Funkcje (definiowanie i wywołanie)

Funkcja jest blokiem kodu, który wykonuje określone zadanie, gdy funkcja ta jest wywoływana. Możesz użyć funkcji po to, aby ten sam kod mógł być wielokrotnie użyty, lepiej zorganizowany i bardziej czytelny. Funkcje mogą mieć parametry i zwracać wartości.

W **Pythonie** istnieją co najmniej cztery podstawowe typy funkcji:

- funkcje wbudowane, które są integralną częścią Pythona (jak na przykład funkcja **print**()). *https://docs.python.org/3/library/functions.html*
- te, które pochodzą z preinstalowanych modułów,
- funkcje zdefiniowane przez użytkownika, które są pisane przez użytkowników dla użytkowników,
- funkcje lambda

1. Definiowanie funkcji i przekazywanie argumentów

Możesz zdefiniować własną funkcję za pomocą słowa kluczowego **def** oraz następującej składni:

```
def nazwa_funkcji(opcjonalne parametry):
    # ciało funkcji
```

Możesz zdefiniować funkcję, która nie przyjmuje żadnych argumentów, np.:

```
def wiadomosc():        # definiowanie funkcji
    print("Cześć")      # ciało funkcji

wiadomosc()             # wywołanie funkcji
```

Możesz zdefiniować funkcję, która przyjmuje argumenty

```
# 1 parametr
def czesc(imie):              # definiowanie funkcji
    print("Cześć,", imię)     # ciało funkcji

imie = input("Podaj swoje imię: ")
czesc(imie)                   # wywołanie funkcji


# 2 parametry
def czesc_wszystkim(imie_1, imie_2):
    print("Cześć,", imie_2)
    print("Cześć,", imie_1)

czesc_wszystkim("Sebastian", "Konrad")
```

Możesz przekazać argumenty funkcji przy użyciu następujących sposobów:

- przekazywanie argumentów pozycyjnych, gdzie znaczenie ma kolejność przekazywania argumentów (przykład 1),
- przekazywanie słów kluczowych, gdzie nie ma znaczenia kolejność przekazywania argumentów (przykład 2),

połączenie powyższych dwóch sposobów przekazywania argumentów (przykład 3).

przykład 1

```
def subtra(a, b):
    print(a - b)


subtra(5, 2)    # daje na wyjściu: 3
subtra(2, 5)    # daje na wyjściu: -3
```

przykład 2

```
def subtra(a, b):
    print(a - b)

subtra(a=5, b=2)    # daje na wyjściu: 3
subtra(b=2, a=5)    # daje na wyjściu: 3
```

Ważne jest, aby pamiętać, że argumenty pozycyjne nie mogą występować po argumentach słów kluczowych. Dlatego jeśli spróbujesz uruchomić następujący fragment kodu:

```
def subtra(a, b):
    print(a - b)

subtra(5, b=2)    # daje na wyjściu: 3
subtra(a=5, 2)    # Syntax Error
```

Python nie pozwoli ci tego zrobić, sygnalizując błąd składni SyntaxError.

Możesz użyć techniki przekazywania argumentów słów kluczowych do predefiniowania wartości dla danego argumentu:

```
def dane (imie, nazwisko="Kowalski"):
    print(imie, nazwisko)

dane ("Andrzej")              # daje na wyjściu: Andrzej Kowalski
dane ("Baśka", "Nowak")       # daje na wyjściu: Baśka Nowak (
                              argument słowa kluczowego zastąpiony
                              przez "Nowak")
```

Zadanie 1

Przeanalizuj i sprawdź poniższe fragmenty kodu

```
def intro(a="James Bond", b="Bond"):
    print("My name is", b + ".", a + ".")

intro()


def intro(a="James Bond", b="Bond"):

    print("My name is", b + ".", a + ".")


intro(b="Sean Connery")
```

2. Zwracane wartości

Możesz używać słowa kluczowego **return**, aby nakazać funkcji zwrócenie pewnych wartości. Instrukcja return kończy działanie funkcji, np.:

przykład 1

```
def mnozenie(a, b):
    return a * b


print(mnozenie(3, 4))    # daje na wyjściu: 12
```

przykład 2

```
def mnozenie(a, b):
    return

print(mnozenie(3, 4))    # daje na wyjściu: None
```

Wynik funkcji można łatwo przypisać do zmiennej, np.:

```
def zyczenia():
    return "Wszystkiego Najlepszego!"

z = zyczenia()

print(z)    # daje na wyjściu: Wszystkiego Najlepszego!
```

Zadanie 2

Przeanalizuj i sprawdź poniższe fragmenty kodu

```
def fibonacci(n):
    if n < 0:
        print("Argument musi być liczbą dodatnią.")
        return
    elif n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)


n = int(input("podaj liczbę naturalną "))
wynik = fibonacci(n)
print(f"{n}-ty element ciągu Fibonacciego wynosi: {wynik}")
```

3. Zakres zmienny w funkcji

Zmienna istniejąca poza funkcją ma zakres wewnątrz ciała funkcji (Przykład 1), chyba że funkcja definiuje zmienną o tej samej nazwie (Przykład 2 oraz Przykład 3), np.:

Przykład 1:

```
var = 2
def pomnoz_przez_var(x):
    return x * var

print(pomnoz_przez_var(7))     # daje na wyjściu: 14
```

Przykład 2:

```
def pomnoz(x):
    var = 5
    return x * var
print(pomnoz(7))              # daje na wyjściu: 35
```

Przykład 3:

```
def mnozenie(x):
    var = 7
    return x * var

var = 3
print(mnozenie(7))     # daje na wyjściu: 49
```

Zmienna istniejąca wewnątrz funkcji ma zakres wewnątrz ciała funkcji (Przykład 4), np.:

Przykład 4:

```
def dodaj(x):
    var = 7
    return x + var
print(dodaj(4))          # daje na wyjściu: 11
print(var)               # NameError
```

Możesz użyć słowa kluczowego global po którym następuje nazwa zmiennej, aby zakres zmiennej był globalny, np.:

```
var = 2
print(var)               # daje na wyjściu: 2
def zwroc_var():
    global var
    var = 5
    return var

print(zwroc_var())       # daje na wyjściu: 5
print(var)               # daje na wyjściu: 5
```