# PYTHON PROGRAMLAMA DİLİ

## iterator, generator

Dr. Öğr. Üyesi M. Ozan AKI

# ITERATOR

İterasyon, elemanların ardışık olarak sırayla işlenmesi anlamına gelir.

Iterator, bunu sağlayan mekanizmadır.

Python dilinde, liste elemanlarının for döngüsünde tek tek ele alınmasını sağlayan yapıdır.

Bir iterator oluşturmak için;

**iter()**

fonksiyonu kullanılır.

İteratör oluşturabilmek için, parametre ile geçilen yapının **iterable** olması gerekir.

```
>>> liste=['bir','iki','uc']
>>> iterator=iter(liste)
>>> type(iterator)
<class 'list_iterator'>
>>>
```

# ITERATOR

İteratör nesnesinin, listedeki bir sonraki elemanı döndüren **next** metodu bulunur.

**next(i):** Verilen i iteratörünün bir sonraki elemanını döndürür.

Eğer listenin sonuna ulaşıldıysa ve döndürecek eleman kalmadıysa;

**StopIteration**

istisna (**exception**) hatası oluşur.

```
>>> liste=['bir','iki','uc']
>>> iterator=iter(liste)
>>> print(next(iterator))
bir
>>> print(next(iterator))
iki
>>> print(next(iterator))
uc
>>> print(next(iterator))
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    print(next(iterator))
StopIteration
>>>
```

# ITERATOR

Bir **iterable** nesne oluşturmak için;

**__iter__()**

**__next__()**

Metotlarını tanımlamamız gerekir.

**__iter__** metodu ile ilk değerleri atayıp nesnenin kendisini döndürürüz

**__next__** metodunda, bir sonraki elemanın ne olacağını belirlenip döndürülür.

```
>>> class ikiser_say:
...     def __init__(self, basla, son):
...         self.sayi = basla
...         self.son = son
...     def __iter__(self):
...         return self
...     def __next__(self):
...         x=self.sayi
...         self.sayi+=2
...         if x < self.son:
...             return x
...         raise StopIteration
...
...
>>> for i in ikiser_say(10,30):
...     print(f'{i} ',end='')
...
...
10 12 14 16 18 20 22 24 26 28
>>>
```

# GENERATOR

Generator, bellekte yer işgal etmeyen bir iterator üretmek için kullanılır.

Bunun için;

**yield**

Anahtar kelimesi kullanılır. **yield** yanında üretilen değerleri bir **iterable** olarak sadece istendiğinde değer gönderir ve bunlar bellekte saklanmaz. Değerlere tekrar ulaşılmak istenirse artık bu değerlere ulaşılamaz.

```
>>> def kareler():
...     for i in range(10):
...         yield i**2
...
...
>>> func_kareler=kareler()
>>> iterator=iter(func_kareler)
>>> print(next(iterator))
0
>>> print(next(iterator))
1
>>> print(next(iterator))
4
>>> print(next(iterator))
9
>>> print(next(iterator))
16
>>>
```

# GENERATOR

Ayrıca, köşeli parantez içinde kullanılan **for** ifadesi, normal parantez ile yazılacak olursa, yine bir **generator** döndürülür.

```
>>> liste=[i for i in range(10)]
>>> type(liste)
<class 'list'>
>>>
>>> generator=(i for i in range(10))
>>> type(generator)
<class 'generator'>
>>>
```