Конспект лекції (2)

Ця лекція по суті — короткий огляд мови Python. Ми проходимося по верхах, а на подальших лекціях занурюємся у всі ці моменти більш ретельно.

Структура мови

▼ Логічна лінія та фізична лінія

Фізичний рядок — це послідовність символів, що завершується символом “кінець рядка” (CR LF).

Логічна лінія складається з однієї або кількох *фізичних ліній* за дотриманням явних або неявних правил *об’єднання ліній*.

https://docs.python.org/uk/3/reference/lexical_analysis.html

```
одинарний_рядок = 'Це одинарний рядок.'
подвійний_рядок = "Це подвійний рядок."
трьохкратний_рядок = '''Це
трьохкратний
рядок.''' # тут три фізичних лінії, проте лише одна логічна лінія
```

▼ Ключові слова

Наступні ідентифікатори використовуються як зарезервовані слова або *ключові слова* мови, і їх не можна використовувати як звичайні ідентифікатори. Вони мають бути написані точно так, як тут написано:

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

▼ Оператори дій

Для виконання дій з математичними значеннями або змінними.

```
+       -       *       **      /       //      %      @
<<      >>      &       |       ^       ~       :=
```

```
<       >       <=      >=      ==      !=
```

Арифметичні оператори:

Додавання (+): Сумує два операнда.

```
a = 5
b = 3
result = a + b  # Результат: 8
```

Віднімання (-): Віднімає один операнд від іншого.

```
a = 5
b = 3
result = a - b  # Результат: 2
```

Множення (*): Перемножує два операнда.

```
a = 5
b = 3
result = a * b  # Результат: 15
```

Ділення (/): Ділить один операнд на інший (завжди повертає число з плаваючою точкою).

```
a = 5
b = 2
result = a / b  # Результат: 2.5
```

Цілочисельне ділення (//): Ділить один операнд на інший і повертає цілу частку.

```
a = 5
b = 2
result = a // b  # Результат: 2
```

Залишок від ділення (%): Повертає залишок від ділення одного операнда на інший.

```
a = 5
b = 2
result = a % b  # Результат: 1
```

Піднесення до ступеня ():** Підносить один операнд до ступеня іншого.

```
a = 2
b = 3
result = a ** b  # Результат: 8
```

Логічні оператори:

Логічне І (and): Повертає True, якщо обидва операнди True.

```
a = True
b = False
result = a and b  # Результат: False
```

Логічне АБО (or): Повертає True, якщо хоча б один із операндів True.

```
a = True
b = False
result = a or b  # Результат: True
```

Логічне НЕ (not): Повертає True, якщо операнд False і навпаки.

```
a = True
result = not a  # Результат: False
```

▼ Роздільники

Служать для відокремлення значень

```
(       )       [       ]       {       }
,       :       .       ;       @
```

Круглі дужки `( )`:

Використовуються для обгортання виразів у виразах, оголошення функцій, передачі аргументів.

```
результат = (2 + 3) * 4
функція_з_аргументами(аргумент1, аргумент2)
```

Квадратні дужки `[ ]`:

Використовуються для створення списків в Python.

```
список = [1, 2, 3, 4]
елемент = список[0]
```

Фігурні дужки `{ }`:

Використовуються для створення словників (dict) та множин (set).

```
словник = {'ключ': 'значення'}
множина = {1, 2, 3}
```

Кома `,`:

Використовується для розділення елементів в структурах даних, аргументів функцій.

```
кортеж = (1, 2, 3)
функція_з_багатьма_аргументами(аргумент1, аргумент2, аргумент3)
```

Двокрапка `:`:

Використовується для позначення початку блоку коду (в умовних і циклічних інструкціях, функціях та ін.).

```
if умова:
    # Блок коду
    дії()

функція():
    # Блок коду
```

Крапка `.`:

Використовується для доступу до атрибутів об'єктів та методів.

```
рядок = "Привіт"
довжина = рядок.__len__()
```

Крапка з комою `;`:

Використовується для розділення інструкцій у тому випадку, коли ми хочемо помістити кілька інструкцій в один рядок.

```
рядок1 = "Привіт"; рядок2 = "Світ"
```

Символ собачки `@`:

Використовується в анотаціях та інших контекстах, наприклад, при визначенні декораторів.

```
@декоратор
def функція():
    # Блок коду
```

Це лише перший огляд деяких роздільників та символів у Python

▼ Спеціальні токени

Символи що служать для обрамлення строкових значень, коментарів та переносів лінії

' " # \

Одинарні лапки `'`:

- Використовуються для обгортання строкових значень.

```
рядок = 'Це строкове значення.'
```

Подвійні лапки `"`:

- Також використовуються для обгортання строкових значень. Обидва типи лапок можуть використовуватися взаємозамінно.

```
рядок = "Це також строкове значення."
```

Хеш-символ `#`:

- Використовується для вставки коментарів у код. Все, що слідує за `#`, у лінії, розглядається як коментар.

```
# Це коментар у Python
```

Екранування `\`:

- Використовується для екранування спеціальних символів в рядках, наприклад, `\'`, `\"`, `\\`.

```
рядок = 'Це символ екранування: \n Новий рядок.'
```

Ці символи важливі для розробки Python-коду та надають можливість працювати з рядками та коментарями.

▼ Літерали

Літерал — це пряме позначення в програмі значення даних (числа, рядка або контейнера). Нижче наведено числові та рядкові літерали в Python.

```
42          # Integer literal
3.14        # Floating-point literal
1.0j        # Imaginary literal
'hello'     # String literal
"world"     # Another string literal
"""Good
night"""    # Triple-quoted string literal, spanning 2 lines
```

Типи даних

▼ Цілі числа

Тип даних цілих чисел у Python позначається як `int`.

- **Представлення:**
  - Цілі числа в Python можуть бути вказані у десятковій, восьмеричній (починається з `0o`), шістнадцятковій (починається з `0x`) та двійковій (починається з `0b`) системах числення.

```
ціле_число = 42
восьмеричне_число = 0o52
шістнадцяткове_число = 0x2A
двійкове_число = 0b101010
```

- **Обмеження:**

  Обмеження цілих чисел у Python залежить від архітектури вашої системи. У зазвичай 32-бітних системах це може бути від -2,147,483,648

до 2,147,483,647, а на 64-бітних системах це може бути від -9,223,372,036,854,775,808 до 9,223,372,036,854,775,807.

- **Величина чисел:**

  У Python 3 цілі числа не мають максимального обмеження в розмірі, так як вони можуть займати стільки байт, скільки потрібно для їх представлення.

```
1, 23, 3493                    # Decimal integer literals
0b010101, 0b110010, 0B01       # Binary integer literals
0x1, 0x17, 0xDA5, 0xda5, 0Xff  # Hexadecimal integer literals
```

▼ Числа з “плаваючою” комою

тип даних `float` використовується для представлення чисел з плаваючою точкою (чисел з десятковою комою)

Літерал із плаваючою комою — це послідовність десяткових цифр, яка включає десяткову крапку (.), суфікс експоненти (e або E, необов’язково, за якими слідують + або -, за якими слідують одна чи більше цифр), або обидва. Початковий символ літералу з плаваючою комою не може бути e або E; це може бути будь-яка цифра або крапка (.), за якою йде цифра.

0., 0.0, .0, 1., 1.0, 1e0, 1.e0, 1.0E0 *# Floating-point literals*

Числа з плаваючою точкою можна виразити як десяткові числа або використовувати наукову нотацію (наприклад, `1.23` або `2.0e-3`).

```
число_з_плаваючою_точкою = 3.14
число_з_науковою_нотацією = 2.0e-3
```

- **Точність:**

  Числа з плаваючою точкою мають обмежену точність через внутрішні обмеження стандарту для чисел з плаваючою точкою в комп'ютерах.

- **Арифметичні Операції:**

  З числами з плаваючою точкою можна виконувати арифметичні операції, так само як із цілими числами.
  УВАГА! результат ділення - **завжди** `float`

```
число1 = 3.14
число2 = 2.0
```

```
сума = число1 + число2
```

- **Обмеження:**

  Числа з плаваючою точкою обмежені можливостями представлення чисел у форматі з плаваючою точкою, тому існує обмеження щодо точності та розміру чисел.

Похибки Порівнянь:

У роботі з числами з плаваючою точкою слід уникати порівнянь на рівність через можливість невеликих похибок через обмежену точність.

```
a = 0.1 + 0.2
b = 0.3
результат = a == b  # Може бути False через невелику похибку представлення.
```

▼ Послідовності (Sequences)

Послідовність — це впорядкований контейнер елементів, проіндексованих цілими числами або хешами. Python має вбудовані типи послідовностей, відомі як рядки (байти або str), кортежі, списки, набори та словники

▼ “Строки”

Тип даних позначається як `str`

відокремлюються одиночною (') або подвійною лапкою (")

Символи можна екранувати через backslash

```
'I\'m a Python fanatic'     # You can escape a quote
"I'm a Python fanatic"      # This way may be more readable
'A not very long string \
that spans two lines'       # Comment not allowed on previous line
# triple-quoted string literal
"""An even bigger
string that spans
three lines*"""            # Comments not allowed on previous lines
```

Строки є незмінними, що означає, що після створення рядка ви не можете змінити його вміст без створення нового рядка (або перезапису вмісту змнної).

Рядки (строки) розглядаються як послідовності символів.

Це означає, що ви можете обробляти рядки, якщо це колекція окремих елементів (символів). Важливі риси рядків як послідовностей включають:

Індексація:

Символи в рядках можна отримати за допомогою індексації. Перший символ має індекс 0, другий - 1, і так далі

```
рядок = "Привіт"
перший_символ = рядок[0]  # Повертає "П"
другий_символ = рядок[1]  # Повертає "р"
```

Зрізи (Slices):

Ви можете отримувати підстрічки (зрізи) з рядка, вказуючи початковий та кінцевий індекси.

```
рядок = "Привіт"
підстрічка = рядок[1:4]  # Повертає "рив"
```

Довжина Рядка:

Ви можете використовувати функцію `len()` для отримання довжини рядка (кількість символів).

```
рядок = "Привіт"
довжина = len(рядок)  # Повертає 6
```

Ітерація:

Рядки можна ітерувати за допомогою циклу `for`, отримуючи кожен символ послідовно.

```
рядок = "Привіт"
for символ in рядок:
    print(символ)
```

▼ Кортежі (Tuples)

Тип даних позначається як `tuple`

ітерована незмінна послідовність даних розділена комами

`(0, “абв”, 3747)`

використовується для збереження групи елементів. Основні характеристики кортежів включають:

Створення Кортежу:
Кортеж можна створити, перераховуючи елементи у круглих дужках. Кожен елемент розділяється комою.

```
кортеж = (1, 2, 3, 'Привіт')
```

Індексація та Зрізи:

Елементи кортежу можна отримати за допомогою індексації, і кортежі також підтримують зрізи.

```
кортеж = (1, 2, 3, 'Привіт')
перший_елемент = кортеж[0]  # Повертає 1
підкортеж = кортеж[1:3]  # Повертає (2, 3)
```

Незмінність:

Кортежі є незмінними (immutable), що означає, що після того, як кортеж створено, ви не можете змінити його елементи. Але ви можете створити новий кортеж.

```
кортеж = (1, 2, 3)
# Це призведе до помилки, так як кортежі неізменні:
кортеж[0] = 10
```

▼ Списки [Lists]

Тип даних позначається як `list`

Список (list) у Python є одним з основних вбудованих типів даних і використовується для зберігання колекції елементів у впорядкованій послідовності. Основні характеристики списків включають:

- **Створення Списку:**

  Список можна створити, перераховуючи його елементи у квадратних дужках. Елементи розділяються комою.

  ```
  список = [1, 2, 3, 'Привіт']
  ```

- **Індексація та Зрізи:**

  Елементи списку можна отримати за допомогою індексації, і списки підтримують зрізи.

```
список = [1, 2, 3, 'Привіт']
перший_елемент = список[0]  # Повертає 1
підсписок = список[1:3]  # Повертає [2, 3]
```

- **Довжина Списку:**

  Ви можете використовувати функцію `len()` для отримання кількості елементів у списку

```
список = [1, 2, 3, 'Привіт']
довжина = len(список)  # Повертає 4
```

Списки є змінними (mutable), тобто ви можете змінювати їх елементи, додавати та видаляти елементи.

▼ Набори {Sets}

Тип данних позначається `set`

НЕ впорядкований набір унікальних елементів, ітерований

Python множини (sets) представляють собою вбудований тип даних, що використовується для зберігання унікальних елементів без порядку. Основні характеристики множин включають:

Створення множини:

Множину можна створити, використовуючи фігурні дужки `{}`. Дублікати автоматично вилучаються.

```
множина = {1, 2, 3, 'Привіт'}
```

▼ Словники {dictionary_key: value}

Тип даних позначається як `dict`

НЕ впорядкований набір пар ключ: значення.

словник (dictionary) є вбудованим типом даних і використовується для зберігання колекції пар ключ-значення. Ключі в словнику можуть бути

різних типів, але вони повинні хешуватися. Значення в словнику є довільними об’єктами і можуть бути будь-якого типу

Основні характеристики словників включають:

Створення Словника:

Словник можна створити, використовуючи фігурні дужки `{}`, та вказуючи пари ключ-значення через двокрапку. Ключі повинні бути унікальними.

```
словник = {'ім_я': 'Василь', 'вік': 25, 'місто': 'Київ'}
```

Доступ до Елементів:

Елементи словника можна отримати за допомогою ключів.

```
name = словник['ім_я']
```

▼ None

`None` використовується для представлення відсутності значення або невизначеності.

Не має методів чи інших атрибутів.

Функції повертають None як свій результат, якщо вони не мають спеціальних операторів повернення, закодованих для повернення інших значень.

▼ Булеві значення

Їх лише два:

- True
- False

УВАГА! результат будь-якого порівняння та логічної операції `is` — булеве значення.

Змінні та об’єкти у Python

▼ Змінні

Змінна — це іменована область пам’яті. Вона може містити будь-який тип даних: числа, рядки, масиви, об’єкти. Змінні використовуються для збереження та отримання значень, які можуть змінюватися під час

виконання програми. Змінна складається з двох основних компонентів: **ім'я** та **значення**.

Ім'я — ідентифікатор, що дозвляє нам посилатися на дані, що зберігаються у пам’яті.

Значення змінної — дані різних типів, що зберігаються в ній.

Python не має строгої типізації, тобто ви можете змінювати тип значень у змінній, однак хорошим тоном програмування буде цього не робити.

У Python можна не вказувати наперед ім’я змінної, якимось чином ініціалізувати чи резервувати її. Достатньо просто задати ім’я = значення.

▼ Об’єкти у Python

У Python все є об’єктом, але поки що достаньо знати що об’єкт може мати:

Атрибути об’єкта

x.y

Елементи об’єкта

x[y]

Методи - виклик операції з об’єктом

x.y()

Математичні операції

▼ Ділення

є аж три операції що до ділення

- математичне ділення 5/2
- ділення націло 5//2
- залишок від ділення 5%2

Єдина числова операція, що викликає виключення:

```
7/0
ZeroDivisionError: division by zero
```

▼ Множення, додавання, віднімання

Тут все просто

+ - *

▼ Піднесення у степінь

pow(a, b) == a ** b

▼ Порівняння

Строгі: equality (==) and inequality (!=)

Нестрогі: <, <=, >, ≥

Не можна порівнювати комплексні числа та (інколи) флоат

Результат порівняння завжди булеве значення (або помилка)

▼ Мінімум або максимум

Вбудовані функції що дозваляють отримати мінімальне \ максимальне значення.

x = min (1, 2, 3)

присвоїть 1 у змінну x

y = max (1, 2, 3)

присвоїть 3 в змінну y

Ввод/вивід даних

▼ Вивід

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

▼ f-string

f-стрічки (formatted string literals) введені у Python 3.6 та є зручним способом форматування рядків. Вони дозволяють вбудовувати значення змінних та виразів безпосередньо в рядок, забезпечуючи більш читабельний та компактний код.

```
year = 2016
event = 'Referendum'
print(f'Results of the {year} {event}') # Виведе: Results of the 2016 Referendum
```

▼ Format

Застарілий спосіб, щоб виводити дані в строках

```
yes_votes = 42_572_654
no_votes = 43_132_495
percentage = yes_votes / (yes_votes + no_votes)
out = '{:-9} YES votes  {:2.2%}'.format(yes_votes, percentage)
print(out)  # 42572654 YES votes  49.67%
```

▼ Ввод

Для присвоєння змінній користувацького вводу використовується `input` :

```
query_sting = "підказка, що буде виведена на екран і повинна пояснити,\
  що ми очікуємо від користувача: "
variable = input(query_sting)
```

input **завжди** повертає дані у форматі строки (str)

Також для вводу ми можемо читати дані з файлу або бази даних, розміщених локально чи в інтернет. Але про це ми поговоримо у подальшому курсі.

Домашне завдання