

1. Lambda Fonksiyonu

`lambda` , Python'da anonim (isimsiz) fonksiyonlar oluşturmak için kullanılan bir anahtar kelimedir. Tek satırda bir fonksiyon tanımlamak için kullanılır. `lambda` ifadesi, `def` ile tanımlanan fonksiyonlar gibi çalışır, ancak isim verilmez ve daha sade bir yapısı vardır.

Yapısı:

```
lambda parametreler: ifade
```

Örnek:

```
# İki sayının toplamını hesaplayan lambda fonksiyonu
toplam = lambda x, y: x + y
print(toplam(5, 3)) # Çıktı: 8
```

2. Map Fonksiyonu

`map()` , bir fonksiyonu bir liste ya da başka bir iterable üzerinde çalıştırıp sonuçları döndüren bir fonksiyondur. Her bir eleman üzerinde belirli bir işlemi otomatik olarak uygular.

Yapısı:

```
map(fonksiyon, iterable)
```

Örnek:

```
# Bir listedeki sayıların karesini almak için map ve lambda kullanımı
sayilar = [1, 2, 3, 4]
kareler = list(map(lambda x: x ** 2, sayilar))
print(kareler) # Çıktı: [1, 4, 9, 16]
```

3. Zip Fonksiyonu

`zip()` , iki ya da daha fazla iterable'ı (örneğin listeleri) bir araya getirip her bir elemandan birer tane olarak bir tuple listesi oluşturur. Eşit uzunlukta olmayan iterable'larda, en kısa olanın uzunluğu kadar elemanla sınırlanır.

Yapısı:

```
zip(iterable1, iterable2, ...)
```

Örnek:

```
# İki listeyi birleştirip her bir çifti bir tuple haline getirme
isimler = ["Ali", "Veli", "Ayşe"]
yaslar = [25, 30, 22]
birlesim = list(zip(isimler, yaslar))
print(birlesim) # Çıktı: [('Ali', 25), ('Veli', 30), ('Ayşe', 22)]
```

4. Enumerate Fonksiyonu

`enumerate()`, bir iterable'ın elemanlarını indeksleriyle birlikte döndürür. Liste ya da başka bir iterable üzerinde döngü kurarken, her bir elemanın indeksine kolayca erişmeyi sağlar.

Yapısı:

```
enumerate(iterable, start=0)
```

Örnek:

```
# Bir listedeki elemanları indeksleri ile birlikte yazdırma
meyveler = ["elma", "armut", "muz"]
for indeks, meyve in enumerate(meyveler):
    print(f"{indeks}: {meyve}")
# Çıktı:
# 0: elma
# 1: armut
# 2: muz
```

5. Filter Fonksiyonu

Python'da `filter()` fonksiyonu, bir liste veya başka bir iterable (örneğin tuple, set) üzerinde belirli bir koşulu sağlayan elemanları seçmek için kullanılır. Bu, verilen bir fonksiyonu kullanarak her bir eleman üzerinde bir test yapar ve yalnızca testi geçen elemanları döndürür.

Yapısı:

```
filter(fonksiyon, iterable)
```

- `fonksiyon`: Her eleman üzerinde çalışacak ve `True` veya `False` döndürecek bir fonksiyon olmalıdır.
- `iterable`: Üzerinde işlem yapılacak liste, set, tuple gibi bir veri yapısıdır.

Fonksiyon her bir eleman üzerinde çalışır ve `True` döndüren elemanlar yeni bir iterable yapısına eklenir. Bu yeni yapı, `filter` fonksiyonunun sonucudur.

Örnek:

Bir listede sadece çift sayıları seçmek için `filter()` ve `lambda` fonksiyonlarını kullanalım.

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

# Çift sayıları filtrelemek için filter ve lambda kullanımı
cift_sayilar = list(filter(lambda x: x % 2 == 0, sayilar))

print(cift_sayilar) # Çıktı: [2, 4, 6, 8, 10]
```

Çalışma Mantığı:

- `lambda x: x % 2 == 0` ifadesi, bir sayının çift olup olmadığını kontrol eder. Eğer sayı çiftse, sonuç `True` olur ve bu sayı yeni listede yer alır.
- `filter()` ise, bu lambda fonksiyonunu `sayilar` listesi üzerinde her bir eleman için uygular ve `True` dönen elemanları seçer.

Başka Bir Örnek:

Bir string listesinde sadece 5 karakterden uzun olanları filtreleyelim:

```
kelimeler = ["elma", "armut", "muz", "kiraz", "portakal", "üzüm"]

# 5 karakterden uzun kelimeleri filtrelemek
uzun_kelimeler = list(filter(lambda x: len(x) > 5, kelimeler))

print(uzun_kelimeler) # Çıktı: ['portakal']
```

Bu örnekte `lambda x: len(x) > 5` ifadesi, her kelimenin uzunluğunu kontrol eder ve sadece 5 karakterden uzun olanlar seçilir.