Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

В.Ю. Каштан, В.В. Гнатушенко, Д.В. Сущевський, Є.О. Обиденний

# Програмування комп’ютерних систем мовою Python. Частина 1

Навчальний наочний посібник

Дніпро
НТУ «ДП»
2023

УДК 351(075)
К31

Затверджено
вченою радою як навчальний наочний посібник для здобувачів ступеня бакалавра
спеціальностей 123 Комп'ютерна інженерія та 126 Інформаційні системи та технології
(протокол №4 від 20.03.2024 року)

**Рецензенти:**
**К.Ю. Островська,** канд. техн. наук, доц., доц. кафедри інформаційних технологій і систем Українського державного університету науки і технологій, м. Дніпро;
**О.С. Волковський,** канд. техн. наук, доц. кафедри комп'ютерних наук та інформаційних технологій Дніпровського національного університету імені Олеся Гончара, м. Дніпро.

**К31 Каштан В.Ю.** Програмування комп'ютерних систем мовою Python. Частина 1 : навч. наоч. посіб. / В.Ю. Каштан, В.В. Гнатушенко, Д.В. Сущевський, Є.О. Обиденний ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Електрон. дані. – Дніпро : НТУ «ДП», 2024. – 189 с. Режим доступу: … (дата звернення). – Назва з екрана.

Складено відповідно до навчальної програми з дисципліни «Програмування комп'ютерних систем мовою Python» за програмою підготовки бакалаврів зі спеціальностей 123 Комп'ютерна інженерія та 126 Інформаційні системи та технології . Розглянуто процес встановлення та налаштування середовища програмування; а також різні типи даних; математичні операції та функції, які можна використовувати у мові Python; умовні оператори і цикли для структурування програмного коду; обробку винятків для забезпечення стійкості програми. Описані концепції є необхідними для розробки ефективних функціональних програм у сфері інформаційних технологій.

Для здобувачів першого (бакалаврського) рівня вищої освіти спеціальностей 123 Комп'ютерна інженерія та 126 Інформаційні системи та технології.

УДК 351(075)

# ПЕРЕДМОВА

Навчальний наочний посібник "Програмування комп'ютерних систем мовою Python. Частина 1" призначено для здобувачів освіти, які тільки починають свій шлях у світ програмування. Цей посібник розкриє перед Вами всі основи застосування мови Python, від встановлення середовища програмування до освоєння умовних операторів, функцій та модулів.

У першій частині посібника ви знайдете матеріали про встановлення та налаштування середовища програмування мовою Python; засвоєння умовних операторів і циклів для структурування програмного коду; визначення функцій та їхнє використання, а також концепції модулів в організації коду; техніки обробки винятків для забезпечення стабільності програм.

У кінці кожної теми наведено перелік практичних завдань, які допоможуть закріпити отримані знання та навички в розробці програм мовою Python.

# ЗМІСТ



# Встановлення та налаштування робочого середовища для Python

# Історія мови

Python – динамічна інтерпретована об'єктно-орієнтована скриптова мова програмування із строгою динамічною типізацією. Розроблена в 1990 році голандським програмістом Гвідо ван Россумом.

Офіційний сайт мови програмування Python

https://www.python.org

*Гвідо ван Россум*

# Використання Python

Python використовується для різних цілей: для створення ігор і веб-застосунків, розробки внутрішніх інструментів для різноманітних проектів.

Мови програмування з часом змінюються - розробники додають в них нові можливості, а також виправляють помилки. Так з'являються різні версії мови. Наприклад, код написаний на Python 2 у більшості випадків не буде працювати у версії Python 3 без внесення додаткових змін.

! Основна команда розробників мови Python припиняє підтримку версії Python 2.x **з 1 січня 2020 року**.

.

# Від машинної мови до мови високого рівня

***Машинна мова*** *- це штучна мова, створена для передачі команд комп'ютеру. За допомогою машинної мови створюються ефективні програми, оскільки розробник отримує доступ до всіх можливостей процесора. Машинна мова -* ***мова низького рівня****.*

Коли процесор виконує інструкції програми, він бере участь у процесі, який є відомим як цикл **fetch - decode - execute** (**отримати - декодувати - виконати**).

- Отримати - **Програма**.
- Декодувати - **Інструкція машинної мови**.
- Виконати - Останній крок циклу.

# Від машинної мови до мови високого рівня


*Оскільки мова асемблера близька за своєю природою до машинної мови, вона є* ***мовою низького рівня****.*

# Від машинної мови до мови високого рівня

Мова програмування **високого рівня** дозволяє створювати складні програми, не знаючи, як працює процесор, і не записуючи великої кількості інструкцій низького рівня.
Наприклад, у **Python** для відображення повідомлення Hello, World! необхідно записати наступну інструкцію:

***print('Hello, World!')***

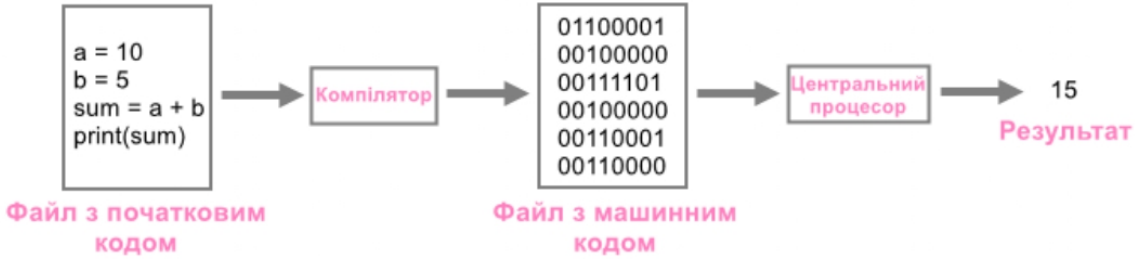
Мовою асемблера

```
section .text
    global _start
_start:
    mov edx, len
    mov ecx, msg
    mov ebx, 1
    mov eax, 4
    int 0x80
    mov eax, 1
    int 0x80

section .data

msg db  'Hello, World!',0xa
len equ $ - msg
```

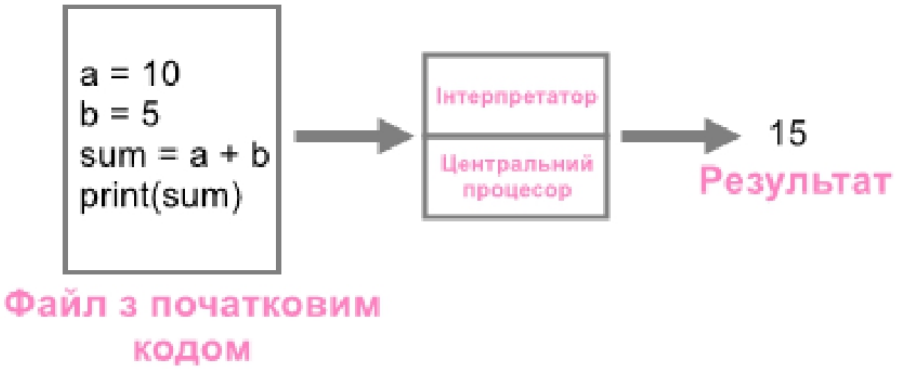
# Python - інтерпретована мова програмування

Для перекладу мови високого рівня на машинну мову доступні два типи програм:

- ❑ Компілятор
- ❑ Інтерпретатор



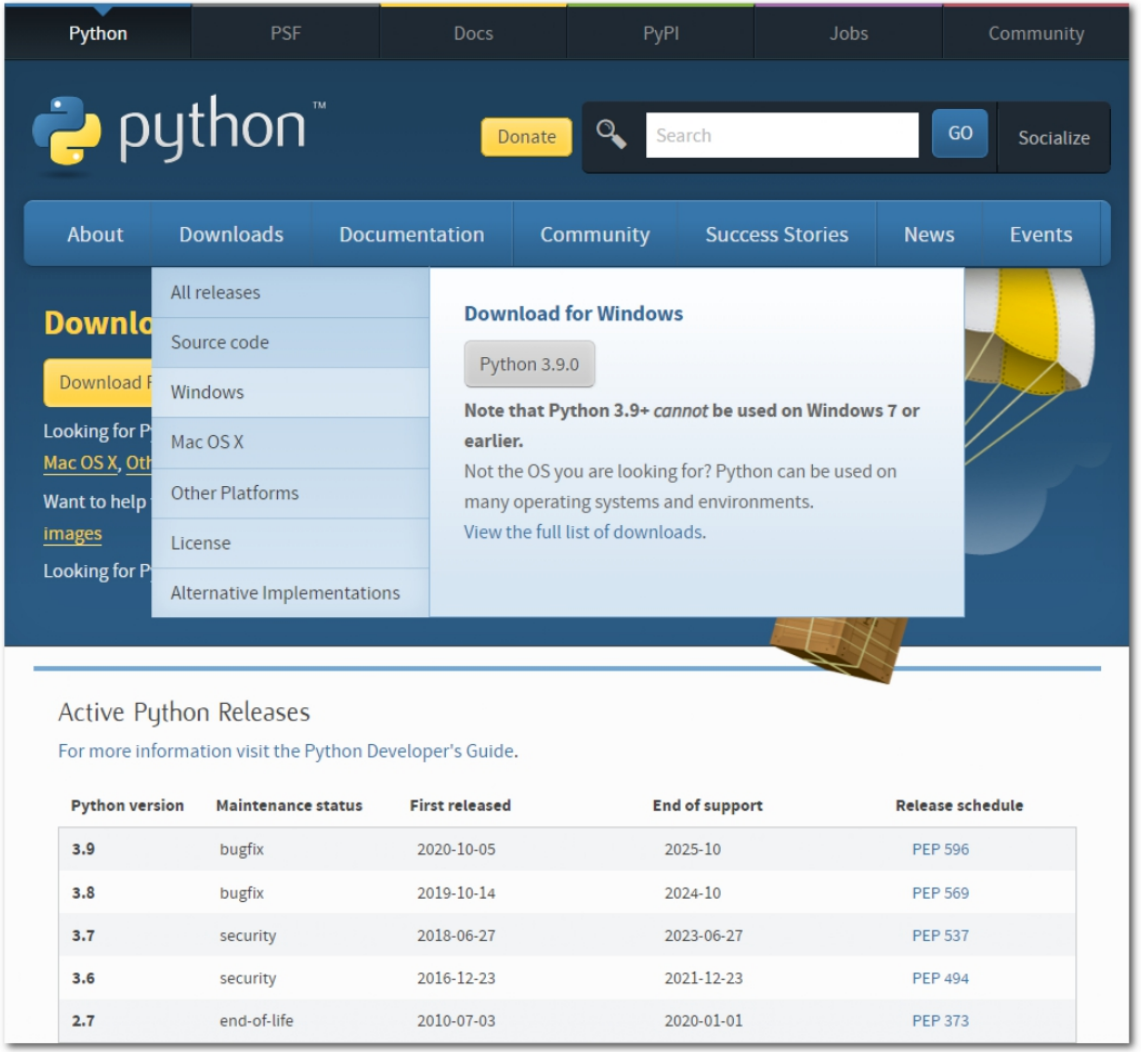

*Виконання програми компілятором*




*Виконання програми інтерпретатором*

# Завантаження Python

Версії інтерпретатора Python для різних операційних систем доступні для безкоштовного завантаження за адресою https://www.python.org/downloads



# Завантаження і встановлення Python

**для Windows**

Для встановлення Python, виконайте наступні дії:

! *Для операційної системи Windows XP остання підтримувана версія Python 3.4.3*.

З’ясуйте розрядність вашої операційної системи.

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

Оберіть версію Python.

Завантажте файл з розширенням .exe відповідної розрядності.

Встановіть Python:

відзначте рекомендований параметр Install launcher for all users не забудьте встановити прапорець Add Python 3.x to PATH (це полегшить правильне налаштування системи)

оберіть варіант налаштування установки Customize installation вкажіть каталог установки C:\PythonX (де X - номер версії)

# Налаштування середовища програмування

Кроки інсталяції IDLE

**На Windows:**

**1.Завантажте Python:** Перейдіть на офіційний веб-сайт Python та завантажте останню версію Python для Windows. Обирайте підходящий інсталятор для вашої системи (32-бітна або 64-бітна).

**2.Запустіть інсталятор:** Відкрийте завантажений файл інсталятора Python та слідуйте інструкціям майстра інсталяції.

**3.Активуйте "Add Python to PATH":** На етапі вибору компонентів переконайтеся, що обрано опцію "Add Python to PATH". Це спростить виклик Python з командного рядка.

**4.Встановіть Python:** Натискайте кнопку "Install Now" та зачекайте, поки інсталятор завершить процес.

**5.Перевірте інсталяцію:** Відкрийте командний рядок (Command Prompt) та введіть python --version для перевірки версії Python. Також, ви можете введенням команди python викликати інтерактивну оболонку Python.

**6.Запустіть IDLE:** Відкрийте IDLE, запустивши команду idle у командному рядку або шукайте IDLE в меню "Пуск".

# Налаштування середовища програмування

Кроки інсталяції Notepad++

**На Windows:**

**1.Завантажте інсталятор:** Перейдіть на офіційний веб-сайт Notepad++ та завантажте останню версію Notepad++.
**2.Запустіть інсталятор:** Відкрийте завантажений файл інсталятора Notepad++ та слідуйте інструкціям майстра інсталяції.
**3.Виберіть мову:** Оберіть мову, яку ви бажаєте використовувати в Notepad++ та натисніть "OK".
**4.Прийміть ліцензійну угоду:** Прочитайте та прийміть ліцензійну угоду, а потім натисніть "Next".
**5.Виберіть тип установки:** Виберіть тип установки (рекомендується залишити стандартний) та натисніть "Next".
**6.Виберіть шлях установки:** Виберіть шлях, куди ви бажаєте встановити Notepad++, і натисніть "Next".
**7.Оберіть компоненти:** Оберіть компоненти за вашими уподобаннями, але залиште їхні значення за замовчуванням, якщо не впевнені. Натисніть "Next".
**8.Оберіть зовнішній вигляд:** Оберіть стиль зовнішнього вигляду для Notepad++ та натисніть "Next".
**9.Оберіть додаткові опції:** Виберіть додаткові опції (рекомендується залишити їхні значення за замовчуванням) та натисніть "Install".
**10.Завершіть інсталяцію:** Після завершення інсталяції натисніть "Finish".
**11.Запустіть Notepad++:** Відкрийте меню "Пуск" та знайдіть Notepad++ у списку програм або запустіть його за ярликом на робочому столі.

# Налаштування середовища програмування

Кроки інсталяції Visual Studio Code

**На Windows:**
**1.Завантажте інсталятор:** Перейдіть на офіційний веб-сайт Visual Studio Code та завантажте відповідну версію для Windows.
**2.Запустіть інсталятор:** Відкрийте завантажений файл інсталятора та слідуйте інструкціям майстра інсталяції.
**3.Виберіть опції інсталяції:** Виберіть опції, які вам необхідні, і натисніть "Next".
**4.Оберіть шлях установки:** Виберіть шлях, куди ви бажаєте встановити Visual Studio Code, і натисніть "Next".
**5.Оберіть створення ярлика:** Оберіть, чи ви хочете створити ярлик на робочому столі та/або в меню "Пуск", і натисніть "Next".
**6.Встановіть:** Натисніть "Install", щоб розпочати інсталяцію.
**7.Завершіть інсталяцію:** Після завершення інсталяції натисніть "Finish".
**8.Запустіть Visual Studio Code:** Відкрийте меню "Пуск" та знайдіть Visual Studio Code у списку програм або запустіть його за ярликом на робочому столі.

# Налаштування середовища програмування

Кроки інсталяції PyScripter

**На Windows:**

**1.Завантажте інсталятор:** Перейдіть на офіційний веб-сайт PyScripter і завантажте останню версію PyScripter для Windows. Зазвичай, вам слід вибрати інсталятор з розширенням ".exe".

**2.Запустіть інсталятор:** Відкрийте завантажений файл інсталятора PyScripter та слідуйте інструкціям майстра інсталяції.

**3.Виберіть опції інсталяції:** Виберіть опції, які вам необхідні, і натисніть "Next". Зазвичай, залишайте значення за замовчуванням, якщо ви не впевнені, які опції обрати.

**4.Виберіть шлях установки:** Оберіть каталог для установки PyScripter, а потім натисніть "Next".

**5.Оберіть ярлик:** Виберіть, чи ви хочете створити ярлик PyScripter на робочому столі та/або в меню "Пуск". Натисніть "Next".

**6.Завершіть інсталяцію:** Натисніть "Install", щоб розпочати інсталяцію. Почекайте, поки встановлення завершиться.

**7.Запустіть PyScripter:** Після завершення інсталяції натисніть "Finish" та відкрийте PyScripter за допомогою створеного ярлика на робочому столі або в меню "Пуск".

# Налаштування середовища програмування

Кроки інсталяції PyCharm

**На Windows:**

**1.Завантажте інсталятор:** Перейдіть на офіційний веб-сайт PyCharm та завантажте версію для Windows.

**2.Запустіть інсталятор:** Відкрийте завантажений файл інсталятора PyCharm та слідуйте інструкціям майстра інсталяції.

**3.Виберіть тип інсталяції:** Оберіть тип інсталяції (Professional або Community) та натисніть "Next". Якщо ви новачок, рекомендується вибрати Community.

**4.Виберіть опції:** Оберіть компоненти, які ви хочете встановити, та натисніть "Next". Зазвичай, залишайте значення за замовчуванням.

**5.Виберіть шлях установки:** Виберіть каталог для установки PyCharm та натисніть "Next".

**6.Оберіть створення ярликів:** Виберіть, чи створити ярлики на робочому столі та/або в меню "Пуск", і натисніть "Next".

**7.Виберіть стартове меню:** Оберіть папку для стартового меню та натисніть "Install".

**8.Завершіть інсталяцію:** Натисніть "Finish", щоб завершити інсталяцію.

**9.Запустіть PyCharm:** Відкрийте меню "Пуск" та знайдіть PyCharm у списку програм або запустіть його за ярликом на робочому столі.

# Завантаження і встановлення Python

Перевіримо успішність встановлення Python на вашому комп'ютері. Для цього використайте комбінацію клавіш ***Win+R***, введіть команду cmd та натисніть OK. У відкритому вікні командного рядка введіть команду python --version та натисніть Enter:

```
> python --version
Python 3.9.0
```

# Середовища програмування для Python

Текстові редактори та інтегровані середовища програмування для

- **Python:**
- **IDLE.**
- **Notepad++.**
- **Visual Studio Code.**
- **PyScripter.**
- **Wing IDE 101.**
- **Geany.**
- **PyCharm .**
- **Windows і macOS.**
- **Thonny**
- **Mu.**

*Для написання програм мовою Python можна використовувати онлайн-середовище repl.it*

# Запуск Python: інтерактивний інтерпретатор

Щоб відкрити вікно терміналу:

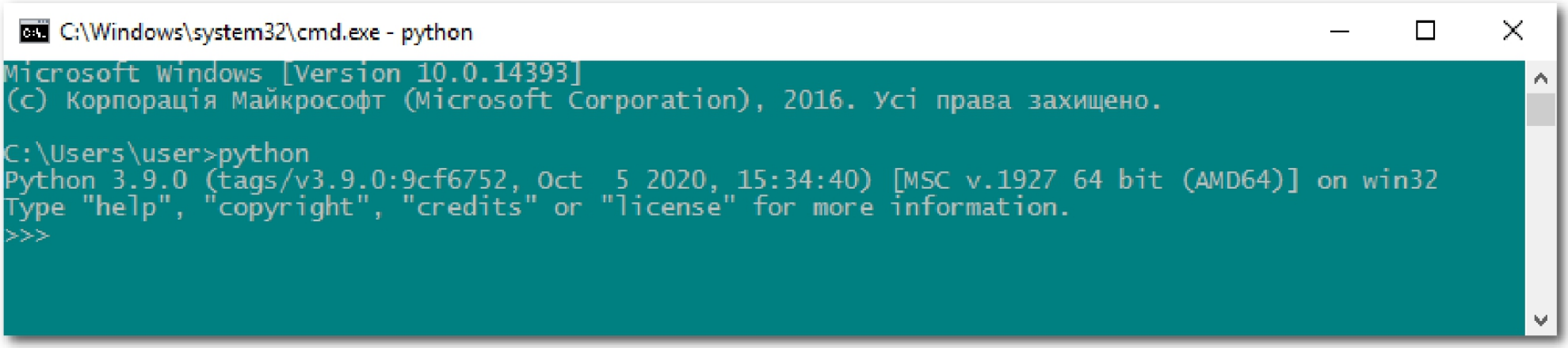
- натисніть сполучення клавіш Win+R на клавіатурі, введіть команду cmd, натисніть OK (для користувачів Windows);
- натисніть сполучення клавіш Ctrl+Alt+T (для користувачів Linux Ubuntu).

У термінальному вікні, що з’явилося, введіть команду (у випадку використання Windows):

```
python
```

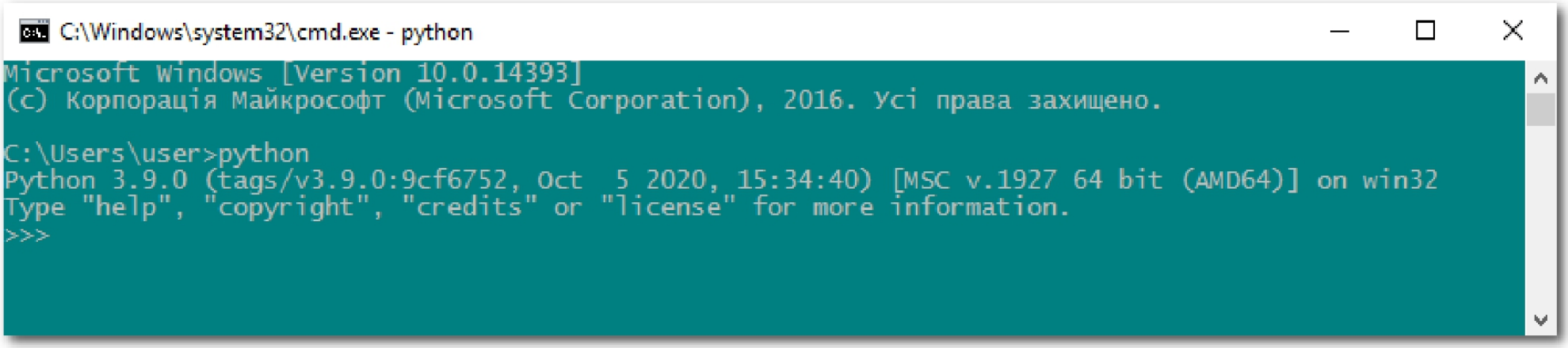
# Запуск Python: інтерактивний інтерпретатор

Якщо на екрані з’явиться запрошення >>> до введення команд, значить система виявила встановлену версію Python:



# Запуск Python: інтерактивний інтерпретатор

Якщо на екрані з’явиться запрошення >>> до введення команд, значить система виявила встановлену версію Python:

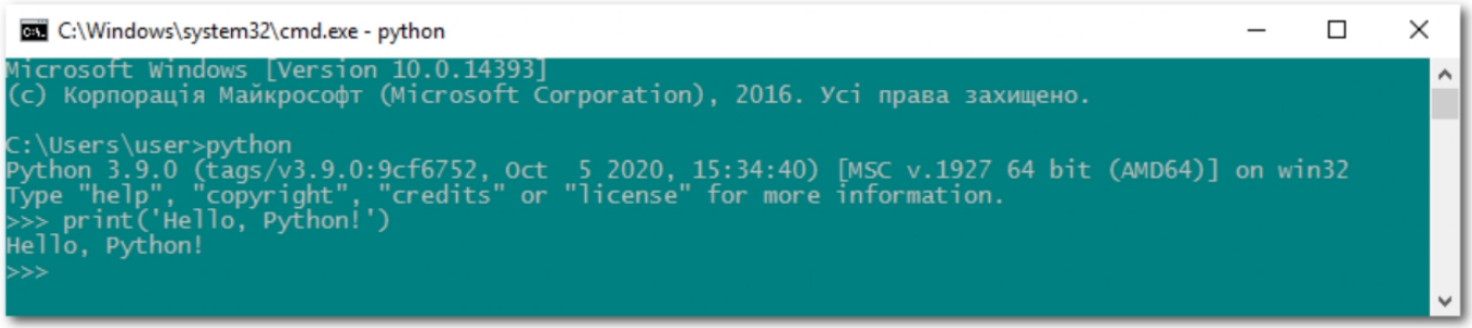


# Запуск Python: інтерактивний інтерпретатор

Введіть в інтерактивному режимі інтерпретатора наступний рядок

```
print('Hello, Python!')
```

натисніть `Enter` і переконайтеся в тому, що на екрані з'явилось повідомлення `Hello, Python!`:




*Режим інтерактивного інтерпретатора **Python** у термінальному вікні **Windows**: виведення текстового повідомлення*

*Функція **print()** входить у стандартну бібілотеку Python. Вона виводить інформацію, вказану в дужках, на екран або записує у файл.*

# Повідомлення про помилку

Наприклад, коли ми введемо в режимі інтерактивного інтерпретатора інструкцію '19' + 81, з’явиться таке повідомлення:

```
>>> '19' + 81
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

*У Python, у разі появи помилки **генерується виняток**, який повідомляє про зміст помилки. В даному випадку, згенерований виняток TypeError повідомляє про несумісність типів під час додавання числа і рядка, іншими словами, рядок можна об’єднувати лише з рядком.*

# Коментарі

В мові Python коментарі вказуються за допомогою символу #. Інтерпретатор Python ігнорує всі символи в коді, які знаходяться після # до кінця рядка. Наприклад:

```
>>> # Привіт, світ!
... print("Hello, world!")
Hello, world!
```

# Синтаксис

Рекомендована (але не обов'язкова) максимальна довжина рядка не повинна перевищувати 80 символів. Якщо ви не можете висловити свою думку в рамках 80 символів, скористайтеся символом **продовження рядка** (\).
Просто помістіть \ в кінець рядка, а **Python** буде діяти так, ніби це все той самий рядок. Наприклад:

```
>>> alphabet = 'abcdefg' + \
... 'hijklmnop' + \
... 'qrstuv' + \
... 'wxyz'
>>> alphabet
'abcdefghijklmnopqrstuvwxyz'
>>> 1 + 2 + \
... 3
6
```

# Стиль Python

Розробники мови є прихильниками певної філософії програмування, яку називають **«The Zen of Python»** («Дзен Пайтона»)

```
import this
```

На відміну від багатьох інших мов, Python обов’язково вимагає, щоб блоки коду забезпечувалися відступами.

*У Python для побудови структури програми використовуються відступи від лівого краю, які створюються за допомогою пропусків (пробілів).*

# Стиль Python

```
n = int(input())
out = []
for i in range(n):
....k = 0
....while k < i + 1:
........out.append(i+1)
........k += 1
....if len(out) > n:
........break
out = out[0:n]
for i in out:
....print(i, end = " ")
```

*Використовуйте 4 пропуски на рівні відступів.*

# ЗАВДАННЯ

1. Встановіть Python 3.
2. Встановіть середовище програмування.
3. Запустіть інтерактивний інтерпретатор Python 3 і використайте його як калькулятор. Наприклад, обчисліть 19 * 81. Запишіть цей добуток і натисніть Enter, щоб побачити результат. Python повинен вивести 1539.
4. Введіть число 43 і натисніть клавішу Enter. Чи з’явилось це число в наступному рядку?
5. Введіть print(43) і натисніть клавішу Enter. Чи з’явилось знову це число в наступному рядку?
6. В інтерактивному інтерпретаторі Python введіть 'Python' + 3. Знайдіть інформацію в мережі Інтернет про помилку, що виникла, за її назвою.
7. Перегляньте принципи Python, ввівши в термінальму сеансі команду **import this**

# Змінні й типи даних у мові Python

# Типи даних

Кожне з даних характеризується **розміром виділеної пам'яті для зберігання**, **ім'ям (ідентифікатором)**, **типом** і **значенням**.

Мови програмування мають в своєму арсеналі деякі **прості (вбудовані) типи даних**. Вони використовуються як базові блоки для програм та складних (спеціалізованих) типів даних.

У Python вбудовані такі прості типи даних:

- **Булевий** (має значення True і False).
- **Цілі числа** (наприклад, 81, 1000).
- **Числа з плаваючою крапкою** (наприклад, 3.14159, 2.5e8, 4000.0).
- **Рядки** (послідовності текстових символів, наприклад Hello, Python!).

.

# Зарезервовані слова Python

| | | | | |
|---|---|---|---|---|
| `False` | `class` | `finally` | `is` | `return` |
| `None` | `continue` | `for` | `lambda` | `try` |
| `True` | `def` | `from` | `nonlocal` | `while` |
| `and` | `del` | `global` | `not` | `with` |
| `as` | `elif` | `if` | `or` | `yield` |
| `assert` | `else` | `import` | `pass` | `break` |

# Об’єкти та змінні

Об’єкт, значення якого неможливо змінити, можна порівняти із закритою скринькою з віконцем: ви можете побачити значення, але не можете змінити його. В рамках тієї ж аналогії, об’єкт, значення якого можна змінити, схожий на відкриту скриньку: ви не тільки можете побачити значення, яке там зберігається, а й змінити його, однак не можете змінити тип об’єкта (скриньки).

В Python для присвоювання змінній певного значення використовується символ = .

! *У математиці символ = означає **«дорівнює»**. У багатьох мовах програмування, включаючи Python, цей символ використовується для позначення **«присвоювання»**.*

# Об’єкти та змінні

У наступному фрагменті програми ціле число 12 присвоюється змінній з ім’ям a, потім на екран виводиться значення, пов’язане в поточний момент з цією змінною:

```
>>> a = 12
>>> a
12
```

З іншого боку, змінні в Python схожі не на контейнери, а на **ярлики** чи **стікери-наклейки**, які прикріплюються до об’єктів із простору імен інтерпретатора Python.

# Об’єкти та змінні

```
>>> a = [5, 6, 7] 1
>>> b = a 2
>>> c = b 3
```

1. Змінна a містить список із трьох числових значень.
2. Присвоюємо змінній b значення a.
3. Присвоюємо змінній c значення b.

# Об’єкти та змінні

А тепер змінимо **перше** значення (з 5 на 100) у списку, який зберігається у змінній b:

```
>>> a = [5, 6, 7]
>>> b = a
>>> c = b
>>> b[0] = 100
>>> print(a, b, c)
[100, 6, 7] [100, 6, 7] [100, 6, 7]
```

# Об’єкти та змінні

*В Python змінні - це просто імена. Присвоювання не копіює значення, воно прикріплює ім’я до об’єкта, який містить дані. Ім’я - це **посилання на якийсь об’єкт**, а не сам об’єкт. Ім’я можна розглядати як стікер-наклейку.*

*Ціле число **12** присвоюється змінній з ім'ям **a**: змінна **a** - це посилання на об’єкт (скриньку для цілих значень), який містить значення **12***

# Об’єкти та змінні

За допомогою інтерактивного інтерпретатора виконайте наступні кроки:

Як і раніше, надайте значення 12 змінній a. Це створить об’єкт-«скриньку», який міститиме цілочисельне значення 12.

Виведіть на екран значення a.

Присвойте a змінній b, змусивши b прикріпитися до об’єкту-«скриньки», який містить значення 12.

Виведіть значення b.

```
>>> a = 12
>>> a
12
>>> b = a
>>> b
12
```

# Об’єкти та змінні

Давайте перевіримо, чи посилаються змінні a і b на один і той же цілочисельний об’єкт зі значенням 12 за допомогою функції **id()**.

```
>>> id(a)
1367960944
>>> id(b)
1367960944
```

Як видно з результатів, при зв’язуванні змінної з існуючою, обидві змінні вказують на один об’єкт. **Змінна при цьому не копіюється!**

Що зберігатиметься у змінній b, якщо записати наступні дві інструкції?

```
>>> b = 'Goodbye!'
>>> b = 'Hello!'
>>> b
'Hello!'
```

# Об’єкти та змінні

Тепер змінна b вказує на новий об’єкт, оскільки стікер-наклейку (змінну) ми переклеїти з одного об’єкта на інший (ідентифікатори змінилися):

```
>>> b = 'Goodbye!'
>>> id(b)
2371957227888
>>> b = 'Hello!'
>>> id(b)
2371957221112
>>> b
'Hello!
```

Ідентифікатори використовуються для перевірки тотожності оператором is, тобто він перевіряє, чи вказують дві змінні на один і той же об’єкт:

```
>>> c = 10
>>> d = c
>>> c is d
True
>>> id(c)
1367960880
>>> id(d)
1367960880
>>> d = 15
>>> c is d
False
>>> id(d)
1367961040
```

# Об’єкти та змінні

Коректний код Python:

```
>>> s = 'Python'
>>> s
'Python'
>>> s = 3
>>> s
3
```

В Python, якщо ви хочете дізнатися тип якогось об’єкта (змінної або значення), слід використовувати стандартну (вбудовану) функцію **type()**.

# Об’єкти та змінні

Спробуємо зробити це для різних значень (81, 99.99, ruby) і змінної (a):

```
>>> type(a)
<Class 'int'>
>>> type(81)
<Class 'int'>
>>> type(99.99)
<Class 'float'>
>>> type('ruby')
<Class 'str'>
```

В Python, якщо ви хочете дізнатися тип якогось об’єкта (змінної або значення), слід використовувати стандартну (вбудовану) функцію **type()**.

# Оператори та вирази

# Пріоритет операторів

| | |
|---|---|
| `(вирази...)`, `[вирази...]`, `{ключ: значення...}`, `{вирази...}` | Вираз в дужках, створення або включення списку/словника/множини |
| `x[індекс]`, `x[індекс:індекс]`, `x(аргументи…)`, `x.атрибут` | Індекс, зріз, виклик функції, посилання на атрибут |
| `**` | Піднесення до степеня |
| `+x`, `-x`, `~x` | Знаки «+» і «-», бітове НІ |
| `*`, `/`, `//`, `%` | Множення, ділення з плаваючою крапкою, цілочисельне ділення, остача від ділення |
| `+`, `-` | Додавання, віднімання |
| `<<`, `>>` | Бітові зсуви |
| `&` | Бітове І |
| `\|` | Бітове АБО |
| `^` | Бітове виключення АБО |

# Пріоритет операторів

| | |
|---|---|
| `^` | Бітове виключення АБО |
| `in`, `not in`, `is`, `is not`, `<`, `<=`, `>`, `>=`, `!=`, `==` | Перевірка на приналежність і рівність |
| `not x` | Булеве (логічне) НІ |
| `and` | Булеве (логічне) І |
| `or` | Булеве (логічне) АБО |
| `if ... else` | Умовний вираз |
| `lambda` | Лямбда-вираз |

# Числа

| Оператор | Опис | Приклад | Результат |
|---|---|---|---|
| `+` | Додавання | `5 + 7` | `12` |
| `-` | Віднімання | `55 - 10` | `45` |
| `*` | Множення | `4 * 6` | `24` |
| `/` | Ділення | `7 / 2` | `3.5` |
| `//` | Цілочисельне ділення | `7 // 2` | `3` |
| `%` | Остача від ділення | `7 % 3` | `1` |
| `**` | Піднесення до степеня | `3 ** 4` | `81` |

# Цілі числа

```
>>> 10
10
>>> 05
  File "<stdin>", line 1
    05
     ^
SyntaxError: invalid token
```

Послідовність цифр вказує на ціле число. Якщо ви поставите знак `+` перед цифрами, число залишиться незмінним:

```
>>> 132
132
>>> +132
132
```

# Цілі числа

Щоб вказати на від'ємне число, поставте перед цифрами знак `-`:

```
>>> -321
-321
```

За допомогою `Python` можна виконувати арифметичні дії як зі звичайним калькулятором:

```
>>> 25 + 9
34
>>> 145 - 37
108
>>> 8 + 3 - 2 + 1 - 106
-96
>>> 6 * 7
42
```

# Цілі числа

Операції **ділення** існує два види. За допомогою оператора `/` виконується **ділення з плаваючою крапкою** (**десяткове ділення**). Навіть, якщо ви ділите ціле число на ціле число, оператор `/` дасть результат з плаваючою крапкою:

```python
>>> 9 / 5
1.8
```

**Цілочисельне ділення** за допомогою оператора `//` дає цілочисельну відповідь, відкидаючи залишок:

```
>>> 9 // 5
1
```

# Цілі числа

Ділення на нуль з допомогою будь-якого оператора викличе помилку:

```
>>> 6 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 8 // 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

Використаємо в обчисленнях змінні та змінимо їх значення:

```
>>> a = 42
>>> a = a - 2
>>> a
40
```

# Цілі числа

Арифметичні оператори можуть використовуватися разом із оператором присвоювання, розміщуючи їх перед символом присвоювання:

```
>>> a = 95
>>> a -= 3 (1)
>>> a
92
>>> a += 8 (2)
>>> a
100
>>> a *= 2 (3)
>>> a
200
>>> a /= 3 (4)
>>> a
66.66666666666667
```

1. Аналогічно виразу `a = a - 3`.
2. Аналогічно виразу `a = a + 8`.
3. Аналогічно виразу `a = a * 2`.
4. Аналогічно виразу `a = a / 3`.

# Об’єкти

# Об’єкти

В Python об'єкти та змінні використовуються для представлення та збереження даних. Об'єкти можуть бути різних типів, таких як числа, рядки, списки, кортежі, словники та інші. Змінні вказують на об'єкти та надають їм імена.

```
# Ціле число
age = 25

# Дійсне число (з плаваючою комою)
height = 1.75

# Комплексне число
complex_number = 3 + 4j
```

**Числові об'єкти та змінні**

```
# Рядок
name = "John Doe"

# Можна використовувати одинарні або подвійні лапки для оголошення рядка
quote = 'Python is a versatile language'

# Форматований рядок (за допомогою f-рядка)
message = f"Hello, {name}! Your height is {height} meters."
```

**Рядкові об'єкти та змінні**

```
# Логічний об'єкт (True/False)
is_adult = True

# Змінна може вказувати на логічний об'єкт
has_permission = False
```

**Логічні об'єкти та змінні**

# Об’єкти

Функція **id()** повертає унікальний ідентифікатор для зазначеного об’єкта - значення його адреси в пам’яті.

Ідентифікатор - це ціле число, яке гарантовано є унікальним та постійним для цього об’єкта протягом його життя.

```
>>> year = 2022
>>> id(year)
2392913595504
>>> import _ctypes
>>> print(_ctypes.PyObj_FromPtr(2392913595504))
2022
```

Для порівняння ідентифікаторів об’єктів ми використали оператор порівняння ==.
Щоб виконати зворотну дію - отримати значення об’єкта за його адресою в пам’яті, можна використати функцію PyObj_FromPtr(), яка надається вбудованим модулем _ctypes:

# Об’єкти

Коли ви створюєте об’єкт типу int у цьому діапазоні, ви, фактично, просто отримуєте посилання на вже існуючий об’єкт, а не на новий об’єкт. А от, наприклад, для числа 257 буде створений новий об’єкт (Цілі об’єкти).

Інтерпретатор Python оптимізований так, що невеликі цілі числа представлені одним об’єктом - це зроблено з метою поліпшення продуктивності. Для великих чисел це вже не виконується.

Інші реалізації Python, такі як Jython, IronPython, можуть мати іншу реалізацію функції id(). Тому, наведені приклади є особливостями реалізації інтерпретатора, а не мовною особливістю Python.

# Об’єкти

У Python є власний механізм автоматичного збору сміття.

Збирання сміття (англ. garbage collection) - одна з форм автоматичного керування оперативною пам’яттю комп’ютера під час виконання програм. Підпрограма - «прибиральник сміття» - вивільняє пам’ять від об’єктів, які не будуть використовуватись програмою у подальшому. Збирання сміття було винайдено Джоном Мак-Карті приблизно 1959 року для розробленої ним мови програмування LISP.

```
>>> year = 2022
>>> y = year
>>> year is y
True
>>> id(year)
1871477591152
>>> id(y)
1871477591152
```

Як бачимо, дві змінні `year` і `y` мають однакову адресу у пам’яті. А тепер виведемо об’єкт за його адресою в пам’яті

```
>>> import _ctypes
>>> print(_ctypes.PyObj_FromPtr(1871477591152))
2022
```

# Перетворення типів: функція int()

Функція **int()** зберігає цілу частину числа і відкидає будь-який залишок.

Найпростіший тип даних в `Python` - **булеві змінні**, значеннями цього типу можуть бути тільки `True` або `False`. При перетворенні в цілі числа вони набувають значень `1` і `0`:

```
>>> int(True)
1
>>> int(False)
0
```

Перетворення числа з плаваючою крапкою в ціле число просто відсікає все, що знаходиться після десяткової крапки:

PYTHON

```
>>> int(98.6)
98
>>> int(1.5e4)
15000
```

# Перетворення типів: функція int()

Розглянемо приклад перетворення текстового рядка, який містить тільки цифри або цифри і знаки `+` і `-`:

```
>>> int('99')
99
>>> int('-23')
-23
>>> int('+12')
12
```

Якщо ви спробуєте перетворити щось не подібне на число, згенерується помилка:

```
>>> int('22 abc')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '22 abc'
```

# Перетворення типів: функція int()

Функції `int()` можна передати необов'язковий другий аргумент з основою системи числення, яка повинна використовуватися при інтерпретації вхідного рядка. Проаналізуємо такий код:

```
>>> int('123.456') 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '123.456'
>>> int('100', 8) 2
64
>>> int('10011', 2) 3
19
>>> int('fa', 16) 4
250
>>> int('123', 3) 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 3: '123'
>>> int('12', 3) 6
5
```

# Перетворення типів: функція int()

1.Перетворення числового рядка '123.456' в ціле число неможливе через виникнення помилки ValueError, оскільки числовий рядок містить крапку.
2.Інтерпретація числового рядка '100' у вісімковій системі числення і перетворення його в десяткове число 64.
3.Інтерпретація числового рядка '10011' у двійковій системі числення і перетворення його в десяткове число 19.
4.Інтерпретація рядка 'fa' у шістнадцятковій системі числення і перетворення його в десяткове число 250.
5.Інтерпретація числового рядка '123' у трійковій системі числення є неможливою через виникнення помилки ValueError, оскільки цифра 3 відсутня у трійковій системі числення.
6.Інтерпретація числового рядка '12' у трійковій системі числення і перетворення його в десяткове число 5.

# Пріоритет операторів

```
>>> 7 * (3 + 4)
49
```

**Числа з плаваючою крапкою**

В Python числа, що мають дробову частину, називаються **дійсними** (або **«числами з плаваючою крапкою»**). Для чисел з плаваючою крапкою використовуються відомі оператори: +, -, *, /, //, **, %.

**Перетворення типів: функція float()**

Булеві значення обробляються як невеликі числа:

```
>>> float(True)
1.0
>>> float(False)
0.0
```

Перетворення значення типу `int` в тип `float`:

PYTHON

```
>>> float(98)
98.0
```

# Пріоритет операторів

**Математичні функції**

Python надає стандартні функції для роботи з числовими даними: **abs()**, **pow()**, **round()**.
Розглянемо як ці функції працюють:

```
>>> abs(-34) 1
34
>>> abs(-34.56) 2
34.56
>>> pow(2, 3) 3
8
>>> pow(-4.5, 2) 4
20.25
>>> round(10.6) 5
11
>>> round(3.5) 6
4
>>> round(8.5) 7
8
>>> round(2.665, 2) 8
2.67
>>> round(2.675, 2) 9
2.67
```

1. Повертає абсолютне значення від'ємного цілого числа.
2. Повертає абсолютне значення від'ємного дробового числа.
3. Піднесення додатного цілого числа до степеня 3.
4. Піднесення від'ємного дробового числа до степеня 2.
5. Якщо не вказано другого аргумента, округлює число до найближчого цілого числа (число 11 ближче знаходиться).
6. Якщо не вказано другого аргумента, округлює число до найближчого цілого числа. Але, якщо два кратні числа однаково рівновіддалені, округлення робиться в напрямку парного вибору (округлення до 4, оскільки 4 парне).
7. Якщо не вказано другого аргумента, округлює число до найближчого цілого числа. Але, якщо два кратні числа однаково рівновіддалені, округлення робиться в напрямку парного вибору (округлення до 8, оскільки 8 парне).
8. Повертає число з плаваючою крапкою, округлене до вказаного числа цифр після десяткової крапки.
9. Обидва числа мають однакове округлення 2.67, хоча число 2.675 повинно було б бути округлене до 2.68. Це не є помилкою. Це особливість представлення десяткових дробів у Python, яка пов'язана з точністю обчислення.

# Математичні функції

# Арифметика з плаваючою крапкою: проблеми і обмеження

Поглянемо, як представляє Python числа 2.665 і 2.675 із 30-ма цифрами після десяткової крапки. Використаємо:

```
>>> '{:.30}'.format(2.665)
'2.665000000000000035527136788 01'
>>> '{:.30}'.format(2.675)
'2.674999999999999822364316059 97'
```

# Вбудовані функції цілих і дійсних чисел

**abs(X)** – повертає абсолютне значення (модуль) числа.

**divmod(A, B)** – повертає пару чисел (P, R), які є цілою частиною P та остачею R при виконанні цілочисельного ділення. Для цілих чисел результат буде таким самим, як і при (A // B, A % B). Для дійсних чисел результатом є (Q, A % B), де Q зазвичай math.floor(A / B), але може бути і на 1 менше. Незважаючи на це значення за виразом Q * B + A % B дуже близьке до A, якщо A % B не рівне нулю, то має такий самий знак, як і B, 0 <= abs(A%B) < abs(B).

```
>>> divmod(7,2)
(3, 1)
```

# Вбудовані функції цілих і дійсних чисел

**pow(X, Y[, Z])** – повертає X в степені Y за модулем Z (обчислюється більш ефективно, чим pow(X, Y) % Z). Двоаргументна форма pow(X, Y) – еквівалент використання оператора піднесення до степеня: X**Y. Якщо параметр Z заданий, то X та Y мають бути цілочисельними, окрім того Y має бути невід'ємним.

```
>>> pow(2,3)
8
>>> pow(2,3,3)
2
```

**round(number[, ndigits])** – повертає число number, округлене до ndigits знаків після десяткової точки.

# Вбудовані функції цілих і дійсних чисел

**max(arg1, arg2, *args, *[, key=func])** – повертає найбільше значення з двох чи більше аргументів. Ця функція має більш широкі можливості, але про них пізніше.

**min(arg1, arg2, *args, *[, key=func])** – повертає найменше значення з двох чи більше аргументів. Ця функція має більш широкі можливості, але про них пізніше.

**int([object], [osn])** – повертає перетворене значення `object` до цілого десяткового числа. `osn` визначає систему числення задання `object` (`osn` від 2 до 36 включно). За замовчуванням `object=0`, `osn=10`.

```
>>> int(4.9)
4
>>> int('11')
11
>>> int('11',2)
3
```

# Вбудовані функції цілих і дійсних чисел

**`float([X])`** – повертає перетворене `X` до дійсного числа.

**`bin(X)`** – повертає рядковий запис цілого числа `X` в двійковій формі.

**`hex(X)`** – повертає рядковий запис цілого числа `X` в шістнадцятковій формі. Для перетворення дійсного числа використовується метод

**`oct(X)`** – повертає рядковий запис цілого числа `X` у вісімковій формі.

**`bool([X])`** – повертає приведене значення `X` до логічного типу (bool), використовуючи стандартну процедуру перевірки істинності.

# Модуль math

**Константи:**

**math.pi**. Число π (`math.pi≈3.141592653589793`).

**math.e**. Число e (`math.e≈2.71828182845904`5).

**math.tau**. Математична константа кола, рівна 2π.

**math.inf**. Додатна нескінченність (дійсне значення). Еквівалентно результату `float('inf')`. Для від'ємної нескінченності використовується: `-math.inf`.

**math.nan**. "не число" ("not a number" (NaN)). Еквівалент результату `float('nan')`.

# Приклад

**Приклад 1**. Написати програму для розрахунку ідеальної ваги чоловіка та жінки за формулою Брокка.

```
zrist=int(input('Ваш зріст в сантиметрах:'))
v_men=(zrist-100)*1.15
v_women=(zrist-110)*1.15
print('Ідеальна вага для чоловіка = ', v_men)
print('Ідеальна вага для жінки = ', v_women)
```

**Приклад 2.** $e^{x+y}+\frac{5}{\cos(y-x)+3}$.

```
import math
x=float(input('X= '))
y=float(input('Y= '))
res=math.exp(x+y)+5/(math.cos(y-x)+3)
print('Знаяення виразу = ', res)
```

# Приклад

**Приклад 3**. Створіть змінну, в яку збережіть свою улюблену книгу, а потім використайте цю змінну для створення повідомлення, яке виведе назву вашої улюбленої книги. Виведіть це повідомлення.

```
# Збережемо улюблену книгу у змінній
ulublena_kniga = "Майстер і Маргарита"

# Створимо повідомлення для виведення улюбленої книги
powidomlennya = f"Моя улюблена книга - '{ulublena_kniga}'."

# Виведемо це повідомлення
print(powidomlennya)
```

# ЗАВДАННЯ

1. Напишіть операції додавання, віднімання, множення і ділення, результатом яких є число 12. Ви повинні написати чотири рядки коду, які виглядають приблизно так: print(5 + 7). Результатом повинні бути чотири рядки, у кожному з яких виводиться число 12.

2. Збережіть своє улюблене число у змінній, а потім за допомогою змінної створіть повідомлення для виведення цього числа. Виведіть це повідомлення.

3. Збережіть будь-яке повідомлення у змінній і виведіть це повідомлення. Потім замініть значення змінної іншим повідомленням і виведіть нове повідомлення. Програму збережіть у файлі, ім’я якої підпорядковується стандартним правилам Python по використанню малих літер і символів підкреслення - наприклад, simple_messages.py.

# Умовні конструкції

# Блок схема


рядку дій коли йде сніг

# Булеві значення

- *При використанні у коді Python, булеві значення True і False записуються без лапок і їх назви починаються з великих літер T і F, тоді як інша частина слова пишеться маленькими буквами.*

```
>>> one = True
>>> one
True
>>> two = false
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'false' is not defined
>>> two = False
>>> two
False
```

# Оператори порівнювання

*Таблиця "Оператори порівнювання"*

| Оператор | Назва |
|---|---|
| `==` | Дорівнює |
| `!=` | Не дорівнює |
| `<` | Менше ніж |
| `>` | Більше ніж |
| `<=` | Менше або дорівнює |
| `>=` | Більше або дорівнює |

# Оператори порівнювання

Приклад

```
>>> a = 10
>>> a == 5
False
>>> a == 10
True
>>> 4 < a
True
>>> a > 12
False
>>> 'breeze' == 'breeze'
True
>>> 'breeze' == 'Breeze'
False
>>> 'fine' != 'rain'
True
>>> 16 == 16.0
True
>>> 16 == '16'
False
>>> 45 >= 11
True
```

*Для перевірки на рівність використовуються два знакм **«дорівнює»** (==); пам'ятайте, що один знак **«дорівнює»** (=) застосовується для присвоювання змінній значенн*

# Булеві оператори

- *Булеві оператори **and** і **or** завжди працюють з двома булевими значеннями (або виразами), тому їх називають **бінарними**.*

*Таблиця істинності для оператора **and***

| Вираз | Результат |
|---|---|
| `True and True` | `True` |
| `True and False` | `False` |
| `False and True` | `False` |
| `False and False` | `False` |

*Таблиця істинності для оператора **or***

| Вираз | Результат |
|---|---|
| `True or True` | `True` |
| `True or False` | `True` |
| `False or True` | `True` |
| `False or False` | `False` |

# Булеві оператори

- *Булевий оператори **not** завжди діє на одне булеве значення (або вираз), тому його називають **унарним**.*

*Таблиця істинності для оператора **not***

| Вираз | Результат |
|---|---|
| `not True` | `False` |
| `not False` | `True` |

- *ПРИКЛАД*

```
>>> True and True
True
>>> True and False
False
>>> True or False
True
>>> not True
False
>>> not False
True
```

# Пріоритет операторів

*Оператори із більш високим пріоритетом розташовуються у таблиці вище.*

*Таблиця "Пріоритети операторів"*

| | |
|---|---|
| `(вирази...)`, `[вирази...]`, `{ключ: значення...}`, `{вирази...}` | Вираз в дужках, створення або включення списку/словника/множини |
| `x[індекс]`, `x[індекс:індекс]`, `x(аргументи...)`, `x.атрибут` | Індекс, зріз, виклик функції, посилання на атрибут |
| `**` | Піднесення до степеня |
| `+x`, `-x`, `~x` | Знаки «+» і «-», бітове НІ |
| `*`, `/`, `//`, `%` | Множення, ділення з плаваючою крапкою, цілочисельне ділення, остача від ділення |
| `+`, `-` | Додавання, віднімання |
| `<<`, `>>` | Бітові зсуви |
| `&` | Бітове І |
| `\|` | Бітове АБО |
| `^` | Бітове виключення АБО |
| `in`, `not in`, `is`, `is not`, `<`, `<=`, `>`, `>=`, `!=`, `==` | Перевірка на приналежність і рівність |
| `not x` | Булеве (логічне) НІ |

# Пріоритет операторів

*На практиці краще використовувати дужки для групування операторів і операндів*

| | |
|---|---|
| `and` | Булеве (логічне) І |
| `or` | Булеве (логічне) АБО |
| `if ... else` | Умовний вираз |
| `lambda` | Лямбда-вираз |

# Таблиці істинності

*Оператор* ***and***

| Вираз | Результат |
|---|---|
| `True and True` | `True` |
| `True and False` | `False` |
| `False and True` | `False` |
| `False and False` | `False` |

*Оператор* ***or***

| Вираз | Результат |
|---|---|
| `True or True` | `True` |
| `True or False` | `True` |
| `False or True` | `True` |
| `False or False` | `False` |

*Оператор* ***not***

| Вираз | Результат |
|---|---|
| `not True` | `False` |
| `not False` | `True` |

# Поєднання булевих значень, операторів порівнювання, булевих операторів

```
>>> (7 < 10) and (6 < 9)
True
>>> (3 < 5) and (16 < 8)
False
>>> (3 == 6) or (32 == 32)
True
>>> 2 + 2 == 4 and not 2 + 2 == 5 and 2 * 2 == 2 + 2
True
```

# Вказівка розгалуження

```
name = 'Mary'
password = 'hurricane'
if name == 'Mary':
....print('Hello, Mary.') 1
....if password == 'swordfish':
........print('Access granted.') 2
....else:
........print('Wrong password.') 3
else:
....print('User not registered.') 4
```

1.Перший  блок коду: починається рядком print('Hello, Mary.) і містить усі наступні рядки коду до другого оператора **else** (не включаючи його).
2.Другий блок коду: знаходиться у першому блоці коду, містить лише один рядок коду print('Access granted.') і відноситься до другого оператора **if**.
3.Третій блок коду: складається тільки із одного рядка print('Wrong password.') і відноситься до першого оператора **else**.
4.Четвертий блок коду: складається із єдиного рядка print('User not registered.') і відноситься до другого оператора **else**.

# Команда if

```
if умова:
    блок коду
```

В Python інструкція **if** включає такі елементи:

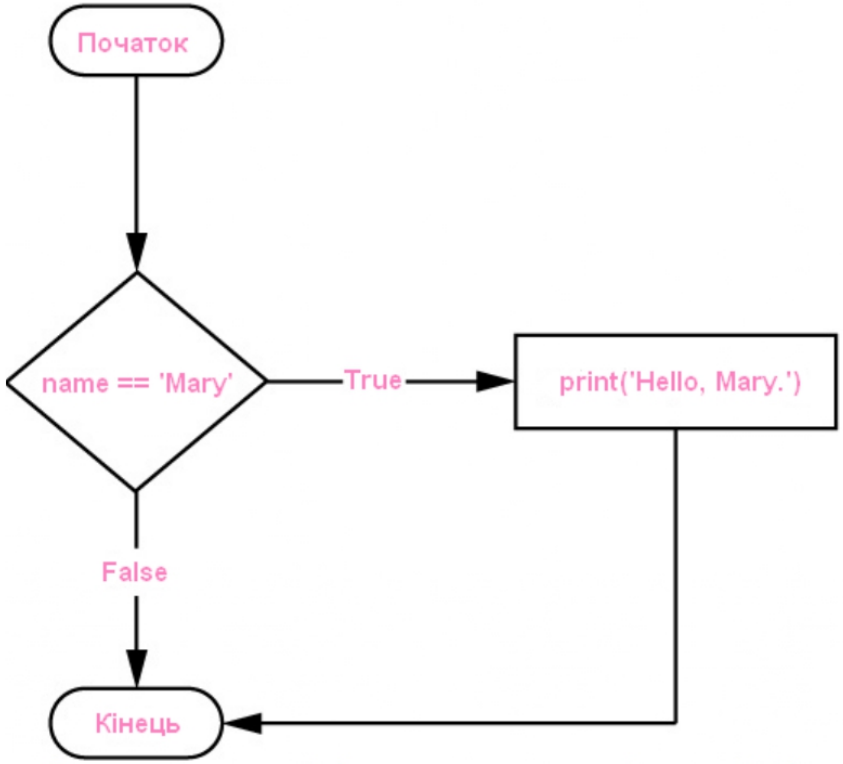
- ключове слово **if**
- умова (вираз, обчислення якого дає одне з двох значень: True або False)
- двокрапка
- блок коду з відступом, який починається в наступному рядку

# Команда if

*Приклад*

```
if name == 'Mary':
    print('Hello, Mary.')
```

```
print('Hello, Mary.')
```




*Блок-схема неповної форми розгалуження: інструкція* ***if***

# Команда else

```
if умова:
    блок коду, коли умова істинна
else:
    блок коду, коли умова хибна
```

В Python команда **else** не має умови і завжди складається з таких елементів:

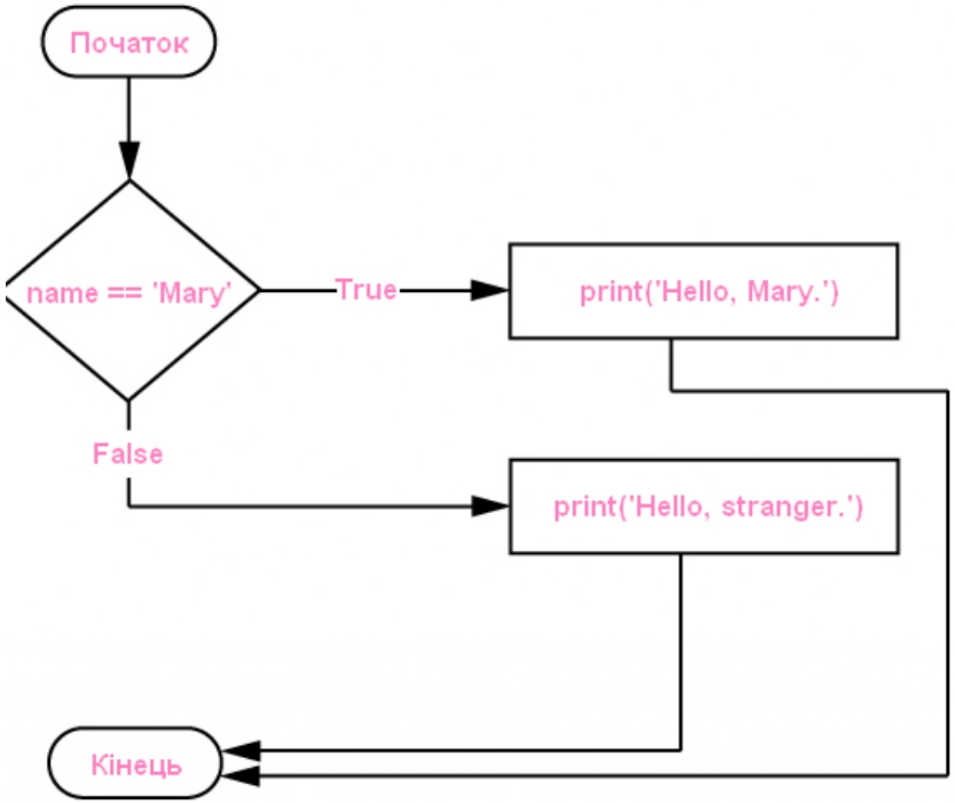
- ключове слово **else**
- двокрапка
- блок коду з відступом, що починається на наступному рядку

# Команда else

ПРИКЛАД

```
if name == 'Mary':
    print('Hello, Mary.')
else:
    print('Hello, stranger.')
```

```
Hello, stranger.
```




*Блок-схема повної форми розгалуження: конструкція* ***if/else***

# Команда elif

```
if умова1:
    блок коду, коли умова1 істинна
elif умова2:
    блок коду, коли умова2 істинна
elif умова3:
    блок коду, коли умова3 істинна
...
```

В Python команда **elif** завжди складається з таких елементів:

- ключове слово **elif**
- умова (вираз, обчислення якого дає одне з двох значень: True або False)
- двокрапка
- блок коду з відступом, що починається на наступному рядку

# Команда elif. Приклад

Приклад фрагменту програми, яка визначає, в яку чверть року потрапляє введений місяць

```
# Введення місяця користувачем
month = input("Введіть назву місяця: ")

# Конструкція if-elif-else для визначення чверті року
if month.lower() in ["грудень", "січень", "лютий"]:
    season = "зима"
elif month.lower() in ["березень", "квітень", "травень"]:
    season = "весна"
elif month.lower() in ["червень", "липень", "серпень"]:
    season = "літо"
elif month.lower() in ["вересень", "жовтень", "листопад"]:
    season = "осінь"
else:
    season = "невідома"

# Виведення результату
print(f"{month} належить до сезону {season}.")
```

# Конструкція if/elif/else

```
if умова1:
    блок коду, коли умова1 істинна
elif умова2:
    блок коду, коли умова2 істинна
elif умова3:
    блок коду, коли умова3 істинна
else:
    блок коду, коли усі умови хибні
```

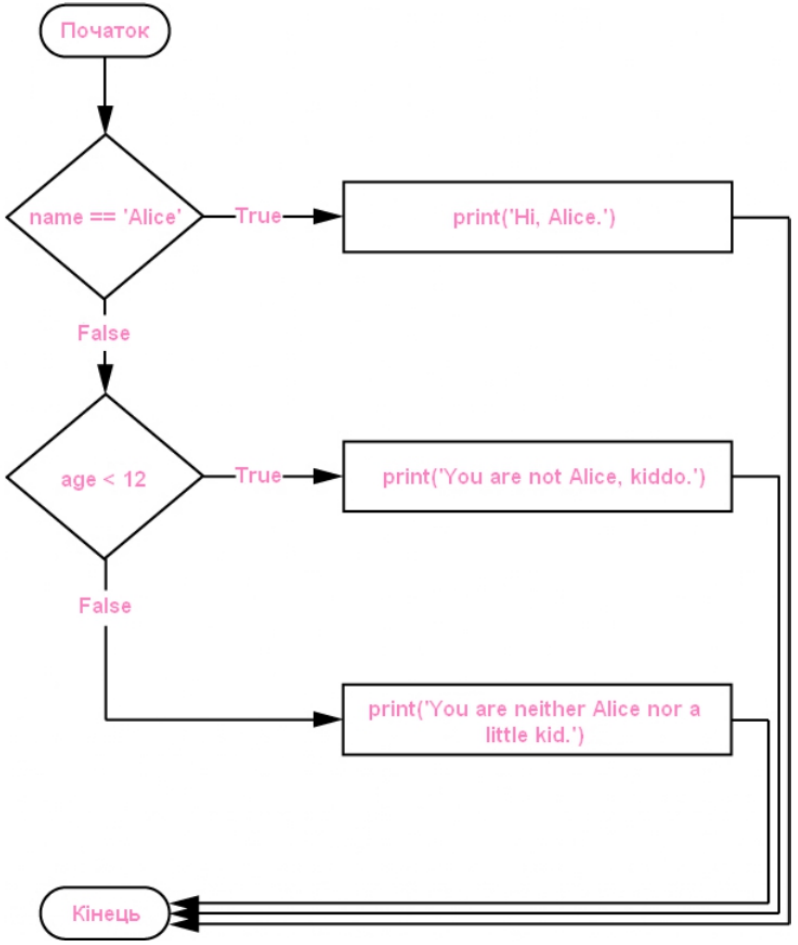
*Якщо умови у всіх інструкціях **if** і **elif** будуть хибними, то виконається блок коду інструкції **else**.*

# Конструкція if/elif/else

*ПРИКЛАД*

```
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
else:
    print('You are neither Alice nor a little kid.')
```

```
You are neither Alice nor a little kid.
```




*Блок-схема: конструкція **if/elif/else***

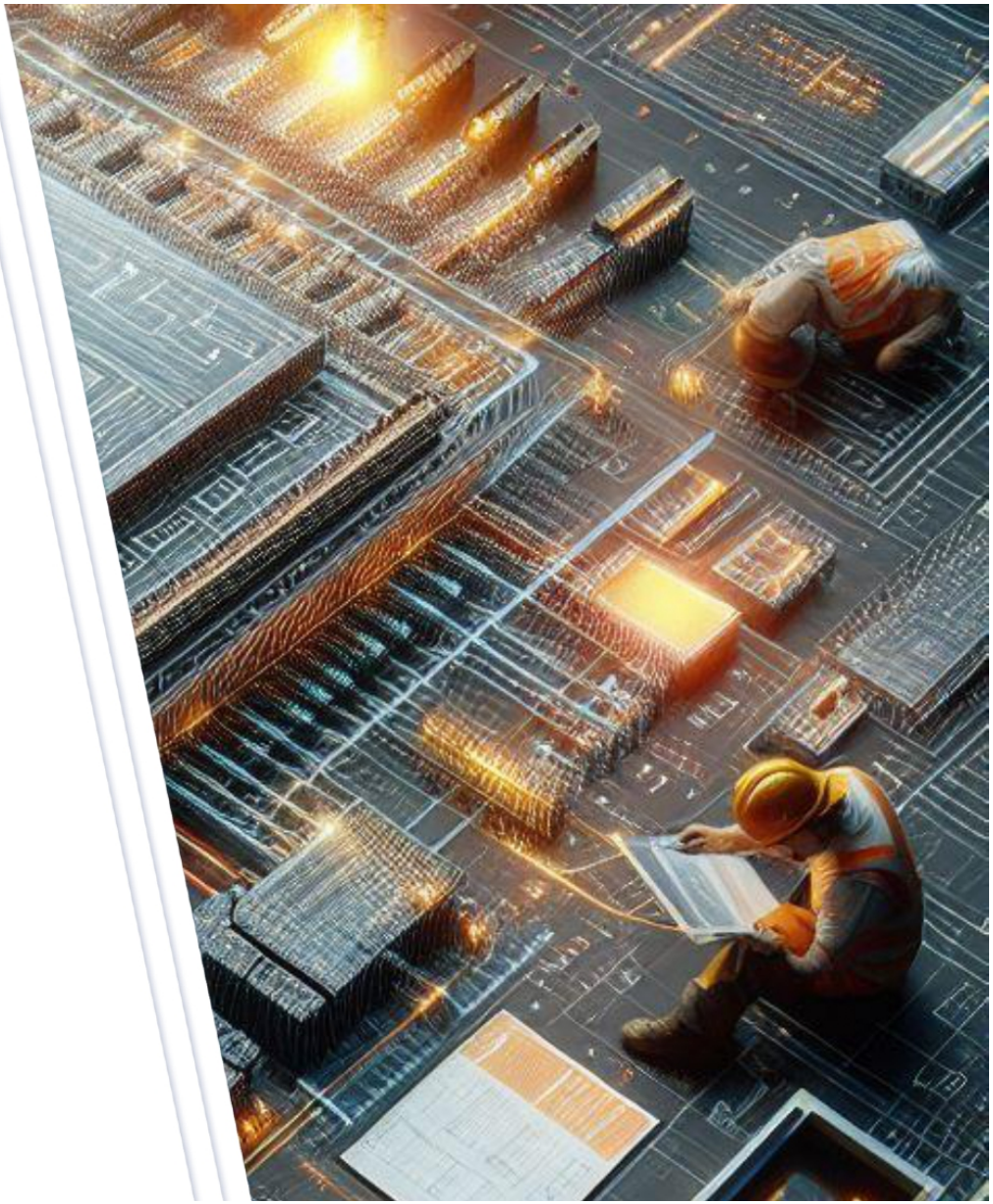
# Конструкція if/elif/else

*ПРИКЛАД*

```
furry = True
small = True
if furry:
    if small:
        print("It's a cat.")
    else:
        print("It's a bear!")
else:
    if small:
        print("It's a lizard!")
    else:
        print("It's a human. Or a hairless bear.")
```

*Що буде результатом ?.....*

# Цикли



# Вказівка повторення

✓ Одним із варіантів циклу у Python є **while**

```
while умова:
    блок коду, коли умова істинна
```

В Python команда **while** завжди складається з таких елементів:

- ключове слово **while**
- умова (вираз, обчислення якого дає одне з двох значень: True або False)
- двокрапка
- блок коду з відступом, що починається на наступному рядку

# Вказівка повторення

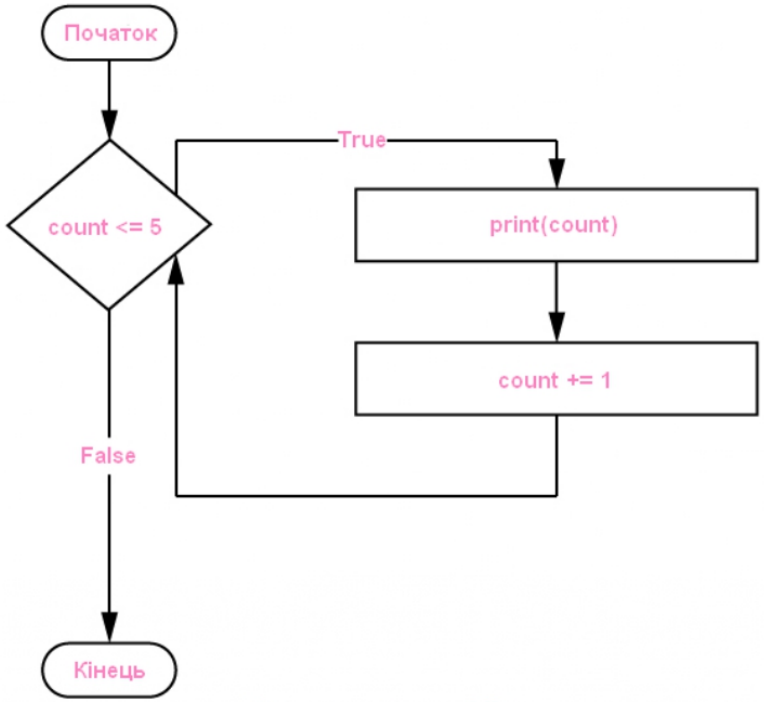
✓ *У циклі* ***while*** *умова завжди перевіряється на початку кожної ітерації - кожен раз, коли виконується цикл.* ***Ітерація*** *- один крок виконання циклу.*

ПРИКЛАД

```
count = 1
while count <= 5:
    print(count)
    count += 1
```

РЕЗУЛЬТАТ

```
1
2
3
4
5
```




*Блок-схема: цикл* ***while***

# Переривання циклу, break

ЩобЯкщо необхідно, щоб цикл виконувався до тих пір, поки щось не станеться, але точно невідомо, коли ця подія трапиться, можна скористатися **нескінченним циклом**, що містить оператор **break**. Якщо програма у процесі виконання досягає команди **break**, то виконання циклу відразу припиняється. отримати певне значення зі словника, ви вказуєте словник і ключ для цього значення:

```
stuff = ''
while True:
    print("String to capitalize [type q to quit]: ")
    stuff = input()
    if stuff == "q":
        break
    print(stuff.capitalize())
print('Thank you!')
```

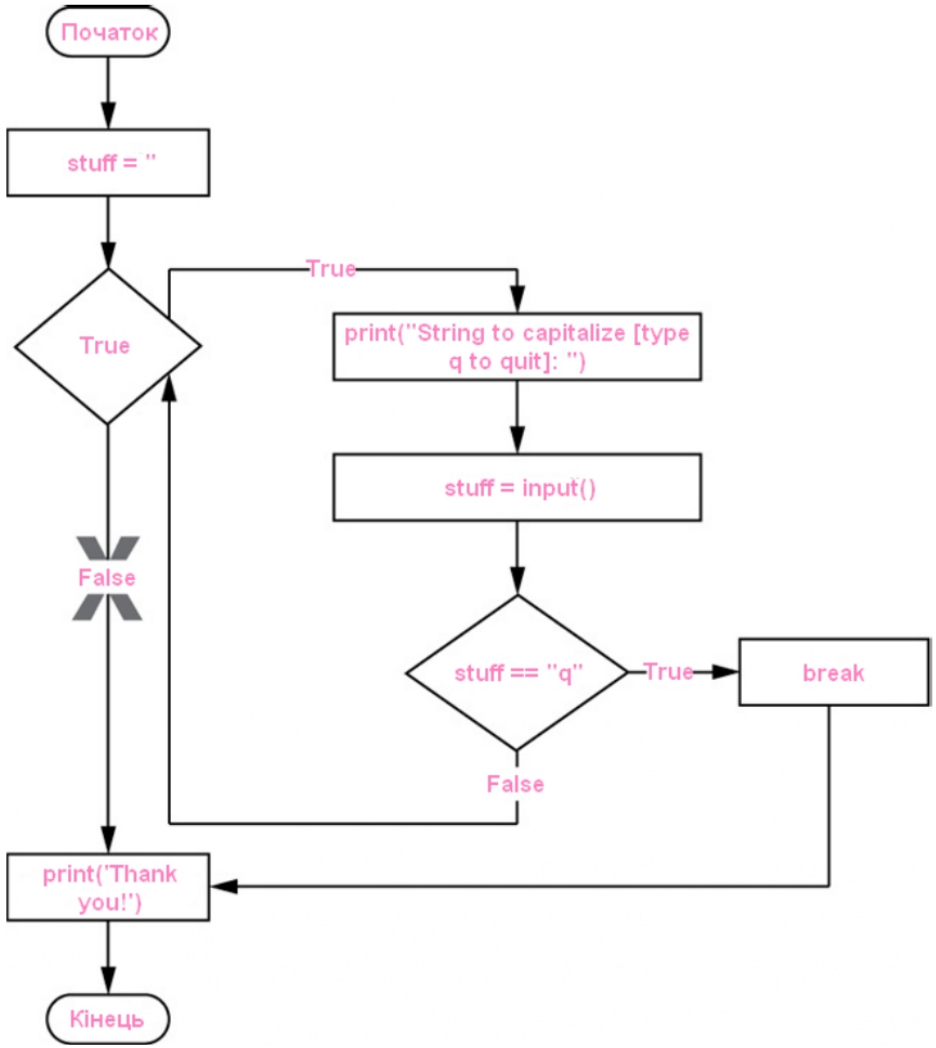
**Результатbreak**

```
String to capitalize [type q to quit]:
test
Test
String to capitalize [type q to quit]:
hey, it works!
Hey, it works!
String to capitalize [type q to quit]:
q
Thank you!
```


*Рядок while True: створює **нескінченний цикл** - умова такого циклу завжди істинна (True). Програма завжди буде виконувати команди циклу і вийде з нього тільки у тому випадку, якщо виконається інструкція **break**. Нескінченний цикл, вийти з якого неможливо, - розповсюджена помилка.*

# Переривання циклу, break

*Блок-схема: нескінченний цикл **while** і команда **break** для виходу з нього*



# Нескінченний цикл і вихід з нього

Якщо ваша програма потрапила в нескінченний цикл, натисніть сполучення клавіш **Ctrl+C**, в результаті чого програма припинить своє виконання.

```
while True:
    print('Hello, world!')
```

Дана програма в процесі виконання без зупинки буде повторювати виведення повідомлення `Hello, world!`, оскільки умова циклу **`while`** завжди істинна (`True`).

Щоб попередити **зациклювання** програми, проаналізуйте, як обробляється значення, що повинно привести до виходу з циклу. Перевірте, щоб виконання хоча б однієї частини програми могло б привести до того, щоб умова циклу стала дорівнювати `False` або була б виконана команда **`break`**. Наприклад, якщо у коді:

```
count = 1
while count <= 5:
    print(count)
    count += 1
```

# Продовження циклу, continue

Іноді потрібно не переривати весь цикл, а лише пропустити, по певній причині, одну ітерацію. Для цих речей використовується оператор **continue**.

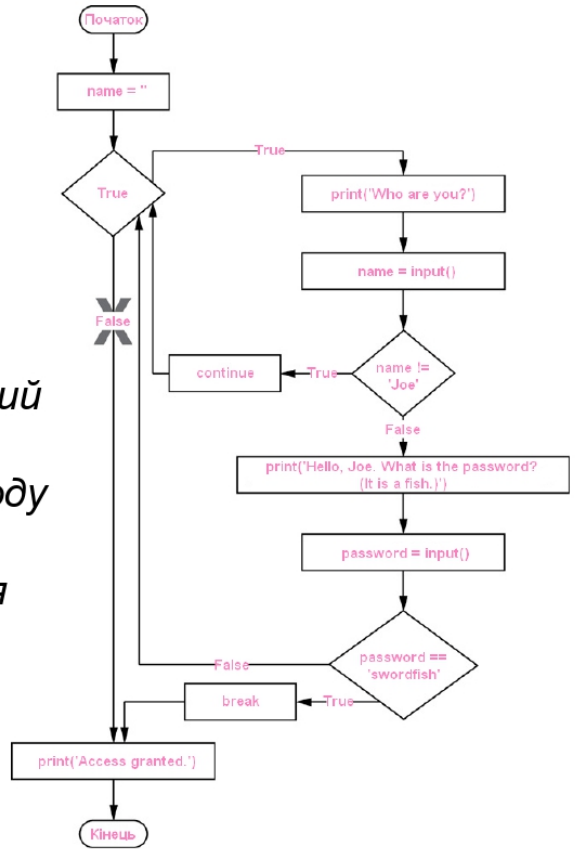
```
name = ''
while True:
    print('Who are you?')
    name = input()
    if name != 'Joe':
        continue
    print('Hello, Joe. What is the password? (It is a fish.)')
    password = input()
    if password == 'swordfish':
        break
print('Access granted.')
```

# Продовження циклу, continue

У випадку проходження ісінтрукції **if**, запитується пароль і, якщо користувач, вводить пароль swordfish, то виконується інструкція **break**, програма перериває цикл і виводить на екран повідомлення Access granted..
В протилежному випадку управління виконанням програми передається в кінець циклу **while** і відразу повертається на його початок.

```
Who are you?
Alice
Who are you?
Jack
Who are you?
Joe
Hello, Joe. What is the password? (It is a fish.)
shark
Who are you?
Joe
Hello, Joe. What is the password? (It is a fish.)
swordfish
Access granted
```

*Блок-схема: нескінченний цикл **while**, команда **break** для виходу з нього, команда **continue** для продовження циклу*



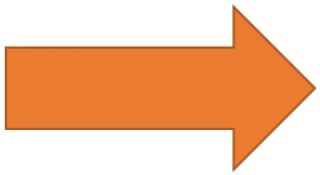
# Цикл for

```
for змінна in range():
    блок коду
```

В Python команда **for** завжди складається з таких елементів:

- ключове слово **for**
- ім’я змінної
- ключове слово **in**
- виклик функції **range()**, в яку можна передати до **трьох цілих чисел**, розділених комами
- двокрапка
- блок коду з відступом, що починається на наступному рядку

```
print('My name is')
for i in range(5):
    print('Anakin Skywalker (' + str(i) + ')')
```

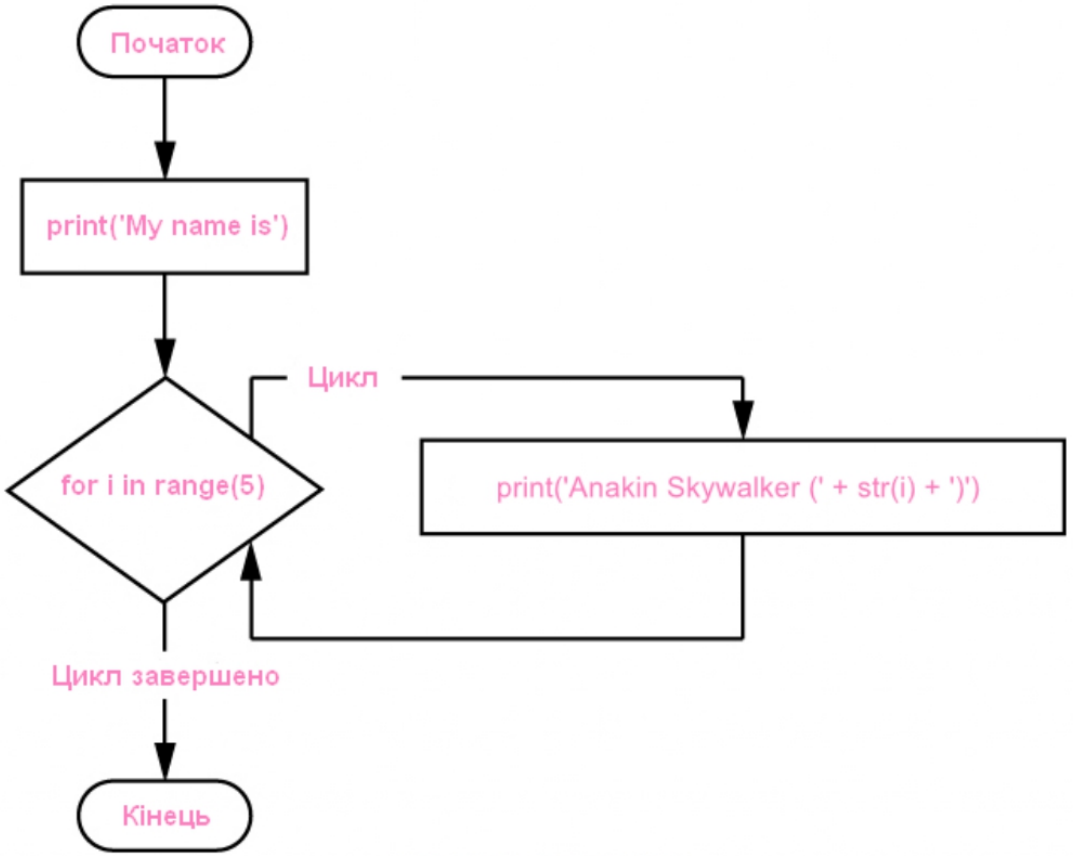


```
My name is
Anakin Skywalker (0)
Anakin Skywalker (1)
Anakin Skywalker (2)
Anakin Skywalker (3)
Anakin Skywalker (4)
```

# Цикл for

*Команди* **break** *і* **continue** *використовуються у циклі* **for** *по аналогії, як і у випадку з циклом* **while**.

```
print('My name is')
i = 0
while i < 5:
    print('Anakin Skywalker (' + str(i) + ')')
    i = i + 1 # i += 1
```




*Блок-схема: цикл* ***for***

# Функція range()

- У функцію **range()** можна передавати аргументи - значення **трьох цілих чисел**, розділених комами.
- Перше число визначає, з якого значення починає змінюватися змінна циклу **for**, друге число - значення, до якого може змінюватися змінна циклу **for** (число не входить у діапазон).

```
for i in range(12, 16):
    print(i)
```

```
12
13
14
15
```

- Також функцію **range()** можна викликати і з трьома аргументами. Перші два - визначають початок і кінець діапазону зміни значень змінної циклу **for**, а третій задає крок цієї зміни.

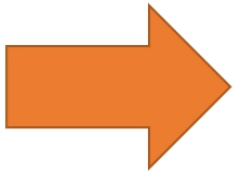
```
for i in range(1, 10, 2):
    print(i)
```

```
1
3
5
7
9
```

# Функція range()

- Функцію **range()** можна викликати також, задавши від’ємний крок.
- range(3, -4, -1):

```
for i in range(3, -4, -1):
    print(i)
```



```
3
2
1
0
-1
-2
-3
```

# Цикл for і послідовності

```
birds = ['pigeon', 'crow', 'owl', 'eagle']
for bird in birds:
    print(bird)
```

Результат:

```
pigeon
crow
owl
eagle
```

- Списки, на зразок birds, є прикладом ітеративних об’єктів у Python поряд з рядками, кортежами, словниками.
- Прохід (ітерація) по кортежу або списку повертає один елемент за раз.

```
word = 'crab'
for letter in word:
    print(letter)
```

Результат:

```
c
r
a
b
```

# Цикл for і послідовності

- *Ітерація по словнику у циклі (або з використанням функції **keys()**) повертає ключі словника.*

```
professions = {'business': 'economist', 'tv': 'newsreader', 'it': 'web developer', 'education':
'teacher'}
for key in professions: # або for key in professions.keys():
    print(key)
```

```
business
education
tv
it
```

- *Щоб виконувати ітерацію по значенням словника, а не за ключами, необхідно використовувати функцію **values()**.*

```
professions = {'business': 'economist', 'tv': 'newsreader', 'it': 'web developer', 'education':
'teacher'}
for value in professions.values():
    print(value)
```

```
teacher
newsreader
web developer
economist
```

# Цикл for і послідовності

- *Для отримання як ключа, так і значення із словника, ви можете використовувати функцію **items()**. Результатом буде кортеж із парами **«ключ: значення»**..*

```
professions = {'business': 'economist', 'tv': 'newsreader', 'it': 'web developer', 'education':
'teacher'}
for item in professions.items():
    print(item)
```

```
('business', 'economist')
('education', 'teacher')
('it', 'web developer')
('tv', 'newsreader')
```

```
professions = {'business': 'economist', 'tv': 'newsreader', 'it': 'web developer', 'education':
'teacher'}
for key, value in professions.items():
    print('Category', key, 'has a profession', value)
```

```
Category business has a profession economist
Category education has a profession teacher
Category it has a profession web developer
Category tv has a profession newsreader
```

# Цикл for і послідовності

- Для отримання впорядкованої копії ключів можна скористатися функцією **sorted()**:

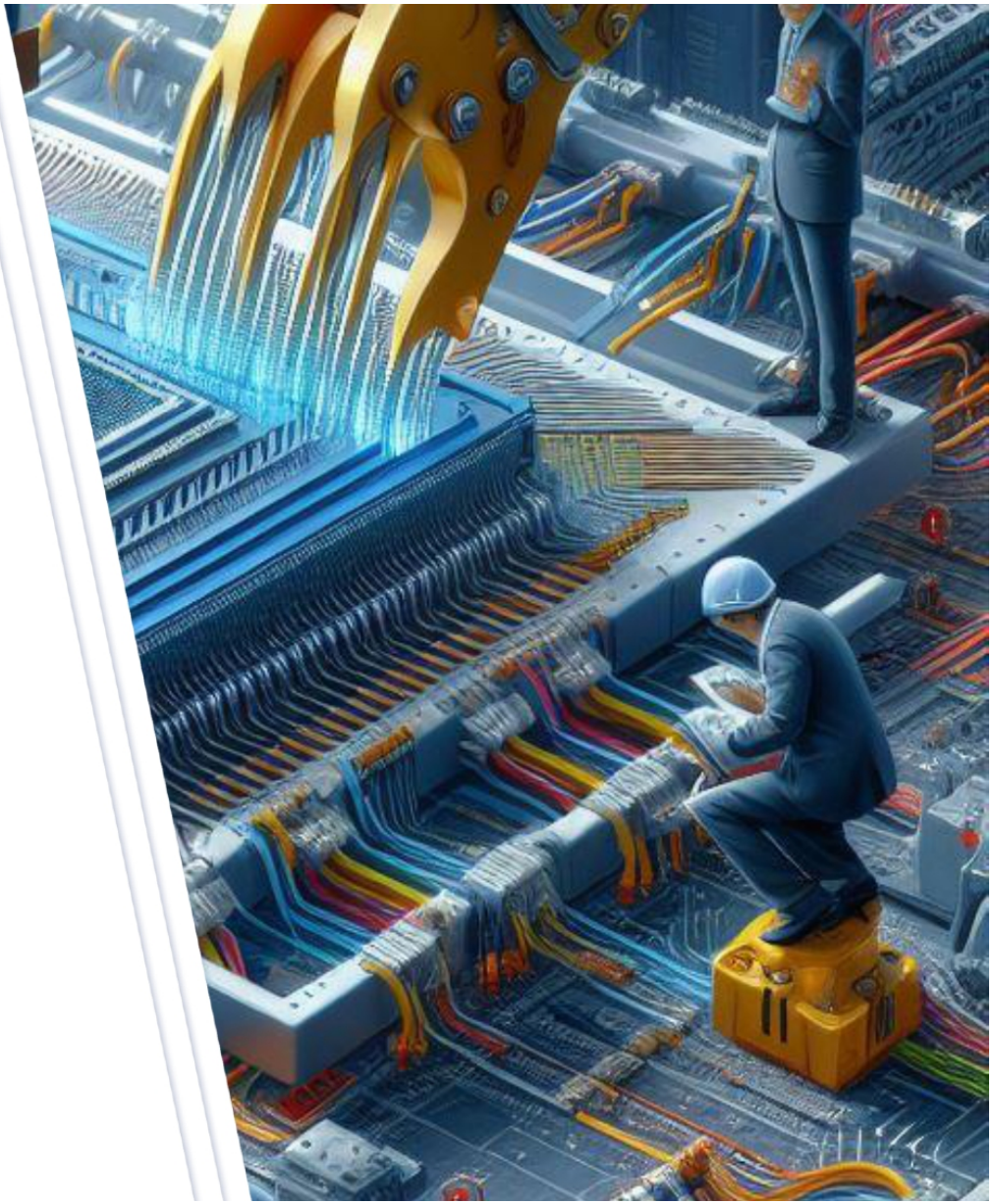
```
favorite_languages = {'john': 'Python', 'catherine': 'C', 'mary': 'Ruby', 'alex': 'Python'}
for name in sorted(favorite_languages.keys()):
    print(name.title() + ", thank you for taking the poll.")
```

```
Alex, thank you for taking the poll.
Catherine, thank you for taking the poll.
John, thank you for taking the poll.
Mary, thank you for taking the poll.
```

# ЗАВДАННЯ

Виведіть вітальне повідомлення для кожного користувача після його входу на сайт. Створіть список з кількох імен користувачів, включаючи ім’я 'Admin'. Перебираючи елементи списку, виведіть повідомлення для кожного користувача. Для користувача з ім’ям 'Admin' виведіть особливе повідомлення - наприклад: Hello Admin, I hope you’re well.. У інших випадках виводиться універсальне привітання - наприклад: Hello Alex, thank you for logging in again.. Додайте команду **if**, яка перевірить, що список користувачів не порожній. Якщо список порожній, виведіть повідомлення: We need to find some users!. Видаліть зі списку всі імена користувачів і переконайтеся у тому, що програма виводить правильне повідомлення.

# Функції



# Функції

***Функція*** *- це міні-програма у межах основної програми.*

Python надає у використання вбудовані функції, на зразок, **len(), print(), input()** тощо. Але можна створювати і власні функції.
Функцію можна:

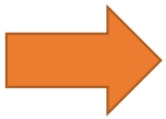
- визначити
- викликати

```
def назва_функції(вхідні параметри):
    блок коду
```

*Імена функцій дотримуються таких же правил, що і імена змінних: імена повинні починатися з* ***букви*** *або* ***знака підкреслення*** *і містити тільки* ***букви****,* ***цифри*** *або* ***знак підкреслення****.*

# Функції

```
def birthday_card():
    print('Happy birthday!')
```



```
Happy birthday!
```

Напишемо ще одну функцію без параметрів, яка **повертає** значення True і викличемо цю функцію в команді **if** для перевірки значення, яке повертає функція:

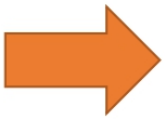
```
# визначення функції
def hungry():
    return True

# виклик функції та перевірка умови
if hungry():
    print('Eat oat flakes!')
else:
    print('Drink water.')
```

```
# визначення функції
def echo(anything):
    return anything
```

# Функції

Функція print() використовується для друкування на екрані значення, яке повернула функція оператором return.

```
print(echo('I enjoy travelling!'))
```



```
I enjoy travelling!
```

*Значення, які передаються у функцію при виклику, називаються **аргументами**. Коли викликається функція з аргументами, значення цих аргументів копіюються у відповідні **параметри** всередині функції.*

```
def like(interest):
    if interest == 'yoga':
        return "I quite like " + interest + "."
    elif interest == "football":
        return "Yes, I play " + interest + "."
    elif interest == 'guitar':
        return "Yes, I play the " + interest + "."
    else:
        return "I'm interested in " + interest + "."
story = like('photography')
print(story)
story = like('yoga')
print(story)
story = like('football')
print(story)
story = like('guitar')
print(story)
```

Функція like() виконає такі дії:

- параметру функції interest присвоїть значення 'photography' ('yoga', 'football', 'guitar')
- пройде по ланцюжку if-elif-else
- поверне рядок з повідомленням
- присвоїть повернений рядок змінній story

# Значення None

❑ ***pass*** *- оператор-заглушка, рівноцінний відсутності операції.*

```
do_nothing()
```

Функція `do_nothing()` відпрацює, але нічого не виведе на екран.

Викличемо цю ж функцію з використанням функції `print()`:

```
print(do_nothing())
```

У результаті отримаємо значення:

```
None
```

```
def do_nothing():
    pass
```

Якщо функція не використовує оператор **return** явно, вона повертає результат None.

# Значення None

```
def is_none(thing):
    if thing is None:
        print("It's None")
    elif thing:
        print("It's True")
    else:
        print("It's False")
```

і виконаємо кілька перевірок:

```
is_none(None)  # It's None
is_none(True)  # It's True
is_none(False) # It's False
is_none(0)     # It's False
is_none(0.0)   # It's False
is_none(())    # It's False
is_none([])    # It's False
is_none({})    # It's False
is_none(set()) # It's False
```

# Функція enumerate()

- Приклад вбудованих функцій Python **range()** і **len()**:

```
popular_sites = ['Google.com', 'Youtube.com', 'Facebook.com', 'Baidu.com', 'Wikipedia.org',
'Yahoo.com' ,'Amazon.com']
for index in range(len(popular_sites)):
    print(index, popular_sites[index])
```

```
0 Google.com
1 Youtube.com
2 Facebook.com
3 Baidu.com
4 Wikipedia.org
5 Yahoo.com
6 Amazon.com
```

- Функція **enumerate()** повертає кортеж (індекс, елемент) для кожного елемента в послідовності:

```
popular_sites = ['Google.com', 'Youtube.com', 'Facebook.com', 'Baidu.com', 'Wikipedia.org',
'Yahoo.com' ,'Amazon.com']
for index, value in enumerate(popular_sites, 1):
    print(index, value)
```

```
1 Google.com
2 Youtube.com
3 Facebook.com
4 Baidu.com
5 Wikipedia.org
6 Yahoo.com
7 Amazon.com
```

# Функція zip()

- Функція **zip()** використовується для паралельної ітерації по декільком послідовностям одночасно.

```
days = ['Monday', 'Tuesday', 'Wednesday']
fruits = ['coconut', 'lemon', 'mango']
drinks = ['coffee', 'tea', 'fruit juice']
desserts = ['marmalade', 'ice cream', 'pie', 'pudding']
for day, fruit, drink, dessert in zip(days, fruits, drinks, desserts):
    print(day, ": drink", drink, "eat", fruit, "enjoy", dessert)
```

```
Monday : drink coffee eat coconut enjoy marmalade
Tuesday : drink tea eat lemon enjoy ice cream
Wednesday : drink fruit juice eat mango enjoy pie
```

- *Функцію **zip()** можна використати, щоб пройти по декільком послідовностям і створити кортежі з елементів із однаковими індексами.*

```
english = 'Monday', 'Tuesday', 'Wednesday'
french = 'Lundi', 'Mardi', 'Mercredi'
```

```
print(list(zip(english, french)))
```

```
[('Monday', 'Lundi'), ('Tuesday', 'Mardi'), ('Wednesday', 'Mercredi')]
```

```
print(dict(zip(english, french)))
```

```
{'Wednesday': 'Mercredi', 'Monday': 'Lundi', 'Tuesday': 'Mardi'}
```

# Функція map()

Створимо список my_str_list = ['1', '2', '3', '4'], що містить числові рядки. Перетворимо поданий список у список цілих чисел

```
result = []
for item in my_str_list:
    result.append(int(item))
print(result)
```

```
[1, 2, 3, 4]
```

```
result = tuple()
for item in my_str_list:
    result += (item,)
print(result)
```

```
('1', '2', '3', '4')
```

```
result = set()
for item in my_str_list:
    result.add(float(item))
print(result)
```

```
{1.0, 2.0, 3.0, 4.0}
```

# Функція map()

*Функція **map()** застосовує вказану функцію (наприклад, **int()**, **float()**, **str()** тощо) до кожного елемента з послідовності (списка, кортежа, множини тощо). Отриманий результат можна перетворити в **list()**, **tuple()**, **set()** тощо.*

```
>>> my_str_list = ['1', '2', '3', '4'] 1
>>> my_str_list
['1', '2', '3', '4']
>>> my_int_list = list(map(int, my_str_list)) 2
>>> my_int_list
[1, 2, 3, 4]
>>> my_float_list = list(map(float, my_str_list)) 3
>>> my_float_list
[1.0, 2.0, 3.0, 4.0]
>>> my_new_str_list = list(map(str, my_float_list)) 4
>>> my_new_str_list
['1.0', '2.0', '3.0', '4.0']
>>> my_tuple = tuple(map(int, my_str_list)) 5
>>> my_tuple
(1, 2, 3, 4)
>>> my_set = set(map(float, my_str_list)) 6
>>> my_set
{1.0, 2.0, 3.0, 4.0}
>>> my_newer_str_list = list(map(str, my_tuple)) 7
>>> print(my_newer_str_list)
['1', '2', '3', '4']
```

1.Створення списку my_str_list з числовими рядками.
2.Використання функції **map()**: функція **int()** застосовується до кожного елемента списку числових рядків my_str_list для перетворення елементів на цілі числа, з яких утворюється список my_int_list цілих чисел.
3.Використання функції **map()**: функція **float()** застосовується до кожного елемента списку числових рядків my_str_list для перетворення елементів на дійсні числа, з яких утворюється список my_float_list дійсних чисел.
4.Використ ання функції **map()**: функція **str()** застосовується до кожного елемента списку дійсних чисел my_float_list для перетворення елементів на рядки, з яких утворюється список my_new_str_list з рядками дійсних чисел.
5.Використання функції **map()**: функція **int()** застосовується до кожного елемента списку числових рядків my_str_list для перетворення елементів на цілі числа, з яких утворюється кортеж my_tuple з цілими числами.
6.Використання функції **map()**: функція **float()** застосовується до кожного елемента списку числових рядків my_str_list для перетворення елементів на дійсні числа, з яких утворюється множина my_set з дійсними числами.
7.Використання функції **map()**: функція **str()** застосовується до кожного елемента кортежу з цілих чисел my_tuple для перетворення елементів на числові рядки, з яких утворюється список my_newer_str_list з числовими рядками.

# Позиційні та іменовані аргументи

- Найбільш поширений тип аргументів - це **позиційні аргументи**, чиї значення копіюються у відповідні параметри функції згідно порядку слідування. Тому, варто пам’ятати кожну позицію аргументів.

```
def music(terminology, musician, genre):
    return {'terminology': terminology, 'musician': musician, 'genre': genre}
```

- Викличемо функцію командою print(music('tune', 'guitarist', 'blues')). Значення tune, guitarist, blues у виклику функції і є **позиційними аргументами**, які будуть скопійовані у відповідні параметри функції і вона поверне словник (порядок у словнику непередбачуваний до версії Python 3.6):

```
{'terminology': 'tune', 'musician': 'guitarist', 'genre': 'blues'}
```

# Позиційні та іменовані аргументи

```
{'terminology': 'tune', 'musician': 'guitarist', 'genre': 'blues'}
```

Аргументи у виклику функції можна вказувати і за допомогою **іменованих аргументів** - імен відповідних параметрів. Порядок слідування аргументів, у цьому випадку, може бути яким завгодно:

```
print(music(terminology='tune', musician='guitarist', genre='blues'))
```

У даному виклику функції `music` : `musician` - це ім'я аргументу, а `'guitarist'` - значення аргументу. Аналогічно і для інших імен і значень аргументів. У результаті словник набуде такого вигляду:

```
{'terminology': 'tune', 'musician': 'guitarist', 'genre': 'blues'}
```

Можна, також, об'єднувати позиційні аргументи та іменовані аргументи, тільки позиційні аргументи завжди вказуються першими:

```
print(music('tune', musician='guitarist', genre='blues'))
```

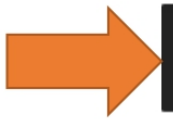
Результат виведення:

```
{'terminology': 'tune', 'musician': 'guitarist', 'genre': 'blues'}
```

# Значення за замовчуванням

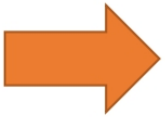
Для кожного параметра функції можна визначити **значення за замовчуванням**.

```
def describe_pet(pet_name, animal_type='parrot'):
    print("I have a " + animal_type + ".")
    print("My " + animal_type + "'s name is " + pet_name.title() + ".")
```



```
describe_pet(pet_name='Jack Sparrow')
```

```
I have a parrot.
My parrot's name is Jack Sparrow.
```

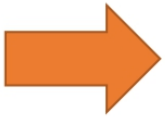


```
describe_pet(pet_name='peppa', animal_type='guinea pig')
```

# Використання аргументів з символами * і **

*Якщо символ * буде використаний усередині функції з параметром, довільна кількість позиційних аргументів буде згрупована у кортеж. В наступному прикладі визначення функції* ***print_days()***

```
def print_days(*args):
    print('Get ready:', args)
```



```
print_days('Wednesday', 'Thursday', 'Friday', 'Weekend...')
```

```
Get ready: ('Wednesday', 'Thursday', 'Friday', 'Weekend...')
```

# Використання аргументів з символами * і **

Є також обов’якові параметри (в даному випадку required1 і required2), у які будуть скопійовані значення позиційних аргументів ('return ticket', 'train') при виклику функції

```
print_travel('return ticket', 'train', 'carriage', 'seat', 'luggage rack')
```

```
For train travel is required: return ticket
For train travel is required too: train
All the rest: ('carriage', 'seat', 'luggage rack')
```

# Використання аргументів з символами * і **

У наступному прикладі визначається функція `print_character()`

```
def print_character(**kwargs):
    print('Person\'s characteristics:', kwargs)
```

при виклику якої з іменованими аргументами

```
print_character(emotions='cheerful', sense='happy', look='slim')
```

виводяться її іменовані аргументи у вигляді словника:

```
Person's characteristics: {'emotions': 'cheerful', 'sense': 'happy', 'look': 'slim'}
```

*При використанні символа * не потрібно обов'язково називати кортеж параметрів args, аналогічно при використанні символів ** називати словник параметрів kwargs, однак це поширена практика у Python.*

# Анонімні функції: інструкція lambda

Python дозволяє в короткій формі оголошувати невеликі анонімні функції - лямбда-функції. Вони ведуть себе точно так само, як і звичайні функції, які оголошуються ключовим словом **def**. Анонімні функції створюються за допомогою інструкції **lambda**:

```
add = lambda x, y: x + y
```

Та ж сама функція `add` може бути визначена за допомогою ключового слова `def`:

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

Виконавши код

```
print(add(7, 8))
```

ми отримаємо однаковий результат в обох випадках:

```
15
```

Крім цього, анонімну функцію не обов'язково присвоювати змінній

```
print((lambda x, y: x + y)(5, 12))
```

# Приклад

Наведено фрагмент прогами Пайтон, яка реалізує функцію, яка приймає список чисел і повертає їхнє максимальне та мінімальне значення.

```python
def знайди_максимальне_і_мінімальне(список_чисел):
    """
    Функція, яка приймає список чисел і повертає їхнє максимальне та мінім
    """
    if not список_чисел:
        return None, None  # Повертаємо None для порожнього списку

    максимальне_значення = мінімальне_значення = список_чисел[0]

    for число in список_чисел:
        if число > максимальне_значення:
            максимальне_значення = число
        elif число < мінімальне_значення:
            мінімальне_значення = число

    return максимальне_значення, мінімальне_значення

# Приклад використання функції
список = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
макс, мін = знайди_максимальне_і_мінімальне(список)

print(f"Максимальне значення: {макс}")
print(f"Мінімальне значення: {мін}")
```

# ЗАВДАННЯ

Напишіть функцію, яка приймає число і повертає його факторіал.

---

Створіть функцію, яка приймає два числа (нижню та верхню межі) і повертає суму чисел у цьому діапазоні.

---

Напишіть функцію, яка перевіряє, чи є задане слово чи фраза паліндромом.

---

Розробіть функцію, яка конвертує температуру з градусів Цельсія в Фаренгейти або навпаки.

# Модулі та пакети

# Теоретичні відомості

- Те, що вважається стандартною бібліотекою в Python, складається з декількох компонентів, до яких входять вбудовані типи даних і константи, які можуть використовуватися без команди імпортування, як, наприклад, числа і списки.
- Будь-якій програмі на мові Python доступний базовий набір вбудованих функцій, в число яких входять такі функції як **print()**, **input()**, **len()** тощо, які можна використовувати у програмі, як кажуть, «з коробки».
- Наприклад, модуль **math** містить математичні функції, модуль **random** - функції для роботи з випадковими числами.

# Що таке модуль?

- Модуль являє собою файл з програмним кодом. Він визначає групу функцій Python або інших об’єктів, а ім’я модуля визначається ім’ям файла.
- Багато стандартних функцій Python не вбудовані в основне ядро мови, а надаються конкретними модулями, що завантажуються в міру необхідності.
- Наприклад, ви пишете для своєї програми модуль з ім’ям firstmodule, який визначає функцію rounded. У тій же програмі може використовуватися модуль secondmodule, який також визначає функцію з ім’ям rounded, яка виконує якусь відмінну від вашої функції rounded дію.

# Простір імен і області видимості

- Простір імен в Python представляє собою спосіб зберігання в Python інформації про активні змінні й про ті об’єкти, на які вони вказують.
- У блоці коду, що виконується в Python, є три простори імен: локальний, глобальний і вбудований.
- В основній програмі визначається глобальний простір імен, тому змінні, що знаходяться у цьому просторі імен, є **глобальними**.

```
animal = 'kangaroo'
def print_global():
    print('inside print_global:', animal)
```

Використавши функцію `print()` і виклик функції `print_global()`

```
print('at the top level:', animal)
print_global()
```

отримаємо значення змінної `animal` в обох випадках:

```
at the top level: kangaroo
inside print_global: kangaroo
```

# Простір імен і області видимості

Але якщо спробувати отримати значення глобальної змінної і змінити його всередині функції

```
animal = 'kangaroo'
def change_and_print_global():
    print('inside change_and_print_global:', animal)
    animal = 'giraffe'
    print('after the change:', animal)
change_and_print_global()
```

то отримаємо помилку:

```
...
... line 3, in change_and_print_global
    print('inside change_and_print_global:', animal)
UnboundLocalError: local variable 'animal' referenced before assignmentassignment
```

# Простір імен і області видимості

- Розглянемо програму з іншою функцією, яка містить змінну з такою ж назвою, що і глобальна змінна animal:

```
animal = 'kangaroo' 1
def change_local():
    animal = 'giraffe' 2
    print('inside change_local:', animal, id(animal)) 3
change_local()
print(animal)
print(id(animal)) 4
```

```
inside change_local: giraffe 47293440
kangaroo
47287896
```

1. Ми присвоїли рядок 'kangaroo' глобальній змінній з іменем animal.
2. Функція change_local() також має змінну animal, але ця змінна знаходиться в локальному просторі імен функції.
3. Ми використали функцію **id()**, щоб вивести на екран унікальне значення об’єкта animal всередині функції.
4. Ми використали функцію **id()**, щоб вивести на екран унікальне значення об’єкта animal ззовні функції.

# Простір імен і області видимості

- Щоб змінна всередині функції була **видимою** в основній програмі, потрібно явно використовувати ключове слово **global**. Перепишемо попередню програму таким чином

```
animal = 'kangaroo'
def change_and_print_global():
    global animal
    animal = 'giraffe'
    print('inside change_and_print_global:', animal)
print(animal) 1
change_and_print_global() 2
print(animal) 3
```

```
kangaroo 1
inside change_and_print_global: giraffe 2
giraffe 3
```

1. До виклику функції change_and_print_global() глобальна змінна animal мала значення 'kangaroo'.
2. Виклик функції change_and_print_global() з використанням ключового слова **global** у функції перетворив локальну змінну animal на глобальну і переписав її значення з 'kangaroo' на 'giraffe'.
3. Тепер змінна animal в основній програмі має значення 'giraffe'.

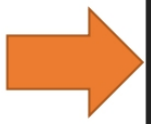
# Простір імен і області видимості

- Якщо не використовувати ключове слово global всередині функції, Python використовує локальний простір імен і змінна буде локальною. Вона зникає після того, як функція завершує роботу.

Python надає дві функції для доступу до вмісту ваших просторів імен:

- locals() - повертає словник, що містить імена локального простору імен
- globals() - повертає словник, що містить імена глобального простору імен

```
animal = 'kangaroo' # глобальна змінна
def change_local():
    animal = 'giraffe' # локальна змінна
    print('locals:', locals()) 1
print(animal)
change_local()
print('locals:', locals()) 2
print('globals:', globals()) 3
print(animal)
```



```
kangaroo
locals: {'animal': 'giraffe'} 1
locals: {'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__':
<_frozen_importlib_external.SourceFileLoader object at 0x0000021BB6A8C748>, '__spec__': None,
'__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>, '__file__':
'C:\\Python36\\namespace.py', '__cached__': None, 'animal': 'kangaroo', 'change_local': <function
change_local at 0x0000021BB6901E18>} 2
globals: {'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__':
<_frozen_importlib_external.SourceFileLoader object at 0x0000021BB6A8C748>, '__spec__': None,
'__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>, '__file__':
'C:\\Python36\\namespace.py', '__cached__': None, 'animal': 'kangaroo', 'change_local': <function
change_local at 0x0000021BB6901E18>} 3
kangaroo
```

# Простір імен і області видимості

- Як бачимо, глобальні та локальні простори імен змінних основної програми збігаються. Вони містять три вхідні пари «ключ: значення», призначені для внутрішнього використання Python:

- рядок документації __doc__

- ім’я основного модуля __name__ (для інтерактивних сеансів і сценаріїв, запущених з файлів, завжди містить __main__)

- модуль, який використовується для вбудованого простору імен (модуль __builtins__).

- Вбудованим простором імен є простір імен модуля __builtins__. Цей модуль містить всі вбудовані функції, такі, як len(), min(), max(), int(), float(), list(), tuple(), range(), str() та інші.
- Зверність увагу, іноді збиває з пантелику той факт, що ви можете перевизначати елементи у вбудованому модулі __builtins__.

# Простір імен і області видимості

- Наприклад, якщо створити список і зберегти його в змінній з ім'ям list, ви в подальшому не зможете використовувати вбудовану функцію **list()** із вбудованого простору імен:

```
>>> list = [1, 2, 3] 1
>>> list((4, 5, 6))  2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'list' object is not callable
```

1. Створення списку із ім'я list, яке вже зарезервоване функцію list().
2. Спроба використати функцію list() для перетворення кортежу (4, 5, 6) у список є невдалою, так як тепер ім'я list - це ім'я змінної, що посилається на числовий список, а не на функцію.

Тепер ім'я list пов'язане з вашою змінною, хоча ви і використовуєте синтаксис вбудованої функції list(). Щоб повернути ситуацію у початковий стан, необхідно перезавантажити середовище інтерпретатора Python.

```
>>> list((4, 5, 6))
[4, 5, 6]
```

У глобальному просторі імен globals, наприклад, основній програмі присвоєно спеціальне ім'я __main__, обрамлене по обидва боки символами __. Такі імена заразервовані для використання всередині Python, тому їх використання для імен власних змінних неприпустимі. Це також стосується зарезервованих імен вбудованих функцій, модулів та інших об'єктів.

# Імпорт модулів: інструкція import

*Щоб використовувати функції, які входять у модуль, необхідно його **імпортувати** (підключити) у програму за допомогою інструкції **import**.*

```
import модуль
```

де **модуль** - це ім’я без розширення **.py** іншого файла Python.
Як тільки модуль імпортований, ви можете використовувати будь-яку функцію, яка входить до його складу.
Наприклад, щоб імпортувати модуль nameofmodule в простір імен

```
import nameofmodule
```

# Модуль random: правила імпорту

*Модуль **random** надає функції для генерації псевдовипадкових чисел, букв, випадкового вибору елементів послідовності. Псевдовипадкові генератори цього модуля непридатні для криптографічного використання.*

*Перед використанням функцій модуля необхідно його імпортувати таким чином:* import назва_модуля . *Для використання функцій модуля необхідно використовувати такий запис:* назва_модуля.назва_функції .

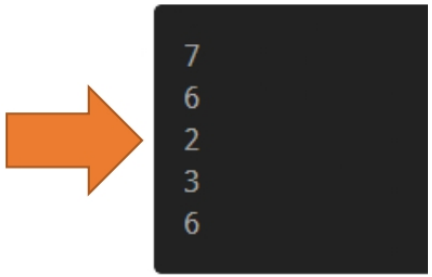
*Таблиця "Функції для генерування псевдовипадкових чисел"*

| Виклик функції | Опис |
|---|---|
| `random.randrange(start, stop, step)` | Випадкове ціле число із діапазону від `start` до `stop` |
| `random.randint(A, B)` | Випадкове ціле число `N` ( `A <= N <= B` ) |
| `random.choice(sequence)` | Випадковий елемент непорожньої послідовності `sequence` |
| `random.sample(population, k)` | Список довжиною `k` із послідовності `population` |
| `random.random()` | Випадкове число від `0` до `1` |

# Модуль random: правила імпорту

*Приклад*

```
import random
for i in range(5):
    print(random.randint(1, 10))
```



```
7
6
2
3
6
```

Для імпортування декількох модулів, використовують такий синтаксис:

```
import модуль1, модуль2, модуль3,...
```

або імпортують у такому вигляді:

```
import модуль1
import модуль2
import модуль3, модуль4
```

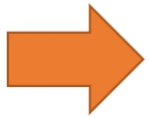
*Всі інструкції* `import` *рекомендують розміщувати у верхній частині файла.*

# Модуль random: правила імпорту

*Оператор import має альтернативну форму використання:*

```
from модуль import функція1, функція2,...
```

```
from random import randint, random
for i in range(5):
    print(randint(1, 10))
print(random())
```



```
8
7
3
6
10
0.12147106708747124 # результат виконання функції random()
```

Функція **random()** викликається без аргументів і генерує випадкове дійсне число від 0 до 1 не включаючи меж даного діапазону.
Ім'я функції, перед яким записане ім'я модуля, наприклад, **random.randint(),** використовувати безпечніше. Тому краще всього користуватися звичайною формою інструкції імпорту: import назва_модуля.

# Модуль random: правила імпорту

*При імпортуванні можна використовувати символ *. Для нашого прикладу з модулем random()*

```
from random import *
```

Для скорочення назв імпортованих модулів (або функцій із модулів) можна також користуватися **псевдонімами**.

```
import модуль as псевдонім # псевдонім для модуля
from модуль import функція as псевдонім # псевдонім для функції
```

```
import random as rn
for i in range(5):
    print(rn.randint(1, 10))
```

# Модуль math

*Таблиця "Математичні функції і константи"*

| Виклик функції/константи | Опис/значення |
|---|---|
| `math.sin(x)` | Синус x (представлений у радіанах) |
| `math.cos(x)` | Косинус x (представлений у радіанах) |
| `math.radians(x)` | Перетворення x (в градусах) у радіани |
| `math.degrees(x)` | Перетворення x (в радіанах) у градуси |
| `math.exp(x)` | Експонентна функція `x` ($e^x$) |
| `math.sqrt(x)` | Квадратний корінь x |
| `math.factorial(x)` | Факторіал цілого числа x |
| `math.e` | `2.718281828459045` |
| `math.pi` | `3.141592653589793` |

# Модуль math

```
import math
```

Використаємо інтерактивний режим інтерпретатора `Python` для дослідження даної бібліотеки. Ця бібліотека містить такі константи, як `pi` (число Пі) і `e` (експонента):

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
```

# Модуль math

Розглянемо кілька корисних функцій бібліотеки `math`:

```
>>> math.fabs(-221.1) 1
221.1
>>> math.floor(56.8) 2
56
>>> math.ceil(38.3) 3
39
>>> math.factorial(7) 4
5040
>>> math.pow(4, 3) 5
64.0
>>> math.sqrt(256) 6
16.0
>>> math.radians(180) 7
3.141592653589793
>>> math.degrees(math.pi) 8
180.0
```

1. Повертає абсолютне значення.
2. Округлення вниз.
3. Округлення вгору.
4. Обчислення факторіалу.
5. Піднесення числа до степеня.
6. Обчислення кореня квадратного з числа.
7. Перетворення значення в градусах у радіани.
8. Перетворення значення в радіанах у градусну міру.

# Модуль math

- Щоб дізнатися увесь список функцій імпортованого модуля math, використаємо функцію dir():

```
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin',
'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf',
'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd',
'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10',
'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh',
'tau', 'trunc']
>>>
```

- *Функція **dir()** повертає список атрибутів об'єкта, впорядкований за алфавітом. Ці атрибути є **методами** - функціями, пов'язаними з об'єктами.*

# Модуль math

- Якщо ви хочете викликати функцію **factorial()**, наприклад, для значення 3, зробити це не вдасться, тому що ім'я factorial не входить у простір імен - у просторі імен лише змінна math.
- Однак, ви можете провести пошук атрибута factorial по змінній math з використанням оператора **крапка** (цей оператор переглядає атрибути об'єкта):

```
>>> math.factorial(3)
6
```

Якщо ви захочете прочитати документацію по функції `factorial()`, скористайтеся функцією `help()`.

*Функція* `help()` *виводить документацію для методу, модуля, класу, функції тощо.*

Якщо вас цікавить, що робить атрибут `pow` в модулі `math`, виконайте наступний код:

```
>>> import math
>>> help(math.pow)
Help on built-in function pow in module math:

pow(...)
    pow(x, y)

    Return x**y (x to the power of y).

>>>
```

# Стандартна бібліотека Python: короткий огляд модулів

У модулі `string` визначена функція `capwords()`, яка переводить у верхній регістр першу букву кожного слова в рядку.

```
import string
s = 'It was nice talking to you! Thank you!'
print(s)
print(string.capwords(s))
```

Результатом роботи функції буде таким:

```
It was nice talking to you! Thank you!
It Was Nice Talking To You! Thank You!
```

# Стандартна бібліотека Python: короткий огляд модулів

```python
import inspect
import string

def is_str(value):
    return isinstance(value, str)

for name, value in inspect.getmembers(string, is_str):
    if name.startswith('_'):
        continue
    print('{0} = {1}'.format(name, repr(value)))
```

```
ascii_letters = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
ascii_lowercase = 'abcdefghijklmnopqrstuvwxyz'
ascii_uppercase = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
digits = '0123456789'
hexdigits = '0123456789abcdefABCDEF'
octdigits = '01234567'
printable = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()*+,-./:;<=>?
@[\\]^_`{|}~ \t\n\r\x0b\x0c'
punctuation = '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
whitespace = ' \t\n\r\x0b\x0c'
```

# Стандартна бібліотека Python: короткий огляд модулів

- Модуль **inspect** може допомогти вам вивчити вміст класу, отримати початковий код методу, отримати і відформатувати список аргументів для функції, або отримати всю інформацію, необхідну для відображення детального трасування (інформація про хід виконання програми).
- Функція isinstance() використовується для перевірки приналежності даних певному типу даних. Результатом виконання функції є значення True або False.
- Функція repr() повертає представлення об’єкта у вигляді текстового рядка.

# Модуль collections, функція OrderedDict()

## Функція **Counter()**

```
from collections import Counter
birds = ['stork', 'sparrow', 'woodpecker', 'owl', 'woodpecker', 'sparrow', 'sparrow']
birds_counter = Counter(birds)
print(birds_counter)
print(birds_counter['woodpecker'])
print(birds_counter['sparrow'])
```

```
Counter({'sparrow': 3, 'woodpecker': 2, 'stork': 1, 'owl': 1})
2
3
```

Визначимо ще один список `ther_birds` і порахуємо скільки птахів з нього зустрічається у списку `birds`:

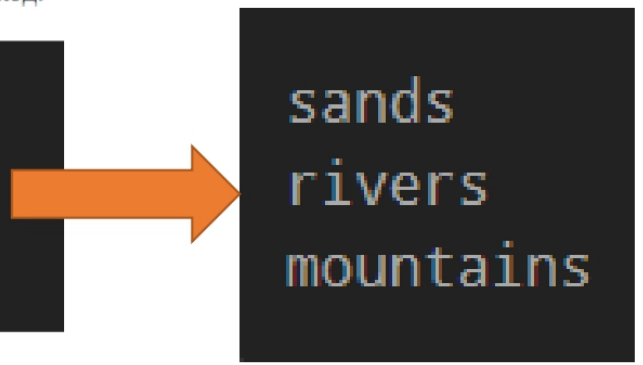
```
from collections import Counter
birds = ['stork', 'sparrow', 'woodpecker', 'owl', 'woodpecker', 'sparrow', 'sparrow']
birds_counter = Counter(birds)
other_birds = ['flamingo', 'nightingale', 'stork', 'sparrow']
for bird in other_birds:
    print('{0} : {1}'.format(bird, birds_counter[bird]))
```

```
flamingo : 0
nightingale : 0
stork : 1
sparrow : 3
```

# Модуль collections, функція OrderedDict()

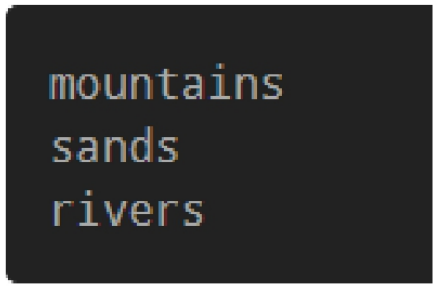
Порядок вставки елементів у словник не можна передбачити (до версії `Python 3.6`). Розглянемо такий код:

```
regions = {
    'sands': 'Sahara',
    'mountains': 'Himalayas',
    'rivers': 'Yangtze',
}
for region in regions:
    print(region)
```




Функція **OrderedDict(),** яка міститься у модулі collections, запам’ятовує порядок, в якому додавалися ключі, і повертає їх в тому ж порядку.

```
from collections import OrderedDict
regions = OrderedDict([
    ('mountains', 'Himalayas'),
    ('sands', 'Sahara'),
    ('rivers', 'Yangtze')
])
for region in regions:
    print(region)
```



# Модуль pprint, функція pprint()

Щоб виводити інформацію на екран використовують функцію **print().** У випадку, коли результати виведення важко прочитати, можна використовувати pretty printer («гарний друк») - функцію **pprint(),** яка міститься у модулі **pprint:**

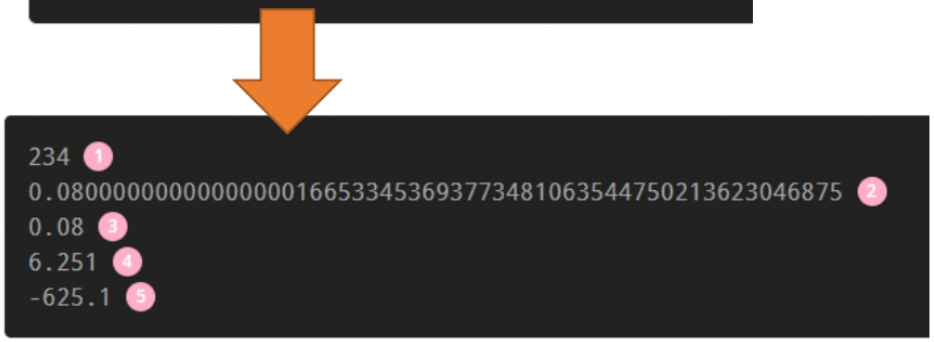
```
from collections import OrderedDict
from pprint import pprint
regions = OrderedDict([
    ('mountains', 'Himalayas'),
    ('sands', 'Sahara'),
    ('rivers', 'Yangtze')
])
print(regions)
pprint(regions)
```

```
OrderedDict([('mountains', 'Himalayas'), ('sands', 'Sahara'), ('rivers', 'Yangtze')])
OrderedDict([('mountains', 'Himalayas'),
             ('sands', 'Sahara'),
             ('rivers', 'Yangtze')])
```

# Модуль decimal, функції Decimal(), getcontext()

- Після імпорту з модуля **decimal** функції **Decimal()**, можна створювати десяткові числа з цілих чисел, рядків, чисел з плаваючою крапкою чи кортежів.

```
from decimal import Decimal
print(Decimal(234)) 1
print(Decimal(0.08)) 2
print(Decimal('0.08')) 3
print(Decimal((0, (6, 2, 5, 1), -3))) 4
print(Decimal((1, (6, 2, 5, 1), -1))) 5
```



```
234 1
0.080000000000000001665334536937734810635447502136230468752
0.08 3
6.251 4
-625.1 5
```

1. Точне представлення цілого числа у десятковій формі.
2. Наближене представлення числа з плаваючою крапкою у десятковій формі. Кількість значущих цифр не може бути точно виражена з використанням апаратного представлення чисел с плаваючою крапкою.
3. Точне представлення числа з плаваючою крапкою у десятковій формі. Числа с плаваючою крапкою можуть попередньо перетворюватися в рядок, перш ніж будуть оброблені Decimal, тим самим виконуваний код явно встановить кількість значущих цифр.
4. Створення дробового десяткового числа на основі кортежу (6, 2, 5, 1), 0 - додатне число, -3 - кількість знаків після десяткової крапки.
5. Створення дробового десяткового числа на основі кортежу (6, 2, 5, 1), 1 - від'ємне число, -1 - кількість знаків після десяткової крапки.

# Модуль decimal, функції Decimal(), getcontext()

Функція `Decimal()` дозволяє виконувати арифметичні операції.

```
from decimal import Decimal
print(Decimal('0.09') + Decimal('0.11'))
print(Decimal(0.09) + Decimal(0.11))
print(Decimal('0.09') - Decimal('0.11'))
```

Результати арифметичних дій:

```
0.20
0.19999999999999999972244424384
-0.02
```

# Модуль decimal, функції Decimal(), getcontext()

Атрибут `prec` контексту керує **точністю обчислення**. На відміну від апаратної двійкової плаваючої крапки, модуль **`decimal`** має змінну точність обчислення (за замовчуванням до `28` місць).

```
from decimal import Decimal, getcontext
d = Decimal('0.123456')
for i in range (1, 7):
    getcontext().prec = i
    print(i, ':', d, d * 1)
```

Точність обчислення атрибут `prec` підтримує **для нових значень, що створюються в результаті виконання математичних операцій**:

```
1 : 0.123456 0.1
2 : 0.123456 0.12
3 : 0.123456 0.123
4 : 0.123456 0.1235
5 : 0.123456 0.12346
6 : 0.123456 0.123456
```

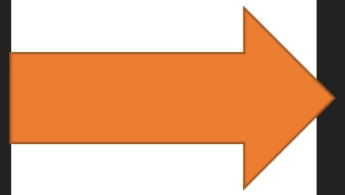
# Модуль decimal, функції Decimal(), getcontext()

У модулі decimal присутні опції, які керують округленням значень в межах заданої точності обчислень. Наведемо приклади деяких таких опцій:

- ROUND_DOWN. Округлення в напрямку нуля.
- ROUND_HALF_DOWN. Округлення в напрямку від нуля, якщо остання цифра рівна або більша 5, інакше - округлення в напрямку до нуля.
- ROUND_HALF_EVEN. Аналогічна ROUND_HALF_DOWN, за винятком того, що у випадках, коли остання цифра дорівнює 5, перевіряється передостання цифра. Якщо передостання цифра парна, округлення результату відбувається вниз, непарна - округлення вгору.
- ROUND_HALF_UP. Аналогічна ROUND_HALF_DOWN, за винятком того, що в тих випадках, коли остання цифра дорівнює 5, значення округлюється у напрямку від нуля.
- ROUND_UP. Округлення в напрямку від нуля.

# Модуль decimal, функції Decimal(), getcontext()

```
import decimal
from decimal import Decimal, getcontext

getcontext().prec = 5
getcontext().rounding = decimal.ROUND_DOWN
print(Decimal('0.52631558947368') * Decimal('1'))

getcontext().prec = 5
getcontext().rounding = decimal.ROUND_HALF_DOWN
print(Decimal('0.52631558947368') * Decimal('1'))

getcontext().prec = 6
getcontext().rounding = decimal.ROUND_HALF_EVEN
print(Decimal('0.52631558947368') * Decimal('1'))

getcontext().prec = 7
getcontext().rounding = decimal.ROUND_HALF_UP
print(Decimal('0.52631558947368') * Decimal('1'))

getcontext().prec = 8
getcontext().rounding = decimal.ROUND_UP
print(Decimal('0.52631558947368') * Decimal('1'))
```



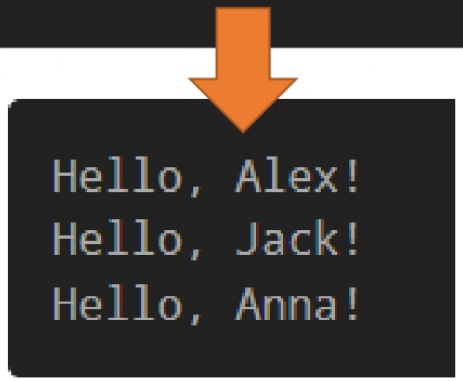
```
0.52631
0.52632
0.526316
0.5263156
0.52631559
```

# Модуль __name__

```
...
if __name__ == '__main__':
    ... # блок коду
```

Давайте розберемось як ця інструкція працює. Для прикладу, визначимо функцію у файлі `users.py`:

```
def greet_users(names):
    """Виведення привітання для кожного користувача у списку."""
    for name in names:
        message = "Hello, " + name.title() + "!"
        print(message)
usernames = ['alex', 'jack', 'anna']
greet_users(usernames)
```



# Модуль __name__

Файл `users.py` вважатимемо **модулем**, який містить єдину функцію. Створимо ще один файл `main_file.py` і помістимо у нього такий код:

```
from users import greet_users (1)
print('This code is executed!') (2)
if __name__ == '__main__': (3)
    print('This code is executed because the main_file.py is not being imported!')
```

1. У файл main_file.py виконується імпорт функції greet_users() із модуля-файла users.py.
2. Цей рядок коду виконується як при запуску файла main_file.py, так і у випадку, коли файл main_file.py імпортуємо у інший файл another_file.py (у даному випадку файл main_file.py стає модулем) таким чином:
3. Умова **if** і відповідний блок коду виконуються лише у випадку, коли ми запукаємо на виконання сам файл **main_file.py**

# Модуль __name__

Результат

```
C:\Python36>python users.py 1
Hello, Alex!
Hello, Jack!
Hello, Anna!

C:\Python36>python main_file.py 2
Hello, Alex!
Hello, Jack!
Hello, Anna!
This code is executed!
This code is executed because the main_file.py is not being imported!

C:\Python36>python another_file.py 3
Hello, Alex!
Hello, Jack!
Hello, Anna!
This code is executed!
```

1. Запуск файла users.py із функцією, яка повертає результи своєї роботи.
2. Запуск файла main_file.py, у який імпортували функцію з файла users.py. Конструкція if __name__ == '__main__': повертає True і блок коду виконується.
3. Запуск файла another_file.py, у який імпортували код файла main_file.py. Конструкція if __name__ == '__main__': повертає False і блок коду не виконується.
4. Python визначає змінну рівня модуля __name__ для будь-якого імпортованого модуля або будь-якого виконуваного файла.

# Модуль __name__

Використання змінної __name__ можна продемонструвати на простому прикладі. Створіть файл some_module.py з наступним кодом

```
print('The __name__ is: {0}'.format(__name__))
```

Коли файл імпортується як модуль:

```
>>> import some_module
The __name__ is: some_module
```

значення змінної `__name__` буде `some_module` і дорівнюватиме імені файла. Тобто, якщо перевірка

```
...
if __name__ == '__main__':
    ... # блок коду
```

# Модуль __name__

```
C:\Python36>python some_module.py
The __name__ is: __main__
```

змінна `__name__` матиме значення `__main__`, перевірка умови

```
...
if __name__ == '__main__':
    ... # блок коду
```

Ідіома Python (Python Trick) - короткий фрагмент початкового коду на Python, який використовується як інструмент навчання. Ідіома Python навчає окремій властивості мови Python шляхом простої ілюстрації або служить в якості мотивуючого прикладу, який дає можливість копнути глибше і розвинути інтуїтивне розуміння.

# Аргументи командного рядка

Створимо файл `test.py`, який міститиме наступні рядки:

```
import sys
print('Program arguments:', sys.argv)
```

Тепер, запустимо цей файл у термінальному вікні, наприклад, `Windows`, перед тим перейшовши у каталог, де файл був збережений:

```
C:\Python36>python test.py
Program arguments: ['test.py']

C:\Python36>python test.py one two three
Program arguments: ['test.py', 'one', 'two', 'three']
```

Змінна argv з модуля **sys** містить список аргументів командного рядка. Значення **argv[0]** - ім'я файла, який запускається (або повний шлях до нього). argv[1], argv[2] - це інші аргументи командного рядка..

# Пакети

- *Пакет - це каталог, який містить файл __init__.py, файли модулів та інші підпакети.*
- *Каталог пакета має обов'язково містити файл __init__.py, тому що так він може бути розпізнаний як пакет. Це убезпечує випадковове імпортування каталогів, що містять посторонній код Python, в якості пакетів.*

# Створення власних пакетів

Створимо каталог boxes, а у ньому каталог sources (він і буде пакетом), який буде містити два модуля: файли daily.py і weekly.py. Кожний з них міститиме функцію forecast(). Версія файла погоди на кожний день повертатиме рядок, а версія файла погоди на кожен тиждень повертатиме список із 7 рядків.

```
from sources import daily, weekly
print("Daily forecast:", daily.forecast())
print("Weekly forecast:")
for number, outlook in enumerate(weekly.forecast(), 1):
    print(number, outlook)
```

У прикладі вище, функція enumerate() розбиває список на частини і відправляє кожний елемент списку в цикл for, додаючи до кожного елемента число - порядковий номер, починаючи з 1.

**Модуль 1** буде знаходитися у файлі по такому шляху: `boxes\sources\daily.py` :

```
def forecast():
    '''Fake daily forecast'''
    return 'like yesterday'
```

**Модуль 2** буде знаходитися у файлі по такому шляху: `boxes\sources\weekly.py` :

```
def forecast():
    """Fake weekly forecast"""
    return ['mist', 'strong winds', 'sleet', 'freezing rain', 'rain', 'fog', 'drizzle']
```

# Створення власних пакетів

Виконання основної програми `weather.py` дасть такий результат:

```
Daily forecast: like yesterday
Weekly forecast:
1 mist
2 strong winds
3 sleet
4 freezing rain
5 rain
6 fog
7 drizzle
```

# Менеджер пакетів pip

***Пакет pip*** *- це система управління пакетами і найпопулярніший спосіб встановити сторонні пакети Python.*

Починаючи з версії мови Python 3.4, pip є стандартною частиною Python.

```
pip install --upgrade pip
```

Для перегляду списку установлених пакетів використовують команду:

```
pip list
```

Для установки пакета використовують команду:

```
pip install назва_пакета
```

Для видалення пакета використовують команду:

```
pip uninstall назва_пакета
```

Для оновлення пакета використовують команду:

```
pip install --upgrade назва_пакета
```

```
pip help
```

# Віртуальні середовища

Рішення цієї проблеми полягає у створенні віртуального середовища, автономного дерева каталогів, що містить установку Python для певної версії Python, а також ряд додаткових пакетів.
Розглянемо створення і використання віртуального середовища.

1. Створення віртуального середовища.

Модуль, який використовується для створення та управління віртуальними середовищами, називається venv. venv, зазвичай, встановлює найновішу версію Python у віртуальне середовище, яка у вас зараз встановлена.

```
python -m venv tutorial-env
```

Дана команда створить каталог **`tutorial-env`**, якщо він не існує, а також створить каталоги всередині нього, що містять копію інтерпретатора `Python`, **`pip`**, стандартну бібліотеку та різні допоміжні файли.

# Віртуальні середовища

2. Після створення віртуального середовища його треба активувати.

У системі `Windows` виконайте:

```
tutorial-env\Scripts\activate.bat
```

У `Linux Ubuntu` виконайте:

```
source tutorial-env/bin/activate
```

# Віртуальні середовища

```
C:\Users\user>python -m venv tutorial-env 1
C:\Users\user>tutorial-env\Scripts\activate.bat 2
(tutorial-env) C:\Users\user>python -V 3
Python 3.6.5
(tutorial-env) C:\Users\user>python 4
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> name = 'tutorial-env'
>>> print('I am a virtual environment', name)
I am a virtual environment tutorial-env
>>>exit() 5
(tutorial-env) C:\Users\user>python -m pip install --upgrade pip 6
...
Successfully installed pip-19.1.1
(tutorial-env) C:\Users\user>pip install requests==2.19.0 7
...
Successfully installed certifi-2019.6.16 chardet-3.0.4 idna-2.7 requests-2.19.0 urllib3-1.23
(tutorial-env) C:\Users\user>deactivate 8
C:\Users\user>
```

1. Створення віртуального середовища tutorial-env.
2. Активація віртуального середовища tutorial-env.
3. Отримання інформації про версію інтерпретатора Python у віртуальному середовищі.
4. Запуск інтерактивного режиму інтерпретатора Python у віртуальному середовищі і виконання деяких команд.
5. Вихід з інтерактивного режиму інтерпретатора Python у віртуальному середовищі.
6. Оновлення пакета pip до останньої версії у віртуальному середовищі (не впливає на пакети pip системної копії Python та інших віртуальних середовищ, якщо такі є).
7. Встановлення пакета requests конкретної версії 2.19.0 лише у віртуальне середовище tutorial-env.
8. Вихід з віртуального середовища tutorial-env за допомогою команди deactivate.

# Приклад

1. Створіть файл із назвою "car.py". У цьому файлі визначте функцію "engine_status()", яка виводить на екран повідомлення "Engine is running". Потім, використовуючи інтерактивний інтерпретатор, імпортуйте модуль "car.py" під псевдонімом "cr" і викличте функцію "engine_status().

```
# Зміст файлу car.py

class Car:
    def __init__(self, brand, year, engine_volume):
        self.brand = brand
        self.year = year
        self.engine_volume = engine_volume

    def get_info(self):
        return f"{self.year} {self.brand} with {self.engine_volume}L
```

```
# Зміст файлу car.py

def engine_status():
    print("Engine is running")
```

```
# Введіть ці команди в інтерактивному інтерпретаторі

from car import Car

# Створіть об'єкт автомобіля
my_car = Car(brand="Toyota", year=2022, engine_volume=2.0)

# Отримайте інформацію про автомобіль та виведіть її
print(my_car.get_info())
```

```
# Введіть ці команди в інтерактивному інтерпретаторі

import car as cr

# Викличте функцію engine_status()
cr.engine_status()
```

# ЗАВДАННЯ

Створіть файл, який називається city.py. У ньому оголосіть функцію library_services(), яка виводить на екран рядок Internet access is open 24 hours. Використайте інтерактивний інтерпретатор, щоб імпортувати модуль city.py і викликати його функцію library_services().

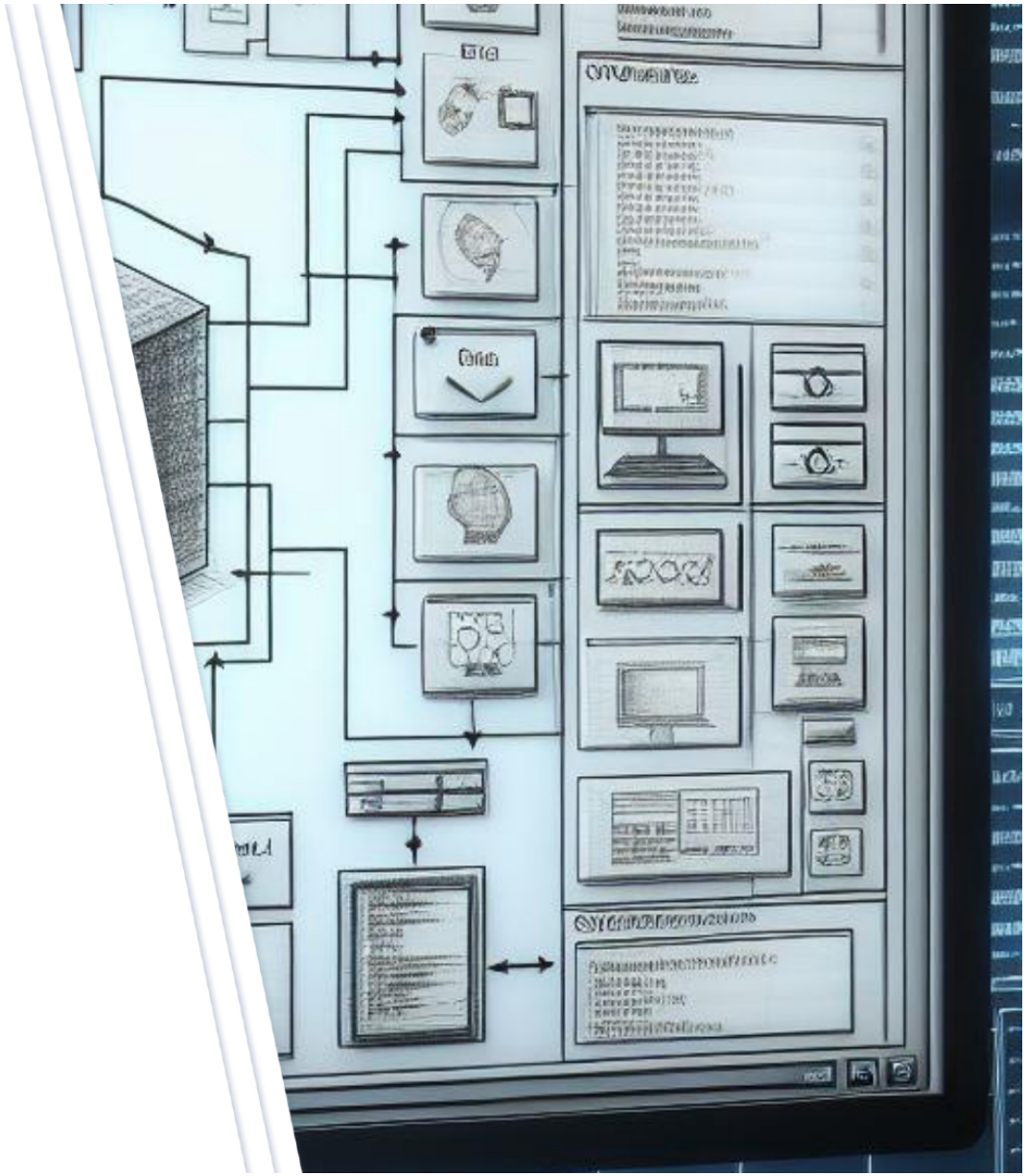
---

У інтерактивному інтерпретаторі імпортуйте модуль city.py з ім’ям ct і викличте його функцію library_services().

---

Залишаючись у інтерпретаторі, імпортуйте безпосередньо функцію library_services() з модуля city.py і викличте її.

# Винятки



# Винятки

- Винятки (або виключення) в програмуванні на Python є способом обробки помилок або несподіваних ситуацій в коді. Вони дозволяють вам визначити, як програма повинна реагувати на певні умови, якщо вони виникають. Винятки в Python використовуються за допомогою конструкцій **try, except і finally.**
- Основна ідея використання винятків полягає в тому, що ви обрамляєте блок коду, який може призвести до помилки, в конструкцію try. Якщо помилка виникає, виконується блок коду в конструкції except. Можна також додатково використовувати else для виконання коду, якщо помилок не виникає, і finally для виконання коду навіть у випадку помилки.

# Винятки

```python
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("Error: Division by zero is not allowed.")
    else:
        print(f"The result of {x} divided by {y} is: {result}")
    finally:
        print("This code will always be executed.")

# Приклад виклику функції
divide(10, 2)  # Успішна операція
divide(5, 0)   # Ділення на нуль, виняток обробляється
```

У цьому прикладі функція divide виконує ділення двох чисел, але вона обрамлена конструкцією try. Якщо ділення викликає помилку ZeroDivisionError, обробка помилок відбувається в блоку except, і виводиться повідомлення про те, що ділення на нуль не допускається. У будь-якому випадку, блок коду в finally виконується навіть у випадку винятку.

# Класи винятків

```python
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("Error: Division by zero is not allowed.")
    else:
        print(f"The result of {x} divided by {y} is: {result}")

divide(10, 2)  # Успішна операція
divide(5, 0)    # Виняток: Division by zero is not allowed.
```

**ZeroDivisionError** – виникає, коли ви спробуєте поділити на нуль

# Класи винятків

```python
def add_numbers(x, y):
    try:
        result = x + y
    except TypeError:
        print("Error: Addition is only allowed for numeric types.")
    else:
        print(f"The sum of {x} and {y} is: {result}")

add_numbers(5, 3)    # Успішна операція
add_numbers("5", 3)  # Виняток: Addition is only allowed for numeric
```

**TypeError –** виникає, коли операція виконується з об'єктами несумісного типу

# Класи винятків

```python
def read_file(filename):
    try:
        with open(filename, 'r') as file:
            content = file.read()
    except FileNotFoundError:
        print(f"Error: File '{filename}' not found.")
    else:
        print("File content:", content)

read_file("existing_file.txt")  # Успішне читання файлу
read_file("nonexistent_file.txt")  # Виняток: File 'nonexistent_file.
```

**FileNotFoundError – виникає, коли намагаєтеся відкрити файл, який не існує.**

```python
def get_positive_number():
    try:
        number = int(input("Enter a positive number: "))
        if number <= 0:
            raise ValueError("Error: Please enter a positive number.")
    except ValueError as e:
        print(e)
    else:
        print("You entered a positive number:", number)

get_positive_number()
```

**ValueError** - виникає, коли функція отримує аргумент правильного типу, але з недопустимим значенням.

# Обробка помилок

- Для управління помилками, що виникають у ході виконання програми, в Python використовуються спеціальні об’єкти, які називаються винятками.
- Якщо при виникненні помилки Python не знає, що робити далі, створюється **об’єкт винятку**. Якщо у програму включений код обробки виняткової ситуації, то виконання програми продовжиться, а якщо немає - програма зупиняється і виводить **трасування** (інформацію про хід виконання програми) зі звітом про помилку.

# Блок try-except

```
short_list = [1, 2, 3]
position = 5
print(short_list[position])
```

Виникає помилка виходу за межі діапазону списку:

```
Traceback (most recent call last):
  File "C:\Python34\test.py", line 246, in <module>
    print(short_list[position])
IndexError: list index out of range
```

Розмістимо свій код у блоці `try` і використаємо блок `except`, щоб обробити помилку:

```
short_list = [1, 2, 3]
position = 5
try:
    short_list[position]
except:
    print('Need a position between 0 and', len(short_list)-1, 'but got', position)
```

```
Need a position between 0 and 2 but got 5
```

# Блок try-except

```
try:
    блок коду
except тип_винятку as назва_змінної:
    блок коду, коли виникла помилка
```

```python
predators = ['tiger', 'wolf', 'bear']
while True:
    value = input('Position [q to quit]? ')
    if value == 'q':
        break
    try:
        position = int(value)
        print(predators[position])
    except IndexError as err:
        print('Bad index:', position)
    except Exception as other:
        print('Something else broke:', other)
```

```
Position [q to quit]? 1
wolf
Position [q to quit]? 0
tiger
Position [q to quit]? 2
bear
Position [q to quit]? 3
Bad index: 3
Position [q to quit]? 2
bear
Position [q to quit]? two
Something else broke: invalid literal for int() with base 10: 'two'
Position [q to quit]? q
```

# Блок try-except

```
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("Division by zero!")
    else:
        print("Result is", result)
    finally:
        print("End of the calculation.")
divide(2, 1)
divide(2, 0)
```

отримаємо наступні результати:

```
Result is 2.0
End of the calculation.
Division by zero!
End of the calculation.
```

# Перелік посилань


1. Пол Беррі. Head First, Python – Фабула, Ганна Якубовська, 2-ге вид., 2021. – 624с. ISBN: 978-617-522-019-1.
2. Олексій Васильєв. Програмування мовою Python – Навчальна книга – Богдан, 2019 – 504с. ISBN: 978-966-10-5611-3.
3. Luciano Ramalho. Fluent Python, 2nd ed. Published by O'Reilly Media, Inc., April 2022. – 1011p. ISBN-10 : 1491946008, ISBN-13 : 978-1491946008.
4. Reuven M. Lerner, Python Workout: 50 ten-minute exercises, 1st ed. – Manning Publications., Juli 2020. – 248p. ISBN-10 : 1617295507, ISBN-13 : 978-1617295508.
5. Michael Inden. Python Challenges: 100 Proven Programming Tasks Designed to Prepare You for Anything, 1st ed. – Apress, April 2022. – 691p. ASIN : B09YLFGDX1.
6. Eric Matthes. Python Crash Course : A Hands-On, Project-Based Introduction to Programming, 2nd ed. No Starch Press, May 2019. – 544p. ISBN-10 : 1593279280, ISBN-13 : 978-1593279288.
7. Eric Matthes. Python Crash Course : A Hands-On, Project-Based Introduction to Programming, 3nd ed. No Starch Press, May 2023. – 552p. ISBN-10 : 1718502702, ISBN-13 : 978-1718502703.
8. Patrick Viafore. Robust Python: Write Clean and Maintainable Code 1st ed. Published by O'Reilly Media, Inc., August 2021. – 378p. ISBN-10 : 1098100662, ISBN-13 : 978-1098100667.
9. Bill Lubanovic. Introducing Python: Modern Computing in Simple Packages 2nd ed. – Published by O'Reilly Media, Inc., December 2019. – 627p. ISBN-10 : 1492051365, ISBN-13 : 978-1492051367.

Навчальне видання

**Каштан** Віта Юріївна
**Гнатушенко** Володимир Володимирович
**Обиденний** Євген Олександрович
**Сущевський** Дмитро Валерійович

**ПРОГРАМУВАННЯ КОМП'ЮТЕРНИХ СИСТЕМ МОВОЮ PYTHON. ЧАСТИНА 1**
Навчальний наочний посібник

Видано в редакції авторів
Підписано до видання …
Електронний ресурс 6,2 авт., арк.

Підготовлено до видання
в Національному технічному університеті «Дніпровська політехніка».
Свідоцтво про внесення до Державного реєстру ДК № 1842 від 11.06.2004
49005, м. Дніпро, просп. Д. Яворницького, 19