Funkcje

Podstawy programowania w języku Python

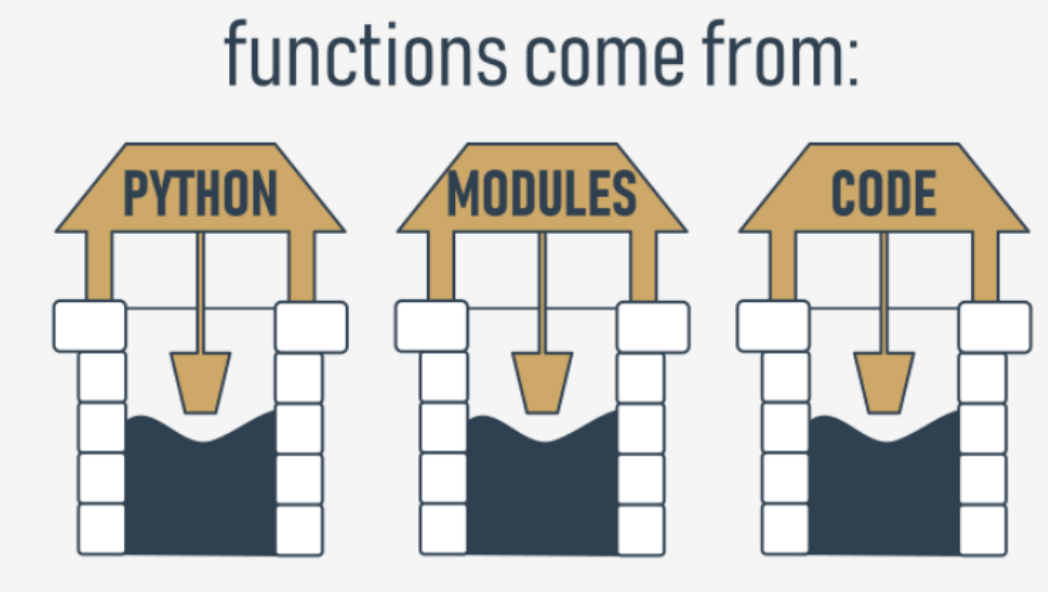
Co to jest funkcja?

- nazwana sekwencja instrukcji, która wykonuje pewne obliczenia
- gdy definiujemy funkcję określamy jej nazwę oraz kolejność instrukcji
- możemy wywołać funkcję po nazwie

```
1  def double_print():
2      print("Hello")
3      print("World!")
4
5  double_print()
```

Skąd pochodzą funkcje?

- są integralną częścią języka Python
- z preinstalowanych modułów języka Python
- bezpośrednio z naszego kodu



Jaka jest rola funkcji?

- grupuje zbiór instrukcji w taki sposób, by mogły one być wykonane w programie więcej niż jeden raz
- najważniejsza struktura programu w Pythonie, służy do maksymalizacji możliwości ponownego wykorzystania kodu i minimalizowania powtarzalności kodu
- narzędzie projektowe pozwalające na rozbicie złożonych systemów, którymi łatwiej jest zarządzać – proceduralny podział na części

Zalety funkcji

- umieszczenie kodu w funkcji sprawia, że uzyskujemy narzędzie, które można wykonać tyle razy ile będzie to potrzebne
- ponieważ kod wywołujący może przekazywać elementy dowolnego typu, funkcje mogą być uniwersalne i działać na różnych obiektach
- kiedy chcemy zmienić sposób działania, wystarczy wprowadzić modyfikację w jednym miejscu
- umieszczenie funkcji w pliku modułu sprawia, że można ją zaimportować i użyć jej ponownie w dowolnym programie

Najprostsza definicja funkcji

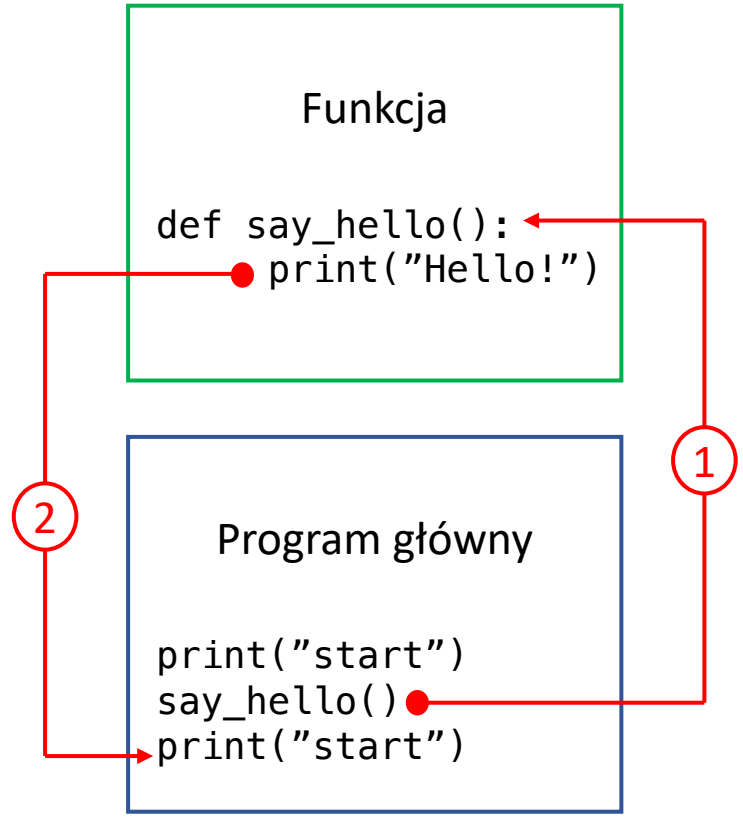
- wiersz nagłówka ***def*** określa nazwę funkcji przypisywaną do obiektu funkcji
- ciało funkcji jest blokiem instrukcji funkcji

```
def nazwa_funkcji():
    cialo_funkcji
```

```
1    def say_hello():
2        print("Hello!")
```

Jak działają funkcje?

- kiedy wywołujemy funkcję, język Python zapamiętuje miejsce, w którym się to stało i przeskakuje do wywołanej funkcji
- następnie wykonywane jest ciało funkcji
- kiedy program dotrze do końca funkcji, zostanie on na powrót skierowany do miejsca, które następuje bezpośrednio po fragmencie, w którym miało miejsce wywołanie funkcji



Ograniczenia przy definiowaniu funkcji

- nie wolno wywoływać funkcji, która nie jest znana w momencie wywołania
- nie wolno posługiwać się tą samą nazwą dla funkcji i dla zmiennej

Funkcje parametryzowane

- do funkcji można przekazywać dane
- dane przekazujemy jako argumenty funkcji podczas jej wywołania
- przekazane dane mogą modyfikować zachowanie funkcji, czyniąc ją bardziej elastyczną i adaptowalną do zmieniających się warunków

```
def nazwa_funkcji(arg1, arg2,…, argN):
    cialo_funkcji
```

```
1    def show_numbers(a, b, c):
2        print("a =", a, "b =", b, "c =", c)
```

Przekazywanie parametrów pozycyjnych

- technika, która przypisuje dany argument (pierwszy, drugi i tak dalej) do danego parametru funkcji (pierwszego, drugiego i tak dalej)
- nazywa się **przekazywaniem argumentu pozycyjnego** (ang. positional parameter passing)
- argumenty przekazywane w ten sposób są nazywane **argumentami pozycyjnymi** (ang. positional arguments)

```
1    def show_numbers(a, b, c):
2        print(a, b, c)
3
4    show_numbers(1, 2, 3)
```

Przekazywanie argumentów słów kluczowych

- konwencja przekazywania argumentów, w której **znaczenie argumentu jest podyktowane jego nazwą** a nie jego pozycją
- nazywa się **przekazywaniem argumentów słów kluczowych** (ang. keyword argument passing)

```
1    def show_numbers(a, b, c):
2        print(a, b, c)
3
4    show_numbers(c=3, a=1, b=2)
```

Wartości domyślne argumentów funkcji

- argumenty funkcji mogą mieć **domyślne (predefiniowane) wartości**
- w przypadku pominięcia odpowiedniego argumentu zostanie użyta wartość domyślna
- domyślne wartości mogą posiadać jedynie te parametry, które są na końcu listy

```
1    def introduce(first_name="Jan", last_name="Kowalski"):
2        print("Część, jestem", first_name, last_name + ".")
```

Wartość None

- specjalny typ w Pythonie (NoneType), który reprezentuje… nic
- to nie to samo co **0** (zero), **False** lub pusty ciąg znaków
- jest słowem kluczowym

```
1    x = None
2    print(x) #wynik: None
```

Zwracanie wartości przez funkcję

- **return** kończy wykonywanie funkcji w wybranym miejscu i pozwala pominąć pewne części funkcji
- **return** zwraca z funkcji podaną wartość, którą można przypisać do zmiennej lub od razu wykorzystać w innych działaniach
- jeśli w **return** nie poda się wartości do zwrócenia to domyślnie zwróci **None**
- jeśli w funkcji nie poda się **return** to domyślnie zostanie on wykonany po ostatniej instrukcji funkcji i także zwróci **None**

```
1    def sum(x, y):
2        return x + y
3
4    print(sum(1, 2))
```

Pytanie

Czy każda funkcja w Pytonie zwraca jakąś wartość?

a) tak, każda funkcja zwraca jakaś wartość, bo każda funkcja wywołuję instrukcję return
b) nie, wartość zwracają tylko funkcje z instrukcją return
c) nie, wartość zwracają tylko funkcje z instrukcją return i wskazaną wartością

Odpowiedź: a)

Każda funkcja wywołuję (czasami niejawnie) instrukcję return. Wywołanie instrukcji return bez wartości oznacza wywołanie return None. None to także wartość.

Pytanie

Co wyświetli się na ekranie po wykonaniu poniższego skryptu?

```
a)  wystąpi błąd TypeError
b)  8 1 8
c)  1 8 1
d)  1 8 8
e)  8 1 1
```

```
1    def my_function(a, b, c=8):
2        print(a, b, c)
3
4    my_function(b=1, a=8)
```

Odpowiedź: b)

Nie ma potrzeby podawać trzeciego argumentu funkcji - przyjmie on wartość domyślną.

Pytanie

Co to znaczy, że argumenty do funkcji przekazywane są pozycyjnie?

a) że istotna jest kolejność argumentów w wywołaniu funkcji
b) że istotna jest kolejność parametrów w definicji funkcji
c) że argumenty zostaną przypisane do odpowiednich parametrów po nazwie podczas wywoływania funkcji

Odpowiedź: a)