Python - funkcje

Dawid Kosiński

Czym są funkcje?

- Są to bloki kodu, umożliwiające wielokrotne wykorzystywanie
- Zwiększają czytelność (ang. readability) kodu poprzez jednoznaczne nazwy co dana funkcja robi
- Budowa funkcji:
    - **def** <nazwa>**(...):**
        <wykonaj kod>
- Aby wywołać daną funkcję należy podać jej nazwę i ( ). Jeśli funkcja wymaga parametrów, należy je przekazać pomiędzy nawiasami, np.
    - get_user(‘tomek’)

Po co wykorzystywać funkcje?

- Umożliwiają zarządzanie kodem z jednego miejsca
- Zwiększają czytelność kodu dzięki jednoznacznym nazwom (zgodnie z zasadą Zen Of Python - Simple is better than complex)
- Lepsza separacja kodu
- Lepsza wymienialność kodu

Dobre praktyki tworzenia funkcji

- Jednoznaczna nazwa, np. jeśli chcemy pobrać użytkownika nazwa powinna być “get_user”
- Powinna robić jedną, konkretną rzecz (ang. single responsibility)
- Posiada docstring - tekstu opisujący co dana funkcja robi i zwraca (jest to mile widziane)

Zwracanie z funkcji

- Nie ma potrzeby deklaracji typu zwracanego z funkcji (tak jak np. w C++)
- Domyślnie funkcja zwraca None lecz może ona zwrócić jakikolwiek obiekt
- Aby zwrócić obiekt należy na końcu funkcji dodać: **return** <obiekt>
- Budowa funkcji ze zwracaniem:
    - **def** calculate_sum**(**a, b**):**
      result = a + b
      return result

Przekazywanie parametrów do funkcji

- Funkcje nie muszą posiadać parametrów, Wtedy niczego nie ma pomiędzy ( ) -> **def** <nazwa>**()**:
- Jeśli funkcja wymaga parametrów wystarczy je podać pomiędzy nawiasami:
  - **def** <nazwa>**(**parametr1, parametr2**):**
- Ilość argumentów jest dowolna lecz nie powinno się przesadzać (jest to zła praktyka)

Argumenty pozycyjne (positional arguments)

- Są **wymagane** podczas chęci wywołania funkcji
- Jak nazwa sama wskazuje, są to argumenty, które są **zależne od pozycji w jakim je zadeklarowano**
- Przykład:
  - **def** subtract**(param1, param2):**
    return param1 - **param2**
- Wykorzystując powyższą funkcję otrzymamy inne wyniki w zależności od tego w jakiej kolejności przekażemy argumenty do funkcji:
  - substract(4, 5) ---> -1 (4 trafiło pod nazwę **param1**, 5 trafiło pod nazwę **param2**)
  - substract(5, 4) ---> 1 (5 trafiło pod nazwę **param1**, 4 trafiło pod nazwę **param2**)

Argumenty nazwane (keyword arguments)

- Są **opcjonalne** podczas chęci wywołania funkcji
- Posiadają **wartość domyślną**, gdy użytkownik nie przekaże argumentu (deklaracja za pomocą ‘<nazwa>=’)
- Można je podawać w losowej kolejności jeśli wskażemy pod jaki argument ma trafić dana zmienna
- Nie podając nazwy argumentu zmienne trafią pod argumenty zgodnie z deklaracja w funkcji
- Przykład:
  - **def** say_hello**(from=**’Bob’, **to=**’Tom’**):**
    return f’{from} says hello to {to}’
- Wywołania:
  - say_hello() -> ‘Bob says hello to Tom’ (wykorzystano **domyślne** wartości: **‘Bob’**, **‘Tom’**)
  - say_hello(**to=’**John’, **from=**’Jerry’) -> ‘Jerry says hello to John’ (**John** trafił pod argument **to**, **Jerry** pod argument **from**)
  - say_hello(**to=**’Sam’) -> ‘Bob says hello to Sam’ (wykorzystano **domyślna** wartość **from** oraz **Sam** trafił pod **to**)
  - say_hello(‘Sam’, ‘Bob’) -> ‘Sam says hello to Bob’ (**Sam** trafił pod argument **from**, **Bob** pod argument **from**)
- **Argumenty nazwane występują PO argumentach pozycyjnych**

Co w przypadku nieznanej liczby argumentów?

- Aby przekazać nieznaną liczbę argumentów pozycyjnych należy wykorzystać: ***args**
  - pod takim parametrem znajdzie się **tupla** z parametrami, które **nie zostały dopasowane** do zadeklarowanych argumentów pozycyjnych
- Aby przekazać nieznaną liczbę nazwanych argumentów należy wykorzystać: ****kwargs**
  - pod takim parametrem znajdzie się **słownik** z parametrami, które **nie zostały dopasowane** do zadeklarowanych argumentów nazwanych
- Przykład:
  - def foo(a, b, k=0, ***args**, ****kwargs**):
    pass
- Wywołania:
  - foo(1, 2, 3) → foo(a=1, b=2, k=3, **args=( )**, **kwargs={ }**)
  - foo(1, 2, 3, 4, 5) → foo(a=1, b=2, k=3, **args=(4, 5)**, **kwargs={ }**)
  - foo(1, 2, **c=3**) → foo(a=1, b=2, k=0, **args=( )**, **kwargs={'c': 3}**)
  - foo(1, 2, **3**, **4, 5**, **c=30**) → foo(a=1, b=2, **k=3**, **args=(4, 5)**, **kwargs={'c': 30}**)

Funkcje lambda

- Są to małe funkcje - mogą posiadać tylko jedno wyrażenie
- Nie posiadają nazwy - są anonimowe
- Wykorzystuje się je tam, gdzie inne funkcje wymagają przekazania funkcji jako parametr
- Budowa:
  - **lambda** <argumenty>**:** <wyrażenie>
- Przykład:
  - my_list = [5, 4, 2, 10, 1, 20, 3, 3, 55]
  - filter(**lambda x: x not in [1, 2, 3]**, my_list) -> filter(..) zwróci odpowiednio liczby: 5, 4, 10, 20, 55