

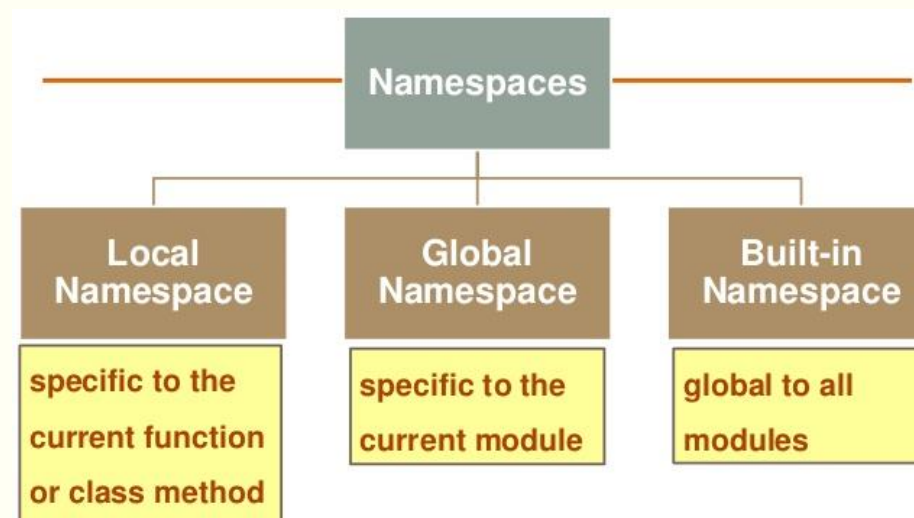
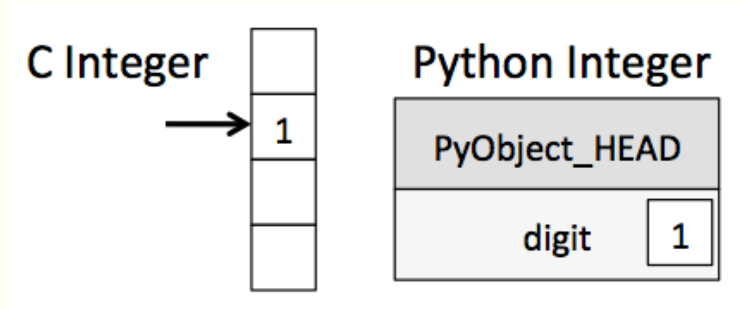


СТРУКТУРА ПРОГРАМИ МОВОЮ PYTHON

Питання 6.2.

Яка структура Python- програми?

- Блочна структура, яка позначається відступами
 - 1) Повторне використання програмного коду за допомогою інструкції `import: import re`
 - 2) Імена та простори імен
 - Завантажена та запущена програма заповнюють пам'ять об'єктами.
 - Завантаження програми розміщує в пам'яті `function object`, який її представляє.
 - У процесі виконання можуть створюватись інші об'єкти.
 - Об'єкти в пам'яті можуть мати назви або бути безіменними.
 - Зазвичай іменування відбувається при створенні об'єктів для того, щоб звернутись до них за потреби.



2) Імена та простори імен

```
def printList(upper_limit, step=2):  
    # variables upper_limit and step belong to this  
    # function's local namespace.  
    print "upper limit: %d" % upper_limit  
    num_list = range(0, upper_limit, step)  
    print num_list  
  
printList(upper_limit=5, step=2)  
a = "foo"  
b = "bar"  
# variables a, b and function printList belong to  
# Global namespace of this module
```

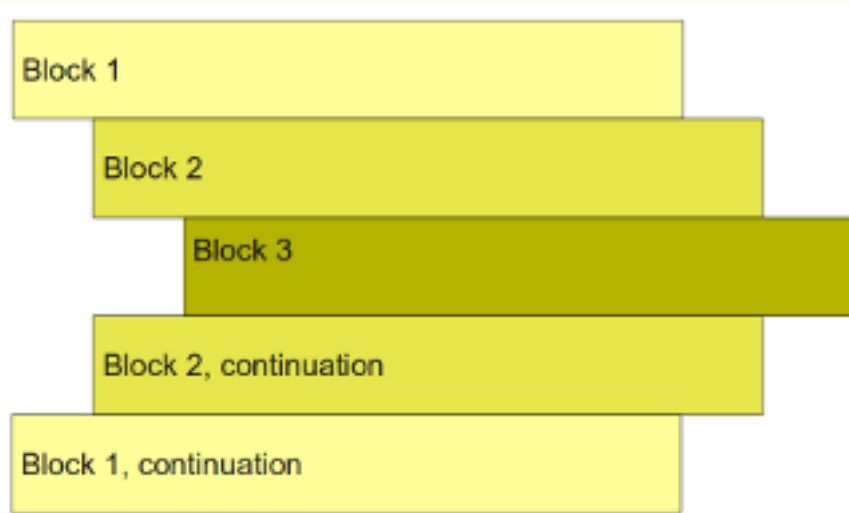
- Кожне ім'я в Python належить деякому простору імен (namespace).
 - Простір імен, який містить усі імена, доступні при запуску Python, називається `__builtins__`.
 - До імен з `__builtins__` не потрібно дописувати спереду назву модуля, оскільки це простір імен найвищого рівня (top-level).

builtin_ namespace

```
>>> dir(__builtins__)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BufferError',
'BytesWarning', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError',
'Exception', 'False', 'FloatingPointError', 'FutureWarning', 'GeneratorExit',
'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'KeyError',
'KeyboardInterrupt', 'LookupError', 'MemoryError', 'NameError', 'None', 'NotImplemented',
'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning',
'ReferenceError', 'RuntimeError', 'RuntimeWarning', 'StandardError', 'StopIteration',
'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'True',
'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError',
'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError',
'Warning', 'ZeroDivisionError', '_', '__debug__', '__doc__', '__import__', '__name__',
'__package__', 'abs', 'all', 'any', 'apply', 'basestring', 'bin', 'bool', 'buffer',
'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'cmp', 'coerce', 'compile',
'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate',
'eval', 'execfile', 'exit', 'file', 'filter', 'float', 'format', 'frozenset',
'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int',
'intern', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals',
'long', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord',
'pow', 'print', 'property', 'quit', 'range', 'raw_input', 'reduce', 'reload', 'repr',
'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str',
'sum', 'super', 'tuple', 'type', 'unichr', 'unicode', 'vars', 'xrange', 'zip']
```

3) Блоки коду

- Циклічні структури (цикли for і while)
- if/then/else інструкції (statements)
- Визначення функцій та класів
- Конструкція with (використовується для об'єктів, зокрема потоків вводу-виводу (streams))



```
if X > Y:
    print Y

for elem in data:
    print elem[0]
    print elem[1]

def inc_abs_value (x):
    if x >= 0:
        inc = 1
    else:
        inc = -1
    return x + inc

with open('pride_and_prejudice.txt', 'r') as fh:
    file_contents = fh.read()
```

Блоки коду завжди починаються після двокрапки (:).

Блоки коду та складені інструкції (Compound statements)

- Складені інструкції (складені оператори мови програмування) – дещо ширше поняття. Включає
 - Оператор розгалуження (if statement)
 - Оператори циклу (while statement, for statement)
 - Оператор перехоплення виключень (try statement)
 - Менеджери контексту (with statement)
 - Визначення функцій
 - Визначення класів
 - Співпрограми (Coroutines)
 - Оголошення функції у співпрограмі (Coroutine function definition)
 - Оператор async for
 - Оператор async with

```
def coroutine(f):  
    def wrap(*args, **kwargs):  
        gen = f(*args, **kwargs)  
        gen.send(None)  
        return gen  
    return wrap  
  
@coroutine  
def calc():  
    history = []  
    while True:  
        x, y = (yield)  
        if x == 'h':  
            print history  
            continue  
        result = x + y  
        print result  
        history.append(result)
```

Прості інструкції (Simple statements)

- У простих інструкціях можуть бути:
 - Інструкції-вирази (Expression statements)
 - Оператор присвоєння (Assignment statement)
 - Оператор assert
 - Оператор pass
 - Оператор del
 - Оператор return
 - Оператор yield
 - Оператор raise
 - Оператори break та continue
 - Оператор import
 - Оператор global
 - Оператор nonlocal

https://docs.python.org/3/reference/simple_stmts.html

```
simple_stmt ::= expression_stmt  
            | assert_stmt  
            | assignment_stmt  
            | augmented_assignment_stmt  
            | annotated_assignment_stmt  
            | pass_stmt  
            | del_stmt  
            | return_stmt  
            | yield_stmt  
            | raise_stmt  
            | break_stmt  
            | continue_stmt  
            | import_stmt  
            | future_stmt  
            | global_stmt  
            | nonlocal_stmt
```

Вирази в мові Python

- Вираз у мові Python – логічна послідовність чисел, рядків, об'єктів та операторів.
 - Саме по собі значення є виразом, а отже, змінна також є виразом.
 - За допомогою виразів можна виконувати операції на зразок додавання, віднімання, конкатенації тощо.
 - Вираз може викликати функцію для визначення результатів.



```
print("The answer is: " + str(round(x * 5, 1)))
```


Оператори мови Python

▪ Оператор присвоєння (Assignment statement):

- `name = expression`
- Мета – пов'язати імена з їх значеннями в програмі.
- Єдиний оператор без ключового слова.
- Python знайде значення виразу (evaluate the expression), звівши його до одного значення (value), а потім зв'яже (bind) ім'я з цим значенням.

```
value = arg[1]
date  = '12/03/2034'
expr  = value * 2
value = value + 1
y      = f(value)
x      = y = 20
```

Моржовий (walrus-) оператор (PEP 572, Python 3.8+) :=

- Альтернативна назва для виразів присвоєння.
 - Наочно продемонстрував Victor Stinner (розробник Python Core) у Twitter:

448 - while True:	446 + while (line := fp.readline()):
449 - line = fp.readline()	447 + if (m := define_rx.match(line)):
450 - if not line:	
451 - break	
452 - m = define_rx.match(line)	
453 - if m:	
454 n, v = m.group(1, 2)	448 n, v = m.group(1, 2)
455 try:	449 try:
456 v = int(v)	450 v = int(v)
457 except ValueError:	451 except ValueError:
458 pass	452 pass
459 vars[n] = v	453 vars[n] = v
460 - else:	454 + elif (m := undef_rx.match(line)):
461 - m = undef_rx.match(line)	455 + vars[m.group(1)] = 0
462 - if m:	
463 - vars[m.group(1)] = 0	
464 return vars	456 return vars

Оператори мови Python

▪ Оператор assert :

- Націлений на налагодження коду, тестує стан.
- Якщо умова true, нічого не робить, просто продовжує роботу. Інакше викидає (raise) виключення AssertionError з опційним повідомленням про помилку.
- Задача – інформувати розробників про невилправлені (unrecoverable) помилки в програмі.

```
# defining the function definition
def divide(num1, num2):
    assert num2 > 0 , "Divisor cannot be zero"
    return num1/num2

# calling the divide function
a1 = divide(12,3)
# print the quotient
print(a1)
# this will give the assertion error
a2 = divide(12,0)
print(a2)
```

```
4.0
Traceback (most recent call last):
  File "D:/T_Code/PythonPackage3/Assert.py", line 10, in
    a2 = divide(12,0)
  File "D:/T_Code/PythonPackage3/Assert.py", line 3, in divide
    assert num2>0 , "Divisor cannot be zero"
AssertionError: Divisor cannot be zero
```

Оператори мови Python

▪ Оператор pass:

- Оператор-заглушка, рівноцінний відсутності операції.
- Під час виконання даного оператора нічого не відбувається.
- Може використовуватись, наприклад, в інструкціях, де тіло є обов'язковим, таких як def, except та in.
- Часто pass використовується там, де код поки ще не з'явився, але планується.

```
1 # pass is just a placeholder for
2 # functionality to be added later.
3 sequence = {'p', 'a', 's', 's'}
4 for val in sequence:
5     pass
```

Оператори мови Python

- **Оператор del:**

- Видаляє елемент з набору даних (тут – список) за індексом, а не значенням.
- `>>> a = [3, 2, 2, 1]`
- `>>> del a[1]`
- `[3, 2, 1]`

- Схожі дії виконують функції `remove()` і `pop()`:

remove видаляє перше значення, яке співпало:

- `>>> a = [0, 2, 3, 2]`
- `>>> a.remove(2)`
- `>>> a`
- `[0, 3, 2]`

pop повертає видалений елемент:

- `>>> a = [4, 3, 5]`
- `>>> a.pop(1)`
- `3`
- `>>> a`
- `[4, 5]`

`del` (на відміну від `pop`) дозволяє видалити діапазон індексів:

```
>>> lst = [3, 2, 2, 1]
>>> del lst[1:]
>>> lst [3]
```

Оператори мови Python

▪ Оператор return:

- Може зустрічатись у довільному місці функції.
- Завершує роботу функції та повертає вказане значення у місце виклику.
- Якщо функція не повертає значення, оператор return використовується без значення, що повертається.
- У функціях, яким не потрібно повертати значення, return може бути відсутнім.

```
1 def max(a, b):  
2     if a > b:  
3         return a  
4     else:  
5         return b  
6  
7 def max3(a, b, c):  
8     return max(max(a, b), c)  
9  
10 print(max3(3, 5, 4))
```

Оператори мови Python

▪ Оператор `yield`:

- Використовується так же, як і слово `return`, проте повертає генератор замість значення.
- Генератори – ітеровані об'єкти, проте загалом можуть використовуватись лише один раз.

```
>>> def generator():
...     for i in (1, 2, 3):
...         yield i
...
>>> g = generator() # create a generator
>>> print(g)
<generator object generator at 0x2e58870>
>>> for i in g:
...     print(i)
1
2
3
```

Генерують значення на льоту в процесі запиту.

```
>>> generator = (x*x for x in xrange(3))
>>> for i in generator:
...     print(i)
0
1
4
```

Оператори мови Python

▪ Оператор `raise`:

- дозволяє переривати стандартний потік виконання коду за допомогою **викидання винятків**;
- якщо після інструкції відсутній вираз, повторно піднімається виняток з даної області коду;
- Якщо в даній області немає активного винятку, викидається `RuntimeError` (до py3.0 – `TypeError`).

```
try:
    # Допустим в функции поднимается FileNotFoundError.
    do()

except OSError:
    # Инструкция raise без выражение поднимет FileNotFoundError повторно.
    raise
```

```
# Объект сформируется из класса автоматически.
raise MyException

# Формируем объект исключения вручную.
raise MyException('Моё исключение')
```


Оператори мови Python


▪ Оператор `break`:

- дозволяє перервати цикл за умови виникнення зовнішнього фактора;
- слід розташувати відразу після оператора циклу (зазвичай після виразу `if`);

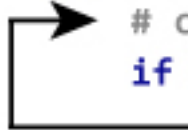
▪ Оператор `continue`:

- дозволяє пропустити частину циклу за умови виникнення зовнішнього фактора та перейти до наступної ітерації циклу.
- розташовується в блоці коду після оператора циклу (зазвичай після `if`).

```
for var in sequence:
    # codes inside for loop
    if condition:
        break
    # codes inside for loop
# codes outside for loop
```



```
for var in sequence:
    # codes inside for loop
    if condition:
        continue
    # codes inside for loop
# codes outside for loop
```



Оператори мови Python

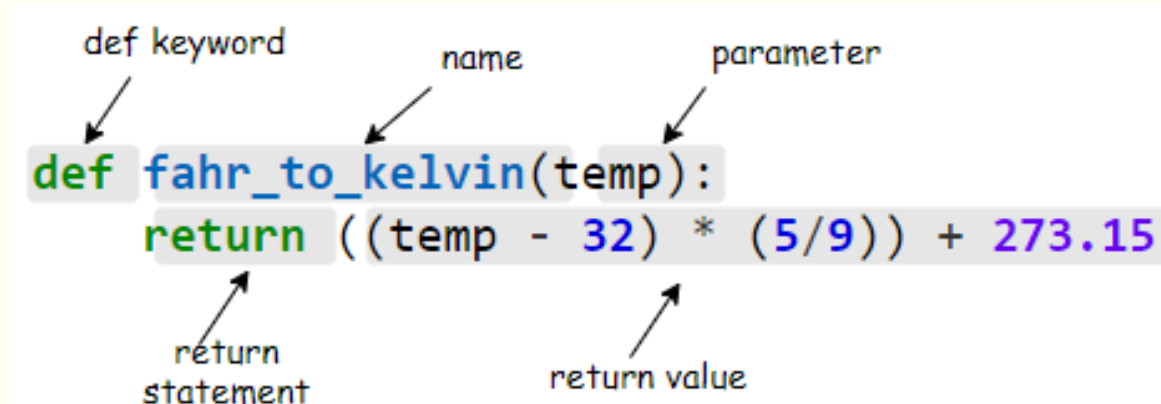
▪ Оператори `global` та `nonlocal`:

- Змінна може бути локальною або глобальною.
- `Nonlocal` аналогічний до `global`, проте націлений на вкладені функції. Це не глобальна, але і не локальна змінна.

```
def method():  
    # Change "value" to mean the global variable.  
    # ... The assignment will be local without "global."  
    global value  
    value = 100  
  
value = 0  
method()  
  
# The value has been changed to 100.  
print(value)
```

```
def method():  
  
    def method2():  
        # In nested method, reference nonlocal variable.  
        nonlocal value  
        value = 100  
  
        # Set local.  
        value = 10  
        method2()  
  
        # Local variable reflects nonlocal change.  
        print(value)  
  
    # Call method.  
    method()
```

4) Функції та їх параметри



The diagram shows a Python function definition with labels pointing to its components:

```
def fahr_to_kelvin(temp):  
    return ((temp - 32) * (5/9)) + 273.15
```

- def keyword**: points to the `def` keyword.
- name**: points to the function name `fahr_to_kelvin`.
- parameter**: points to the parameter `temp`.
- return statement**: points to the `return` keyword.
- return value**: points to the expression `((temp - 32) * (5/9)) + 273.15`.

- Функція може бути пустою (stub):

```
1 def empty_function():  
2     pass
```

- Кожна функція видає певний результат.
 - Якщо ви не вказуєте на видачу конкретного результату, вона видасть результат `None` (нічого).

Функція main() у Python

- Python-функція main() виконується лише тоді, коли скрипт запускається як програма.
 - Ми можемо імпортувати python-програму в якості модуля – тоді метод main() має не виконуватись.
- Інтерпретатор послідовно виконує код та не викликає метод, якщо він не є частиною цього коду.
 - Спеціальний підхід, щоб визначити метод main() у програмі лише тоді, коли програма запускається напряму:

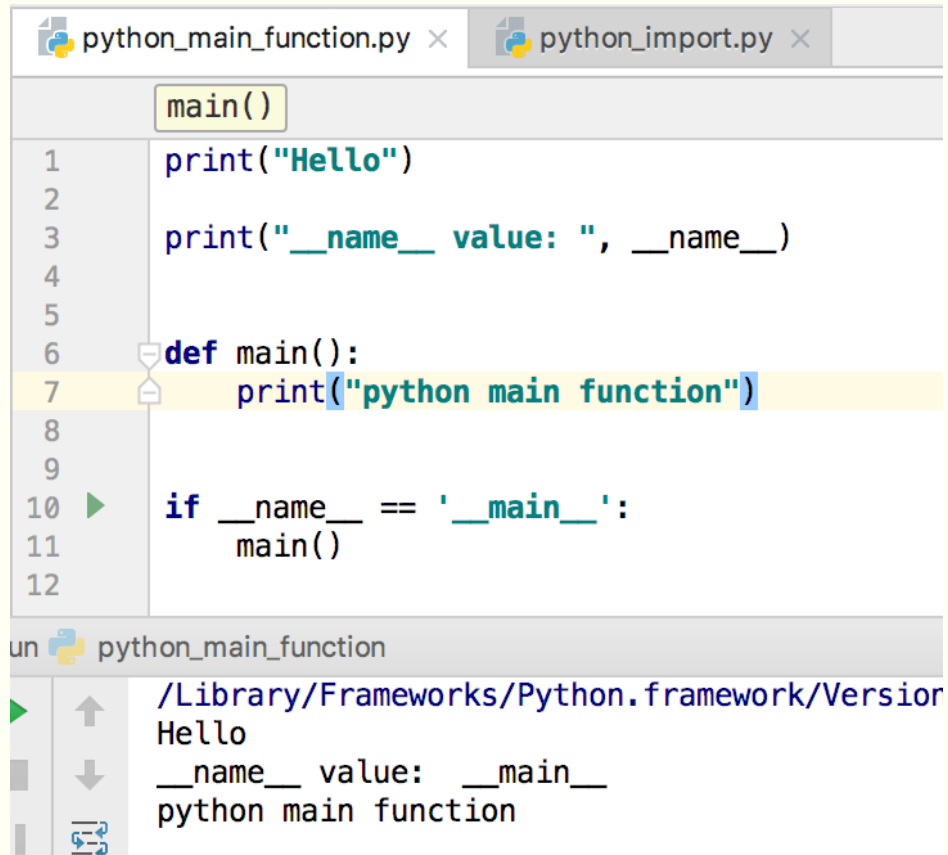
```
1 print("Hello")
2 print("__name__ value: ", __name__)
3
4
5 def main():
6     print("python main function")
7
8
9 if __name__ == '__main__':
10     main()
```

Інтерпретатор задає неявній змінній __name__ значення __main__.

- Якщо файл з кодом імпортується у вигляді модуля, інтерпретатор присвоює __name__ назву модуля.
- Умова поверне false і метод main() не буде виконуватись.

```
Hello
__name__ value: __main__
python main function
```

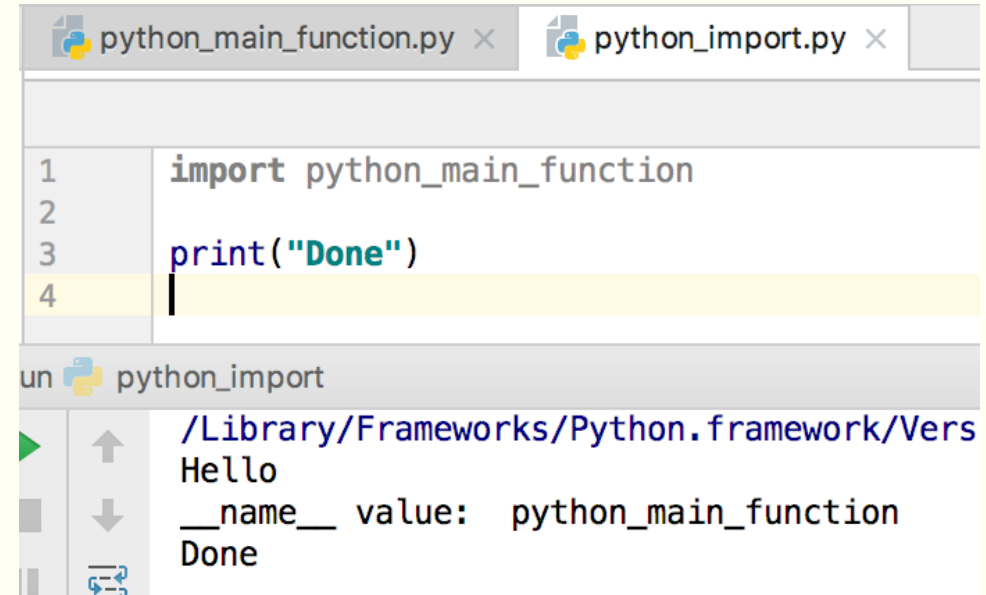
Функція main() у якості модуля



```
python_main_function.py x python_import.py x
main()
1 print("Hello")
2
3 print("__name__ value: ", __name__)
4
5
6 def main():
7     print("python main function")
8
9
10 if __name__ == '__main__':
11     main()
12
```

un python_main_function

/Library/Frameworks/Python.framework/Vers
Hello
__name__ value: __main__
python main function



```
python_main_function.py x python_import.py x
import python_main_function
2
3 print("Done")
4
```

un python_import

/Library/Frameworks/Python.framework/Vers
Hello
__name__ value: python_main_function
Done

- Перші 2 рядки виводу – з файлу `python_main_function.py` source file.
- Значення `__name__` відрізняється, тому метод `main()` не виконується.

Аргументи функцій

- Де у функцій аргументи, а де параметри?
 - Параметр (формальний параметр) – змінна, яка є частиною сигнатури функції.
 - Аргумент (фактичний параметр) – вираз, який використовується при виклику функції.
 - `i` та `f` – параметри,
 - `anInt` та `2.0` – аргументи.

```
void Foo(int i, float f)
{
    // Do things
}

void Bar()
{
    int anInt = 1;
    Foo(anInt, 2.0);
}
```

Функція може приймати будь-яку кількість аргументів чи не приймати їх зовсім.

Поширені функції з

- довільною кількістю аргументів,
- позиційними та іменованими аргументами (обов'язковими і необов'язковими).

Обов'язкові та необов'язкові аргументи

```
>>> def func(a, b, c=2): # c – необов'язковий аргумент
...     return a + b + c
...
>>> func(1, 2) # a = 1, b = 2, c = 2 (за умовчанням)
5
>>> func(1, 2, 3) # a = 1, b = 2, c = 3
6
>>> func(a=1, b=3) # a = 1, b = 3, c = 2
6
>>> func(a=3, c=6) # a = 3, c = 6, b не визначено
Traceback (most recent call last):
  File "", line 1, in
    func(a=3, c=6)
TypeError: func() takes at least 2 arguments (2 given)
```

Необов'язкові та іменовані (keyword) аргументи

- **def** `help(object, spacing=10, collapse=1):`
 - Аргументи `spacing` і `collapse` можна опустити при виклику.
 - Аргумент `object` не має значення за умовчанням, тому повинен вказуватись завжди.
- Python дозволяє передавати аргументи в довільному порядку за іменем.
- Різні варіанти виклику функції `help()`:
 - `help(odbcHelper)`
 - `help(odbcHelper, 12)`
 - `help(odbcHelper, collapse=0)`
 - `help(spacing=15, object=odbcHelper)`

Позиційні аргументи та їх довільна кількість

- Функція може приймати змінну кількість позиційних аргументів, перед ім'ям ставиться *:
 - args - це кортеж з усіх переданих аргументів функції, і зі змінною можна працювати так само, як і з кортежем.

```
>>> def func(*args):
```

```
...     return args
```

```
...
```

```
>>> func(1, 2, 3, 'abc')
```

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

```
>>> func()
```

```
()
```

```
>>> func(1)
```

```
(1,)
```

Функція може приймати і довільну кількість іменованих аргументів, перед ім'ям ставиться **:

У змінній kwargs зберігається словник.

```
>>> def func(**kwargs):
```

```
...     return kwargs
```

```
...
```

```
>>> func(a=1, b=2, c=3)
```

```
{'a': 1, 'c': 3, 'b': 2}
```

```
>>> func()
```

```
{}
```

```
>>> func(a='python')
```

```
{'a': 'python'}
```

Ще Python-програма може включати:

- 5) Класи та їх атрибути

```
class Car():  
  
    def __init__(self, make, model, year, color):  
        self.make = make  
        self.model = model  
        self.year = year  
        self.color = color  
  
    def outputvalues(self):  
        #returns all parameter values as a tuple  
        vals = (self.make, self.model, self.year, self.color)  
        return vals  
  
mycar = Car("BMW", "X5", 2018, "Black")  
print(mycar.outputvalues())  
  
( 'BMW', 'X5', 2018, 'Black' )
```

- 6) Файли та потоки вводу-виводу (IO streams)



ДЯКУЮ ЗА УВАГУ!

Наступне питання: Управляючі оператори мови програмування Python

- PEP8