

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 pierwszy

$A = \{x^2: x \in \langle 0, 9 \rangle\}$

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

$C = \{x: x \in A \text{ i } x \text{ jest liczbą nieparzystą}\}$

W pythonie zapiszemy to:

#wersja z pętlą

a = []

for x *in* range(10):

a.append(x**2)

print(*a*)

b = []

for y *in* range(6):

b.append(3**y)

print(*b*)

c = []

for z *in* *a*:

if z % 2 == 1:

c.append(z)

```
print(c)

#wersja z python comprehension
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 == 1]

print(a)
print(b)
print(c)
```

Przykład drugi

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 wykorzystaniem pętli")
print(lista)

print()
```

```
#wersja z python comprehension
lista2 = [i for i in liczby if i % 2 == 0]

print(lista2)
```

Przykład trzeci zagnieżdżenia

```
#wersja z zagnieżdżonymi pętlami
lista = []

for i in [1, 2, 3]:
    for j in [4, 5, 6]:
        lista.append((i,j))
```

```
print(lista)
```

#wersja z python comprehension

```
lista2 = [(i,j) for i in [1, 2, 3] for j in [4, 5, 6]]
```

```
print(lista2)
```

Przykład czwarty związany ze zamianą klucza z wartością w słowniku

#wersja z pętlą

```
skroty = {"PZU": "Państwowy zakład ubezpieczeń",  
          "ZUS": "Zakład ubezpieczeń społecznych",  
          "PKO": "Państwowa kasa oszczędności"}
```

```
odwrocone = {}
```

```
for key,value in skroty.items():
```

```
    odwrocone[value] = key
```

```
print(odwrocone)
```

#wersja z python comprehension

```
odwrocone2 = {value: key for key, value in skroty.items()}
```

```
print(odwrocone2)
```

Funkcje

W pythonie możemy definiować własne funkcje, które będziemy traktować jak podprogramy lub 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 pierwszy

Chcemy zdefiniować funkcję, która będzie obliczać pierwiastki równania kwadratowego

```
def row_kwadratowe(a,b,c):  
    delta = b**2 - 4 * a * c  
    if delta < 0:  
        print("brak pierwiastków")  
        return -1  
    elif delta == 0:  
        print("jedne pierwiastek")  
        x = (-b) / (2 * a)  
        return x  
    else:  
        print("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))
```

Przykład drugi

Definiujemy funkcję 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 funkcje dla wartości domyślnych  
print(dlugosc_odcinka())
```

#wywołujemy funkcje dla własnych podanych wartości

#są to argumenty pozycyjne czyli ważna jest kolejność podania wartości

```
print(dlugosc_odcinka(1,2,3,4))
```

#wywołujemy funkcje podając mieszane wartości

#dwie pierwsze interpretowane są jako x1 i y1 jak podano w definicji funkcji

```
print(dlugosc_odcinka(2, 2, y2=2, x2=1))
```

#wywołujemy funkcje podając wartości nie w kolejności

```
print(dlugosc_odcinka(y2=5, x1=2, y1=2, x2=6))
```

#wywołujemy funkcje podając tylko dwa argumenty a reszta domyślne

```
print(dlugosc_odcinka(x2=5, y2=5))
```

Przykład trzeci

Symbol * oznacza dowolną ilość argumentów przechowywanych w krotce

```
def ciag(* liczby):
```

```
    # jeżeli nie ma argumentów to
```

```
    if len(liczby) == 0:
```

```
        return 0
```

```
    else:
```

```
        suma = 0
```

```
        #sumujemy elementy ciągu
```

```
        for i in liczby:
```

```
            suma += i
```

```
        #zwracamy wartość sumy
```

```
        return suma
```

#wywołanie gdy nie ma argumentów

print(ciaag())

#podajemy argumenty

print(ciaag(1, 2, 3.5, 4, 5, 6, 7, 8))

Przykład czwarty

****** 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=['gry', 'filmy'])

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 ją do 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))
```

```
#wyswietla 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():
```

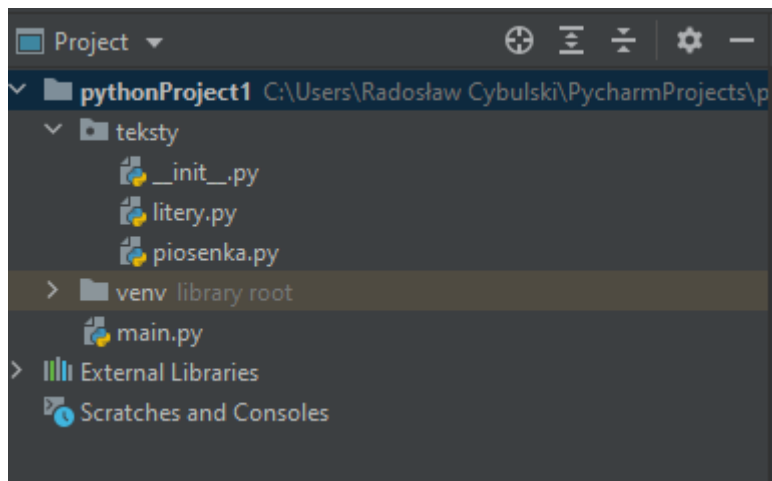
```
    print("La la la la la")
```

```
def zespol():
```

```
    print("Boysband")
```

```
    print("Girl'n'dance")
```

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



Zawartość pliku `__init__.py`

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


Zadania

Zad1

Zdefiniuj następujące zbiory, wykorzystując Python comprehension:

$$A = \{1-x: x \in \langle 1, 10 \rangle\}$$

$$B = \{1, 4, 16, \dots, 4^7\}$$

$$C = \{x: x \in B \text{ i } x \text{ jest liczbą podzielną przez } 2\}$$

Zad2

Wygeneruj losowo 10 elementów, zapisz je do listy1, następnie wykorzystując Python Comprehension zdefiniuj nową listę, która będzie zawierała tylko parzyste elementy

Zad3

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.

Zad4

Zdefiniuj funkcję, która sprawdzi czy trójkąt jest prostokątny.

Zad5

Zdefiniuj funkcję która policzy pole trapezu. Funkcja ma przyjmować wartości domyślne.

Zad6.

Zdefiniuj funkcję która będzie liczyć iloczyn elementów ciągu.

Parametry funkcji a_1 (wartość początkowa), b (wielkość o ile mnożone są kolejne elementy), ile (ile elementów ma mnożyć)

Ponadto funkcja niech przyjmuje wartości domyślne: $a = 1$, $b = 4$, $ile = 10$

Zad7

Napisz funkcję za pomocą operatora $*$, która wykona te same działanie co w zadaniu 6.

Zad8

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

Zad9

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