Iteratoriai

Iteratorius - objektas, kurį galima iteruoti. Tai objektas, kuris grąžina duomenis, kas kartą naudojant metodą next().

Iteratorius galima kurti iš bet kokių objektų, kuriuos galime iteruoti su for ciklais, pvz.: string, list, dict ir t.t.

Tarkime žodis "Kėdė" yra objektas iš kurio galime sukurti iteratorių:

```
iteratorius = iter("Kėdė")
print(type(iteratorius))


# <class 'str_iterator'>
```

kai tik bus iškviestas iteratoriaus metodas next(), jis grąžins sekantį iteruojamo objekto segmentą, kol nebeliks ko iteruoti ir išmes StopIteration error:

```
print(next(iteratorius))
print(next(iteratorius))
print(next(iteratorius))
print(next(iteratorius))
print(next(iteratorius))


# K
# ė
# d
# ė
# Traceback (most recent call last):
#   File "/home/blablabla/iterators.py", line 8, in <module>
#     print(next(iteratorius))
# StopIteration
```

Atkreipkite dėmesį, kaip iteratorius įsimena savo būseną(state), ir žino, kad reikia spausdinti sekantį simbolį. Mūsų nuolat naudojami for ciklai yra sukurti iteratorių pagrindu. Parašykime primityvų for ciklą:

```
numeriai = [1, 2, 3]


for num in numeriai:
      print(num)


# 1
# 2
# 3
```

O dabar tą pačią iteraciją atlikime be for sintaksės:

```
iteratorius = iter(numeriai)
while True:
      try:
            print(next(iteratorius))
      except StopIteration:
            break
# 1
# 2
# 3
```

O dabar parašykime šiek tiek universalesnę funkciją:

```
def iteruoklis(objektas, func):
```

```
    iteratorius = iter(objektas)
    while True:
        try:
            result = next(iteratorius)
        except StopIteration:
            break
        else:
            func(result)
```

Mūsų funkcija priima objektą (pvz. list'ą ar kt.) iš karto jį paverčia iteratoriumi. Tuomet, kol yra ką iteruoti, iteruoja ir praleidžia per mūsų funkciją argumentuose func. Pvz.:

```
broliai = ['jurgis', 'antanas', 'aloyzas', 'martynas']
iteruoklis(broliai, print)


# jurgis
# antanas
# aloyzas
# martynas
```

Šiuo atveju pritaikėme Python integruotą funkciją print, taigi mums bus paprasčiausiai atspausdinti brolių vardai. Galime panaudoti savo sukurtą funkciją, tarkime:

```
def kubu(x):
    print(x**3)


nums = [1, 2, 3, 4, 5]
iteruoklis(nums, kubu)


# 1
# 8
# 27
# 64
# 125
```

Generatoriai

Generatoriai yra iteratorių rūšis. Jie yra paprastesnis būdas kurti iteratorius;

- Generatoriai kuriami naudojant generatorių funkcijas;
- Generatorių funkcijų ypatumas yra tas, kad vietoje return naudojame yield;

skirtumai tarp paprastų ir generatoriaus funkcijų (generator functions):

Funkcija	Generatoriaus funkcija
naudoja return	naudoja yield
grąžina rezultatą 1 kartą	gali grąžinti rezultatus daug kartų
grąžina *return* vertę	grąžina generatorių

Pvz.:

```
def skaiciuojam_iki(iki):
      count = 1
      while count <= iki:
            yield count
            count +=1
```

čia yra generatoriaus funkcija. Iš jos galime susikurti generatorių ir analogiškai, kaip ir su kitais iteratoriais, galime iškvieti funkciją next:

```
counter = skaiciuojam_iki(5)
print(next(counter))
print(next(counter))
print(next(counter))
print(next(counter))
print(next(counter))
print(next(counter))


# 1
# 2
# 3
# 4
# 5
# Traceback (most recent call last):
#   File "/home/blablabla/uzduotys.py", line 11, in <module>
#     print(next(counter))
# StopIteration
```

Kaip matome, veiksmas vyksta vienodai, kaip ir su iteratoriais, nes generatorius ir yra iteratorius.

Beje, generatorius galima paversti į list'us:

```
sarasas = list(counter)
print(sarasas)
# [1, 2, 3]
```

Ir juos iteruoti su paprastais for ciklais:

```
for i in counter:
      print(i)


# 1
# 2
# 3
```

o dabar padidinkime iki reikšmę iki 10 ir atlikim eksperimentą:

```
counter = skaiciuojam_iki(10)
print(next(counter))
print(next(counter))
print(next(counter))
sarasas = list(counter)
print(sarasas)


# 1
# 2
# 3
# [4, 5, 6, 7, 8, 9, 10]
```

šis pavyzdys iliustruoja, kaip veikia generatoriai - jie nekaupia viso turinio atmintyje, o tik įsidėmi momentinę reikšmę, kurios pagrindu generuoja sekančią. 1, 2, 3 reikšmės buvo panaudotos ir išmestos, o sąrašas suformuotas tik iš to, kas liko. Dėl to generatoriai veikia gerokai sparčiau už įprastus for loops.

Yra dar vienas metodas generatoriams kurti - generator expressions. Pamenate list comprehension?:) Tai yra analogiškas sintaksinis palengvinimas kurti generatoriams. Pvz.:

```
g = (num**2 for num in range(1, 50))
print(type(g))


print(next(g))
print(next(g))
print(next(g))
# ir t.t.


# <class 'generator'>
# 1
# 4
# 9
```