Lab 4 – funkcje

Najprostsza definicja funkcji ma posta:

```
def my_function():
    print("hello world")
```

Nie posiada ona parametrów a jej wywołanie wygląda następująco:

```
my_function()
```

```
hello world
```

Argumenty i parametry:

Parametr jest zmienną - informacją jaką przekazania wymaga funkcja aby mogła się wykonać. Argumentem jest konkretna wartość do niej przekazana.

Definicja funkcji witająca użytkownika oraz wyświetlająca jego imię:

```
def my_function_with_parameter(your_name):
    print(f"hello {your_name}")
```

Oraz wywołanie z argumentem:

```
my_function_with_parameter("Alice")
```

```
hello Alice
```

Prosta funkcja sprawdzająca czy liczba jest liczbą pierwszą (od cyfry 2):

```
def prime(number):
    is_prime = True;
    for i in range(2, number):
        if number % i == 0:
            is_prime = False
            break
    if is_prime:
        print(f"{number} is prime")
    else:
        print(f"{number} is not prime")
```

Argumenty pozycyjne:

Stosowanie argumentów pozycyjnych polega na podawaniu wartości argumentów w kolejności ich występowania na liście deklaracji parametrów w definicji metody.

```
def my_function_with_parameters(your_name,your_surname):
    print(f"Hello to: \nname:{your_name}\nsurname:{your_surname}")
```

```
my_function_with_parameters("Alice","Novak")
my_function_with_parameters("Novak","Alice")
```

W obu przypadkach funkcja zadziała, jednak w drugim pomylimy imię z nazwiskiem, ponieważ kolejność podawanych argumentów ma znaczenie:

```
Hello to:
name:Alice
surname:Novak

Hello to:
name:Novak
surname:Alice
```

Argumenty w postaci słów kluczowych:
Kolejność nie ma znaczenia, za to trzeba podawać nazwy parametrów przy wywoływaniu funkcji:

```
my_function_with_parameters(your_surname="Novak",
your_name="Alice")
```

```
Hello to:
name:Alice
surname:Novak
```

Funkcja nie zmieniła się! Zmienił się sposób wywołania.

Wartości domyślne:

W Pythonie parametry funkcji mogą posiadać wartości domyślne. Jeżeli funkcja zostanie wywołana bez podania argumentu, zostanie użyta wartość domyślna danego parametru.

```
def my_function_default(your_name="John",your_surname="Doe"):
    print(f"Hello from default to: \nname:{your_name}\nsurname:
{your_surname}")
```

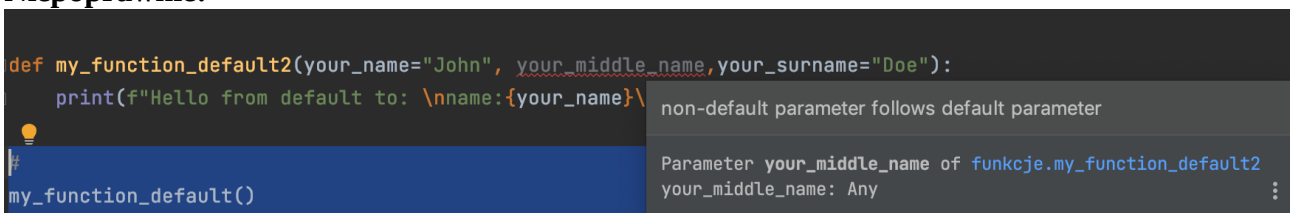
Wywołania mogą wyglądać np. tak:

```
my_function_default()
my_function_default(your_name="Alice")
my_function_default("Alice", "Novak")
my_function_default(your_name="Alice", your_surname="Novak")
```

```
Hello from default to:
name:John
surname:Doe
Hello from default to:
name:Alice
surname:Doe
Hello from default to:
name:Alice
surname:Novak,
Hello from default to:
```

name:Alice
surname:Novak

Gdyby jeden z parametrów nie był domyślny ważna jest kolejność występowania :
Niepoprawnie:



Dobrze:

```
def my_function_default2(your_middle_name,your_name="John",your_surname="Doe"):
    print(f"Hello from default to: \nname:{your_name}\nsurname:{your_surname}")

def my_function_default2(your_middle_name,your_name="John",your_surname="Doe"):
    print(f"Hello from default to: \nname:{your_name}\nsurname:{your_surname}")
```

Wartości opcjonalne:

Niech tym razem drugie imię - middle name będzie opcjonalne – nie każdy posiada drugie imię. Jeżeli ktoś nie posiada drugiego imienia przywitamy go posługując się tylko imieniem i nazwiskiem:

```
def hello(name, surname, middle_name=''):
    person_data = name
    if middle_name:
        person_data += " " + middle_name
    person_data += " " + surname
    print("Hello " + person_data + " !!")


hello("John", "Doe")
hello("John", "Doe", "Robert")
```

Hello John Doe !!
Hello John Robert Doe !!

Funkcja może zwrócić wynik działania zamiast wyświetlać coś na ekranie:
#

```
def hello_return(name="no", surname="name", middle_name=''):
    person_data = name
    if middle_name:
        person_data += " " + middle_name
    person_data += " " + surname
    return person_data

print("Hello "+hello_return()+" !")
```

Domyślne wartości to „no”, „name”, oraz ‘’.
Zwrócony został łańcuch znaków „no name” a następnie użyty w funkcji print():
Hello no name !

Jako parametr, możemy przekazać listę, zmiany wprowadzone przez funkcję w liście są trwałe:

```
def hello_list(names):
    for i in range(0, len(names)):
        names[i] = "hello " + names[i]

names = ["Alice", "Bob"]

hello_list(names)
print(names)
```

['hello Alice', 'hello Bob']
…

Aby nie zmieniać listy na stałe można pracować na kopii listy: (w przykładzie kopiowanie płytkie, dobre dla typow prostych)

```
names = ["Alice", "Bob"]
hello_list(names[:])
print(names)
```

```
list_of_names = ["Alice", "Bob"]
def hello_list_copy(names):
    for i in range(0, len(names)):
        names[i] = "hello " + names[i]
    return(names)
print("COPY: " + str(hello_list_copy(list_of_names[:])))
print("ORIGINAL: " + str(list_of_names))
```

COPY: ['hello Alice', 'hello Bob']
ORIGINAL: ['Alice', 'Bob']

Nie będzie też można zmienić krotki:
Krotka (ang. tuple) to uporządkowana i niezmienna kolekcja elementów.
)

```
lista_krotka = ("a", "b")
print("krotka:" + str(lista_krotka))
hello_list_copy(lista_krotka)
```

TypeError: 'tuple' object does not support item assignment

Wiele argumentów:

Do funkcji można też przekazać nieznaną ilość argumentów:

```
def hello_multiply(*people,lang):
    greet="Hello "
    if lang == "PL":
        greet="Witaj "
```

```
    if lang == "FR":
        greet="Bonjour "
    for p in people:
        print(greet+p)
hello_multiply("Alice","Bob","Kenny",lang="PL")
```

Witaj Alice
Witaj Bob
Witaj Kenny

Zadania:

1. Napisz funkcję obliczającą i zwracającą ilość potrzebnych opakowań paneli w danym pomieszczeniu, zakładając prostokątną podłogę i prostokątne panele. Dane wejściowe to długość i szerokość podłogi. (do powierzchni pomieszczenia należy dodać 10%) długość i szerokość panela oraz ilość paneli w opakowaniu. (10%)

np.:

```
print("Potrzeba : " + str(panel_calc(4, 4, 0.20, 1, 10)))
```

Potrzeba : 9

2. Napisz funkcję sprawdzającą czy podane liczby są liczbami pierwszymi w szybszy sposób niż w przykładzie. Do funkcji można przekazać dowolną liczbę argumentów (liczby). Liczby 0 i 1 nie są liczbami pierwszymi. (10%)

np.:

```
prime(0, 1, 2, 3, 4, 5)
```

0 is not prime
1 is not prime
2 is prime number
3 is prime number
4 is not prime
5 is prime number

3. Napisz funkcję szyfrującą wiadomość szyfrem cezara. Dla ułatwienia należy przekształcić wiadomość tak aby zawierała tylko wielkie lub małe litery.

Funkcja przyjmuje:

wiadomość – tekst do zaszyfrowania,

klucz – liczbę o ile należy przesunąć litery w alfabecie

oraz zwraca zaszyfrowaną wiadomość w formie łańcucha znaków -string. (40%)

Funkcja szyfruje tylko litery – inne znaki wstawia do końcowej zaszyfrowanej wiadomości bez zmian(10%)

Funkcja rozwiązuje problem klucza przesuwającego litery poza zakres tablicy z alfabetem oraz problem podania klucza o dowolnej wielkości(20%).

Funkcja opcjonalnie przyjmuje dowolny alfabet. Domyślnie używa angielskiego(10%).

Przykład:

```
enc = caesar_cipher(data, 5)
```

zamieni:

The Project Gutenberg eBook of Alice’s Adventures in Wonderland, by Lewis Carroll
na:

```
ymj uwtojhy lzyjsgjwl jgttp tk fqnhj’x fiajsyzwjx ns btsijwqfsi,
gd qjbnx hfwwtqq
```

```
enc = caesar_cipher(data, 3,["a","B"])
```

Zamieni wszystkie „a” na duże „B” ze względu na podany alfabet:

```
the project gutenberg ebook of Blice’s Bdventures in wonderlBnd,
by lewis cBrroll
```