WIZUALIZACJA DANYCH

Ćwiczenie 3

Python 3: Python 3: Funkcje, moduły i operacje na plikach

Python Comprehension

Jest to mechanizm służący do generowania kolekcji (lista, słownik, zbiór) na podstawie jednowierszowej definicji. Równoważne definicje zawsze można podać za pomocą pętli. Czasami zaś wystarczy przepisać na język Python definicję matematyczną zbioru.

Możliwa składnia

```
#Zamiast pisać w pętli
lista = []
for element in zakres:
     if pewien_warunek_na(element):
          lista.append(„Cos sie dzieje z:” + element)

#możemy zapisać w jednej linijce
lista = [„Cos sie dzieje z:” + element for element in zakres if
pewien_warunek_na(element)]
```

Przykład

Mamy zdefiniowane zbiory:

$A=\{x^2: x\in<0,9>\}$

$B=\{1, 3, 9, 27,\ldots, 3^5\}$

C={x: x∈A i x jest liczbą nieparzystą}

W Python zapiszemy to:

```
A=[x**2 for x in range(10)]

B=[3**i for i in range(6)]

C=[x for x in A if x % 2 != 0]

print(A)
print(B)
print(C)
```

Przykład

```
#Chcemy uzyskać liczby parzyste z podanego zakresu
```

```
#Wersja z pętlą

liczby=[1,2,3,4,5,6,7,8,9,10]
lista=[]
for i in liczby:
    if i % 2 == 0:
        lista.append(i)

print("Liczby parzyste uzyskane z wykorystaniem pętli")
print(lista)

#wersja z Python comprehension

lista2=[i for i in liczby if i % 2 == 0]
print(lista2)
```

Przykład

```
#Zagnieżdżanie
#Zamiast pisać tak:
lista=[]
for i in [1, 2, 3]:
    for j in [4, 5, 6]:
      if i != j:
          lista.append((i,j))
print(lista)

#można to zrobić krócej
lista2=[(i,j) for i in [1, 2, 3] for j in [4, 5, 6]]
print(lista2)
```

Przykład

```
#Słowniki i zamiana klucza z wartością

skroty={"PZU": "Państwowy zakład Ubezpieczeń",
          "ZUS": "Zakład Ubezpieczeń Społecznych",
          "PKO": "Powszechna Kasa Oszczędności"}

odwrocone={value: key for key, value in skroty.items()}
print("Oryginalny słownik")
print(skroty)
print("Słownik odwrócony")
print(odwrocone)
```

Zad. 1

Zdefiniuj następujące zbiory, wykorzystując Python comprehension:

A={1/x: x∈<1,10>}

B={1, 2, 4, 8,…, 2^{10}}

C={x: x∈B i x jest liczbą podzielną przez 4}

Zad. 2

Wygeneruj losowo macierz 4x4 i wykorzystując Python Comprehension zdefiniuj listę, która będzie zawierała tylko elementy znajdujące się na przekątnej.

Zad. 3

Utwórz słownik z produktami spożywczymi do kupienia. Klucz to niech będzie nazwa produktu a wartość - jednostka w jakiej się je kupuje (np. sztuki, kg itd.). Wykorzystaj Python Comprehension do zdefiniowania nowej listy, gdzie będą produkty, których wartość to sztuki.

Funkcje

W Pythonie możemy definiować własne funkcje, które będziemy traktować jak podprogramy albo jak funkcje w matlabie.

Składnia:

```
def nazwa_funkcji(arg_pozycyjny, arg_domyslny=wartosc, *arg_4, **arg_5):
      instrukcje
      return wartość
```

Definicja instrukcji to instrukcja która tworzy obiekt. Funkcje możemy wywoływać z argumentami lub bez ale zawsze musimy używać nawiasów (nawet jak nie ma argumentów). Funkcja może zwracać jedną lub wiele wartości, które będą zwrócone jako krotka

Przykład

Chcemy zdefiniować funkcję, która będzie obliczać pierwiastki równania kwadratowego:

```
import math

def row_kwadratowe(a, b, c):
    delta = b**2 - 4 * a * c
    if (delta < 0):
        print("Brak pierwiastków")
        return -1
    elif (delta == 0):
        print("Jeden pierwiastek")
        x = (-b) / (2 * a)
        return x
    else:
        print("Równanie ma dwa pierwiastki")
        x1= (- b - math.sqrt(delta)) / (2 * a)
        x2= (- b + math.sqrt(delta)) / (2 * a)
        return x1, x2

print(row_kwadratowe(6,1,3))
print(row_kwadratowe(1,2,1))
print(row_kwadratowe(1,4,1))
```

Zad. 4

Zdefiniuj funkcję, która będzie badać monotoniczność funkcji liniowej:

y = a x + b

Funkcja jest rosną ca gdy a>0

malejąca jeżeli a<0

stała gdy a=0

i w zależności od tego będzie wyświetlać odpowiedni komunikat

Zad. 5

Napisz funkcję, która będzie sprawdzać czy dwie proste są równoległe czy prostopadłe:

Proste dane równaniami $y=a_1x+b_1$, $y=a_2x+b_2$, są

równolegle gdy $a_1=a_2$

prostopadłe gdy $a_1a_2=-1$

Przykład

```
#Definiujemy funkcje z wartościami domyślnymi
import math

def dlugosc_odcinka(x1 = 0, y1 = 0, x2 = 0, y2 = 0):
    return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)

#wywołujemy dla wartości domyślnych
print(dlugosc_odcinka())

#wywolujemy dla własnych podanych wartości
#są to argumenty pozycjne czyli ważna jest kolejnosć podania wartości
print(dlugosc_odcinka(1, 2, 3, 4))

#Wywolujemy funkcje podając mieszane wartości
#Dwie pierwsze są interpretowane jako x1 i y1 jak podano w definicji funkcji
print(dlugosc_odcinka(2, 2, y2 = 2, x2 = 1))

#wywołujemy funkcje pdoając wartości nie w kolejności
print(dlugosc_odcinka(y2 = 5, x1 = 2, y1 = 2, x2 = 6))

#wywołujemy funkcję podając tylko dwa argumenty a reszta domyślne
print(dlugosc_odcinka(x2 = 5, y2 = 5))
```

Zad. 6

Zdefiniuj funkcję, która oblicza długość przeciwprostokątnej, wykorzystując twierdzenie pitagorasa. Proszę podać wartości domyślne dla funkcji

Zad. 7

Zdefiniuj funkcję, która zwraca sumę dowolnego ciągu arytmetycznego.

Funkcja niech przyjmuje jako parametry: a_1 (wartość początkowa), r (wielkość o ile rosną kolejne elementy) i ile_elementów (ile elementów ma sumować). Ponadto funkcja niech przyjmuje wartości domyślne: a_1= 1, r=1, ile=10.

Przykład

```
#symbol * oznacza dowolną ilość argumentów przechowywanych w krotce

def ciag(* liczby):
    #jeżeli nie ma argumentów to
    if len(liczby) == 0:
        return 0.0
    else:
        suma = 0.0

        #sumujemy elementy ciągu
        for i in liczby:
            suma += i
        #zwracamy wartość sumy
        return suma
#wywołanie gdy brak argumentów
print(ciag())
#podajemy argumenty
print(ciag(1,2,3,4,5,6,7,8))
```

Zad. 8

Wykorzystując poprzedni przykład zdefiniuj funkcję, która będzie liczyć iloczyn elementów ciągu.

Przykład

```
# ** czyli dwie gwiazdki oznaczają że możemy użyć
# dowolną ilość argumentów z kluczem
def to_lubie(** rzeczy):
    for cos in rzeczy:
        print("To jest ")
        print(cos)
        print(" co lubie ")
        print(rzeczy[cos])


to_lubie(slodycze="czekolada", rozrywka=["disco-polo", "moda na sukces"])
```

Zad. 9

Napisz funkcję, która wykorzystuje symbol **. Funkcja ma przyjmować listę zakupów w postaci: klucz to nazwa produktu a wartość to ilość. Funkcja ma zliczyć ile jest wszystkich produktów w ogóle i zwracać tę wartość.

Moduły i pakiety

Żeby użyć funkcji matematycznych potrzebowaliśmy zaimportować plik math.

Taki plik nazywa się modułem i są tam zapisane po prostu kody w języku Python. Jeśli takich plików będziemy mieć kilka to możemy utworzyć z nich pakiet.

Import modułów systemowych

Jeden import modułu powinien być w jednej linii np.

```
import sys
```

można również zapisać import modułu w postaci:

```
from math import *
```

Import modułu zamieszczamy na początku pliku. Ewentualnie za komentarzami. Zaleca się następującą kolejność importów:

- biblioteki standardowe
- powiązane biblioteki zewnętrzne
- lokalne aplikacje/biblioteki

Tworzenie swojego modułu

- Tworząc swój moduł piszemy funkcje i zapisujemy w pliku z rozszerzeniem .py
- Następnie dołączamy do nowego skryptu swój moduł używając instrukcji.

Przykład

Zawartość pliku litery, który będzie naszym modułem

```
#plik litery

def wyswietl(a):
    print(a)


def dlugosc(a):
    return len(a)
```

Teraz możemy już wykorzystać funkcje z modułu litery (to będzie nowy skrypt):

```
import litery

a = "Ala ma kota"
litery.wyswietl(a)
print(litery.dlugosc(a))

#wyświetla wszystkie zmienne oraz nazwy modułów, które się w
nim znajdują
print(dir(litery))
```

Tworzenie swojego pakietu

Pakiet składa się z kilku modułów i najczęściej zapisywany w określonym folderze, gdzie nazwa folderu oznacza nazwę pakietu. Jeżeli chcemy stworzyć pakiet musimy utworzyć katalog dodać tam moduły a następnie dorzucić pliku o nazwie __init__.py, w którym powinien się znaleźć sposób importu plików. Dla stylu **`import pakiet.moduł`** plik zostaje pusty dla stylu **`from pakiet import *`** **`w`** pliku zapisujemy zmienną __all__ która zawiera wszystkie moduły, które mogą być zaimportowane.

Przykład

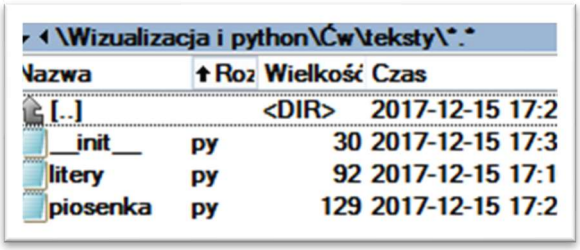
Tworzymy jeszcze jeden moduł

```
#piosenka.py
```

```
def spiew():
    print("La la la la la")


def zespoly():
    print("Boysband")
    print("Girl'n'dance")
```

Tworzymy teraz katalog teksty i wrzucamy tam nasze moduły oraz edytujemy pliki __init__.py



Rys. 1. Zawartość katalogu teksty, gdzie jest omawiany pakiet

Zawartość pliku __init__.py

```
__all__ = ["litery", "piosenka"]
```

Zad. 10

Stwórz pakiet liczby zespolone z dwoma modułami.

Jeden moduł ma zawierać dwie funkcje, które z podanej liczby zespolonej zwracają część rzeczywistą i część urojoną

Drugi moduł ma wykonywać dodawanie i odejmowanie dwóch liczb zespolonych.

Przetestuj działanie tego pakietu.

Zad. 11

Stwórz pakiet ciągi. Jeden moduł niech dotyczy działań i wzorów związanych z ciągami arytmetycznymi a drugi niech dotyczy działań i wzorów związanych z ciągami geometrycznymi.

Bibliografia

[1] Jacqueline Kazil, [online], Katharine Jarmul, Data Wrangling with Python, wyd. 1, O'Reilly, 2016, dostęp: 13 września 2017, Dostępny w Internecie: http://pdf.th7.cn/down/files/1603/Data%20Wrangling%20with%20Python.pdf

[2] Wes McKinney, [online], Python for Data Analysis, wyd. 1, O'Reilly, 2013, dostęp: 13 września 2017, Był dostępny w Internecie: http://www3.canisius.edu/~yany/python/Python4DataAnalysis.pdf

[3] Marian Mysior, *Ćwiczenia z języka Python*, Warszawa, Mikom, 2003