Laboratorium 4 (Programowanie 2 – 410-KS1-2PRO3)
Kierunek kognitywistyka i komunikacja
Funkcje wieloargumentowe i podawanie oraz przekazywanie ich argumentów c.d.
(położenia argumentów opcjonalnych; dowolna liczba argumentów nazwanych).
Funkcje jako argumenty innych funkcji.
Zmienne lokalne i globalne w funkcjach.
Funkcje anonimowe – wyrażenia *lambda*.

Funkcje wieloargumentowe cd.

1. **(dowolna liczba argumentów nazwanych) Napisz funkcję (nazwa funkcji *moja_funkcja_1*), która będzie miała dowolną ilość argumentów nazwanych (wykorzystaj ***kwargs*) i będzie na konsoli wypisywała wszystkie podane argumenty.**
2. **Wykorzystując operatory specjalne */, *, *args oraz **kwargs argumentów funkcji wyznacz dopuszczalne ich kombinacje. Przyjmij założenie, że mogą występować wszystkie typy argumentów tzn. tylko pozycyjne, pozycyjne i nazwane oraz tylko nazwane. Jako ciało funkcji użyj *pass* lub funkcja ma wyświetlać na konsoli typy argumentów i ich wartości.**
3. **(położenie argumenty opcjonalnych) Wykorzystując operatory specjalne */, *, *args oraz **kwargs argumentów funkcji wyznacz dopuszczalne ich kombinacje zawierające argumenty opcjonalne (*None*). Przyjmij założenie, że mogą występować wszystkie typy argumentów tzn. tylko pozycyjne, pozycyjne i nazwane oraz tylko nazwane. Funkcje mają wyświetlać na konsoli typy argumentów i ich wartości.**

Funkcje jako argumenty innych funkcji.

1. **Przeanalizuj następujący kod:**

```
def funkcja2(tekst):
    return tekst.upper()

def funkcja3(tekst):
    return tekst.lower()

def funkcja1(funkcja, tekst=None):
    if tekst is None:
        tekst = "To Ja!"
    res = funkcja(tekst)
    print(res)

funkcja1(funkcja2, "Puk puk!")
funkcja1(funkcja3, 'Kto tam?')
```

2. **W oparciu o powyższy przykład stwórz dwie funkcje, z których jedna przyjmuje jako argument drugą z nich. Przetestuj je.**

Zmienne lokalne i globalne w funkcjach.

1. **(zmienne wolne i lokalne) Przeanalizuj następujący kod:**

```
def f():
    print(s)
```

```
s = "Python"
f()

def f():
    s = "Perl"
    print(s)

s = "Python"
f()
print(s)
```

Co zaobserwowałaś/łeś?

2. **Napisz dowolną funkcję, która będzie używała zmiennej lokalnej i przetestuj ją**
3. **(zmienne lokalne i globalne) Przeanalizuj następujący kod:**

```
def f():
    global s
    print(s)
    s = "dog"
    print(s)

def g():
    s = "snake"
    print(s)

s = "cat"
f()
print(s)
g()
print(s)
```

Co zaobserwowałaś/łeś?

4. **Napisz dowolną funkcję, która będzie używała zmiennej globalnej i przetestuj ją**

Funkcje anonimowe – *wyrażenia lambda*.

1. **Przeanalizuj powyższy fragment kodu:**

```
print((lambda : print('Cześć'))())
print((lambda x: x**2)(2))
print((lambda x, y: x**y)(2, 3))
```

Jakim funkcjom standardowym one odpowiadają? Zapisz te funkcje

2. **Wykorzystując zarówno standardową definicję funkcji (użyj pętli *for*), jak i *wyrażenie lambda* napisz funkcję, która przyjmuje dowolną liczbę argumentów pozycyjnych i wyznacza ich sumę.**
3. **(sposoby podawania argumentów w wyrażeniach lambda) Napisz funkcje anonimowe np. liczącą sumę liczb, dla których argumenty będą podawane jako jeden typ tzn. jako argumenty pozycyjne; nazwane; domyślne; nieokreślona liczba argumentów pozycyjnych; nieokreślona liczba argumentów nazwanych.**
4. **(złożenie wyrażeń lambda) Przeanalizuj powyższy fragment kodu:**

```
print((lambda x, y, lmbfunc: x*lmbfunc(y))(10, 2, lambda y: y*y))
```

Jakie obliczenia są wykonywane?

5. **(użycie wyrażeń lambda z if … else) Przeanalizuj powyższy fragment kodu:**

```
print((lambda x: 'OK' if x % 2 == 0 else 'Bad')(3))
```

6. **Korzystając z poprzedniego przykładu napisz funkcję anonimową, która po podaniu liczby wypiszę na konsoli słowo *dobrze* gdy liczba będzie co najmniej równa 4, *średnio*, gdy liczba będzie co najmniej równa 3 oraz *źle* w pozostałych przypadkach.**

7. **(użycie funkcji anonimowej z *reduce, filter, map*) Przeanalizuj powyższy fragment kodu:**

```
from functools import reduce
moja_lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
suma = reduce((lambda x, y: x+y), moja_lista)
print(suma)

nowa_lista_f = list(filter(lambda x: (x % 2 == 0), moja_lista))
print(nowa_lista_f)

nowa_lista_m = list(map(lambda x: x % 2, moja_lista))
print(nowa_lista_m)
```

8. **Rozwiąż zadanie 3 biorąc funkcję wyznaczająca iloczyn podanej dowolnej ilości liczb wykorzystując raz funkcję *reduce*, a raz używając metody *prod* z pakietu NumPy. W obu wypadkach należy użyć funkcji anonimowych.**
9. **(fabryki funkcji z wyrażeniami lambda) Przenalizuj powyższy kod:**

```
multiplier = (lambda x: (lambda y: x*y))

doubler = multiplier(2)
print(doubler(10))

tripler = multiplier(3)
print(tripler(10))
```

   Co możesz o nim powiedzieć?
10. **W oparciu o poprzednie zadanie wykorzystując wyłącznie *wyrażenia lambda* napisz funkcję będącą fabryką funkcji, która będzie podnosiła do potęgi naturalnej. Przy jej pomocy zdefiniuj funkcje liczące drugą, trzecią i czwartą potęgę liczby. Przetestuj je.**