Wizualizacja danych - Wykład 5

Funkcje

Funkcje

```
def functionname( parameters ):
   "function_docstring"
   function_suite
   return [expression]
```

```
1  def printme(str):
2      """Funkcja wyświetlająca string"""
3      print(str)
4      return
5
6
7  printme("abc")
8  print(printme.__doc__)
```

```
abc
Funkcja wyświetlająca string
```

Przekazywanie przez referencję

```
 1 def changeme(lista):
 2     print("Przed zmianą: ", lista)
 3     lista[2] = 50
 4     print("Po zmianie: ", lista)
 5     return
 6
 7
 8 mylist = [10, 20, 30]
 9 changeme(mylist)
10 print("Poza funkcją: ", mylist)
```

```
Przed zmianą:  [10, 20, 30]
Po zmianie:  [10, 20, 50]
Poza funkcją:  [10, 20, 50]
```

```python
def changeme(lista):
    lista = [2, 3, 4]
    print("Wewnątrz funkcji: ", lista)
    return


lista = [10, 20, 30]
changeme(lista)
print("Poza funkcją: ", lista)
```

```
Wewnątrz funkcji:  [2, 3, 4]
Poza funkcją:  [10, 20, 30]
```

```
def changeme():
    global lista
    lista = [2, 3, 4]
    print("Wewnątrz funkcji: ", lista)
    return


changeme()
print("Poza funkcją: ", lista)
```

```
Wewnątrz funkcji:  [2, 3, 4]
Poza funkcją:  [2, 3, 4]
```

Obowiązkowy argument (ang. positional argument)

```
1 def printme(str):
2     print(str)
3     return
4
5
6 printme()
```

```
## TypeError: printme() missing 1 required
positional argument: 'str'
```

Keyword argument

```
1 def kwadrat(a):
2     return a*a
3
4
5 print(kwadrat(a=4))
```

```
16
```

Domyślny argument

```
1 def sumsub(a, b, c=0, d=0):
2     return a - b + c - d
3
4
5 print(sumsub(12, 4))
6 print(sumsub(3, 4, 5, 7))
```

```
8
-3
```

```python
def srednia(first, *values):
    return (first + sum(values)) / (1 + len(values))


print(srednia(2, 3, 4, 6))
print(srednia(45))
```

```
3.75
45.0
```

```python
def f(**kwargs):
    print(kwargs)


f()
f(pl="Polish", en="English")
```

```
{}
{'pl': 'Polish', 'en': 'English'}
```

Inne symbole

- symbol / oznacza, że wcześniejsze argumenty są pozycyjne
- symbol * oznacza, że późniejszej argumenty są typu keyword

```
1 def f(a, b, /, c, d, *, e, f):
2     print(a, b, c, d, e, f)
3
4
5 f(10, 20, 30, d=40, e=50, f=60)
```

```
10 20 30 40 50 60
```

Funkcje matematyczne

Link do dokumentacji

https://docs.python.org/3/library/math.html

```
1 import math
2
3 a=0
4 b=math.sin(2*math.pi)
5 print(b)
6 print(math.isclose(a,b, rel_tol=1e-09, abs_tol=1e-09))
```

```
-2.4492935982947064e-16
True
```

Wyrażenia lambda

Skąd nazwa? https://pl.wikipedia.org/wiki/Rachunek_lambda

Wyrażenie

```
1 def identity(x):
2     return x
```

zapisujemy jako

```
lambda x: x
```

Jeśli uruchomimy to w konsoli, to możemy wywołać

```
1 _(5)
```

To tzw. przykład funkcji anonimowej.

By uruchomić to też w skrypcie, to potrzebujemy argumentu:

```
1 print((lambda x: x + 1)(2))
```

lub

```
1 add_one = lambda x: x + 1
2 print(add_one(2))
```

Możemy też tworzyć złożenia funkcji (funkcje wyższych rzędów):

```
1 high_ord_func = lambda x, func: x + func(x)
2 print(high_ord_func(2, lambda x: x * x))
3 print(high_ord_func(2, lambda x: x + 3))
```

```
6
7
```

Możemy również skorzystać z różnych opcji argumentów jak dla funkcji:

```
1 print((lambda x, y, z: x + y + z)(1, 2, 3))
2 print((lambda x, y, z=3: x + y + z)(1, 2))
3 print((lambda x, y, z=3: x + y + z)(1, y=2))
4 print((lambda *args: sum(args))(1,2,3))
5 print((lambda **kwargs: sum(kwargs.values()))(one=1, two=2, three=3))
6 print((lambda x, *, y=0, z=0: x + y + z)(1, y=2, z=3))
```

```
6
6
6
6
6
6
```

Typ

```
1 string = 'abc'
2 print(lambda string: string)
```

```
<function <lambda> at 0x0000017D75EF1120>
```

Różnice względem zwykłej funkcji:

```
1  def cube(y):
2      print(f"Finding cube of number:{y}")
3      return y * y * y
4
5  lambda_cube = lambda num: num ** 3
6  print("invoking function defined with def keyword:")
7  print(cube(30))
8  print("invoking lambda function:", lambda_cube(30))
```

```
invoking function defined with def keyword:
Finding cube of number:30
27000
invoking lambda function: 27000
```

Przykłady praktyczne

```
1  r = lambda a: a + 15
2  print(r(10))
3  r = lambda x, y: x * y
4  print(r(12, 4))
```

```
25
48
```

```
subject_marks = [('English', 88), ('Science', 90), ('Maths', 97), ('Social
print("Original list of tuples:")
print(subject_marks)
subject_marks.sort(key=lambda x: x[1])
print("\nSorting the List of Tuples:")
print(subject_marks)
```

```
Original list of tuples:
[('English', 88), ('Science', 90), ('Maths', 97), ('Social sciences', 82)]

Sorting the List of Tuples:
[('Social sciences', 82), ('English', 88), ('Science', 90), ('Maths', 97)]
```

```python
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Original list of integers:")
print(nums)
print("\nEven numbers from the said list:")
even_nums = list(filter(lambda x: x % 2 == 0, nums))
print(even_nums)
print("\nOdd numbers from the said list:")
odd_nums = list(filter(lambda x: x % 2 != 0, nums))
print(odd_nums)
```

```
Original list of integers:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Even numbers from the said list:
[2, 4, 6, 8, 10]

Odd numbers from the said list:
[1, 3, 5, 7, 9]
```

```
array_nums1 = [1, 2, 3, 5, 7, 8, 9, 10]
array_nums2 = [1, 2, 4, 8, 9]
print("Original arrays:")
print(array_nums1)
print(array_nums2)
result = list(filter(lambda x: x in array_nums1, array_nums2))
print("\nIntersection of the said arrays: ", result)
```

```
Original arrays:
[1, 2, 3, 5, 7, 8, 9, 10]
[1, 2, 4, 8, 9]

Intersection of the said arrays:  [1, 2, 8, 9]
```

“Obiektowość”

Programowanie obiektowe w Pythonie

```
class Employee:
    """Common base class for all employees"""
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

    def displayCount(self):
        print("Total Employee %d" % Employee.empCount)

    def displayEmployee(self):
        print("Name : ", self.name, ", Salary: ",
              self.salary)


emp1 = Employee("John", 2000)
```

```
Name :  John , Salary:  2000
Name :  Anna , Salary:  5000
```