

# Python: функції та змінні

Прикладна аналітика при розробці ІТ

Ігор Мірошниченко

КНУ імені Тараса Шевченка, ФІТ

Прикладна аналітика при розробці ІТ

# ЗМІСТ

- Про мене
- DataCamp Group
- Мотивація
- Функції
- Змінні
- Рядки
- Числа
- Власні функції

# ПРО МЕНЕ

- Мірошниченко Ігор Вікторович
- кандидат економічних наук, доцент
- доцент кафедри технологій управління, ФІТ, КНУ імені Тараса Шевченка
- доцент кафедри математичного моделювання та статистики, ІІТЕ, КНЕУ
- викладач Міжнародного інституту бізнесу (МВА)

✉ [ihor.miroshnychenko@kneu.ua](mailto:ihor.miroshnychenko@kneu.ua)

☞ Data Mirosh

☞ [@ihormiroshnychenko](https://www.linkedin.com/in/ihormiroshnychenko)

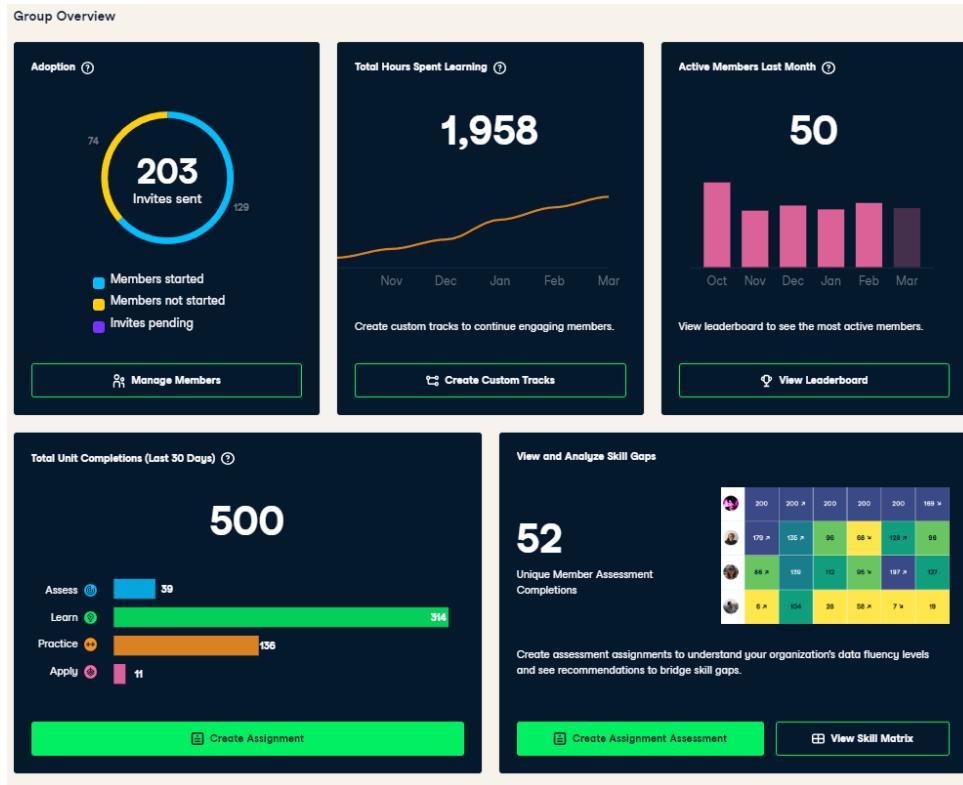
☞ [@aranaur](https://www.twitter.com/aranaur)

☞ [aranaur.rbind.io](https://aranaur.rbind.io)

# DATA CAMP GROUP



# DATA CAMP GROUP



**Career Tracks**

Our career tracks are hand-picked by industry experts. You will learn all you need to start a new career in the data science field.

**All**    Power BI    Python    R    SQL    Tableau

② Which technology should I choose?

**R Programmer**

PREPARE FOR CERTIFICATION

48 hours 12 courses

**Data Scientist with R**

88 hours 22 courses

**Data Analyst with R**

36 hours 9 courses

**Python Programmer**

67 hours 17 courses

**Data Scientist with Python**

PREPARE FOR CERTIFICATION

36 hours 9 courses

**Data Analyst with Python**

Track Completed

# DATA CAMP GROUP



## Як долучитися?

1. Приєднатися до телеграм-каналу Data Mirosh
2. Зареєструйтесь на DataCamp
3. Приєднайтесь до класу за посиланням

### Примітка

Клас буде активний **з 11 жовтня 2023 року до 11 квітня 2023 року**, після чого буде відкрито наступний потік. Слідкуйте за оновленнями.

# МОТИВАЦІЯ

Прикладна аналітика при розробці ІТ

# ОСНОВНІ ГРАВЦІ

- **R / Python / Julia** - мови програмування
-  **SQL** - мова для роботи з базами даних
-  **Статистика** - наука про збір, обробку, аналіз та інтерпретацію даних

# ВСТАНОВЛЕННЯ РУТНОН

Windows

Linux

MacOS

1. Завантажити та запустити Python актуальної версії.
2. Відкрити командний рядок: [cmd](#)
3. Виконати у командному рядку: `pip install numpy`
4. Виконати у командному рядку: `pip install pandas`
5. Виконати у командному рядку: `pip install jupyter`
6. Запустити **jupyter**: виконати у командному рядку: `jupyter-notebook`



Порада

Рекомендую використовувати Chocolatey: <https://aranaur.rbind.io/blog/2023/01>

# ХМАРНІ СЕРВІСИ

1. Google Colab
2. Jupyter Notebook
3. Posit.cloud

# IDE

1. Visual Studio Code
2. PyCharm
3. RStudio

# ФУНКЦІЇ

Прикладна аналітика при розробці ІТ

# ФУНКЦІЇ

- **Функція** - це дія або дієслово, яке дозволяє вам робити щось у програмі.
- **Аргументи** - це вхідні дані для функції, які якимось чином впливають на її поведінку.

```
1 print('Привіт, світ!')
```

Привіт, світ!

У програмуванні така дія називається побічним ефектом (анг. **side effects**). Він може бути візуальним, звуковим, виконувати записи файл або базу даних тощо.

# БАГИ

**Баг** - це помилка у програмі. Вони можуть приймати найрізноманітніші форми і наша задача навчитися виправляти їх.

```
1 print('Привіт, світ!')
```

```
SyntaxError: incomplete input (4260497653.py, line 1)
```



# HARD CODING

```
1 input('Як тебе звати? ') # Гаррі  
2 print('Привіт, Гаррі')
```

Привіт, Гаррі

```
1 input('Як тебе звати? ') # Герміона  
2 print('Привіт, Гаррі')
```

Привіт, Гаррі



# ЗМІННІ

Прикладна аналітика при розробці ІТ



# ЗАГАЛЬНЕ ПРО ЗМІННІ

**Змінна** - це просто контейнер для якогось значення всередині комп'ютера або всередині вашої програми.

При виборі імені змінної давайте дотримуватись певних правил, щоб наш код виконувався без помилок та його було зручно читати:

- Ім'я змінної починається з літери;
- Для імен змінних використовуватимемо маленькі літери з підкресленням замість пробілу;
- Ім'я змінної не повинно співпадати з назвою ключових слів Python:

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

# ОПЕРАТОР ПРИСВОЄННЯ

Для створення змінних використовується оператор присвоєння `=`.

```
1 name = input('Як тебе звати? ') # Гаррі
2 print('Привіт, name')
```

Привіт, name

```
1 name = input('Як тебе звати? ') # Гаррі
2 print('Привіт,')
3 print(name)
```

Привіт,
Гаррі

# КОМЕНТАРІ ДО КОДУ

**Коментарі** - це примітки до вашого коду. Вони не виконуються і не впливають на роботу програми.

Для створення коментарів використовується **символ #**.

```

1 # Запитати користувача про ім'я
2 name = input('Як тебе звати? ') # Gappi
3
4 # Вивести привітання
5 print('Привіт, ')
6 print(name)

```

А що якщо необхідно написати коментар, який займе декілька рядків? В такому випадку можна використати спеціальну техніку використовуючи потрійні лапки (одинарні або подвійні):

```

1 """
2 Запитати користувача про ім'я
3 і вивести привітання
4 """
5 name = input('Як тебе звати? ') # Gappi
6 print('Привіт, ')
7 print(name)

```

# ПСЕВДОКОД

**Псевдокод** - це неформальна форма запису. Це просто використання природної мови, щоб висловити свої думки лаконічно, методично, алгоритмічно тощо.

```
1 # Запитати користувача про ім'я  
2  
3 # Вивести привітання
```

Псевдокод - це гарний спосіб структурувати список справ, особливо якщо ви поки не знаєте, як писати код.

# ВИРІШЕННЯ РІЗНИМИ СПОСОБАМИ

Давайте у середині функції `print()` “додамо” змінну `name`.

```

1 # Запитати користувача про ім'я
2 name = input('Як тебе звати? ') # Гаррі
3
4 # Вивести привітання
5 print('Привіт, ' + name)

```

Привіт, Гаррі

Позбавимось естетичної помилки, додавши пробіл між словами:

```

1 # Запитати користувача про ім'я
2 name = input('Як тебе звати? ') # Гаррі
3
4 # Вивести привітання
5 print('Привіт, ' + name) # додали пробіл після коми

```

Привіт, Гаррі

Передамо декілька агрументів:

```

1 # Запитати користувача про ім'я
2 name = input('Як тебе звати? ') # Гаррі
3
4 # Вивести привітання
5 print('Привіт,', name)

```

Привіт, Гаррі



# РЯДКИ

Прикладна аналітика при розробці ІТ



# ДОКУМЕНТАЦІЯ

Весь цей час ми працювали з рядками - `str`, послідовність тексту.

Давайте повернемося до одного з попередніх варіантів написання програми з подвійним використанням функції `print()`:

```

1 # Запитати користувача про ім'я
2 name = input('Як тебе звати? ') # Гаррі
3
4 # Вивести привітання
5 print('Привіт, ')
6 print(name)

```

Привіт,  
Гаррі

Чи можна вирішити “проблему” іншим способом? Так:

`print(*objects, sep=' ', end='\n', file=None, flush=False)`

## Примітка

Документація до Python доступна на [docs.python.org](https://docs.python.org).

# ДОКУМЕНТАЦІЯ

## Варіант 1:

```
1 # Запитати користувача про ім'я
2 name = input('Як тебе звати? ') # Гаррі
3
4 # Вивести привітання
5 print('Привіт, ', end='')
6 print(name)
```

Привіт, Гаррі

## Варіант 2:

```
1 # Запитати користувача про ім'я
2 name = input('Як тебе звати? ') # Гаррі
3
4 # Вивести привітання
5 print('Привіт', name, sep=', ')
```

Привіт, Гаррі



# F-РЯДКИ

**f-рядки** - це рядки, які містять обчислювальні вирази.

Щоб створити f-рядок, ми використовуємо літеру **f** перед першими лапками або апострофом.

Вирази, які ми хочемо обчислити, ми пишемо в фігурних дужках **{}**:

```
1 # Запитати користувача про ім'я
2 name = input('Як тебе звати? ') # Гаррі
3
4 # Вивести ім'я
5 print(f'Привіт, {name}')
```

Привіт, Гаррі



# F-РЯДКИ

Раніше використовувались такі способи:

## 1. За допомогою оператора %:

```
1 name = "Гаррі"
2 age = 11
3 print("My name is %s and I am %d years old" % (name, age))
```

My name is Гаррі and I am 11 years old

## 2. За допомогою методу format:

```
1 name = "Гаррі"
2 age = 11
3 print("Моє ім'я {}. Мені {} років.".format(name, age))
```

Моє ім'я Гаррі. Мені 11 років.

```
1 name = "Гаррі"
2 age = 11
3 print("Моє ім'я {1}. Мені {0} років.".format(name, age))
```

Моє ім'я 11. Мені Гаррі років.

```
1 print("Моє ім'я {name}. Мені {age} років.".format(name='Гаррі', age=11))
```

Моє ім'я Гаррі. Мені 11 років.



# МЕТОДИ РЯДКІВ

Методи - це функції, які виконуються на об'єкті.

```

1 # Запитати користувача про ім'я
2 name = input('Як тебе звати? ') # ' гаррі '
3
4 # Вивести ім'я
5 print(f'Привіт, {name}')

```

Привіт, гаррі

Як бачите, ім'я виводиться з пробілами на початку і в кінці. Якщо ми хочемо видалити пробіли з початку і кінця рядка, ми можемо використати метод `.strip()`:

```

1 # Запитати користувача про ім'я
2 name = input('Як тебе звати? ') # ' гаррі '
3
4 # Вивести ім'я
5 print(f'Привіт, {name.strip()}')

```

Привіт, гаррі

# МЕТОДИ РЯДКІВ

Вже краще, але що робити з ім'ям, яке введене з маленької літери? Ми можемо використати метод `.capitalize()`, щоб перетворити першу літеру рядка на велику:

```
1 # Запитати користувача про ім'я
2 name = input('Як тебе звати? ') # ' гаррі '
3
4 # Вивести ім'я
5 print(f'Привіт, {name.strip().capitalize()}')
```

Привіт, Гаррі

# МЕТОДИ РЯДКІВ

Тепер розглянемо приклад, коли користувач вводить не тільки ім'я, але і прізвище:

```

1 # Запитати користувача про ім'я
2 name = input('Як тебе звати? ') # ' гаррі поттер '
3
4 # Вивести ім'я
5 print(f'Привіт, {name.strip().capitalize()}')

```

Привіт, Гаррі поттер

Метод `.capitalize()` перетворює **першу літеру** рядка на велику. Але якщо нам необхідно записати і ім'я, і прізвище з великої літери? Ми можемо використати метод `.title()`:

```

1 # Запитати користувача про ім'я
2 name = input('Як тебе звати? ') # ' гаррі поттер '
3
4 # Вивести ім'я
5 print(f'Привіт, {name.strip().title()}')

```

Привіт, Гаррі Поттер

# МЕТОДИ РЯДКІВ

З іншої сторони, якщо ми хочемо вивести тільки ім'я, а користувач ввів ім'я та прізвище, ми можемо використати метод `.split()`. Цього разу я не буду додавати зайвих пробілів до імені:

```
1 # Запитати користувача про ім'я
2 name = input('Як тебе звати? ') # 'гаррі поттер'
3
4 # Розділити рядок на дві частини
5 first, last = name.split(" ")
6
7 # Вивести ім'я
8 print(f'Привіт, {first.capitalize()}')
```

Привіт, Гаррі

# СПЕЦІАЛЬНІ СИМВОЛИ В РЯДКАХ

Часто для форматування тексту необхідно використовувати **спеціальні символи** або **екрановані послідовності**:

Ось деякі з найбільш використовуваних:

- `\n` — перенесення рядка
- `\t` - табуляція
- `\'` - однаарна лапка
- `\"` - подвійна лапка
- `\\"` — зворотний слеш

# СПЕЦІАЛЬНІ СИМВОЛИ В РЯДКАХ

Розберемо їх використання відразу на прикладі:

```
1 string1 = '\'Грифіндор\' - чемпіон з квідичу!\n1996 - 1997'
2 print(string1)
```

'Грифіндор' - чемпіон з квідичу!  
1996 - 1997

```
1 print("Привіт\nсвіт!")
```

Привіт  
світ!

```
1 print("C:\\\\Users\\\\user\\\\Desktop\\\\file.txt")
```

C:\\Users\\user\\Desktop\\file.txt

```
1 print("Ім'я:\tГаррі")
```

Ім'я: Гаррі

```
1 print('Кам\'яні стіни, як і в «Грінготсі», освітлювали смолоскипи')
```

Кам'яні стіни, як і в «Грінготсі», освітлювали смолоскипи

```
1 print("Кам'яні стіни, як і в «Грінготсі», освітлювали смолоскипи")
```

Кам'яні стіни, як і в «Грінготсі», освітлювали смолоскипи

## Примітка

Документація до методів рядків у Python доступна за посиланням.

# РЯДКИ ТА ЇХ НЕЗМІНЮВАНІСТЬ

Рядки належать до незмінних об'єктів у Python. Які практичні наслідки виникають з того факту, що рядки, на відміну від списків, є незмінними об'єктами?

Щоб показати, що рядки є незмінюваними об'єктами, ми можемо використати такий приклад коду:

```
1 my_string = "abcde"
2 my_string[0] = "f"
```

`TypeError: 'str' object does not support item assignment`

Якщо ми хочемо створити новий рядок на основі існуючого, то ми повинні створити новий об'єкт рядка:

```
1 my_string = "abcde"
2 new_string = "f" + my_string[1:]
3 print(new_string)
```

`fbcde`

# ЧИСЛА

Прикладна аналітика при розробці ІТ

# ЦІЛІ ЧИСЛА

**Цілі числа**, `int` (англ. *integer*) - це окремий тип даних у Python.

Операції з числами:

- `+`: додавання
- `-`: віднімання
- `*`: множення
- `**`: зведення в ступінь
- `/`: ділення
- `//`: ціличислове ділення, арифметична операція, результатом якої є ціла частина частки, отриманого поділом одного цілого числа на інше ціле число
- `%`: залишок від ділення

# ЦІЛІ ЧИСЛА

Давайте подивимось, чи зможемо ми створити власний маленький калькулятор:

```

1 # Запитати користувача про перше число
2 first_number = input('Введіть перше число: ') # '1'
3
4 # Запитати користувача про друге число
5 second_number = input('Введіть друге число: ') # '2'
6
7 # Вивести результат додавання
8 print(f'Результат додавання: {first_number} + {second_number} = {first_number + second_number}')

```

Результат додавання: 1 + 2 = 12



Функція `input()` завжди повертає рядок. Щоб вирішити цю проблему, нам потрібно перетворити рядок у ціле число. Для цього ми можемо використати функцію `int()`:

```

1 # Запитати користувача про перше число
2 first_number = int(input('Введіть перше число: ')) # '1'
3
4 # Запитати користувача про друге число
5 second_number = int(input('Введіть друге число: ')) # '2'
6

```

# ЦІЛІ ЧИСЛА

В загалі то ми можемо написати весь наш калькулятор у одному рядку:

```
1 print(f'Результат додавання: {int(input("Введіть перше число: ")) + int(input("Введіть друге число: "))}'
```

Результат додавання: 3

Однак далеко не завжди перетворення типів проходить без помилок, наприклад, ми не зможемо зробити таке перетворення:

```
1 a = 'my number is 42'
2 b = int(a) # отримаємо помилку
```

ValueError: invalid literal for int() with base 10: 'my number is 42'

Це означає, що рядок '*my number is 42*' не може бути відразу представлена як ціле число. Ми могли б взяти з цього рядка лише число **42** і перевести його до цілого числа. Скоріше ми навчимося так робити.

## Примітка

Якщо ви хочете дізнатися більше про цілі числа, то можете прочитати документацію.

# ДРОБОВІ ЧИСЛА

Python також підтримує **числа з плаваючою комою** (англ. *floating point numbers*). Це числа, які мають дробову частину. Наприклад, [3.14](#) - число з плаваючою комою.

Щоб використати число з плаваючою комою, ми можемо використати [float\(\)](#):

```

1 # Запитати користувача про перше число
2 first_number = float(input('Введіть перше число: ')) # '1.2'
3
4 # Запитати користувача про друге число
5 second_number = float(input('Введіть друге число: ')) # '3.4'
6
7 # Вивести результат додавання
8 print(f'Результат додавання: {first_number} + {second_number} = {first_number + second_number}')

```

Результат додавання: 1.2 + 3.4 = 4.6

## Примітка

Якщо ви хочете дізнатися більше про числа з плаваючою комою, то можете прочитати [документацію](#).

# ОКРУГЛЕННЯ

Для округлення чисел з плаваючою комою ми можемо використати функцію `round()`:

`round(number, ndigits=None)`

- `number` - число, яке ми хочемо округлити
- `ndigits` - кількість знаків після коми, до якої ми хочемо округлити число. Якщо ми не вказуємо цей аргумент, то число буде округлено до найближчого цілого.

```

1 # Запитати користувача про перше число
2 first_number = float(input('Введіть перше число: ')) # '1.2'
3
4 # Запитати користувача про друге число
5 second_number = float(input('Введіть друге число: ')) # '3.4'
6
7 # Вивести результат додавання
8 print(f'Результат додавання: {first_number} + {second_number} = {round(first_number + second_number)}')

```

Результат додавання: 1.2 + 3.4 = 5

# ФОРМАТУВАННЯ ЧИСЕЛ

Давайте додамо 1 до 999:

```

1 # Запитати користувача про перше число
2 first_number = float(input('Введіть перше число: ')) # '1'
3
4 # Запитати користувача про друге число
5 second_number = float(input('Введіть друге число: ')) # '999'
6
7 # Вивести результат додавання
8 print(f'Результат додавання: {first_number} + {second_number} = {round(first_number + second_number)}')

```

Результат додавання: 1 + 999 = 1000

Все ок, але в різних частинах світу використовують різне форматування чисел для зручності читання:

```

1 # Запитати користувача про перше число
2 first_number = float(input('Введіть перше число: ')) # '1'
3
4 # Запитати користувача про друге число
5 second_number = float(input('Введіть друге число: ')) # '999'
6
7 # Вивести результат додавання
8 print(f'Результат додавання: {first_number} + {second_number} = {first_number + second_number:,}')

```

Результат додавання: 1 + 999 = 1,000



# ФОРМАТУВАННЯ ЧИСЕЛ

Для округлення, замість `round()` можна використати спеціальну нотацію `{:.2f}`:

```

1 # Запитати користувача про перше число
2 first_number = float(input('Введіть перше число: ')) # '2'
3
4 # Запитати користувача про друге число
5 second_number = float(input('Введіть друге число: ')) # '3'
6
7 # Вивести результат ділення
8 print(f'Результат ділення: {first_number} / {second_number} = {first_number / second_number:.2f}')

```

Результат ділення: 2 / 3 = 0.67

## Примітка

Іноді для розв'язання задачі нам потрібно створити змінну і зберегти в ній нескінченність у математичному сенсі слова. Для цього нам допоможе така форма запису:

```
1 x = float('inf')
```

`float('-inf')` - спеціальне значенням для чисел із плаваючою крапкою з негативною нескінченністю.

# ВЛАСНІ ФУНКЦІЇ

# КЛЮЧОВЕ СЛОВО `def`

Для створення власних функцій ми використовуємо ключове слово `def`:

```

1 def hello(name):
2     print('Привіт,', name)
3
4 name = input('Введіть ваше ім'я: ') # 'Гаррі'
5 hello(name)

```

Привіт, Гаррі

## Важливо

Зверніть увагу на **відступи** в нашому коді. Відступ дорівнює **4 пробілам** або одного **табулятора**. В Python відступи дуже важливі, тому що вони вказують на те, що код належить до певного блоку.

```

1 name = 'Гаррі'
2
3 def hello(name):
4     print('Привіт,', name)
5
6 hello(name)

```

IndentationError: expected an indented block after function definition on line 3 (2611194764.py, line 4)

# ЗНАЧЕННЯ ЗА ЗАМОВЧУВАННЯМ

Іноді нам потрібно задати значення **за замовчуванням** для аргументів функції.

Наприклад, якщо ми хочемо вивести привітання, але не знаємо імені користувача, то ми можемо використати значення за замовчуванням:

```
1 def hello(name='світ'):  
2     print('Привіт, ', name)  
3  
4 hello()
```

Привіт, світ

# ГОЛОВНА ФУНКЦІЯ `main()`

Ми можемо зануритися глибше і назвати свою функцію `main()`.

Це **не обов'язкова** вимога, але це певна **конвенція** у світі програмування. Це означає, що це головна функція, яка виконується, коли ми запускаємо нашу програму. Давайте спробуємо це зробити:

```

1 def main():
2     name = input('Введіть ваше ім\'я: ') # 'Гаррі'
3     hello(name)
4
5 def hello(name):
6     print('Привіт, ', name)
7
8 main()

```

Привіт, Гаррі

Викликаючи головну функцію `main()` таким чином, це позбавляє нас від проблем з порядком записів у коді. Таким чином, ми можемо організувати свій код і впорядкувати його.

# ОБЛАСТЬ ВИДИМОСТІ

Змінні, які ви визначаєте всередині функції, не будуть доступні за межами цієї функції:

```

1 def main():
2     name = 'Гаррі'
3     hello()
4
5 def hello():
6     print('Привіт,', name)
7
8 main()

```

NameError: name 'name' is not defined

Щоб зробити змінну доступною за межами функції, ми можемо використати ключове слово **global**:

```

1 def main():
2     global name
3     name = 'Рон'
4     hello()
5
6 def hello():
7     print('Привіт,', name)
8
9 main()

```

# ДЯКУЮ ЗА УВАГУ!

☰ Матеріали курсу

✉ ihor.miroshnychenko@kneu.ua

↳ Data Mirosh

↳ @ihormiroshnychenko

↳ @aranaur

↳ aranaur.rbind.io

