Laboratorium 4

Funkcje w języku Python

Zagadnienia:

- Funkcje klasyczne i ich właściwości
- Argumenty specjalne *args i **kwargs
- Funkcje anonimowe
- Dokumentowanie funkcji

Funkcje klasyczne i ich właściwości

Funkcje to bloki kodu, które wykonują określone operacje i mogą być wielokrotnie wywoływane w różnych miejscach programu. Są one używane do organizacji i uproszczenia kodu, co znacząco zwiększa czytelność i sposób zarządzania projektami. Funkcje w języku Python mogą przyjmować dowolną liczbę argumentów (również argumenty domyślne) oraz zwracać dowolną liczbę wyników (lub nie zwracać ich wcale):

```
def diff(x = 0, y = 0):
    return x-y

print(diff())
print(diff(5,3))
print(diff(1))
print(diff(y=2))
```

Widzimy również, że wywołanie może odbywać się z argumentami zgodnie z ich kolejnością w definicji lub za pomocą nazwy argumentu w definicji (kolejność nie ma znaczenia) oraz w sposób mieszany (argumenty pozycyjne muszą poprzedzać argumenty zdefiniowane po nazwie):

```
print(diff(y=2,1)) #BŁĄD!
```

Argumenty przekazywane są do funkcji zawsze przez wartość, co oznacza, że operacje są wykonywane na kopiach zmiennych, a nie na oryginałach. Modyfikacja argumentów możliwa jest tylko w przypadku, obiektów których stan można zmieniać po ich utworzeniu (mutable), np. takich jak listy:

```
def square(x):
return x**2

x = 3
print(square(x))
print(x)


def square(x):
    x[0] = x[0]**2
    return x
```

```
x = [2,3,4,5]
print(square(x))
print(x)
```

Zmienne definiowane wewnątrz funkcji mają zakres lokalny i nie są widzialne poza funkcją, chyba że zostaną jawnie zdefiniowane jako globalne (global). Natomiast zmienne zdefiniowane poza funkcją mają zakres globalny i są widzialne we wszystkich funkcjach:

```
result = 0
def diff(x = 0, y = 0):
    result = x-y
    return result

print(diff(5,3))
print(result)
```

```
def diff(x = 0, y = 0):
    global result
    result = x-y
    return result

print(diff(5,3))
print(result)
```

W Pythonie funkcje są obiektami pierwszej klasy, co oznacza, że można je przypisywać do zmiennych, przekazywać jako argumenty do innych funkcji lub zwracać z funkcji:

```
# Przypisywanie funkcji do zmiennej:
def greet(name):
    return f"Hello, {name}!"

say_hello = greet
print(say_hello("Alice"))
```

```
# Przekazywanie funkcji jako argumentu:
def diff(x, y):
    return x - y

def apply_function(func, a, b):
    return func(a, b)

result = apply_function(diff, 5, 3)
print(result)
```

```
# Zwracanie funkcji z funkcji:
def outer_function(x):
    def inner_function(y):
        return x + y
    return inner_function

add_five = outer_function(5)
print(add_five(10))
```

Argumenty specjalne args i kwargs

Specjalne argumenty *args i **kwargs pozwalają na przekazywanie zmiennej liczby argumentów do funkcji:

- *args – umożliwia przekazanie dowolnej liczby argumentów pozycyjnych do funkcji, a dostęp do nich uzyskujemy za pomocą indeksów (krotka)

  ```
  def mean(*args):
      return sum(args) / len(args)

  result = mean(2, 4, 6, 8, 10)
      print(result)
  ```

- **kwargs – umożliwia przekazanie dowolnej liczby argumentów nazwanych w postaci klucz=wartość, a dostęp do nich uzyskujemy za pomocą klucza (słownik):

  ```
  def print_dict(**kwargs):
      for key, value in kwargs.items():
          print(f"{key}: {value}")

  print_dict(name = 'John', surname = 'Rambo', age = '33')
  ```

Funkcje anonimowe

Funkcje anonimowe w Pythonie (funkcje lambda) to jednowierszowe funkcje nie posiadające nazwy definiowane za pomocą słowa kluczowego lambda. Można je przekazywać jako argumenty do innych funkcji lub wyrażeń. Stosuje się je gdy istnieje potrzeba zastosowania krótkiej chwilowej funkcji, która nie będzie później wykorzystywana:

```
diff = lambda x,y: x-y
print(diff(5,3))
```

Często stosuje się je w połączeniu z funkcjami wyższego rzędu, np.:

- map() – zastosowanie funkcji do każdego elementu w sekwencji:

  ```
  numbers = [1, 2, 3, 4, 5, 6]
  squares = map(lambda x: x**2, numbers)
  print(list(squares))
  ```

- filter() – filtrowanie elementów w sekwencji na podstawie warunków:

```
evens = filter(lambda x: x % 2 == 0, numbers)
print(list(evens))
```

- reduce() – redukowanie sekwencji do pojedynczej wartości

```
from functools import reduce
product = reduce(lambda x, y: x * y, numbers)
print(product)
```

Dokumentowanie funkcji

Dokumentowanie funkcji w Pytonie realizowane jest za pomocą docstringów (stringi dokumentacyjne). Umieszcza się je wewnątrz funkcji bezpośrednio po definicji i otacza potrójnymi cudzysłowami (komentarz wielowierszowy):

```
def diff(x, y):
    """
    Difference of two numbers
    Parameters:
    x (int, float): First numer
    y (int, float): Second numer


    Returns:
    int, float: Difference of two numbers
    """

    return x-y
```

Zadania

1. Wykorzystując funkcje anonimowe napisz funkcję, która umożliwi sortowanie listy liczb całkowitych względem liczby cyfr każdej z nich, wyświetlanie liczby o największej i najmniejszej liczbie cyfr.
2. Wykorzystując wyrażenia lambda oraz funkcję map napisz funkcję, która dla listy napisów sformatuje każdy z nich w taki sposób, aby rozpoczynał się od wielkiej litery alfabetu i kończył kropką.
3. Napisz funkcję, która będzie przyjmowała dowolną liczbę argumentów, a następnie:
    a. Jeżeli argumentami były liczby zwróci ich średnią
    b. Jeżeli argumentami były napisy zwróci medianę ich długości
    c. Jeżeli argumenty były innego typu wyświetli odpowiedni komunikat.
4. Napisz funkcję, która będzie przyjmowała dowolną liczbę argumentów, a następnie:
    a. Jeżeli argumenty były przekazane przez pozycję, utworzy z nich listę i ją wyświetli oddzielając elementy tabulatorami
    b. Jeżeli argumenty były przekazane przez nazwę, zapisze każdy z nich do pliku tekstowego w formacie klucz -> wartość. Poszczególne argumenty mają być oddzielone znakami końca linii
    c. Jeżeli argumenty były przekazane w sposób mieszany utworzy z nich słownik i wyświetli zawartość.
5. Do każdej funkcji dodaj dokumentujący ją docstring.