

Київський національний університет імені Тараса Шевченка

РЕФЕРАТ

з дисципліни “Основи програмування”  
на тему “Функціональне програмування в Python”

студентки  
механіко-математичного факультету  
1 курсу спеціальності “Математика”  
заочної форми навчання  
Косяк Анастасії Олександрівни

КИЇВ - 2021

Функціональне програмування

Функціональне програмування в Python

Модуль itertools

Модуль functools

Модуль operator

Висновки

Посилання на використані джерела

# Функціональне програмування

Функції є важливим елементом для структурування коду в багатьох мовах програмування.

Загалом, під функціональним програмуванням мається на увазі використання функцій як найкращий спосіб досягнення чистого та легкопідтримуваного коду. Якщо говорити точніше, то функціональне програмування - це набір підходів до написання коду, яке зазвичай називають парадигмою.

Функціональне програмування має певні переваги та застосовується в багатьох мовах та фреймворках, і зараз є популярним підходом в розробці програмного забезпечення.

Розглянемо основні принципи функціонального програмування.

## Чисті функції (pure functions)

В функціональному програмуванні чиста функція (pure function) - це функція, чиї результати залежать лише від вхідних параметрів і чиї операції не мають жодних побічних ефектів (side effects), тобто такі функції лише повертають певне значення, але не мають зовнішнього впливу.

Чисті функції архітектурно прості. В ООП (об'єктно-орієнтованому програмуванні) кожен об'єкт може мати методи (так називаються функції, що належать об'єктам) і ці методи можуть змінювати стан об'єкту. Чисті функції ж за означенням не можуть змінювати стан об'єктів.

## Незмінність (immutability)

Ще одним принципом в функціональному програмуванні є незмінність даних за межами функції. На практиці це значить уникати модифікації вхідних параметрів функції і повертати з функції нове значення. Це дозволяє уникати побічних ефектів (side effects).

## Першокласні функції (first-class functions)

Першокласна функція - це функція, яка вважається окремою незалежною сутністю, і яку можна використовувати як аргумент, як окрему змінну, повертати як результат функції.

### Функції вищого порядку (high-order functions)

Функція, яка приймає іншу функцію як параметр або повертає функцію називається функцією вищого порядку. Тобто функція вищого порядку - це функція, що оперує над іншою функцією.

---

## Функціональне програмування в Python

В попередньому розділі ми розглянули поняття функціонального програмування, а також основні принципи цієї парадигми. Тепер пропоную більш детально поговорити про те, як підтримується функціональне програмування саме в Python.

### Модуль `itertools`

Модуль `itertools` - це модуль, який надає різні функції, що оперують над ітераторами для створення більш складних ітераторів. Цей модуль працює як швидкий інструмент, який або використовується сам по собі, або в комбінації для того, щоб утворити алгебру ітераторів.

В модулі `itertools` є декілька типів ітераторів:

- `infinite` ітератори
- `combinatoric` ітератори
- `terminating` ітератори

Розглянемо ці типи більш детально.

#### Infinite iterators

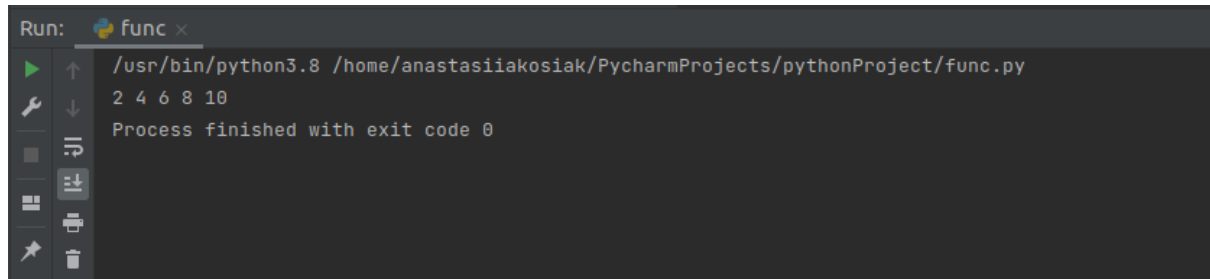
Ітератори можуть бути нескінченними - тобто завдяки ним можна обходити та отримувати нескінченну кількість елементів. Такі типи ітераторів називаються `infinite iterators`.

В Python є три типи нескінченних ітераторів:

- `count(start, step)`: цей ітератор починає друкувати із числа, який вказаний як початок і може друкувати до нескінченності. Якщо вказаний крок (`step`), то числа пропускаються. За замовчуванням крок дорівнює 1.

```
import itertools
```

```
for i in itertools.count(2, 2):
    if i > 10:
        break
    else:
        print(i, end=" ")
```



Скріншот №1. Результат виконання коду із Listing №1

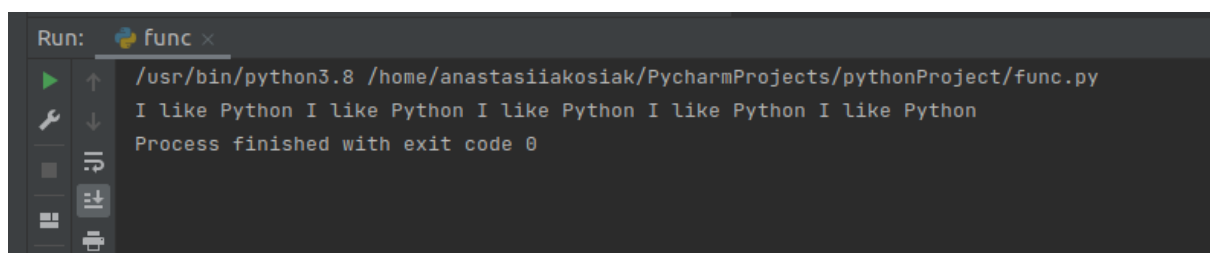
- `cycle(iterable)`: цей ітератор друкує усі значення в порядку, в якому вони були передані контейнеру. Він починає знову друкувати елементи з початку, коли всі елементи друкуються в циклічній манері.

```
import itertools

words = ['I', 'like ', 'Python']

iterators = itertools.cycle(words)

for i in range(10):
    print(next(iterators), end=" ")
```

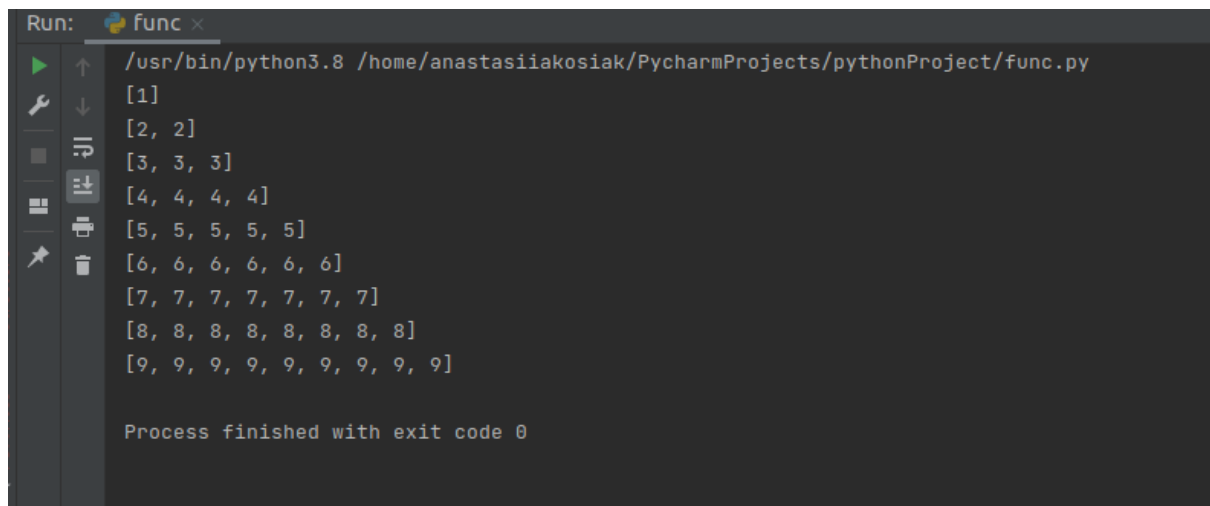


Скріншот №2. Приклад виконання коду із Listing №2.

- `repeat(val, num)`: цей ітератор повторно друкує переане знчення нескінечнну кількість разів. Якщо вказати `num`, то значення буде надруковано ту кількість разів.

```
import itertools

for i in range(1, 10):
    print(list(itertools.repeat(i, i)))
```



```
Run: func x
/usr/bin/python3.8 /home/anastasiakosiak/PycharmProjects/pythonProject/func.py
[1]
[2, 2]
[3, 3, 3]
[4, 4, 4, 4]
[5, 5, 5, 5, 5]
[6, 6, 6, 6, 6, 6]
[7, 7, 7, 7, 7, 7, 7]
[8, 8, 8, 8, 8, 8, 8, 8]
[9, 9, 9, 9, 9, 9, 9, 9, 9]

Process finished with exit code 0
```

Скріншот № 3. Результати виконання коду із Listing №3.

### Combinatoric iterator

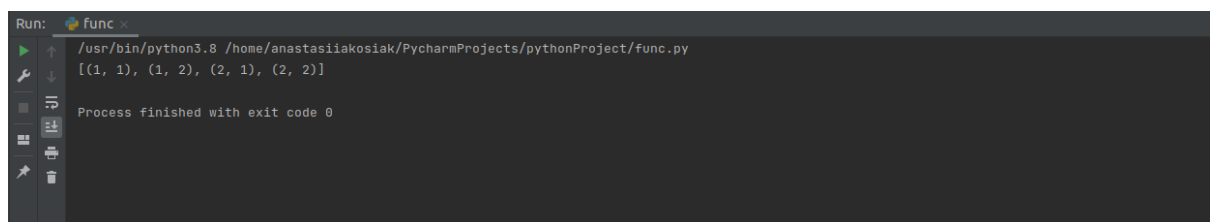
Комбінаторні ітератори використовуються, аби полегшити створення операцій комбінаторики, наприклад, комбінації, перестановки і добуток множин.

В Python є 4 типи комбінаторних ітераторів:

- `product()`: рахує декаторовий добуток множин. Повертає кортеж (tuple) в посортованому порядку.

```
from itertools import product

print(list(product([1, 2], repeat=2)))
```



```
Run: func x
/usr/bin/python3.8 /home/anastasiakosiak/PycharmProjects/pythonProject/func.py
[(1, 1), (1, 2), (2, 1), (2, 2)]

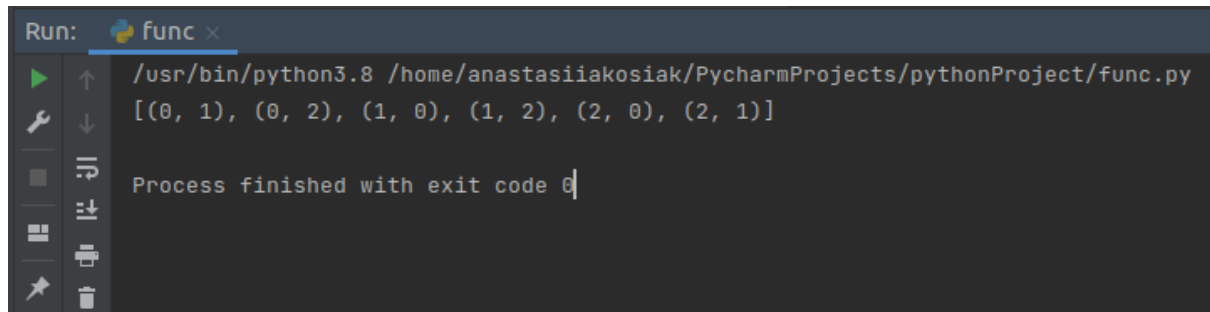
Process finished with exit code 0
```

Скріншот № 4. Результати виконання коду із listing №4

- `permutations()`: генерує усі можливі перестановки. Ця функція приймає iterable об'єкт та `group_size`.

```
from itertools import permutations
```

```
print(list(permutations(range(3), 2)))
```



The screenshot shows a Python IDE window titled 'Run: func x'. The command prompt displays the execution of a script: `/usr/bin/python3.8 /home/anastasiakosiak/PycharmProjects/pythonProject/func.py`. The output is a list of permutations: `[(0, 1), (0, 2), (1, 0), (1, 2), (2, 0), (2, 1)]`. Below the output, it states 'Process finished with exit code 0'.

Скріншот №5. Результат виконання коду із listing №5

- `combinations()`: цей ітератор друкує усі можливі комбінації без замін в посортованому вигляді.

```
from itertools import combinations  
  
print(list(combinations(range(7), 2)))
```

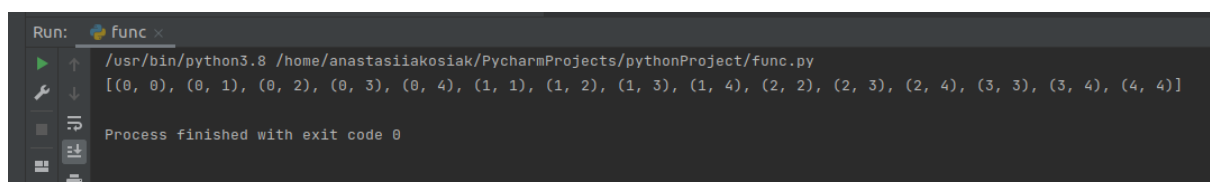


The screenshot shows a Python IDE window titled 'Run: func x'. The command prompt displays the execution of a script: `/usr/bin/python3.8 /home/anastasiakosiak/PycharmProjects/pythonProject/func.py`. The output is a list of combinations: `[(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 3), (2, 4), (2, 5), (2, 6), (3, 4), (3, 5), (3, 6), (4, 5), (4, 6), (5, 6)]`. Below the output, it states 'Process finished with exit code 0'.

Скріншот №6. Результат виокнання коду із listing №6

- `combinations_with_replacement()`: комбінації із замінами.

```
from itertools import combinations_with_replacement  
  
print(list(combinations_with_replacement(range(5), 2)))
```



The screenshot shows a Python IDE window titled 'Run: func x'. The command prompt displays the execution of a script: `/usr/bin/python3.8 /home/anastasiakosiak/PycharmProjects/pythonProject/func.py`. The output is a list of combinations with replacement: `[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (2, 3), (2, 4), (3, 3), (3, 4), (4, 4)]`. Below the output, it states 'Process finished with exit code 0'.

Скріншот №7. Результат виокнання коду із listing №7

## Terminating iterators

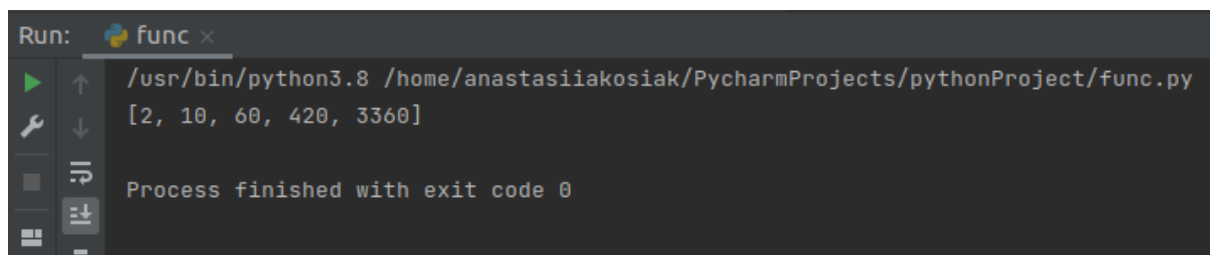
Termination ітератори дозволяють виконувати певні перетворення над послідовностями залежно від використаного методу.

Наведемо декілька різних типів terminating ітераторів:

- `accumulate(iter, func):`

```
import itertools
import operator

li = [2, 5, 6, 7, 8]
print(list(itertools.accumulate(li, operator.mul)))
```



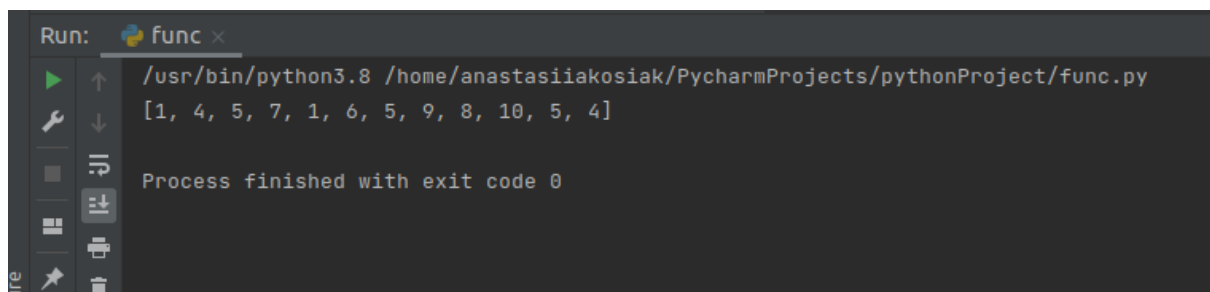
Скріншот №8. Результат виконання коду із listing №8

- `chain(iter1, iter2,...)`

```
import itertools

list1 = [1, 4, 5, 7]
list2 = [1, 6, 5, 9]
list3 = [8, 10, 5, 4]

print(list(itertools.chain(list1, list2, list3)))
```



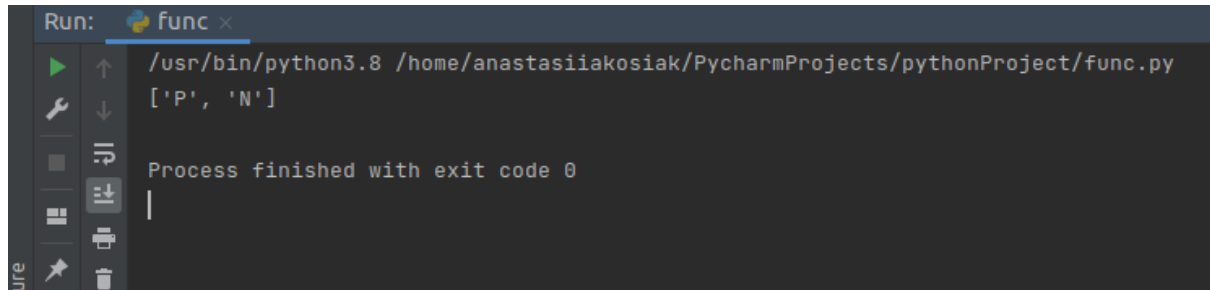
Скріншот №9. Результат виконання коду із listing #9.

- `compress(iter, selector)`



```
import itertools

print(list(itertools.compress('PYTHON', [1, 0, 0, 0, 0, 1])))
```



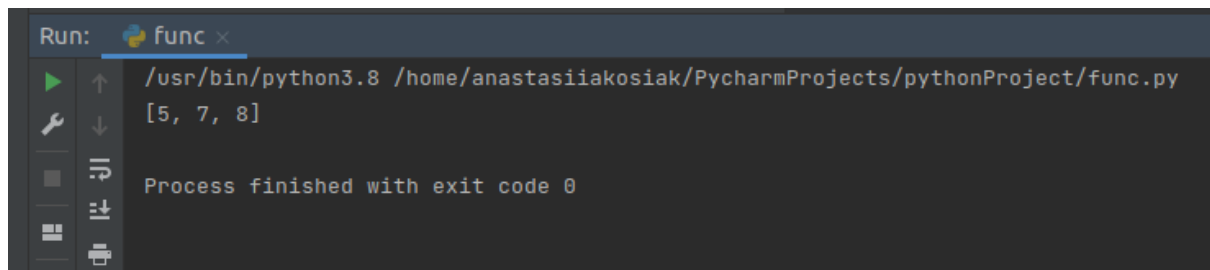
Скріншот №10. Результат виконання коду із listing #10.

- `dropwhile(func, seq)`

```
import itertools

li = [2, 4, 5, 7, 8]

print(list(itertools.dropwhile(lambda x: x % 2 == 0, li)))
```



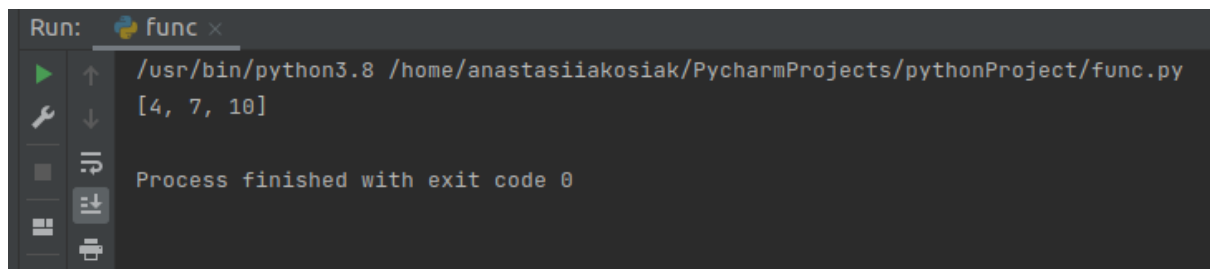
Скріншот №11. Результат виконання коду із listing #11.

- `islice(iterable, start, stop, step)`

```
import itertools

li = [2, 4, 5, 7, 8, 10, 20]

print(list(itertools.islice(li, 1, 6, 2)))
```



```
Run: func x
/usr/bin/python3.8 /home/anastasiakosiak/PycharmProjects/pythonProject/func.py
[4, 7, 10]
Process finished with exit code 0
```

Скріншот №12. Результат виконання коду із listing #12.

## Модуль functools

Модуль functools призначений для роботи із функціями вищого порядку.

Розглянемо основні функції functools.

- partial: partial функція може використовуватись для створення нової функції із частковим додаванням аргументів та ключових слів. Також partial може використовуватись для створення функції із різними налаштуваннями. Наприклад,

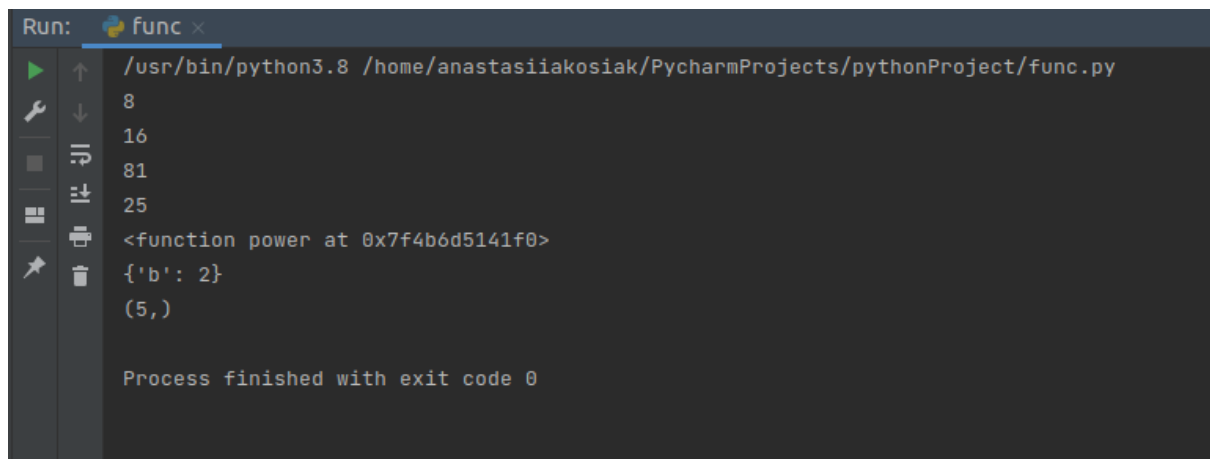
```
from functools import partial

def power(a, b):
    return a ** b

pow2 = partial(power, b=2)
pow4 = partial(power, b=4)
pow5 = partial(power, 5)

print(power(2, 3))
print(pow2(4))
print(pow4(3))
print(pow5(2))

print(pow2.func)
print(pow2.keywords)
print(pow5.args)
```



```
Run: func x
/usr/bin/python3.8 /home/anastasiakosiak/PycharmProjects/pythonProject/func.py
8
16
81
25
<function power at 0x7f4b6d5141f0>
{'b': 2}
(5,)

Process finished with exit code 0
```

Скріншот №13. Результат виконання коду із listing #13.

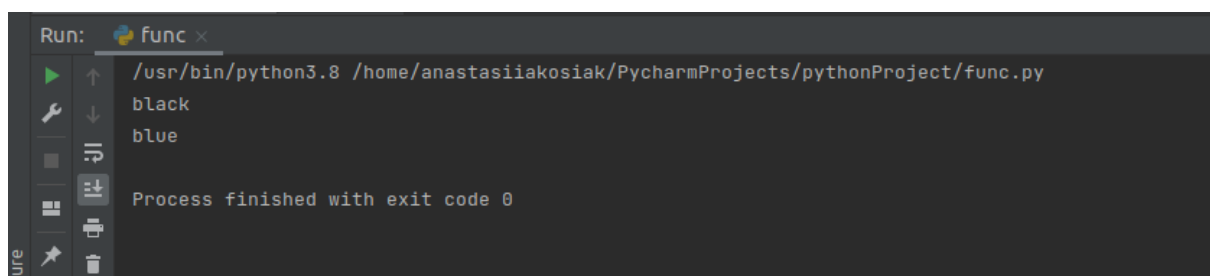
- `partialmethod(func, *args, **keywords)`: дефініція функції вже визначеної функції для певних аргументів. Але це лише дескриптор методу, а не функція для виклику.

```
from functools import partialmethod

class Test:
    def __init__(self):
        self.color = 'black'
    def _color(self, type):
        self.color = type

    set_blue = partialmethod(_color, type='blue')
    set_pink = partialmethod(_color, type='pink')
    set_brown = partialmethod(_color, type='brown')

obj = Test()
print(obj.color)
obj.set_blue()
print(obj.color)
```



```
Run: func x
/usr/bin/python3.8 /home/anastasiakosiak/PycharmProjects/pythonProject/func.py
black
blue

Process finished with exit code 0
```

Скріншот №14. Результат виконання коду із listing #14

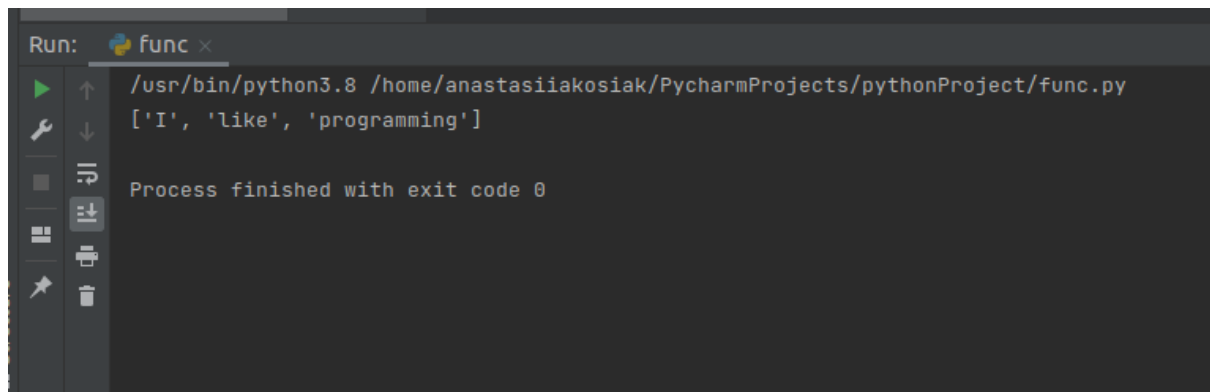
- `str_to_key`: перетворює функцію порівняння в key функцію. Функція порівняння має повертати 0, 1, -1 для різних умов. Вона може використовуватися як ключ у

sorted(), min(), max() функціях.

```
from functools import cmp_to_key

def cmp_fun(a, b):
    if a[-1] > b[-1]:
        return 1
    elif a[-1] < b[-1]:
        return -1
    else:
        return 0

li = ['like', 'I', 'programming']
sortedList = sorted(li, key=cmp_to_key(cmp_fun))
print(sortedList)
```



Скріншот №15. Результат виконання коду із listing #15.

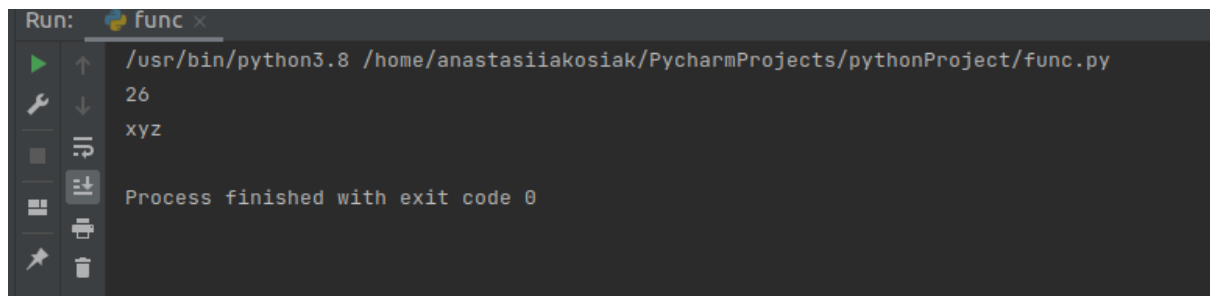
- reduce

```
from functools import reduce

li1 = [2, 4, 7, 9, 1, 3]
li1_sum = reduce(lambda a, b: a + b, li1)

li2 = ["abc", "xyz"]
li2_max = reduce(lambda a, b: a if a > b else b, li2)

print(li1_sum)
print(li2_max)
```



```
Run: func x
/usr/bin/python3.8 /home/anastasiakosiak/PycharmProjects/pythonProject/func.py
26
xyz
Process finished with exit code 0
```

Скріншот №16. Результат виконання коду із listing #16.

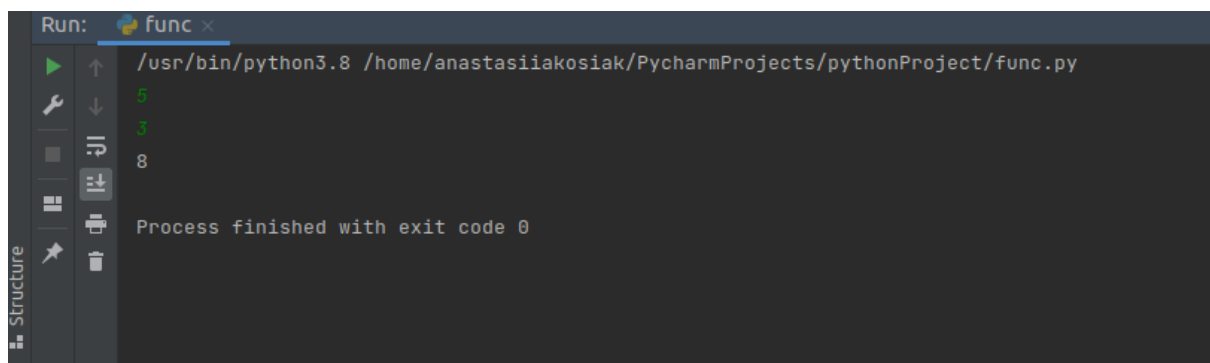
## Модуль operator

Модуль operator дозволяє виконувати різноманітні операції над двома числами.

- **add(x, y):**  $x + y$

```
import operator
x = int(input())
y = int(input())

res = operator.add(x, y)
print(res)
```



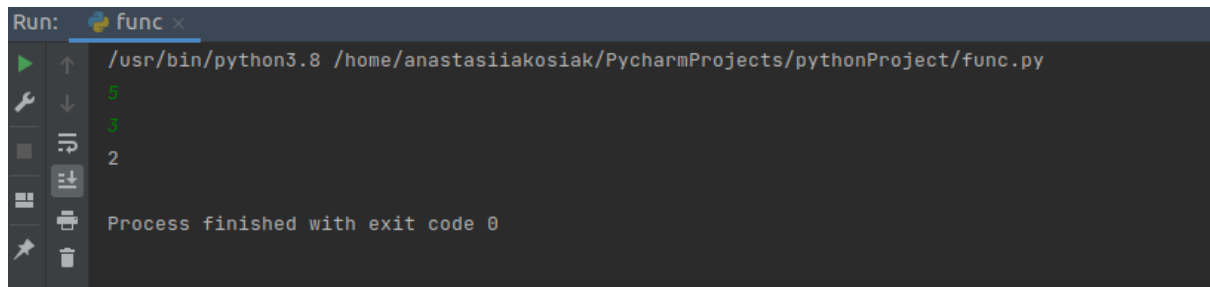
```
Run: func x
/usr/bin/python3.8 /home/anastasiakosiak/PycharmProjects/pythonProject/func.py
5
3
8
Process finished with exit code 0
```

Скріншот № 17. Результати виконання коду №3 із Listing №17

- **sub(x, y):**  $x - y$

```
import operator
x = int(input())
y = int(input())

res = operator.sub(x, y)
print(res)
```

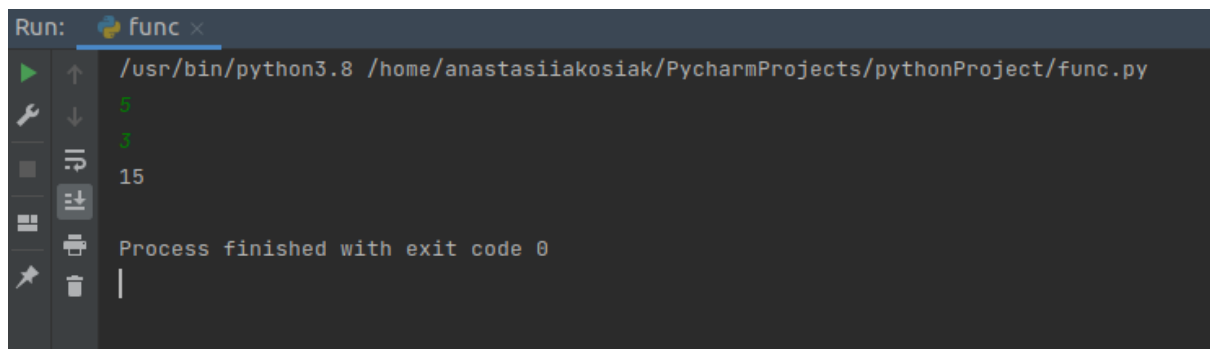


```
Run: func x
/usr/bin/python3.8 /home/anastasiakosiak/PycharmProjects/pythonProject/func.py
5
3
2
Process finished with exit code 0
```

Скріншот № 18. Результати виконання коду із Listing № 18

- **mul(x, y): x \* y**

```
# Import operator module
import operator
# Take two input numbers from user
x = int(input("Enter first integer number: "))
y = int(input("Enter second integer number: "))
# Multiply both input numbers
mulResult = operator.mul(x, y)
# Print result
print("Multiplication result of numbers given by you is: ", mulResult)
```



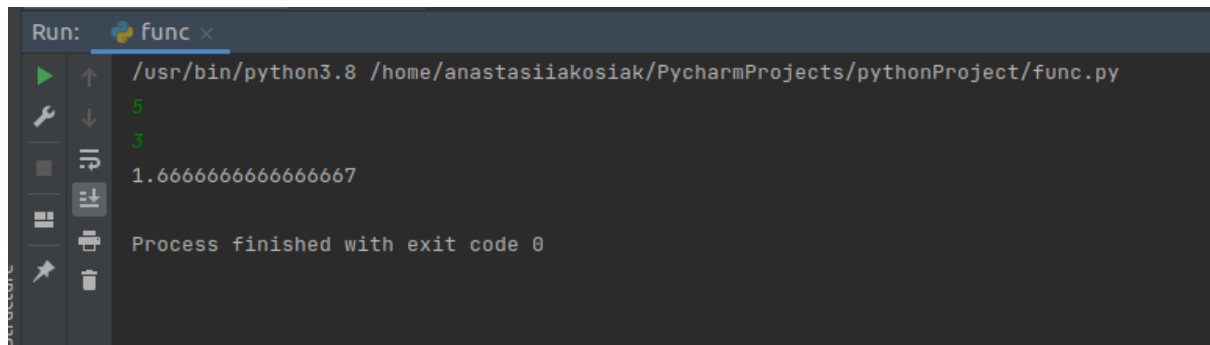
```
Run: func x
/usr/bin/python3.8 /home/anastasiakosiak/PycharmProjects/pythonProject/func.py
5
3
15
Process finished with exit code 0
|
```

Скріншот № 19. Результати виконання коду із Listing №19.

- **truediv(x, y): x % y**

```
import operator
x = int(input())
y = int(input())
res = operator.truediv(x, y)

print(res)
```



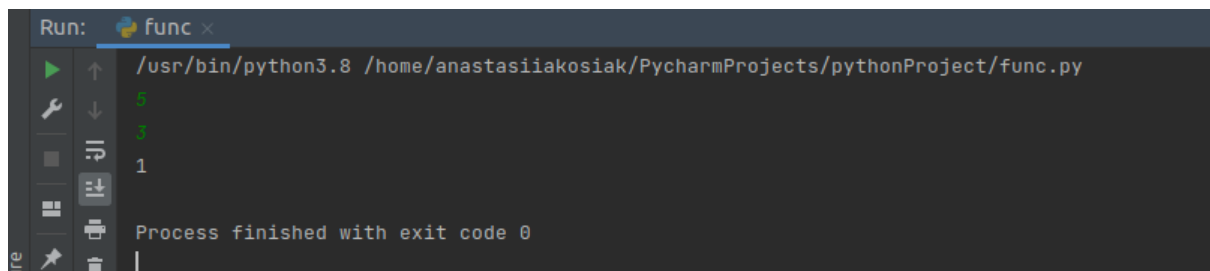
```
Run: func x
/usr/bin/python3.8 /home/anastasiakosiak/PycharmProjects/pythonProject/func.py
5
1.6666666666666667
Process finished with exit code 0
```

Скріншот № 20. Результати виконання коду із Listing №20.

- **floordiv(x, y):** ділення із округленням в меншу сторону

```
import operator
x = int(input())
y = int(input())

res = operator.floordiv(x, y)
print(res)
```



```
Run: func x
/usr/bin/python3.8 /home/anastasiakosiak/PycharmProjects/pythonProject/func.py
5
1
Process finished with exit code 0
```

Скріншот № 21. Результати виконання коду із Listing №21.

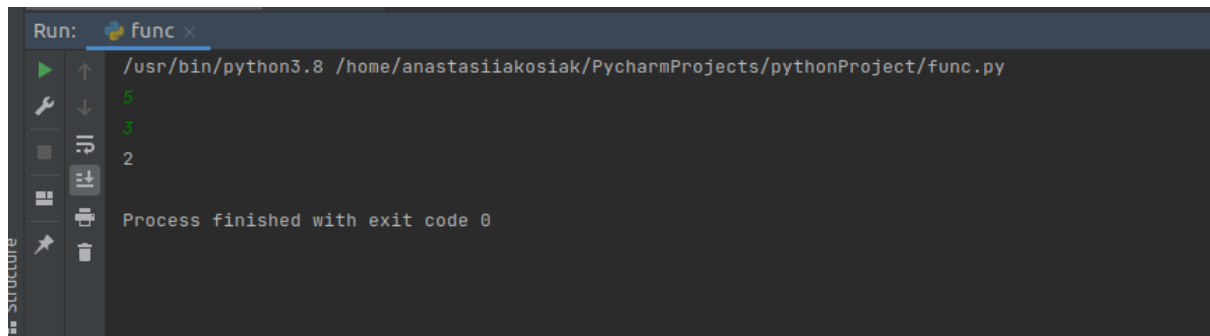
- **mod(x, y):**  $x \bmod y$

```
import operator

x = int(input())
y = int(input())

res = operator.mod(x, y)

print(res)
```

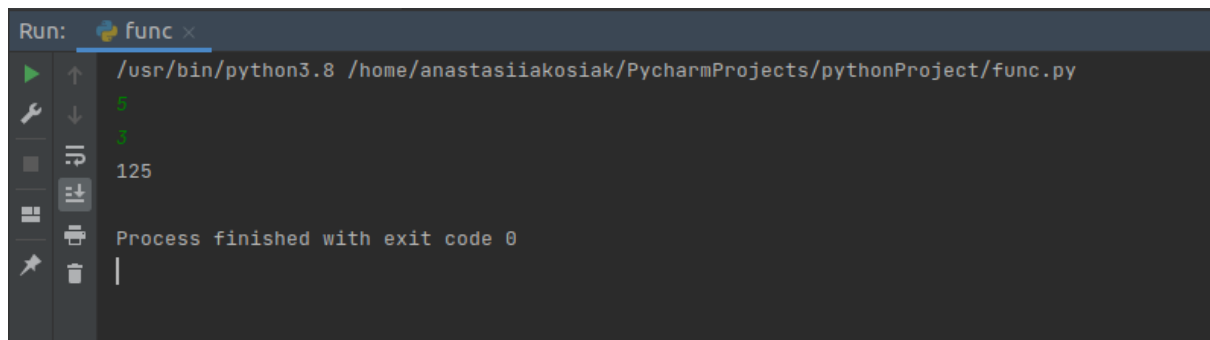


Скріншот № 22. Результати виконання коду із Listing №22.

- **pow(x, y):**  $x^y$

```
import operator
x = int(input())
y = int(input())

res = operator.pow(x, y)
print(res)
```



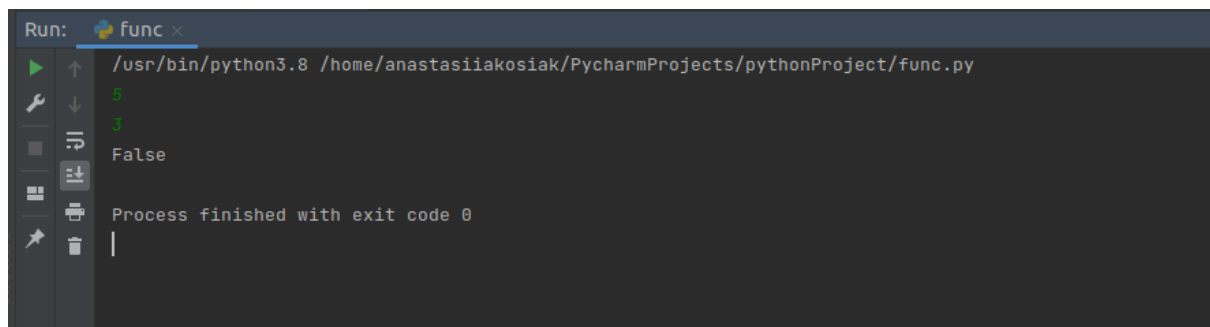
Скріншот № 23. Результати виконання коду із Listing №23.

- **lt(x, y):**  $x < y$

```
import operator
x = int(input())
y = int(input())

res = operator.lt(x, y)
print(res)
```





Скріншот №24. Результати виконання коду із Listing №24.

- **le(x, y):**  $x \leq y$

```
import operator

x = int(input())
y = int(input())

res = operator.le(x, y)
print(res)
```

- **gt(x, y):**  $x > y$

```
import operator

x = int(input())
y = int(input())

res = operator.gt(x, y)
print(res)
```

- **ge(x, y):**  $x \geq y$

```
import operator

x = int(input())
y = int(input())

res = operator.ge(x, y)
print(res)
```

- **eq(x, y):**  $x = y$

```
import operator
x = int(input())
y = int(input())

res = operator.eq(x, y)
print(res)
```

- **ne(x, y):  $x \neq y$**

```
import operator

x = int(input())
y = int(input())

res = operator.ne(x, y)
print(res)
```

---

## Висновки

Під час підготовки реферату я ознайомилась із поняттям функціонального програмування, зрозуміла основні принципи цієї парадигми, а також дізналась про реалізацію функціонального програмування в мові Python.

В Python існують декілька модулів, які дозволяють створювати програми в парадигмі функціонального програмування. Деякі з них було розглянуто в цьому рефераті, а саме: модуль `itertools`, модуль `functools`, а також модуль `operator`.

---

## Посилання на використані джерела

Офіційна документація Python: <https://docs.python.org/3/library/>