**Модуль 3: Функції**

Функції – це фрагменти програми, які багаторазово використовуються. Вони дозволяють дати ім'я певному блоку команд для того, щоб згодом запускати цей блок за вказаним ім'ям у будь-якому місці програми і скільки завгодно разів. Це називається викликом функції.

**Створення та виклик функцій**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson03/lesson-03#%D1%81%D1%82%D0%B2%D0%BE%D1%80%D0%B5%D0%BD%D0%BD%D1%8F-%D1%82%D0%B0-%D0%B2%D0%B8%D0%BA%D0%BB%D0%B8%D0%BA-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D0%B9)

Функції визначаються за допомогою зарезервованого слова def. Після цього слова вказується ім'я функції, за яким йде пара дужок, у яких можна вказати імена деяких змінних, та заключна двокрапка в кінці рядка. Далі слідує блок команд, що складають функцію. На прикладі можна бачити, що насправді це дуже просто:

def say\_hello():  
 print('Привіт, Світ!') *# блок, що належить функції*  
 *# Кінець функції say\_hello()*  
  
*# виклик функції*  
say\_hello()  
  
*# ще один виклик функції*  
say\_hello()

Ми визначили функцію з ім'ям say\_hello, використовуючи описаний вище синтаксис. Ця функція не приймає параметрів, тому в дужках не оголошені якісь змінні. Параметри функції – це деякі вхідні дані, які ми можемо передати функції, щоб отримати результат, що відповідає їм.

**Аргументи функції**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson03/lesson-03#%D0%B0%D1%80%D0%B3%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B8-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%97)

Функції можуть приймати параметри, тобто деякі значення, що передаються функції для того, щоб вона щось зробила з ними. Ці параметри схожі на змінні, за виключенням того, що значення цих змінних вказуються при виклику функції, та під час роботи функції їм вже присвоєні їх значення. Параметри вказуються в дужках при оголошенні функції та розділяються комами. Аналогічно ми передаємо значення, коли викликаємо функцію. Зверніть увагу на термінологію: імена, вказані при оголошенні функції, називаються параметрами, тоді як значення, які ви передаєте у функцію при її виклику, – аргументами.

def print\_max(a, b):  
 if a > b:  
 print(a, 'максимально')  
 elif a == b:  
 print(a, 'дорівнює', b)  
 else:  
 print(b, 'максимально')  
  
print\_max(3, 4) *# пряма передача значень*  
  
x = 5  
y = 7  
print\_max(x, y) *# передача змінних у якості аргументів*

Тут ми визначили функцію з ім'ям print\_max, яка використовує два параметри з іменами a та b. Ми знаходимо найбільше число із застосуванням простого оператора if..else та виводимо це число. При першому виклику функції print\_max ми напряму передаємо числа як аргументи. У другому випадку ми викликаємо функцію зі змінними як аргументи, print\_max(x, y) призначає значення аргументу x параметру a, а значення аргументу y – параметру b. В обох випадках функція print\_max працює однаково.

**Повернення результату**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson03/lesson-03#%D0%BF%D0%BE%D0%B2%D0%B5%D1%80%D0%BD%D0%B5%D0%BD%D0%BD%D1%8F-%D1%80%D0%B5%D0%B7%D1%83%D0%BB%D1%8C%D1%82%D0%B0%D1%82%D1%83)

У Python немає синтаксичної різниці між функціями і процедурами. По суті, функція вміє повертати деякий результат своєї роботи, а процедура нічого не повертає та результатом її роботи може бути зміна стану вже існуючих змінних. Така форма використання функцій максимально наближена до функцій, з якими ми знайомі з уроків математики.

Для повернення значення з функції необхідно вказати, що повернути після ключового слова return. Наприклад, функція, що виконує операцію додавання.

def plus(a, b):  
 c = a + b  
 return c  
  
res = plus(3, 4)  
print(res) *# Виведе 7*

Або ще коротше:

def plus(a, b):  
 return a + b  
  
print(plus(3, 4)) *# Виведе 7*

**Локальні змінні**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson03/lesson-03#%D0%BB%D0%BE%D0%BA%D0%B0%D0%BB%D1%8C%D0%BD%D1%96-%D0%B7%D0%BC%D1%96%D0%BD%D0%BD%D1%96)

При оголошенні змінних всередині визначення функції вони жодним чином не пов'язані з іншими змінними з таким самим ім'ям за межами функції, тобто імена змінних є локальними у функції. Це називається областю видимості змінної. Область видимості усіх змінних обмежена блоком, в якому вони оголошені, починаючи з точки оголошення імені.

x = 50  
  
def func():  
 x = 2  
 print('Зміна локального x на', x) *# Зміна локального x на 2*  
  
func()  
print('x як і раніше', x) *# x як і раніше 50*

Далі ми назначаємо x значення 2 всередині функції func, але це "локальний" x, який жодним чином не впливає на "глобальний" x поза тілом функції. Тому коли ми заміняємо значення x у функції, x, оголошений в основному блоці, залишається незачепленим. Останнім викликом функції print ми виводимо значення x, вказане в основному блоці, підтверджуючи таким чином, що воно не змінилося при локальному присвоюванні значення в раніше викликаній функції.

**Global**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson03/lesson-03#global)

Щоб присвоїти деяке значення змінній, визначеній на вищому рівні програми (тобто не у якійсь зоні видимості, як функції), необхідно вказати Python, що її ім'я не локальне, а глобальне (global). Зробимо це за допомогою зарезервованого слова global. Без застосування зарезервованого слова global неможливо присвоїти значення змінній, визначеній за межами функції.

x = 50  
  
def func():  
 global x  
 print('x дорівнює', x) *# x дорівнює 50*  
 x = 2  
 print('Змінюємо глобальне значення x на', x) *# Змінюємо глобальне значення x на 2*  
  
func()  
print('Значення x складає', x) *# Значення x складає 2*

Зарезервоване слово global використовується для того, щоб оголосити, що x – це глобальна змінна, а значить, коли ми присвоюємо значення імені x всередині функції, ця зміна відобразиться на значенні змінної x в основному блоку програми. Використовуючи одне зарезервоване слово global, можна оголосити відразу декілька змінних: global x, y, z. Використовувати global варто з великою обережністю і не потрібно зловживати такою можливістю.

**Ключові аргументи**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson03/lesson-03#%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%B2%D1%96-%D0%B0%D1%80%D0%B3%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B8)

Якщо є деяка функція з великою кількістю параметрів, і при її виклику вимагається вказати тільки деякі з них, значення цих параметрів можуть задаватися за їх ім'ям – це називається ключові параметри. В цьому випадку для передачі аргументів функції використовується ім'я (ключ), замість позиції (як було досі). Є дві переваги такого підходу: по-перше, використання функції стає легшим, оскільки немає потреби відстежувати порядок аргументів; по-друге, можна задавати значення тільки деяким обраним аргументам, за умови, що інші параметри мають значення аргументу за замовчуванням.

def func(a, b=5, c=10):  
 print('a дорівнює', a,', b дорівнює', b,', а c дорівнює', c)  
  
func(3, 7) *# a дорівнює 3, b дорівнює 7, а c дорівнює 10*  
func(25, c=24) *# a дорівнює 25, b дорівнює 5, а c дорівнює 24*  
func(c=50, a=100) *# a дорівнює 100, b дорівнює 5, а c дорівнює 50*  
Функція з ім'ям func має один параметр без значення за замовчуванням, за яким слідують два параметри зі значеннями за замовчуванням. При першому виклику func(3, 7), параметр a отримує значення 3, параметр b отримує значення 7, а c отримує своє значення за замовчуванням, що дорівнює 10.

При другому виклику func(25, c=24) змінна a отримує значення 25 на підставі позиції аргументу. Після цього параметр c отримує значення 24 за ім'ям, тобто як ключовий параметр. Змінна b отримує значення за замовчуванням, що дорівнює 5.

При третьому зверненні func(c=50, a=100) ми використовуємо ключові аргументи для усіх вказаних значень. Зверніть увагу на те, що ми вказуємо значення для параметра c перед значенням для a, навіть попри те, що у визначенні функції параметр a вказаний раніше c.

def say(message, times=1):  
 print(message \* times)  
  
say('Привіт') *# Привіт*  
say('Світ', 5) *# СвітСвітСвітСвітСвіт*

Функція з ім'ям say використовується для виведення на екран рядка, вказаного число разів. Якщо ми не вказуємо значення за замовчуванням, рядок виводиться один раз. Ми досягаємо цього зазначенням значення аргументу за замовчуванням, що дорівнює 1 для параметра times. При першому виклику say ми вказуємо тільки рядок, і функція виводить його один раз. При другому виклику say ми вказуємо також і аргумент, позначаючи таким чином, що ми хочемо сказати фразу 5 разів.

Значеннями за замовчуванням можуть бути забезпечені тільки параметри, що знаходяться у кінці списку параметрів. Таким чином, у списку параметрів функції параметр зі значенням за замовчуванням не може передувати параметру без значення за замовчуванням. Це пов'язано з тим, що значення надаються параметрам відповідно до їх положення. Наприклад, def func(a, b=5) — допустимо, а def func(a=5, b) – не допустимо.

**Змінна кількість параметрів**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson03/lesson-03#%D0%B7%D0%BC%D1%96%D0%BD%D0%BD%D0%B0-%D0%BA%D1%96%D0%BB%D1%8C%D0%BA%D1%96%D1%81%D1%82%D1%8C-%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80%D1%96%D0%B2)

Іноді буває необхідно визначити функцію, здатну приймати будь-яку кількість параметрів. Цього можна досягти за допомогою зірочок:

def total(a=5, \*numbers, \*\*phone\_book):  
 print('a', a)  
 *# прохід по всіх елементах кортежу*  
 for single\_item in numbers:  
 print('single\_item', single\_item)  
  
 *#прохід по всіх елементах словника*  
 for first\_part, second\_part in phone\_book.items():  
 print(first\_part,second\_part)

print(total(10, 1, 2, 3, Jack=1123, John=2231, Inge=1560))

В результаті в консолі ми побачимо:

a 10  
single\_item 1  
single\_item 2  
single\_item 3  
Jack 1123  
John 2231  
Inge 1560  
None

Коли ми оголошуємо параметр із зірочкою (наприклад, \*numbers), всі позиційні аргументи, починаючи з цієї позиції до кінця, будуть зібрані в кортеж під ім'ям numbers. Аналогічно, коли ми оголошуємо параметри із двома зірочками (\*\*phone\_book), всі ключові аргументи, починаючи з цієї позиції до кінця, будуть зібрані в словник під ім'ям phone\_book.

**Контейнери для зберігання аргументів функцій**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson03/lesson-03#%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%B9%D0%BD%D0%B5%D1%80%D0%B8-%D0%B4%D0%BB%D1%8F-%D0%B7%D0%B1%D0%B5%D1%80%D1%96%D0%B3%D0%B0%D0%BD%D0%BD%D1%8F-%D0%B0%D1%80%D0%B3%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%96%D0%B2-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D0%B9)

Для зберігання "зайвих" аргументів застосовуються контейнери Python, кортежі і словники. У ці контейнери збираються усі аргументи, які не визначені вами явно, але вони були передані при виклику функції.

Звичайно, в контейнери Python збирає аргументи, тільки якщо ви вкажете зробити так за допомогою синтаксису з \* і/або \*\*.

У прикладі вище numbers — це кортеж, а phone\_book — словник.

**Кортежі**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson03/lesson-03#%D0%BA%D0%BE%D1%80%D1%82%D0%B5%D0%B6%D1%96)

Кортежі у Python — це впорядковані незмінні множини елементів. Елементом кортежу може бути будь-який тип даних. Кортежі не можна змінювати, не можна додавати/видаляти/переставляти елементи.

Щоб створити порожній кортеж, існують два способи, хоча і не зовсім зрозуміло навіщо потрібний порожній кортеж :-)

my\_tuple = tuple()  
another\_tuple = ()

Створення ж непорожніх кортежів відбувається наступним чином:

not\_empty = (1, 2, 3)  
Доступ до елементів кортежу відбувається за індексом за допомогою синтаксису квадратних дужок:

not\_empty = (2, 4, 6)  
not\_empty[0] *# 2*  
not\_empty[1] *# 4*  
not\_empty[2] *# 6*

Індексом слугує суворо ціле число. Індексація починається з 0. Ще з кортежами вміє працювати цикл for та може одразу перебрати усі елементи кортежу, як в прикладі з total.

**Словники**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson03/lesson-03#%D1%81%D0%BB%D0%BE%D0%B2%D0%BD%D0%B8%D0%BA%D0%B8)

Словник — це контейнер, який зберігає пари ключ-значення. Ключем може бути будь-який незмінний тип даних Python (число, рядок, кортеж тощо). Значенням словника може бути будь-який тип даних Python, включаючи призначені для користувача типи.

Порожній словник можна створити одним з двох способів:

empty\_dict = {}  
another\_empty\_dict = dict()  
Створити непорожній словник можна наступним чином:

some\_dict = {  
 "key": "value",  
 1: "one",  
}

Для створення заповненого деякими значеннями словника, достатньо перелічити пари ключ-значення через кому всередині фігурних дужок, ключ йде першим, потім двокрапка та значення.

У прикладі показано застосування в якості ключів рядка і числа.

Запис пари ключ-значення у вже існуючий словник відбувається за допомогою квадратних дужок і оператора присвоєння =:

not\_empty = {"key": "value"}  
not\_empty["new\_key"] = "new value"  
print(not\_empty) *# {"key": "value", "new\_key": "new value"}*

**Рекурсія**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson03/lesson-03#%D1%80%D0%B5%D0%BA%D1%83%D1%80%D1%81%D1%96%D1%8F)

Рекурсивна функція — це функція, що визначається в термінах самої себе і здатна викликати саму себе. Це означає, що функція викликатиме себе і повторюватиме свою поведінку до тих пір, доки не буде виконана деяка умова для повернення результату.

Найчастіший приклад використання рекурсивних функцій — це обчислення факторіалу. Спершу нагадаємо визначення факторіалу з математики: факторіал натурального числа n визначається як добуток усіх натуральних чисел від 1 до n включно. Наприклад: 5! = 1 · 2 · 3 · 4 · 5 = 120. Це саме визначення можна записати рекурсивно:

* 5! = 5 · 4!
* 4! = 4 · 3!
* 3! = 3 · 2!
* 2! = 2 · 1!
* 1! = 1

def factorial(n):  
 if n <= 1:  
 return 1  
 else:  
 return n \* factorial(n - 1)  
  
factorial(5) *# 120*

Ми визначили функцію factorial, яка приймає як аргумент деяке число n. Якщо n менше або дорівнює 1, то факторіал такого числа вже відомий та дорівнює 1. В іншому випадку , щоб знайти факторіал числа n, потрібно помножити n на факторіал n - 1, а факторіал n - 1 ми знаходимо, використовуючи функцію factorial. Викликати ж factorial ми будемо до тих пір, доки на черговій ітерації n не стане менше або дорівнює 1.

Рекурсивні функції зручні в ситуаціях, коли ми не знаємо заздалегідь, скільки разів потрібно буде викликати функцію, наприклад, при розборі папок на диску. Застосунок не знає заздалегідь, наскільки глибока структура папок і який у них рівень вкладеності. І щоб перебрати усі файли в усіх вкладених папках, функція повинна викликати сама себе, коли зустрічає чергову папку. Така функція, яка викликає сама себе за деяких умов, називається рекурсивною.

**Імпорт пакетів та модулів**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson03/lesson-03#%D1%96%D0%BC%D0%BF%D0%BE%D1%80%D1%82-%D0%BF%D0%B0%D0%BA%D0%B5%D1%82%D1%96%D0%B2-%D1%82%D0%B0-%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%96%D0%B2)

У Python є великий набір пакетів та модулів з готовими корисними функціями та інструментами. Якщо вам потрібно зробити щось, чого немає в стандартному невеликому наборі функцій, ви можете імпортувати пакет, де є необхідна функція, щоб скористатися нею. Імпортування у Python відбувається за допомогою ключового слова import, після якого ви можете вказати один або декілька пакетів, які ви хочете імпортувати.

import math  
  
sin\_pi = math.sin(math.pi)

У цьому прикладі ми імпортували пакет математичних функцій і констант math та зберегли значення синуса π у sin\_pi

Для того щоб викликати функцію з імпортованого пакету, потрібно вказати ім'я пакету і через крапку ім'я функції або константи в цьому пакеті. Цей синтаксис дуже схожий на виклик метода, ми робили так зі словниками та списками раніше.

Є й інший спосіб: можна імпортувати з пакета тільки те, що нам необхідно за допомогою виразу from ... import ...:

from math import pi, sin  
  
sin\_pi = sin(pi)  
Цей код робить те саме, що і код вище. Використовуйте той синтаксис, який вам здається зручнішим.

Імпортувати код можна не лише із стандартних або встановлених пакетів (модулей), але й з власноручно написаних модулів Python.

Модулем Python є будь-який текстовий файл з розширенням py, який містить код мовою Python. Це означає, що коли ви створюєте скрипт hello.py, який містить ось такий код:

def say\_hello(name):  
 print(f'Hello {name}')  
ви можете імпортувати функцію say\_hello з hello.py за домопогою виразу: from hello import say\_hello у будь-якому модулі в тій самій папці, що і hello.py.

Це дуже зручно і дозволяє структурувати ваш код, розділяючи його на окремі файли (модулі).

Важливо розуміти, що під час імпорту модуля Python виконує увесь код, що міститься в модулі. Саме через це модуль з ось таким вмістом:

def say\_hello(name):  
 print(f'Hello {name}')  
  
print("You imported hello.py")  
say\_hello('user')  
Під час імпорту (виконання виразу from hello import say\_hello) виведе у консоль:

You imported hello.py  
Hello user  
Далеко не завжди така поведінка бажана.

**Точка входу**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson03/lesson-03#%D1%82%D0%BE%D1%87%D0%BA%D0%B0-%D0%B2%D1%85%D0%BE%D0%B4%D1%83)

Що ж робити, коли ми хочемо зробити скрипт, що виконується (який можна викликати із консолі командою python [ім'я скрипта]), але зберегти можливість імпорту з цього модуля, не викликаючи його? В таких випадках нам може допомогти службова змінна Python: \_\_name\_\_. Річ у тому, що якщо скрипт викликаний безпосередньо, то він є "точкою входу" та \_\_name\_\_ == "\_\_main\_\_". Якщо ж цей модуль виконується під час імпорту, то \_\_name\_\_ == "hello" (наприклад для модуля, який називається hello.py). Таким чином ми можемо модифікувати наш модуль hello.py:

def say\_hello(name):  
 print(f'Hello {name}')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print("You imported hello.py")  
 say\_hello('user')

Тоді під час імпорту функції say\_hello із hello.py код у блоці if ... не буде виконаний, а якщо ж в консолі виконати python hello.py, то буде.

Для зручності прийнято весь код, який потрібно виконати, коли модуль викликається із консолі (не імпортується), поміщати у функцію main:

def say\_hello(name):  
 print(f'Hello {name}')  
  
def main():  
 print("You imported hello.py")  
 say\_hello('user')  
  
for arg in sys.argv:  
 print(arg)  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Так заведено і функцію main ще називають "точкою входу", оскільки робота застосунку починається з виклику цієї функції. Ви, звичайно, можете назвати цю функцію як завгодно, але називати її саме main вважається хорошим тоном.