Podstawy programowania w języku Python

Dzień 4

Agenda

Dzień 4

- jeszcze trochę o listach – kopiowanie, referencja
- definiowanie funkcji – argumenty, return, zakres zmiennych
- import i modułu
- docstring PEP257

Listy są typami referencyjnymi

jeśli przypiszemy listę do innej zmiennej to tak naprawdę przypiszemy **adres** w pamięci do listy

```
nowa_lista = lista.copy()
nowa_lista = list(stara_lista)
nowa_lista = stara_lista[:]
```

Listy są typami referencyjnymi

do głębokiego kopiowania (kopiowanie wszystkiego jako wartość) używamy modułu ***copy*** i metody ***deepcopy()***

```
import copy
nowa_lista = copy.deepcopy(stara_lista)
```

Tworzenie funkcji

```
def funkcja():
    # kod funkcji który zostanie wykonany przy jej wywołaniu


def do_nothing():
    pass

do_nothing()                # nic nie zrobiła, czyli tak jak chcieliśmy :)
wynik = do_nothing          # funkcja jest obiektem (jak wszystko w pythonie)
wynik()                     # więc możemy przypisać ją do zmiennej i ją wykonać
```

Tworzenie funkcji

```
1  give_square(35)
2
3  def give_square(x):
4      print(x**2)
5
```

NIE

```
1  def give_square(x):
2      print(x**2)
3
4  give_square(35)
5
```

TAK

```
give_square(35)
NameError: name 'give_square' is not defined
```

Argumenty funkcji

```
def do_nothing():                          # brak argumentów
    pass

def do_nothing(x):                         # jeden argument
    pass

def do_nothing(x, y, z):                   # 3 argumenty
    pass

def do_nothing(*args, **kwargs):           # dowolna ilość argumentów
    pass
```

Argumenty domyśle

```
def do_nothing(x, y=10):
    pass

def do_nothing(x, y, imie="Ola", wiek="18"):
    Pass

def do_nothing(x, y, imie="Ola", wiek="18", miasto):
    pass

def do_nothing(y=10):
    pass
```

- argumenty domyślne **muszą** być po argumentach wymaganych
- argument domyślny jest sprawdzany **tylko przy pierwszym** wywołaniu funkcji – uwaga na typy referencyjne

Wywoływanie

```
def do_nothing(x, y, imie="Ola", wiek=18):
    pass

do_nothing(1)
do_nothing(1, 2, "Iza")
do_nothing(1, 2, "Iza", 22)
do_nothing(1, 2, imie="Iza", 22)
do_nothing(1, 2, imie="Iza", wiek=22)
do_nothing(1, 2, wiek=22, imie="Iza")
```

return

Funkcja może wykonywać coś w sobie:

```
def wyswietl(x):
    print(x)
```

lub zwracać dowolne wartości

```
def dodaj(x, y):
    suma = x + y
    return suma

wynik = dodaj(2, 3)
```

Atrybuty domyśle - referencja

```
def dodaj_imie(imie, imiona=[]):
    imiona.append(imie)
    return imiona

>>> print(dodaj_imie("Iza"))
["Iza"]
>>> print(dodaj_imie("Kasia"))
["Iza", "Kasia"]
>>> print(dodaj_imie("Gosia"))
["Iza", "Kasia", "Gosia"]
```

```
def dodaj_imie(imie, imiona=None):
    if(imiona == None):
        imiona = []
    imiona.append(imie)
    return imiona

>>> print(dodaj_imie("Iza"))
["Iza"]
>>> print(dodaj_imie("Kasia"))
["Kasia"]
>>> print(dodaj_imie("Gosia"))
["Gosia"]
```

Zakres zmiennych (scope)

Zmienne lokalne funkcji są do wykorzystania tylko w tej funkcji (i głębiej).
Zmiany za pomocą **global** i **nonlocal**

```
imie = ”Jola”
def zmien_imie():
    imie = ”Teresa”

>>> print(imie)
???
>>>zmien_imie()
>>>print(imie)
???
```

docstring - PEP257

```
def do_nothing(x, y):
    """Does absolutely nothig"""
    pass

def give_square(x):
    """Return square of given number

    (number) → number
    """
    return x**2

give_square.__doc__
```

DZIĘKUJĘ