Iterators and Generators(Silsilələr və törəmələr)

May 23, 2022

0.0.1 Silsilə və Törəmələr

Bundan əvvəlki mövzumuzda python ilə gələn quraşdırılmış bir neçə funksiyanı öyrəndik.Bəzi funksiyaları isə həmin bəhsə yerləşdirmədim.Bu funksiyalar iter() və next() funksiyalarıdır.Həmin funksiyalar pythonda xüsusi yer alır.Bu bəhsdə silsilə və törəmələr nədir gəlin nəzər yetirək.

Iterators(silsilə-təkrarlama) İteratorlar,sayıla bilən dəyərə malik obyektlərdir.Python dilində ifadə etdiyimiz siyahılar,kortej,sətir tipi verilənlər silsilə obyektlərdir.silsilələr **iter** () və **next** () metodlarından ibarət silsilə protokolunu həyata keçirən bir obyektdir.

Xatırlayırsınızsa verilən tipləri üzərində təkrar təkrar gəzərək elementləri əldə edə bilirdik

[2]:
```
siyahı = ['Python','C++','Java','Rust']
for i in siyahı:
    print(i)
```

```
Python
C++
Java
Rust
```

[4]:
```
print(siyahı[0])
```

```
Python
```

[5]:
```
print(siyahı[1])
```

```
C++
```

[7]:
```
kortej = (1,2,3,4,5)
for i in enumerate(kortej,1):
    print(i)
```

```
(1, 1)
(2, 2)
(3, 3)
(4, 4)
(5, 5)
```

[15]:
```
string = 'python'
for i in string:
    print(i)
```

```
p
y
t
h
o
n
```

biz fordövr operatorundan istifadə etdikdə,python həmin obyektin **iter**() metoduna yönəlir,

obyektin silsilə olması üçün mütləq **iter**() və **next**() metodunu qeyd etməliyik.Yuxarıda qeyd etdiyim kimi bu metod sayılan verilən tiplərində zatən mövcuddur.Və həmin verilən tipləri obyekt,silsilələr sayılır

[18]:
```
print(dir(siyahı))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
'__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__',
'__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__',
'__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index',
'insert', 'pop', 'remove', 'reverse', 'sort']
```

siyahı obyekti qeyd etdik və **iter**() metodun var olduğunu obyektdə gördük.

[20]:
```
siyahı = iter(siyahı)
```

[21]:
```
print(dir(siyahı))
```

```
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__length_hint__', '__lt__',
'__ne__', '__new__', '__next__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__setstate__', '__sizeof__', '__str__', '__subclasshook__']
```

Və siyahımızı silsilə obyekti olaraq qeyd etdik.Diqqət etsəniz **next** metodu əlavə olundu

next() funksiyası vasitəsilə silsilə elementlərini əldə edək

[23]:
```
print(next(siyahı))
```

```
Python
```

[24]:
```
print(next(siyahı))
```

```
C++
```

[25]:
```
print(next(siyahı))
```

```
Java
```

[26]:
```
print(next(siyahı))
```

```
Rust
```

[27]:
```
print(next(siyahı))
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-27-13fa648a5019> in <module>()
----> 1 print(next(siyahı))

StopIteration:
```

Və elementlər bitdiyi üçün Stopİteration xətası aldıq.Yuxarıda for operatoru vasitəsilə elementləri əldə etdikdə,arxa planda silsilə obyekt hazırlanır daha sonra next() metodundan istifadə olunur.

Həmçinin next metodlarından da istifadə edə bilərik

[28]:
```
kortej = ('Azerbaijan','USA','Italy')
kortej = iter(kortej)
```

[30]:
```
print(kortej.__next__())
```

```
Azerbaijan
```

[31]:
```
print(kortej.__next__())
```

```
USA
```

[32]:
```
print(kortej.__next__())
```

```
Italy
```

[33]:
```
print(kortej.__next__())
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-33-dad4c82d3c64> in <module>()
----> 1 print(kortej.__next__())

StopIteration:
```

[ ]:
```

```

biz xətalar və istisnalar bəhsində xətaları necə önlədiyimizi öyrənmişdik.Həmin metodları və iki funksiyanı tətbiq edərək bunun qarşısını alaq

[35]:
```
kortej = ('Azerbaijan','USA','Italy')
kortej = iter(kortej)
while True:
    try:
        print(kortej.__next__())
    except StopIteration:
        break
```

```
Azerbaijan
USA
Italy
```

[68]:
```
kortej = ('Azerbaijan','USA','Italy')
kortej = iter(kortej)
while True:
    try:
        print(next(kortej))
    except StopIteration:
        break
```

```
Azerbaijan
USA
Italy
```

[76]:
```
class Iterable():
    def __init__(self):
        self.iterable = 0
    def __iter__(self):
        return self
    def __next__(self):
        if self.iterable <= 10:
            a = self.iterable
            self.iterable += 1
            return a
        else:
            raise StopIteration
```

[77]:
```
check = Iterable()
myiter = iter(check)
```

[78]:
```
for i in myiter:
    print(i)
```

```
0
```

```
1
2
3
4
5
6
7
8
9
10
```

[ ]:

0.0.2 Generasiya.Törəmələr (yield ifadəsi)

Silsilələrdən sonra törəmələr və ya generasiya adlandırdığımız ifadələri öyrənək.Python dilində ifadə etdiyimiz elementlər yaddaşda xüsusi yer tutur.Məsələn siyahılar kortejlərə nisbətən daha çox yer tutmasına görə,biz kortejləri istifadə etməyimiz daha məqsədəuyğun sayılır.Amma bu hər kod blokunda keçərli olmur.Bəzən tam siyahılara ehtiyacımız olur.Siz kiçik kodlar yazdığınızdan dolayı yaddaş məsələsinin fərqinə varmayacaqsınız.Və deyə bilərsinizki çalışdırıram anında qarşıma çıxır burda nə problem olarki.Bu həmişə belə deyil,siz proqramlaşdırma dilini öyrənirsinizsə təbiki irəlidə daha çox kod blokları yazacaqsınız və ya data analitikası,maşın öyrənmə kimi texniki funksiyalardan istifadə etsəniz,zaman və yaddaş məsələsinin fərqinə varacaqsınız.Eləcədə kiçik ölçülü minikompyuterlərdən(Asus Tinkerboard,Nvidia Jetson,Raspberry Pi,microPython) istifadə etdikdə yaddaş zaman anlayışları daha çox önəm daşıyacaq.

Generasiya etdiyimiz silsilələr yer tutmayaraq,yalnız və yalnız çağırıldığı zaman törənən obyekt nümunələridir.Nümunələrlə bu daha aydın olacaq.

[102]:
```
import sys
```

[103]:
```
def G():
    num = []
    for i in range(4):
        num.append(i)
    return num
```

[104]:
```
g=G()
```

[105]:
```
print(g)
```

```
[0, 1, 2, 3]
```

[106]:
```
print(sys.getsizeof(g))
```

```
96
```

[118]:
```
def Generasiya():
    yield 1
    yield 2
    yield 3
```

[119]:
```
generator = Generasiya()
```

[120]:
```
print(sys.getsizeof(generator))
```

```
88
```

[121]:
```
print(generator)
```

```
<generator object Generasiya at 0x00000239EC01B5C8>
```

Sadəcə ünvanını göstərdi,amma elementlərimiz hal hazırda yoxdur.Biz obyekti iterasiya etmədiyimiz müddətcə elementləri əldə edə bilməyəcəyik

[122]:
```
generator = iter(generator)
```

[123]:
```
print(dir(generator))
```

```
['__class__', '__del__', '__delattr__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__lt__', '__name__', '__ne__',
'__new__', '__next__', '__qualname__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'close',
'gi_code', 'gi_frame', 'gi_running', 'gi_yieldfrom', 'send', 'throw']
```

[124]:
```
for i in generator:
    print(i)
```

```
1
2
3
```

Elementləri ekrana çap etdirdik.generasiya etdiyimiz elementləri heç bir dəyişəndə saxlaya bilmirik.Yalnız və yalnız çağırıldıqda elementləri törədir.İkinci dəfə elementləri görmək istəsək,funksiyanı yenidən çağırmağımız lazmdır.Aşağıda bunun şahidi olacaqsınız

[ ]:
```

```

[125]:
```
for i in generator:
    print(i)
```

Çağırsaq da elementləri görə bilmədik.Qeyd etdiyim kimi,heç bir dəyişəndə saxlanılmamasıdır.Yaddaşda yer tutmur.

[127]:
```
def fib(limit):
    a, b = 0, 1
    while a < limit:
        yield a
        a, b = b, a + b
```

[128]:
```
f = fib(7)
```

[129]:
```
print(f)
```

```
<generator object fib at 0x00000239EC01BAF0>
```

[130]:
```
print(next(f))
```

```
0
```

[131]:
```
print(next(f))
```

```
1
```

[132]:
```
print(next(f))
```

```
1
```

[133]:
```
print(next(f))
```

```
2
```

[134]:
```
print(next(f))
```

```
3
```

[135]:
```
print(next(f))
```

```
5
```

[136]:
```
print(next(f))
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-136-476d6081df2c> in <module>()
----> 1 print(next(f))

StopIteration:
```

[138]:
```
for i in fib(5):
    print(i)
```

```
0
1
1
2
3
```

[139]:
```
#Building Generators With Generator Expressions
```

[140]:
```
num = [i for i in range(1,11)]
```

[141]:
```
num
```

[141]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Yuxarıda list_comprehension hazırladıq.Və dəyərləri ekranda gördük.ədəd diapazonun artıraq

[142]:
```
num = [i for i in range(10000)]
```

[155]:
```
print(sys.getsizeof(num)) #optimize memory
```

```
87624
```

Yaddaşda tutduğu həcmi ekrana çap etdirdik.İndi isə siyahılardan generasiya metodu ilə ədədlər diapazonu hazırlayaq.Bunun üçün qapalı mötərizə deyil,normal mötərizədən istifadə edəcəyik

[145]:
```
generator = (i for i in range(10000))
```

[146]:
```
print(generator)
```

```
<generator object <genexpr> at 0x00000239EC01BFC0>
```

Və generasiya etdik.Hal-hazırda elementlər mövcud deyil,çağırmadığımız müddətcə heç bir dəyişəndə saxlanılmır.Yaddaş həcminə baxaq

[147]:
```
print(sys.getsizeof(generator))
```

```
88
```

Fərqi yəqinki görürsünüz

Elementləri əldə etmək üçün isə obyekti silsilə obyektinə çevirməliyik

[148]:
```
generator = iter(generator)
```

[149]:
```
print(type(generator))
```

```
<class 'generator'>
```

[150]:
```
print(next(generator))
```

```
0
```

[151]:
```
print(next(generator))
```

```
1
```

[152]:
```
print(next(generator))
```

```
2
```

[153]:
```
print(next(generator))
```

```
3
```

[154]:
```
print(next(generator))
```

```
4
```

[ ]:
```

```