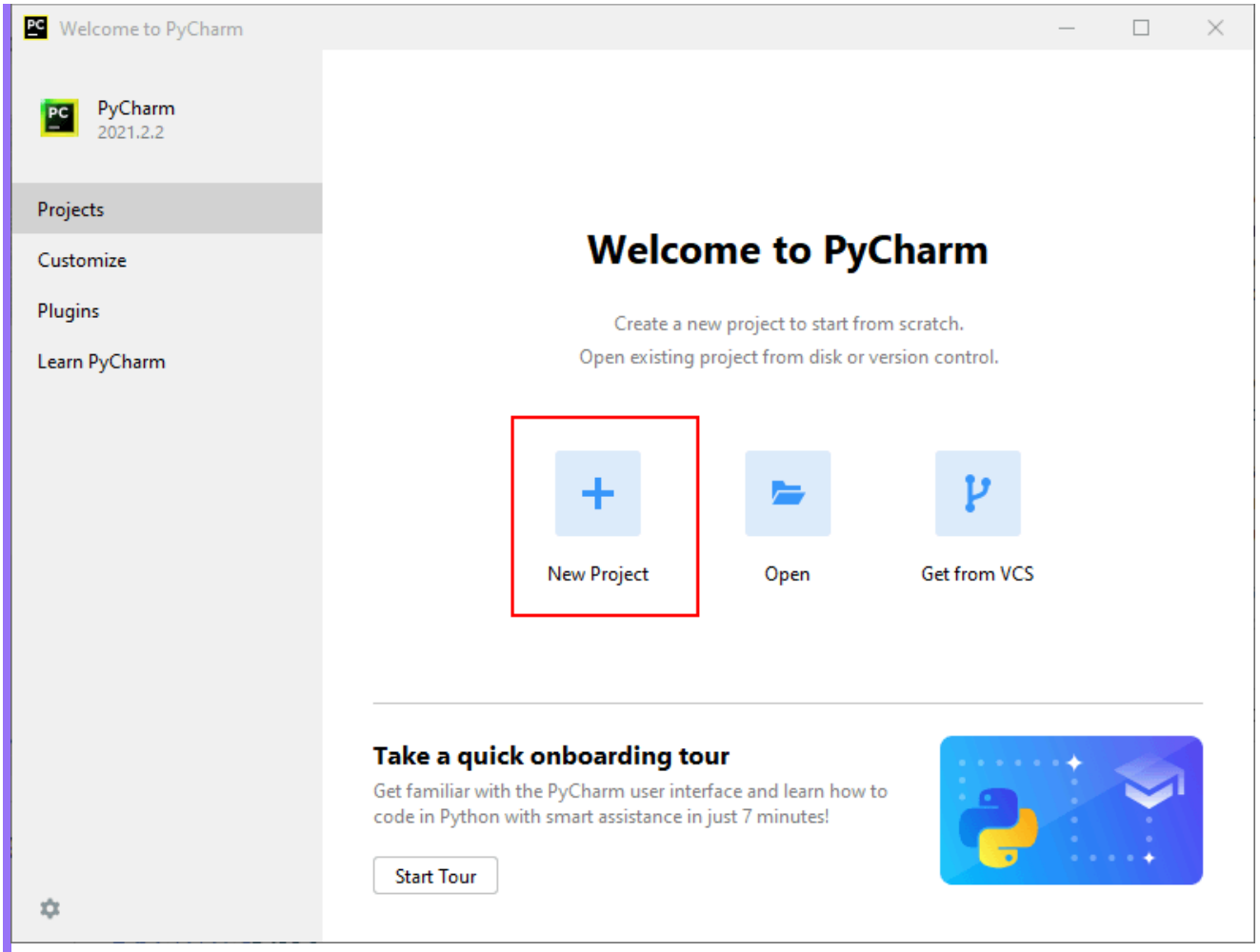
У нашому випадку скористаємось безкоштовним випуском Community. Для цього перейдемо на
сторінку завантаження та завантажимо інсталяційний файл PyCharm Community.



Після завантаження виконаємо його встановлення.

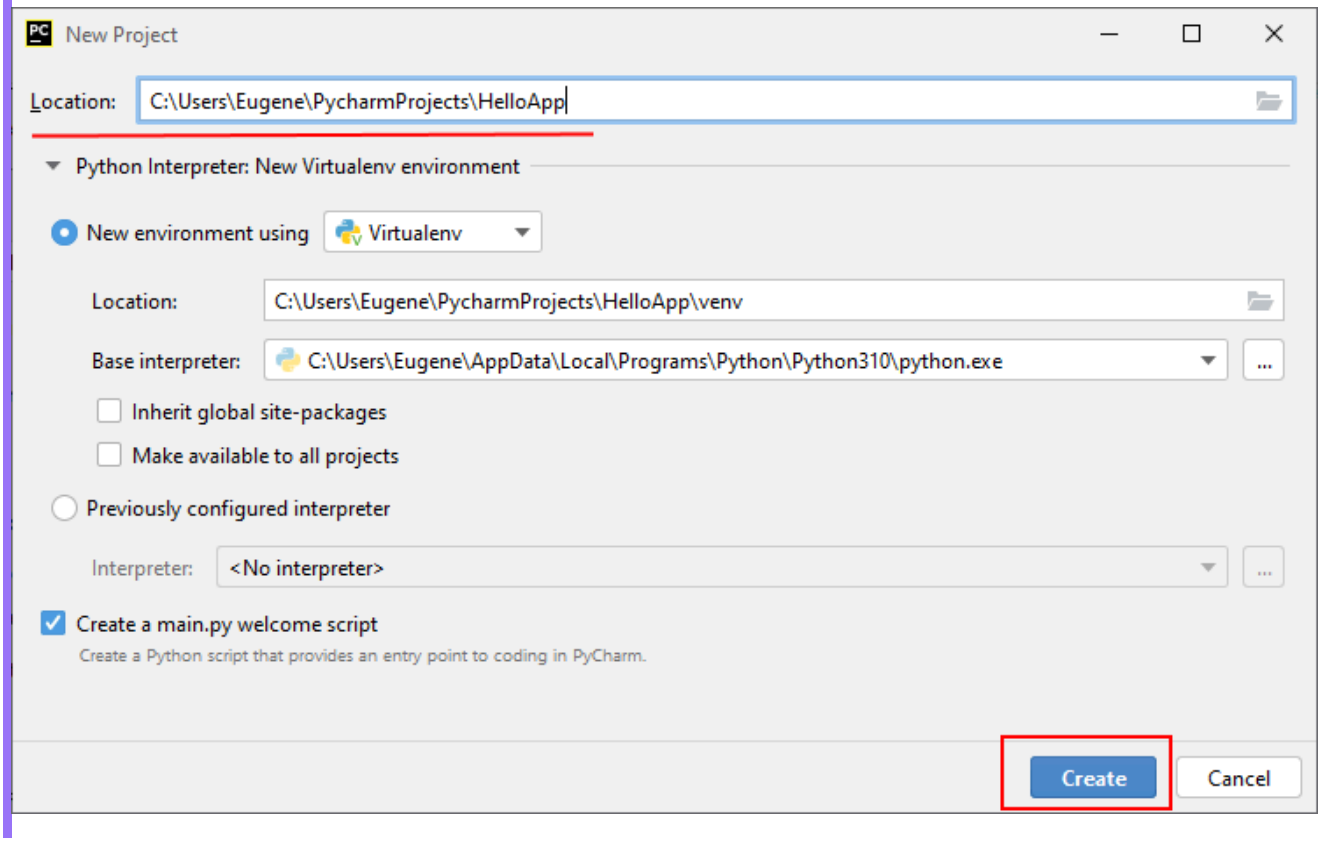


Після завершення встановлення запустимо програму. При першому запуску відкривається початкове вікно:



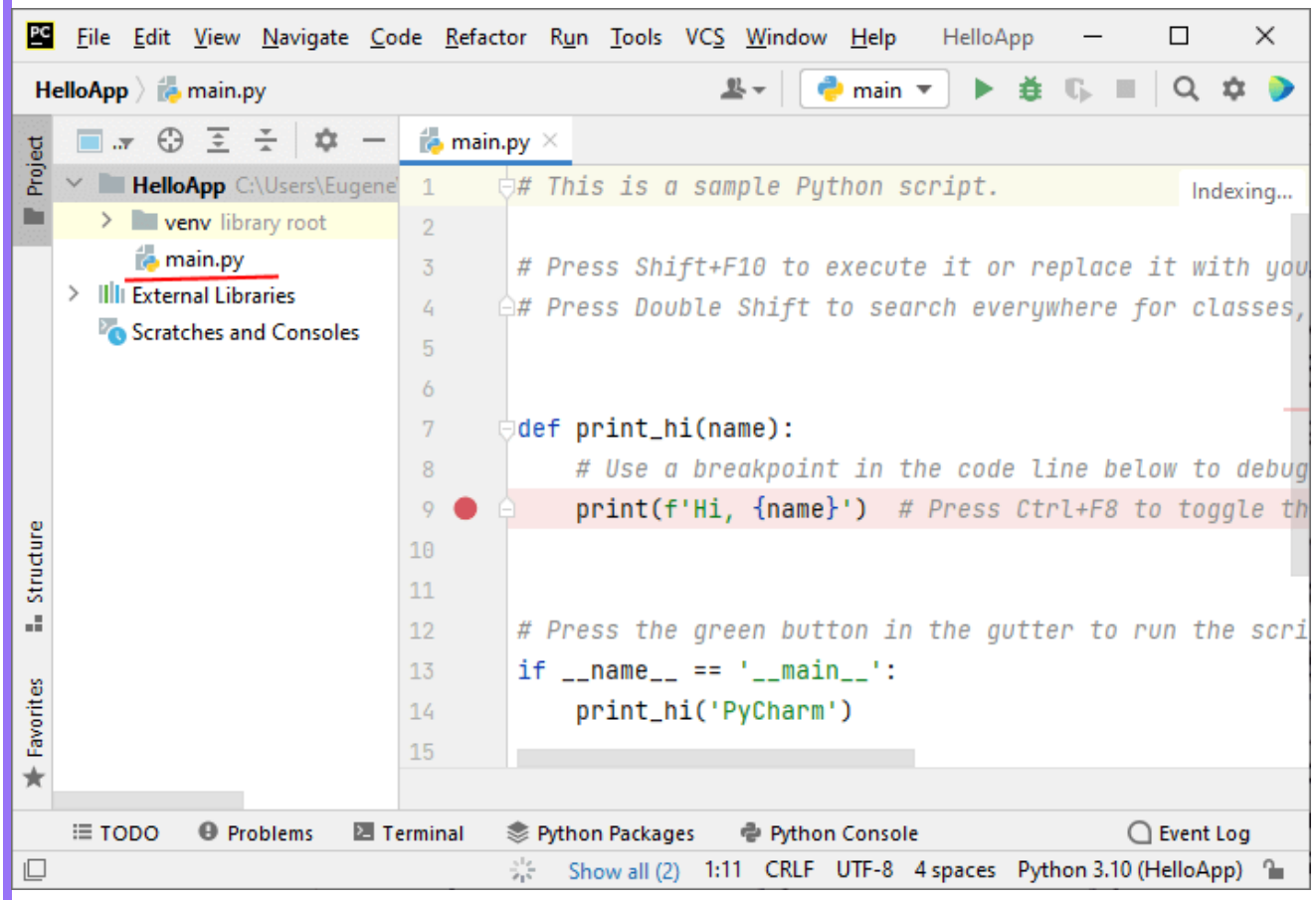
Створимо проект і для цього оберемо пункт New Project .

Далі нам відкриється вікно для налаштування проекту. У полі Location необхідно вказати шлях до проекту. У моєму випадку проект буде розміщено в папці HelloApp. Власне назва папки і буде назвою проекту.



Крім шляху до проекту, всі інші налаштування залишимо за замовчуванням і натиснемо на кнопку Create для створення проекту.

Після цього буде створено порожній проект:

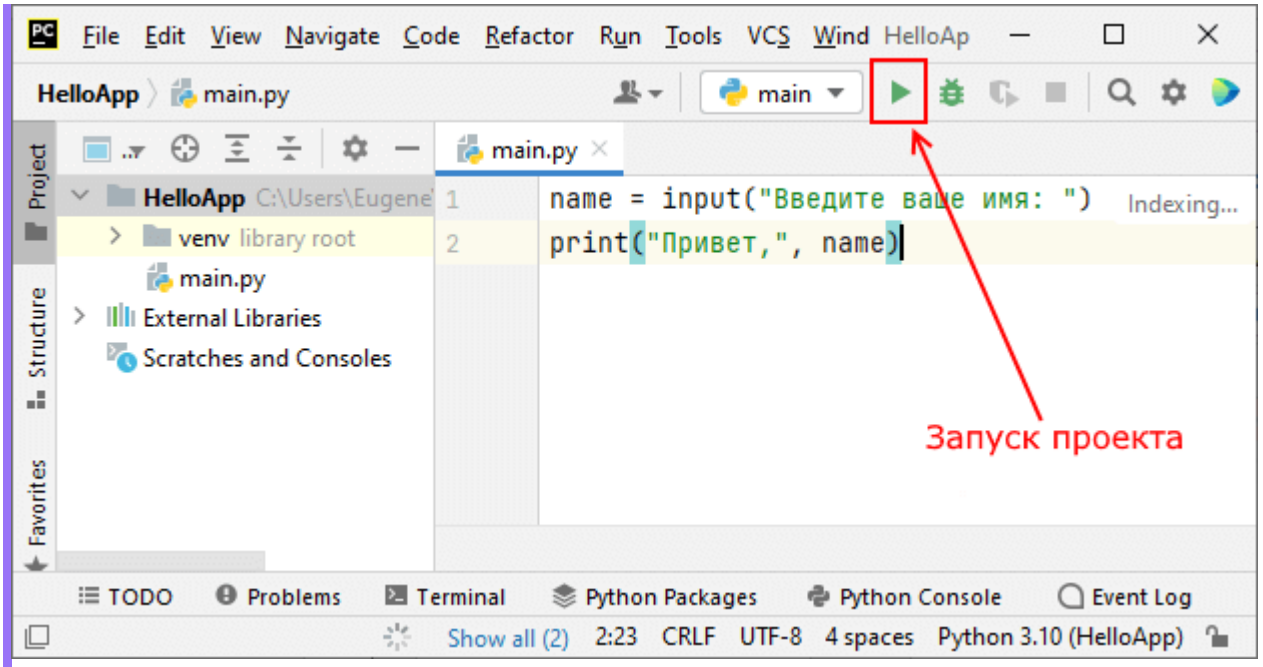


У центрі середовища буде відкрито файл main.py з деяким вмістом за промовчанням.

Тепер створимо найпростішу програму. Для цього змінимо код файлу main.py так:

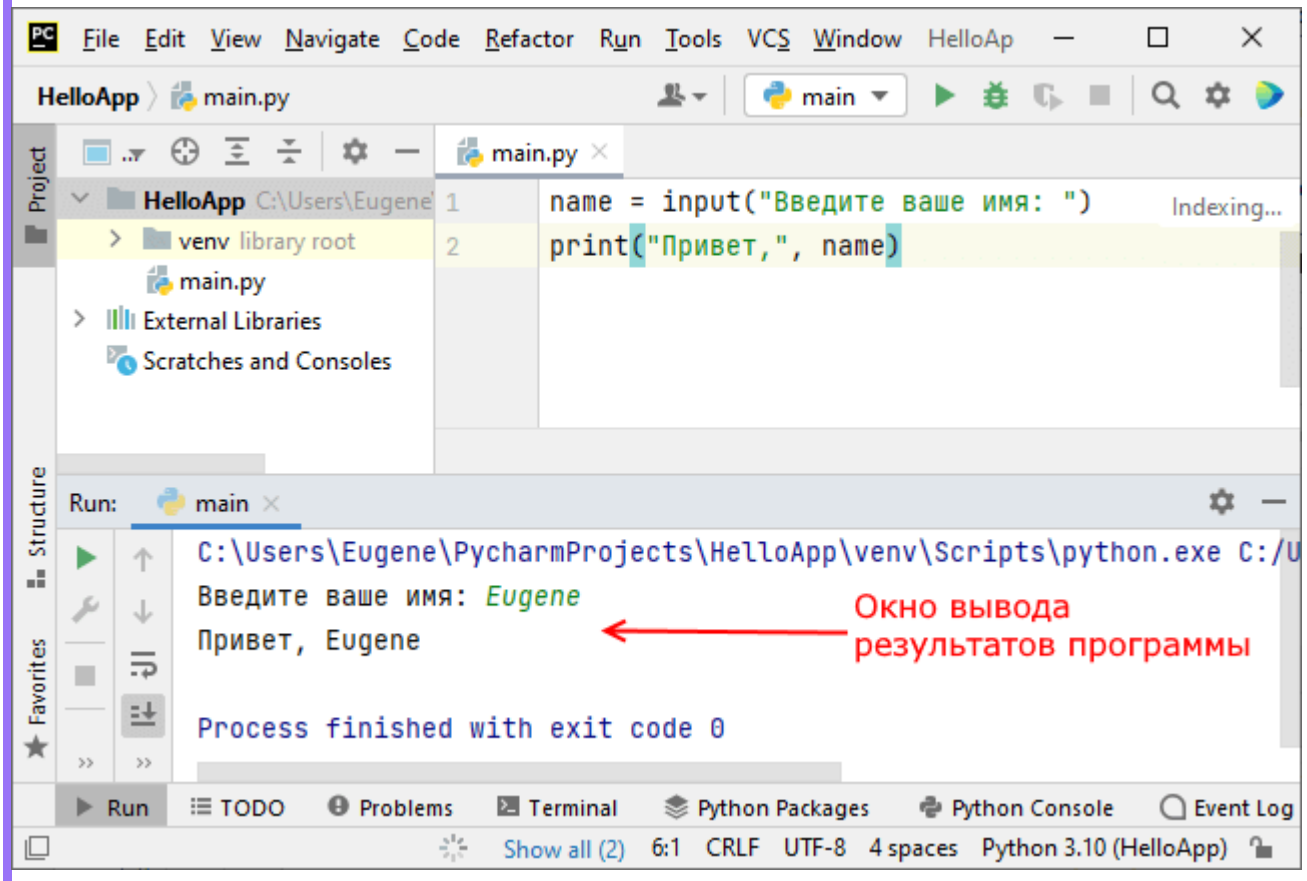
```
name = input("Введіть ваше ім'я: ")
print("Привіт", name)
```

Для запуску скрипту натиснемо на зелену стрілку в панелі інструментів програми:



Також для запуску можна перейти в меню Run і там натиснути на підпункт Run 'main')

Після цього внизу IDE відобразиться вікно виведення, де потрібно буде ввести ім'я і де після цього буде виведено привітання:

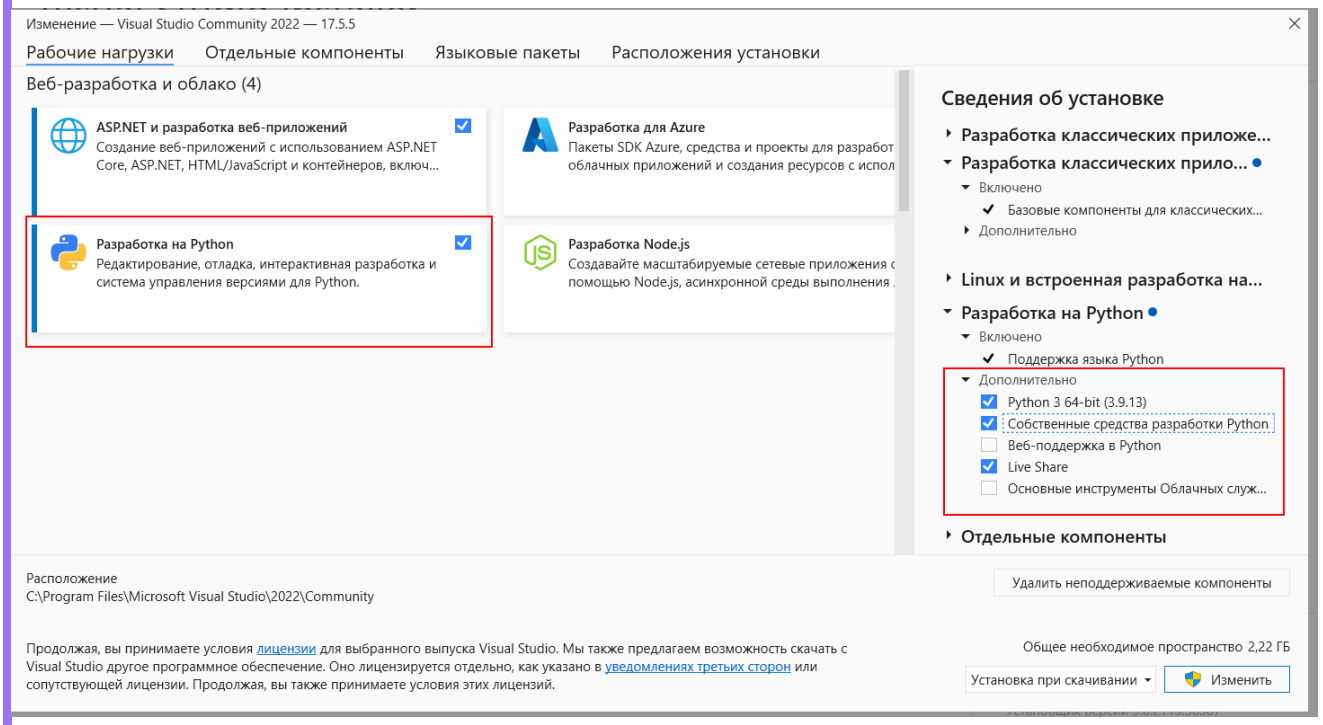


Python у Visual Studio

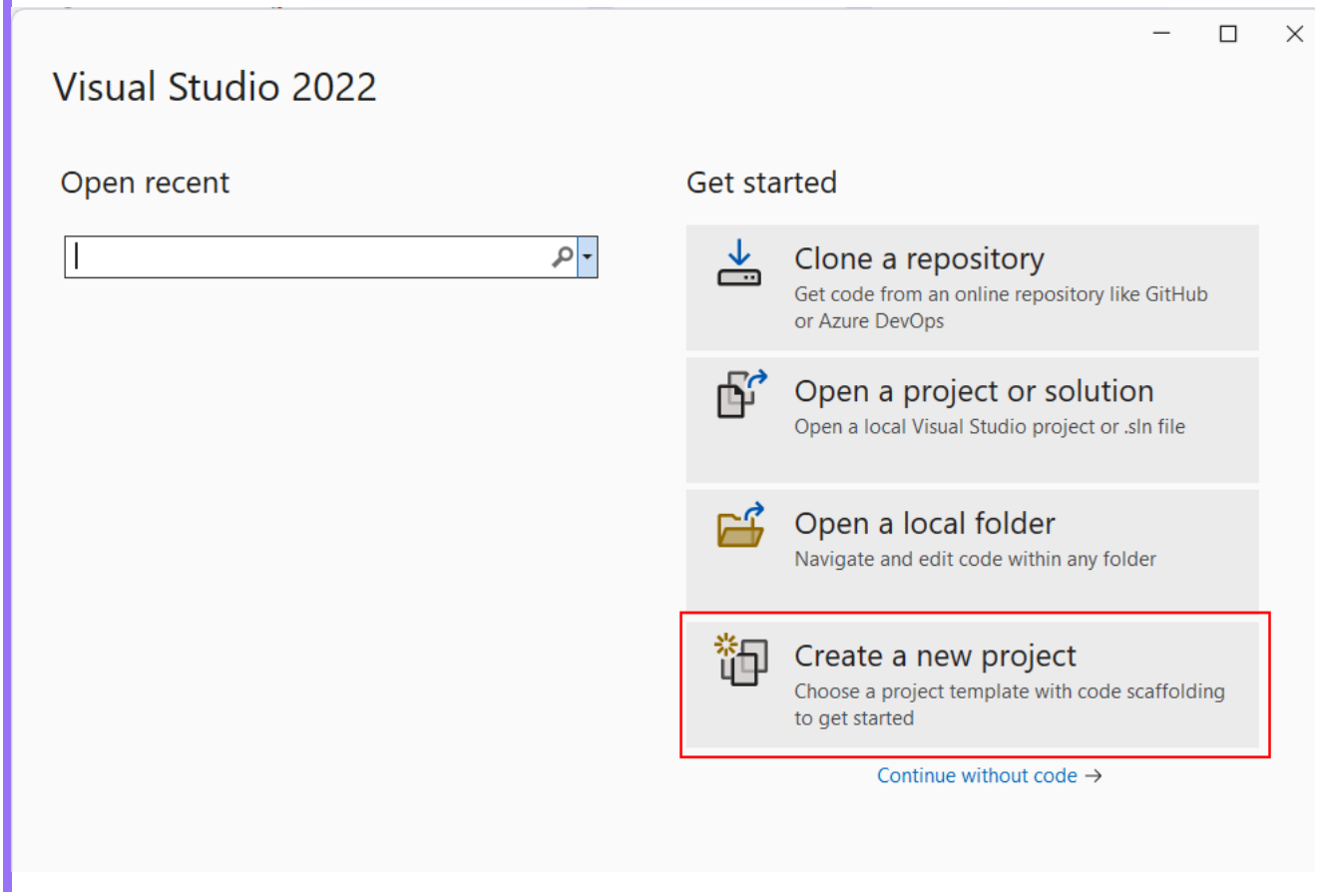
Одним із середовищ розробки, що дозволяє працювати з Python, є Visual Studio. Перевагою даної IDE порівняно, скажімо, з PyCharm, слід зазначити насамперед те, що в її безкоштовній редакції Visual Studio Community безкоштовно доступні ряд функцій і можливостей, які в тому ж PyCharm доступні лише платній версії Professional

Edition. Наприклад, це веб-розробка, у тому числі за допомогою різних фреймворків. У той же час засоби для розробки на Python у Visual Studo доступні поки що тільки у версії для Windows.

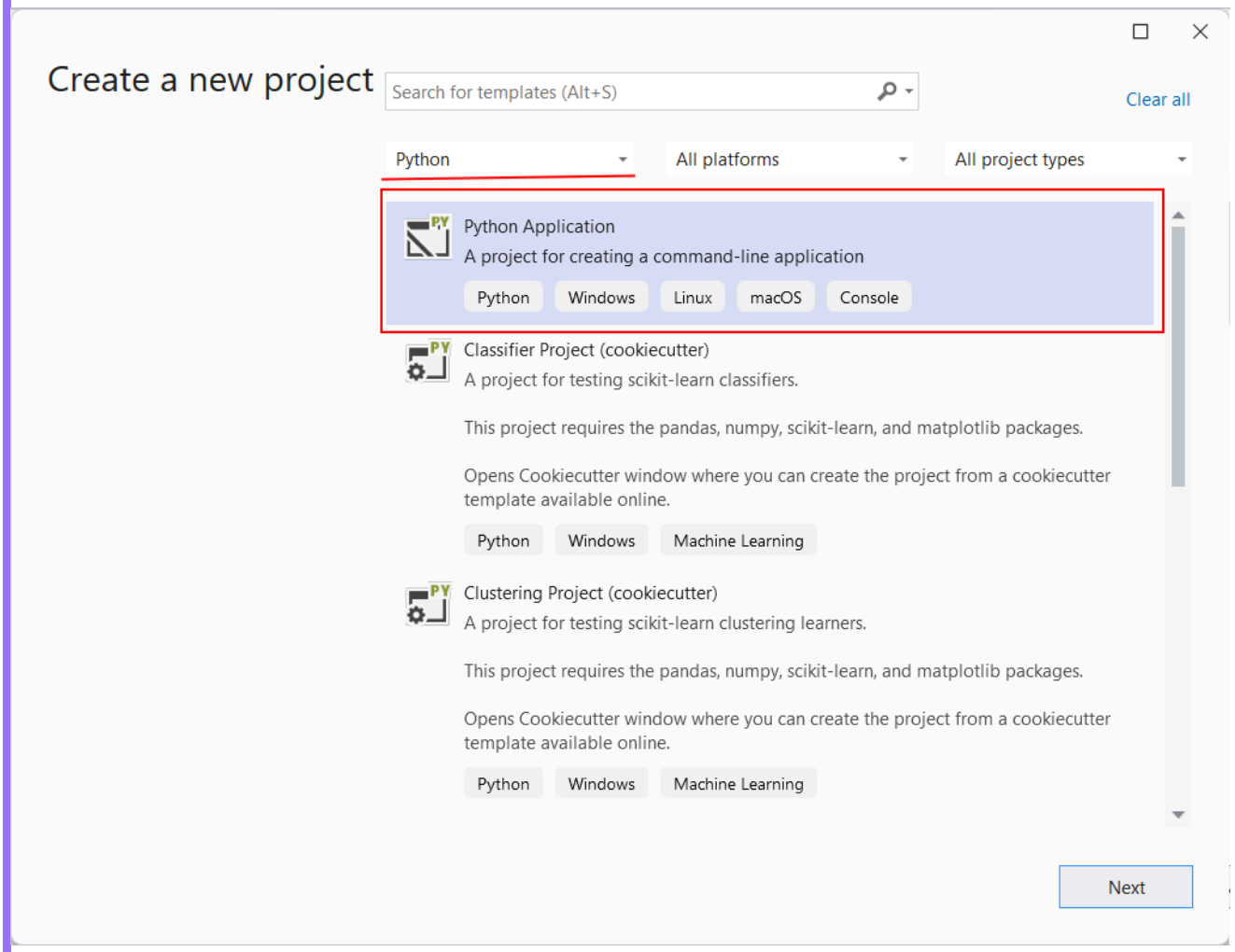
Отже, завантажимо інсталяційний файл Visual Studio Community за посиланням https://visualstudio.microsoft.com/ru/vs/community/ . Після запуску інсталяційного файлу виберемо серед опцій пункт Розробка на Python :



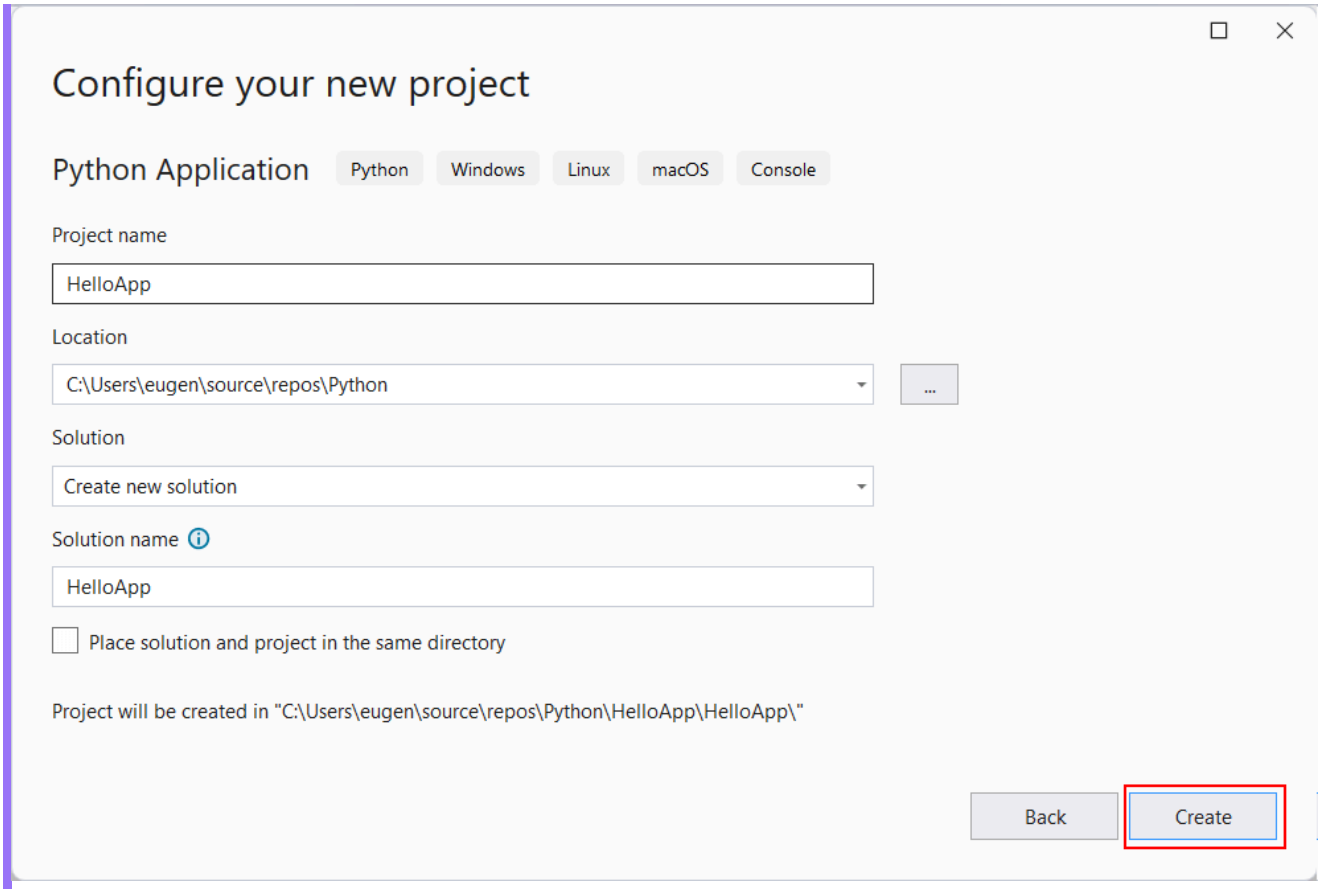
Після інсталяції Visual Studio запустимо її і у вікні програми оберемо Create a new project :



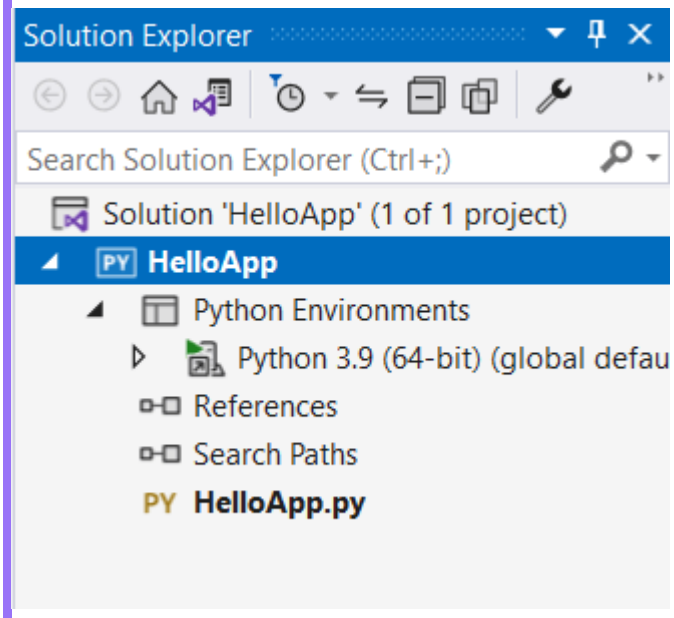
Далі у вікні створення нового проекту виберемо шаблон Python Application :



На наступному вікні вкажемо назву та шлях до проекту. Наприклад, у моєму випадку проект називатиметься "HelloApp":



Натисніть кнопку Create, і Visual Studio створить новий проект:



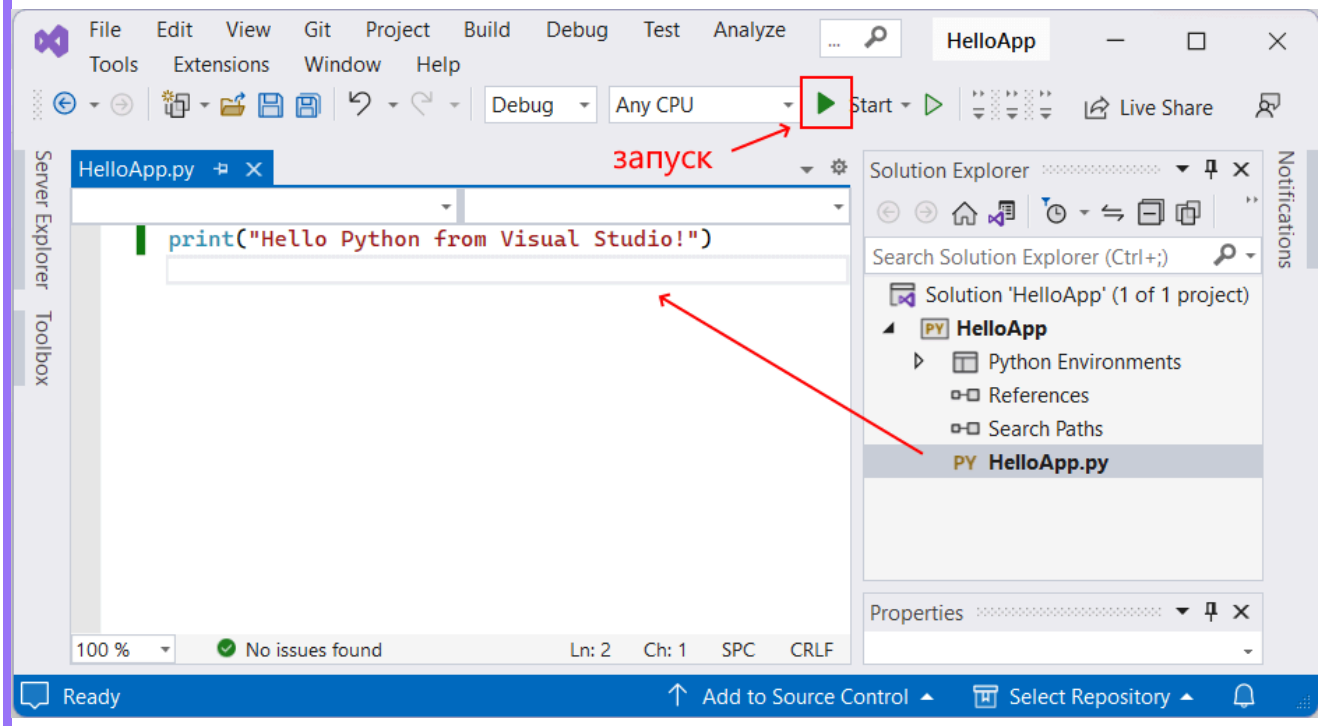
Праворуч у вікні Solution Explorer (Обозреватель рішень) можна побачити структуру проекту. За замовчуванням ми можемо побачити такі елементи:

- Python Environments : тут можна побачити всі використовувані середовища, зокрема тут можна версію Python, яка використовується.
- References : цей вузол містить всі зовнішні залежності, які використовуються поточним проектом
- Search Paths : цей вузол дозволяє вказати шляхи пошуку для модулів Python
- HelloApp.py : власне файл Python з вихідним кодом

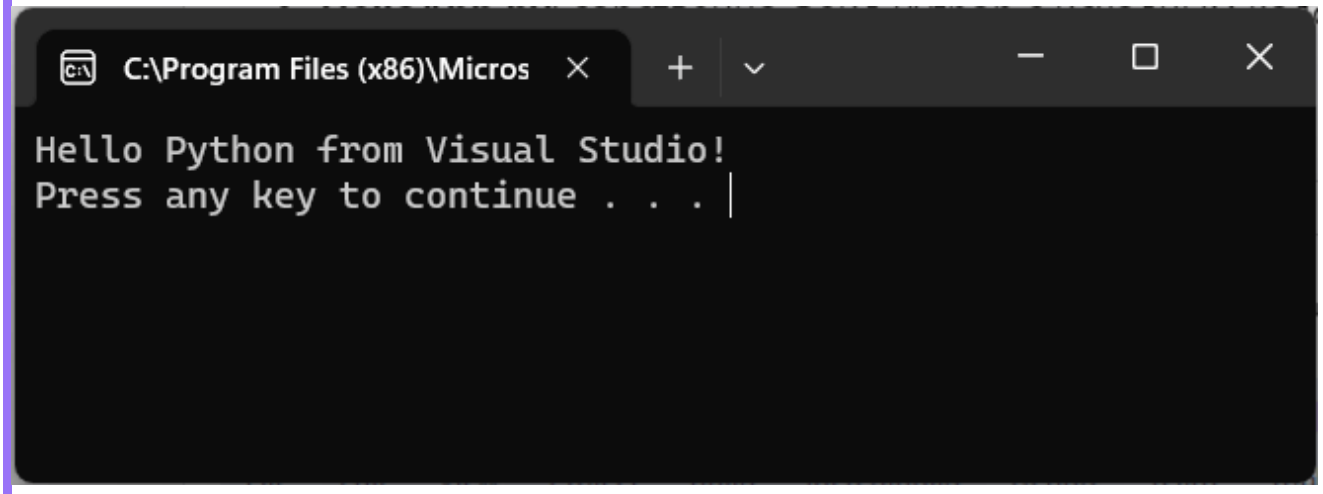
За промовчанням у Visual Studio вже відкрито файл HelloApp.py, але він поки що порожній. Додамо до нього наступний рядок:

```
print("Hello Python from Visual Studio!")
```

І потім на панелі інструментів натиснемо на зелену стрілочку для запуску:



В результаті запуску відобразиться консоль, яка виведе потрібний рядок:



Розділ 2. Основи Python

Введення у написання програм

Програма Python складається з набору інструкцій. Кожна інструкція міститься на новий рядок. Наприклад:

```
print(2 + 3)
print("Hello")
```

Велику роль у Python відіграють відступи. Неправильно поставлений відступ фактично є помилкою. Наприклад, у наступному випадку ми отримаємо помилку, хоча код буде практично аналогічний наведеному вище:

```
print(2 + 3)
    print("Hello")
```

Тому варто поміщати нові вказівки спочатку рядка. У цьому одна з важливих відмінностей пайтон від інших мов програмування, як C# або Java.

Однак, варто враховувати, що деякі конструкції мови можуть складатися з декількох рядків. Наприклад, умовна конструкція if :

```
if 1 < 2:
    print("Hello")
```

У разі якщо 1 менше 2, то виводиться рядок " Hello " . І тут має бути відступ, оскільки інструкція print("Hello") використовується не як така, бо як частина умовної конструкції if. Причому відступ, згідно з посібником з оформлення коду , бажано робити з такої кількості прогалин, яка кратна 4 (тобто 4, 8, 16 і т.д.) Хоча якщо відступів буде не 4, а 5, то програма також працюватиме.

Таких конструкцій не так багато, тому особливої плутанини щодо де треба, а де не треба ставити прогалини, не повинно виникнути.

Реєстрозалежність

Python - регістрозалежна мова, тому вирази `print` та `Print` або `PRINT` представляють різні вирази. І якщо замість методу print для виведення на консоль ми спробуємо використати метод Print:

```
Print("Hello World")
```

то в нас нічого не вийде.

Коментарі

Для позначки, що робить та чи інша ділянка коду, застосовуються коментарі. При трансляції та виконанні програми інтерпретатор ігнорує коментарі, тому вони не

впливають на роботу програми. Коментарі в Python бувають блокові та малі.

Рядкові коментарі передуються знаком решітки -# . Вони можуть розташовуватися на окремому рядку:

```
# Виведення на консоль
# повідомлення Hello World
print("Hello World")
```

Будь-який набір символів після знак# представляє коментар. Тобто у прикладі вище перші два рядки коду є коментарями.

Також вони можуть розташовуватися на тому ж рядку, що й інструкції мови після інструкцій, що виконуються:

```
print("Hello World")# Виведення повідомлення на консоль
```

У блокових коментарях до і після тексту коментаря ставляться три одинарні лапки: `'''текст комментария'''`. Наприклад:

```
'''
    Виведення на консоль
    повідомлення Hello World
'''
print("Hello World")
```

Основні функції

Python надає низку вбудованих функцій. Деякі їх використовуються дуже часто, особливо у початкових етапах вивчення мови, тому розглянемо їх.

Основною функцією виведення інформації на консоль є функція print() . Як аргумент у цю функцію передається рядок, який ми хочемо вивести:

```
print("Hello Python")
```

Якщо нам необхідно вивести кілька значень на консоль, ми можемо передати їх у функцію print через кому:

```
print("Full name:", "Tom", "Smith")
```

У результаті всі передані значення склеяться через прогалини в один рядок:

```
Full name: Tom Smith
```

Якщо функція print відповідає за виведення, функція input відповідає за введення інформації. Як необов'язковий параметр ця функція приймає запрошення до введення та повертає введений рядок, який ми можемо зберегти в змінний:

```
name = input("Введіть ім'я: ")
print("Привіт", name)
```

Консольний висновок:

```
Введіть ім'я: Нікіта
Привіт Нікіта
```

Змінні та типи даних

Змінні

Змінні призначені для зберігання даних. Назва змінної в Python повинна починатися з алфавітного символу або символу підкреслення і може містити алфавітно-цифрові символи і знак підкреслення. Крім того, назва змінної не повинна співпадати з назвою ключових слів мови Python. Ключових слів не так багато, їх легко запам'ятати:

```
False await else import pass
None break except in raise
True class finally is return
and continue for lambda try
as def from nonlocal while
assert del global not with
async elif if або yield
```

Наприклад, створимо змінну:

```
name = "Tom"
```

Тут визначено змінну `name`, яка зберігає рядок "Tom".

У пайтоні застосовується два типи найменування змінних: camel case та underscore notation .

Camel case має на увазі, що кожне нове підслівне в найменуванні змінної починається з великої літери. Наприклад:

```
userName = "Tom"
```

Underscore notation передбачає, що підслів в найменуванні змінної поділяються знаком підкреслення. Наприклад:

```
user_name = "Tom"
```

І також треба враховувати регістрозалежність, тому змінні `name` і `Name` представлятимуть різні об'єкти.

```
дві різні змінні
name = "Tom"
Name = "Tom"
```

Визначивши змінну ми можемо використовувати в програмі. Наприклад, спробувати вивести її на консоль за допомогою вбудованої функції
print :

```
name = "Tom"# Визначення змінної name
print(name)# виведення значення змінної name на консоль
```

Наприклад, визначення та застосування змінної в середовищі PyCharm:

 Змінні в Python

Відмінною рисою змінної є те, що ми можемо змінювати її значення протягом роботи програми:

```
name = "Tom"# змінної name дорівнює "Tom"
print(name)# виводить: Tom
name = "Bob"# змінюємо значення на "Bob"
print(name)# виводить: Bob
```

Типи даних

Змінна зберігає дані одного з типів даних. У Python існує безліч різних типів даних. В даному випадку розглянемо тільки базові типи:
bool , int , float , complex і str .

Логічні значення

Тип bool представляє два логічні значення: True (вірно, істина) або False (невірно, брехня). Значення
True служить у тому, щоб показати, що щось істинно. Тоді як значення False , навпаки, показує, що щось хибне. Приклад змінних даного типу:

```
isMarried = False
print(isMarried)# False
isAlive = True
print(isAlive)# True
```

Цілі числа

Тип int є цілим числом, наприклад, 1, 4, 8, 50. Приклад

```
age = 21
print("Вік:", age)# Вік: 21
count = 15
print("Кількість:", count)# Кількість: 15
```

За замовчуванням стандартні числа розцінюються як числа у десятковій системі. Але Python також підтримує числа у двійковій, вісімковій та шістнадцятковій системах.

Для вказівки, що число є двійковою системою, перед числом ставиться префікс 0b :

```
a = 0b11
b = 0b1011
c = 0b100001
print(a)# 3 у десятковій системі
print(b)# 11 у десятковій системі
print(c)# 33 у десятковій системі
```

Для вказівки, що число представляє вісімкову систему, перед числом ставиться префікс 0o :

```
a = 0o7
b = 0o11
c = 0o17
print(a)# 7 у десятковій системі
print(b)# 9 у десятковій системі
print(c)# 15 у десятковій системі
```

Для вказівки, що число є шістнадцятковою системою, перед числом ставиться префікс 0x :

```
a = 0x0A
b = 0xFF
c = 0xA1
print(a)# 10 у десятковій системі
print(b)# 255 у десятковій системі
print(c)# 161 у десятковій системі
```

У будь-якій системі ми не передали число в функцію print для виведення на консоль, воно за умовчанням буде виводитися в десятковій системі.

Дробові числа

Тип float є число з плаваючою точкою, наприклад, 1.2 або 34.76. У якості роздільника цілої та дробової частин використовується крапка.

```
height = 1.68
pi = 3.14
weight = 68.
print(height)# 1.68
print(pi)# 3.14
print(weight)# 68.0
```

Число з плаваючою точкою можна визначати в експоненційному записі:

```
x = 3.9e3
print(x)# 3900.0
x = 3.9e-3
print(x)# 0.0039
```

Число float може мати лише 18 значущих символів. Так, у цьому випадку використовуються лише два символи – 3.9. І якщо число занадто велике чи занадто мало, ми можемо записувати число у подібній нотації, використовуючи експоненту. Число після експоненти вказує ступінь числа 10, яке треба помножити основне число - 3.9.

Комплексні числа

Тип complex представляє комплексні числа у форматі - після уявної частини вказується суфікс j `вещественная_часть+мнимая_частьj`

```
complexNumber = 1+2j
print(complexNumber)# (1+2j)
```

Рядки

Тип str представляє рядки. Рядок представляє послідовність символів, укладену в одинарні або подвійні лапки, наприклад "hello" та "hello". У Python 3.x рядки представляють набір символів у кодуванні Unicode

```
message = "Hello World!"
print(message)# Hello World!
name = 'Tom'
print(name)# Tom
```

При цьому, якщо рядок має багато символів, його можна розбити на частини та ці частини розмістити на різних рядках коду. У цьому випадку весь рядок полягає в круглі дужки, а її окремі частини - у лапки:

```
text = ("Laudate omnes gentes laudate"
        "Magnificat in secula")
print(text)
```

Якщо ми хочемо визначити багаторядковий текст, такий текст полягає у потрійні подвійні чи одинарні лапки:

```
'''
Це коментар
'''
text = '''Laudate omnes gentes laudate
Magnificat in secula
Et anima mea laudate
Magnificat in secula
'''
print(text)
```

При використанні потрійних одинарних лапок не варто плутати їх із коментарями: якщо текст у потрійних одинарних лапках присвоюється змінним, то це рядок, а не коментар.

Керуючі послідовності у рядку

Рядок може містити ряд спеціальних символів - послідовностей, що управляють. Деякі з них:

- \ : дозволяє додати всередину рядки слєш
- ' : дозволяє додати всередину рядка одинарну лапку
- " : дозволяє додати всередину рядка подвійну лапку
- \n : здійснює перехід на новий рядок

- \t : додає табуляцію (4 відступи)

Застосуємо кілька послідовностей:

```
text = "Message:\n\"Hello World\""
print(text)
```

Консольний висновок програми:

```
Message:
"Hello World"
```

Хоча подібні послідовності можуть допомогти в деяких справах, наприклад, помістити в рядок лапку, зробити табуляцію, перенесення на інший рядок. Але вони також можуть заважати. Наприклад:

```
path = "C:\python\name.txt"
print(path)
```

Тут змінна path містить певний шлях до файлу. Однак усередині рядка зустрічаються символи "\n", які будуть інтерпретовані як послідовність, що управляє. Так, ми отримаємо наступний консольний висновок:

```
C:\python
ame.txt
```

Щоб уникнути подібної ситуації, перед рядком ставиться символ r

```
path = r"C:\python\name.txt"
print(path)
```

Вставка значень у рядок

Python дозволяє встроювати рядок значення інших змінних. Для цього всередині рядка змінні розміщуються у фігурних дужках {}, а перед усім рядком ставиться символ f :

```
userName = "Tom"
userAge = 37
user = f"name: {userName} age: {userAge}"
print(user)# name: Tom age: 37
```

У цьому випадку на місце `{userName}` вставлятиметься значення змінної userName. Аналогічно замість `{userAge}`

буде вставлятися значення змінної userAge.

Динамічна типізація

Python є мовою з динамічною типізацією. А це означає, що змінна не прив'язана до певного типу.

Тип змінної визначається виходячи із значення, яке їй надано. Так, при присвоєнні рядка в подвійних чи одинарних лапках змінна має тип str . При присвоєнні цілого числа Python автоматично визначає тип змінної як int . Щоб визначити змінну як об'єкт float, їй надається дробове число, в якому роздільником цілої та дробової частини є точка.

При цьому в процесі роботи програми ми можемо змінити тип змінної, надавши їй значення іншого типу:

```
userId = "abc"# тип str
print(userId)
userId = 234# тип int
print(userId)
```

За допомогою вбудованої функції type() динамічно можна дізнатися про поточний тип змінної:

```
userId = "abc"# тип str
print(type(userId))# <class 'str'>
userId = 234# тип int
print(type(userId))# <class 'int'>
```

Консольне введення та виведення

Виведення на консоль

Для виведення інформації на консоль призначена вбудована функція print() . При виклику цієї функції їй у дужках передається значення:

```
print("Hello METANIT.COM")
```

Цей код виведе нам на консоль рядок "Hello METANIT.COM".

Відмінною особливістю цієї функції є те, що за умовчанням вона виводить значення на окремому рядку. Наприклад:

```
print("Hello World")
print("Hello METANIT.COM")
print("Hello Python")
```

Тут три дзвінки функції `print()` виводять деяке повідомлення. Причому при виведенні на консоль кожне повідомлення розміщуватиметься на окремому рядку:

```
Hello World
Hello METANIT.COM
Hello Python
```

Така поведінка не завжди зручна. Наприклад, ми хочемо, щоб усі значення виводилися на одному рядку. Для цього нам потрібно настроїти поведінку функції за допомогою параметра end . Цей параметр визначає символи, які додаються в кінці до рядка і . При застосуванні параметра end виклик функції `print()` виглядає так:

```
print(значення, end = кінцеві_символи)
```

За промовчанням `end` дорівнює символу `"\n"`, який задає переклад наступного рядка. Власне тому функція print за замовчуванням виводить значення, що їй передається, на окремому рядку.

Тепер визначимо, щоб функція не робила переведення на наступний рядок, а виводила значення на тому ж рядку:

```
print("Hello World", end=" ")
print("Hello METANIT.COM", end=" ")
print("Hello Python")
```

Тобто тепер значення, що виводяться, будуть розділятися пробілом:

```
Hello World Hello METANIT.COM Hello Python
```

Причому це може бути не один символ, а набір символів:

```
print("Hello World", end=" and ")
print("Hello METANIT.COM", end=" and ")
print("Hello Python")
```

В даному випадку повідомлення, що виводяться, будуть відокремлюватися символами `" and "`:

```
Hello World and Hello METANIT.COM and Hello Python
```

Консольне введення

Поряд із виведенням на консоль ми можемо отримувати введення користувача з консолі, отримувати дані, що вводяться. Для цього Python визначена функція input() . У цю функцію надсилається запрошення до введення. А результат введення ми можемо зберегти змінну. Наприклад, визначимо код для введення користувачем імені:

```
name = input("Введіть своє ім'я: ")
print(f"Ваше ім'я: {name}")
```

У цьому випадку функцію `input()` передається запрошення до введення у вигляді рядка "Введіть своє ім'я: ". Результат функції - результат введення користувача передається до змінної `name` . Потім ми можемо вивести значення цієї змінної консоль за допомогою функції `print()` . Приклад роботи коду:

```
Введіть своє ім'я: Nikita
Ваше ім'я: Nikita
```

Ще приклад із введенням кількох значень:

```
name = input("Your name: ")
age = input("Your age: ")
print(f"Name: {name} Age: {age}")
```

Приклад роботи програми:

```
Your name: Tom
Your age: 37
Name: Tom Age: 37
```

Варто враховувати, що усі введені значення розглядаються як значення типу str , тобто рядки. І навіть якщо ми вводимо число, як у другому випадку у коді вище, то Python все одно розглядатиме введене значення як рядок, а не як число.

Арифметичні операції з числами

Python підтримує всі поширені арифметичні операції:

- Додавання двох чисел:

```
print(6 + 2)# 8
```

-

  Віднімання двох чисел:

```
print(6 - 2)# 4
```

- *

  Розмноження двох чисел:

```
print(6 * 2)# 12
```

- /

  Розподіл двох чисел:

```
print(6 / 2)# 3.0
```

- //

  Цілочисельне розподіл двох чисел:

```
print(7 / 2)# 3.5
print(7 // 2)# 3
```

  Ця операція повертає цілий результат поділу, відкидаючи дробову частину

- **

  Зведення у ступінь:

```
print(6 ** 2)# Зводимо число 6 до ступеня 2. Результат - 36
```

- %

  Отримання залишку від розподілу:

```
print(7 % 2)# Отримання залишку від розподілу числа 7 на 2. Результат - 1
```

  В даному випадку найближче число до 7, яке ділиться на 2 без залишку, це 6. Тому залишок від поділу дорівнює 7 - 6 = 1

При послідовному використанні кількох арифметичних операцій їх виконання здійснюється відповідно до пріоритету. Спочатку виконуються операції з великим пріоритетом. Пріоритети операцій у порядку зменшення наведені в наступній таблиці.

Операції	Напрям
**	Справа наліво
* // // %	Зліва направо
+ -	Зліва направо

Нехай у нас виконується такий вираз:

```
number = 3 + 4 * 5 ** 2 + 7
print(number)# 110
```

Тут на початку виконується зведення у ступінь (5**2) як операція з великим пріоритетом, далі результат множиться на 4 (25*4), потім відбувається додавання (3+100) і далі знову йде додавання (103+7).

Щоб перевизначити порядок операцій, можна використовувати дужки:

```
number = (3 + 4) * (5 ** 2 + 7)
print(number)# 224
```

Слід зазначити, що у арифметичних операціях можуть брати участь як цілі, і дробові числа. Якщо однієї операції бере участь ціле число (int) і з плаваючою точкою (float), то ціле число наводиться до типу float.

Арифметичні операції із присвоєнням

Ряд спеціальних операцій дозволяють використовувати результат операції першому операнду:

- +=

  Присвоєння результату додавання
- -=

  Присвоєння результату віднімання
- *=

  Присвоєння результату множення
- /=

  Присвоєння результату від розподілу
- //=

  Присвоєння результату цілісного поділу
- **=

Присвоєння ступеня числа

- %=

Присвоєння залишку від розподілу

Приклади операцій:

```
number = 10
number += 5
print(number)# 15
number -= 3
print(number)# 12
number *= 4
print(number)# 48
```

Округлення та функція round

При операціях з числами типу float треба враховувати, що результати операцій з ними може бути не зовсім точним. Наприклад:

```
first_number = 2.0001
second_number = 5
third_number = first_number/second_number
print(third_number)# 0.40002000000000004
```

В даному випадку ми очікуємо отримати число 0.40002, проте в кінці через ряд нулів з'являється ще якась четвірка. Або ще один вираз:

```
print(2.0001 + 0.1)# 2.1001000000000003
```

У разі вище заокруглення результату ми можемо використовувати вбудовану функцію round() :

```
first_number = 2.0001
second_number = 0.1
third_number = first_number + second_number
print(round(third_number))# 2
```

У функцію `round()` передається число, яке треба округлити. Якщо функцію передається одне число, як у прикладі вище, воно округляється до цілого.

Функція `round()` також може приймати друге число, яке вказує, скільки знаків після коми має містити число, що отримується:

```
first_number = 2.0001
second_number = 0.1
third_number = first_number + second_number
print(round(third_number, 4))# 2.1001
```

У разі число `third_number` округляється до 4 знаків після коми.

Якщо в функцію передається лише одне значення - тільки округлене число, воно округляється то найближчого цілого

Приклади заокруглень:

```
округлення до цілого числа
print(round(2.49))# 2 - округлення до найближчого цілого 2
print(round(2.51))# 3
```

Однак якщо округлена частина дорівнює однаково віддалена від двох цілих чисел, то округлення йде до найближчого парного:

```
print(round(2.5))# 2 - найближче парне
print(round(3.5))# 4 - найближче парне
```

Округлення проводиться до найближчого кратного 10 в ступені мінус округлена частина:

```
округлення до двох знаків після коми
print(round(2.554, 2))# 2.55
print(round(2.5551, 2))# 2.56
print(round(2.554999, 2))# 2.55
print(round(2.499, 2))# 2.5
```

Однак слід враховувати, що функція `round()` не є ідеальним інструментом. Наприклад, вище за округлення до цілих чисел застосовується правило, згідно з яким, якщо округлена частина однаково віддалена від двох значень, округлення проводиться до найближчого парного значення. У Python у зв'язку з тим, що десяткова частина числа не може бути точно представлена у вигляді числа float, це може призводити до деяких не зовсім очікуваних результатів. Наприклад:

```
округлення до двох знаків після коми
print(round(2.545, 2))# 2.54
print(round(2.555, 2))# 2.56 - округлення до парного
print(round(2.565, 2))# 2.56
print(round(2.575, 2))# 2.58
print(round(2.655, 2))# 2.65 - округлення не до парного
```

```
print(round(2.665, 2))# 2.67
print(round(2.675, 2))# 2.67
```

Подібно до проблеми можна почитати до документації .

Додаткові матеріали

- Запитання для самоперевірки
- Вправи для самоперевірки

Умовні вирази

Ряд операцій представляють умовні висловлювання. Всі ці операції приймають два операнди і повертають логічне значення, яке Python представляє тип bool . Існує тільки два логічні значення - True (вираз істинно) і False (вираз хибно).

Операції порівняння

Найпростіші умовні вирази становлять операції порівняння, які порівнюють два значення. Python підтримує наступні операції порівняння:

- ==

  Повертає True, якщо обидва операнди рівні. Інакше повертає False.
- !=

  Повертає True, якщо обидва операнди НЕ рівні. Інакше повертає False.
- (більше ніж)

  Повертає True, якщо перший операнд більший за другий.
- < (менше ніж)

  Повертає True, якщо перший операнд менший за другий.
- = (більше чи одно)

  Повертає True, якщо перший операнд більше або дорівнює другому.
- <= (менше чи одно)

  Повертає True, якщо перший операнд менший або дорівнює другому.

Приклади операцій порівняння:

```
a = 5
b = 6
```

```
result = 5 == 6# зберігаємо результат операції у змінну
print(result)# False - 5 не дорівнює 6
print(a != b)# True
print(a > b)# False - 5 менше 6
print(a < b)# True
bool1 = True
bool2 = False
print(bool1 == bool2)# False - bool1 не дорівнює bool2
```

Операції порівняння можуть порівнювати різні об'єкти - рядки, числа, логічні значення, проте обидва операнди операції повинні представляти один і той же тип.

Логічні операції

Для створення складових умовних виразів використовуються логічні операції. У Python є такі логічні оператори:

- Оператор and (логічне множення) застосовується до двох операндів:

  ```
  x and y
  ```

  Спочатку оператор and оцінює вираз x, і якщо він дорівнює False, то повертається його значення. Якщо воно дорівнює True, то оцінюється другий операнд – y і повертається значення y.

  ```
  age = 22
  weight = 58
  result = age > 21 and weight == 58
  print(result)# True
  ```

  У разі оператор and порівнює результати двох выражений: `age > 21` `weight == 58`. І якщо обидва ці висловлювання повертають True, то оператор and також повертає True (формально повертається значення останнього операнда).

  Але операндами оператора and необов'язково виступають значення True і False. Це можуть бути будь-які значення. Наприклад:

  ```
  result = 4 and "w"
  print(result)# w, тому що 4 дорівнює True, тому повертається значення останнього операнда
  result = 0 and "w"
  print(result)# 0, тому що 0 еквівалентно False
  ```

У даному випадку число 0 і порожній рядок "" розглядаються як False, всі інші числа та непусті рядки еквівалентні True

- or (логічне додавання) також застосовується до двох операндів:

```
x or y
```

Спочатку оператор or оцінює вираз x, і якщо він дорівнює True, то повертається його значення. Якщо воно дорівнює False, то оцінюється другий операнд – y і повертається значення y. Наприклад

```
age = 22
isMarried = False
result = age > 21 or isMarried
print(result)# True, тому що вираз age > 21 дорівнює True
```

Також оператор or може застосовуватися до будь-яких значень. Наприклад:

```
result = 4 or "w"
print(result)# 4, тому що 4 еквівалентно True, тому повертається
значення першого операнда
result = 0 or "w"
print(result)# w, тому що 0 еквівалентно False, тому повертається
значення останнього операнда
```

- not (логічне заперечення)

Повертає True, якщо вираз дорівнює False

```
age = 22
isMarried = False
print(not age > 21)# False
print(not isMarried)# True
print(not 4)# False
print(not 0)# True
```

Оператор in

Оператор in повертає, `True` якщо в деякому наборі значень є певне значення. Він має таку форму:

```
значення in набор_значень
```

Наприклад, рядок представляє набір символів. І за допомогою оператора `in` ми можемо перевірити, чи є в ній якийсь підрядок:

```
message = "hello world!"
hello = "hello"
print(hello in message)# True - підрядок hello є в рядку "hello world!"
gold="gold"
print(gold in message)# False - підрядки "gold" немає в рядку "hello
world!"
```

Якщо нам треба навпаки перевірити, чи немає в наборі значень будь-якого значення, ми можемо використовувати модифікацію оператора - not in . Вона повертає `True` , якщо в наборі значень немає певного значення:

```
message = "hello world!"
hello = "hello"
print(hello not in message)# False
gold="gold"
print(gold not in message)# True
```

Додаткові матеріали

- Запитання для самоперевірки

Умовна конструкція if

Умовні конструкції використовують умовні вирази і в залежності від їх значення спрямовують виконання програми по одному із шляхів. Одна з таких конструкцій - це конструкція if . Вона має таке формальне визначення:

```
if логічний_вираз:
    інструкції
[elif логічний вираз:
    інструкції]
[else:
    інструкції]
```

У найпростішому вигляді після ключового слова if йде логічний вираз. І якщо цей логічний вираз повертає True, то виконується наступний блок інструкцій, кожна з яких повинна починатися з нового рядка і повинна мати відступи від початку виразу `if` (відступ бажано робити в 4 пробіли або кількість прогалин, яке кратно 4):

```
language = "english"
if language == "english":
```

```
    print("Hello")
print("End")
```

Оскільки в даному випадку значення змінної мови дорівнює "english", то буде виконуватися блок if, який містить тільки одну інструкцію - `print("Hello")`. У результаті консоль виведе такі рядки:

```
Hello
End
```

Зверніть увагу на останній рядок, який виводить повідомлення "End". Вона не має відступів від початку рядка, тому вона не належить до блоку if і виконуватиметься у будь-якому випадку, навіть якщо вираз у конструкції if поверне False.

Але якби ми поставили відступи, то вона також належала б до конструкції if:

```
language = "english"
if language == "english":
    print("Hello")
    print("End")
```

Блок else

Якщо раптом нам треба визначити альтернативне рішення на той випадок, якщо вираз у if поверне False, то ми можемо використати блок else :

```
language = "російський"
if language == "english":
    print("Hello")
else:
    print("Привіт")
print("End")
```

Якщо вираз `language == "english"` повертає True, виконується блок if, інакше виконується блок else. І оскільки в даному випадку умова `language == "english"` повертає False, то виконуватиметься інструкція із блоку `else`.

Причому інструкції блоку else також повинні мати відступи від початку рядка. Наприклад, у прикладі вище `print("End")` немає відступу, тому вона входить у блок `else` і виконуватиметься незалежно, чому одно умова `language == "english"`. Тобто консоль нам виведе наступні рядки:

```
Привіт
```

```
End
```

Блок else також може мати кілька інструкцій, які повинні мати відступ від початку рядка:

```
language = "російський"
if language == "english":
    print("Hello")
    print("World")
else:
    print("Привіт")
    print("світ")
```

elif

Якщо необхідно ввести кілька альтернативних умов, можна використовувати додаткові блоки elif , після якого йде блок інструкцій.

```
language = "німецький"
if language == "english":
    print("Hello")
    print("World")
elif language == "німецький":
    print("Hallo")
    print("Welt")
else:
    print("Привіт")
    print("світ")
```

Спочатку Python перевіряє вираз `if` . Якщо воно дорівнює True, виконуються інструкції з блоку if. Якщо ця умова повертає False, то Python перевіряє вираз із `elif` .

Якщо вираз після `elif` одно `True` , то виконуються інструкції з блоку `elif` . Але якщо воно одно,
`False` то виконуються інструкції з блоку `else`

За потреби можна визначити декілька блоків `elif` для різних умов. Наприклад:

```
language = "німецький"
if language == "english":
    print("Hello")
elif language == "німецький":
    print("Hallo")
elif language == "french":
    print("Salut")
```

```
else:
    print("Привіт")
```

Вкладені конструкції if

Конструкція if у свою чергу сама може мати вкладені конструкції if:

```
language = "english"
daytime = "morning"
if language == "english":
    print("English")
    if daytime == "morning":
        print("Good morning")
    else:
        print("Good evening")
```

Тут конструкція if містить вкладену конструкцію if/else. Тобто якщо змінна language дорівнює "english", тоді вкладена конструкція if/else додатково перевіряє значення змінної daytime - чи дорівнює вона рядку "morning" чи ні. І в даному випадку ми отримаємо наступний консольний висновок:

```
English
Good morning
```

Варто враховувати, що вкладені вирази if також повинні починатися з відступів, а інструкції у вкладених конструкціях повинні мати відступи. Відступи, розставлені належним чином, можуть змінити логіку програми. Так, попередній приклад не аналогічний наступному:

```
language = "english"
daytime = "morning"
if language == "english":
    print("English")
if daytime == "morning":
    print("Good morning")
else:
     print("Good evening")
```

Подібним чином можна розміщувати вкладені конструкції if/elif/else у блоках elif та else:

```
language = "російський"
daytime = "morning"
if language == "english":
    if daytime == "morning":
```

```
        print("Good morning")
    else:
        print("Good evening")
else:
    if daytime == "morning":
        print("Доброго ранку")
    else:
        print("Добрий вечір")
```

Додаткові матеріали

- Запитання для самоперевірки
- Вправи для самоперевірки

Цикли

Цикли дозволяють виконувати певну дію залежно від дотримання певної умови. У мові Python є такі типи циклів:

- while
- for

Цикл while

Цикл while перевіряє істинність деякого умови, і якщо умова істинно, то виконує інструкції циклу. Він має таке формальне визначення:

```
while умовний_вираз:
   інструкції
```

Після ключового слова while вказується умовний вираз, і доки цей вираз повертає значення `True`, виконуватиметься блок інструкцій, що йде далі.

Усі інструкції, які відносяться до циклу while, розташовуються на наступних рядках і повинні мати відступ від початку ключового слова while.

```
number = 1
while number < 5:
    print(f"number = {number}")
    number += 1
print("Робота програми завершена")
```

В даному випадку цикл while буде виконуватися, поки змінна number менше 5.

Сам блок циклу і двох інструкцій:

```
print(f"number = {number}")
number += 1
```

Зверніть увагу, що вони мають відступи від початку оператора while - у разі від початку рядка. Завдяки цьому Python може визначити, що вони належать циклу. У самому циклі спочатку виводиться значення змінної number, а потім їй надається нове значення. .

Також зверніть увагу, що остання інструкція `print("Работа программы завершена")` не має відступів від початку рядка, тому вона не входить до циклу while.

Весь процес циклу можна представити так:

1. Спочатку перевіряється значення змінної number - чи воно менше 5. І оскільки спочатку змінна дорівнює 1, то ця умова повертає `True`, і тому виконуються інструкції циклу

   Інструкції циклу виводять на консоль рядок `number = 1`. І далі значення змінної number збільшується на одиницю - тепер вона дорівнює 2. Одноразове виконання блоку інструкцій циклу називається ітерацією . Тобто таким чином у циклі виконується перша ітерація.
2. Знову перевіряється умова `number < 5`. Воно, як і раніше, дорівнює True, тому що number = 2, тому виконуються інструкції циклу

   Інструкції циклу виводять на консоль рядок `number = 2`. І далі значення змінної number знову збільшується на одиницю - тепер вона дорівнює 3. Таким чином, виконується друга ітерація.
3. Знову перевіряється умова `number < 5`. Воно, як і раніше, дорівнює True, тому що number = 3, тому виконуються інструкції циклу

   Інструкції циклу виводять на консоль рядок `number = 3`. І далі значення змінної number знову збільшується на одиницю - тепер вона дорівнює 4. Тобто виконується третя ітерація.
4. Знову перевіряється умова `number < 5`. Воно, як і раніше, дорівнює True, тому що number = 4, тому виконуються інструкції циклу

   Інструкції циклу виводять на консоль рядок `number = 4`. І далі значення змінної number знову збільшується на одиницю - тепер вона дорівнює 5. Тобто виконується четверта ітерація.
5. І знову перевіряється умова `number < 5`. Але тепер воно дорівнює False , тому що number = 5, тому виконуються вихід із циклу. Усі цикл – завершився. Далі вже виконуються дії, визначені після циклу. Таким чином, даний цикл проведе чотири проходи або чотири ітерації

У результаті під час виконання коду ми отримаємо наступний консольний висновок:

```
number = 1
number = 2
number = 3
number = 4
Роботу програми завершено
```

Для циклу while також можна визначити додатковий блок else , інструкції якого виконуються, коли умова дорівнює False:

```
number = 1
while number < 5:
    print(f"number = {number}")
    number += 1
else:
    print(f"number = {number}. Робота циклу завершена")
print("Робота програми завершена")
```

Тобто в даному випадку спочатку перевіряється умова та виконуються інструкції while. Потім, коли умова стає рівною False, виконуються інструкції з блоку else. Зверніть увагу, що інструкції блоку else також мають відступи від початку конструкції циклу. У результаті ми отримаємо наступний консольний висновок:

```
number = 1
number = 2
number = 3
number = 4
number =5. Робота циклу завершена
Роботу програми завершено
```

Блок else може бути корисним, якщо умова спочатку дорівнює False, і ми можемо виконати деякі дії з цього приводу:

```
number = 10
while number < 5:
    print(f"number = {number}")
    number += 1
else:
    print(f"number = {number}. Робота циклу завершена")
print("Робота програми завершена")
```

В даному випадку умова `number < 5` спочатку дорівнює False, тому цикл не виконує жодної ітерації і відразу переходить до блоку else.

Цикл for

Інший тип циклів представляє конструкція for . Цей цикл пробігається по набору значень, поміщає кожне значення змінну, потім у циклі ми можемо з цієї змінної робити різні дії. Формальне визначення циклу for:

```
for змінна in набор_значень:
    інструкції
```

Після ключового слова for йде назва змінної, в яку будуть розміщуватись значення. Потім після оператора
in вказується набір значень та двокрапка.

А з наступного рядка розташовується блок інструкцій циклу, які повинні мати відступи від початку циклу.

При виконанні циклу Python послідовно отримує всі значення набору і передає їх змінну. Коли всі значення набору будуть перебрані, цикл завершує свою роботу.

Як набор значень, наприклад, можна розглядати рядок, який по суті представляє набір символів. Подивимося на прикладі:

```
message = "Hello"
for c in message:
    print(c)
```

У циклі визначається змінну `c` , після оператора `in` в якості набору, що перебирається, вказана змінна message, яка зберігає рядок "Hello". У результаті цикл for буде перебирати послідовно всі символи з рядка message і поміщати їх у змінну c. Блок самого циклу складається з однієї інструкції, яка виводить значення змінної з консоль. Консольний висновок програми:

```
H
e
l
l
o
```

Нерідко у зв'язці з циклом for застосовується вбудована функція range() , яка генерує числову послідовність:

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

Якщо функцію range передається один параметр, він означає максимальне значення діапазону чисел. У цьому випадку генерується послідовність від 0 до 10 (не включно). У результаті ми отримаємо наступний консольний висновок:

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

Також у функцію range() можна передати мінімальне значення діапазону

```
for n in range(4, 10):
    print(n, end=" ")
```

Тут генерується послідовність від 4 до 10 (не включаючи). Консольний висновок:

```
4 5 6 7 8 9
```

Також у функцію range() можна передати третій параметр, який вказує на збільшення:

```
for n in range(0, 10, 2):
    print(n, end=" ")
```

Тут генерується послідовність від 0 до 10 (не включаючи) із збільшенням 2.
Консольний висновок:

```
0 2 4 6 8
```

Цикл for також може мати додатковий блок else , який виконується після завершення циклу:

```
message = "Hello"
for c in message:
    print(c)
else:
    print(f"Останній символ: {c}. Цикл завершений");
print("Робота програми завершена")# інструкція не має відступу, тому не
відноситься до else
```

В даному випадку ми отримаємо наступний консольний висновок:

```
H
e
l
l
o
Останній символ: o. Цикл завершено
Роботу програми завершено
```

Варто зазначити, що блок else має доступ до всіх змінних, визначених у циклі for.

Вкладені цикли

Одні цикли в собі можуть містити інші цикли. Розглянемо з прикладу виведення таблиці множення:

```
i = 1
j = 1
while i < 10:
    while j < 10:
        print(i * j, end="\t")
        j += 1
    print("\n")
    j = 1
    i += 1
```

Зовнішній цикл `while i < 10:` спрацьовує 9 разів поки змінна і не дорівнюватиме 10. Усередині цього циклу спрацьовує внутрішній цикл `while j < 10:`. Внутрішній цикл також спрацьовує 9 разів поки змінна j не дорівнюватиме 10. Причому всі 9 ітерацій внутрішнього циклу спрацьовують в рамках однієї ітерації зовнішнього циклу.

У кожній ітерації внутрішнього циклу на консоль виводиться добуток чисел і та j. Потім значення змінної j збільшується на одиницю. Коли внутрішній цикл закінчив роботу, значень змінної j скидається до 1, а значення змінної і збільшується на одиницю і відбувається перехід до наступної ітерації зовнішнього циклу. І все повторюється, поки змінна і не дорівнюватиме 10. Відповідно внутрішній цикл спрацює всього 81 раз для всіх ітерацій зовнішнього циклу. У результаті ми отримаємо наступний консольний висновок:

```
1 2 3 4 5 6 7 8 9
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

Подібним чином можна визначати вкладені цикли for:

```
for c1 in "ab":
    for c2 in "ba":
        print(f"{c1}{c2}")
```

У цьому випадку зовнішній цикл проходить рядком "ab" і кожен символ поміщає в змінну c1. Внутрішній цикл проходить рядком "ba", поміщає кожен символ рядка змінну c2 і виводить поєднання обох символів на консоль. Тобто в результаті ми отримаємо всі можливі поєднання символів a і b:

```
ab
aa
bb
ba
```

Вихід із циклу. break і continue

Для управління циклом ми можемо використовувати спеціальні оператори break та continue . Оператор break здійснює вихід із циклу. А оператор continue виконує перехід до наступної ітерації циклу.

Оператор break може використовуватися, якщо циклі утворюються умови, які несумісні з його подальшим виконанням. Розглянемо наступний приклад:

```
number = 0
while number < 5:
    number += 1
    if number == 3 :# якщо number = 3, виходимо з циклу
        break
    print(f"number = {number}")
```

Тут цикл while перевіряє умову `number < 5` . І поки number не дорівнює 5, передбачається, що значення number буде виводитися на консоль. Однак усередині циклу також перевіряється інша умова: `if number == 3` . Тобто, якщо значення number дорівнює 3, то за допомогою оператора break виходимо з циклу. І в результаті ми отримаємо наступний консольний висновок:

```
number = 1
number = 2
```

На відміну від оператора, `break` оператор continue виконує перехід до наступної ітерації циклу без його завершення. Наприклад, у попередньому прикладі замінимо break на continue:

```
number = 0
while number < 5:
    number += 1
    if number == 3 :# якщо number = 3, переходимо до нової ітерації циклу
```

```
        continue
    print(f"number = {number}")
```

І в цьому випадку якщо значення змінної number дорівнює 3, наступні інструкції після оператора continue не виконуватимуться:

```
number = 1
number = 2
number = 4
number = 5
```

Додаткові матеріали

- Запитання для самоперевірки
- Вправи для самоперевірки