

## ▼ List Comprehension

List Comprehension, mevcut bir listenin değerlerine dayalı olarak yeni bir liste oluşturmak istediğinizde daha kısa bir sözdizimi sunar.


- Biz daha öncesinden öğrendiğimiz şekilde bir boş liste tanımlayıp for döngüsü yardımıyla append (ekleme) komutu ile listemizi oluşturup çıktı alabiliriz.

```
1 l = []
```

```
1 for i in range(6):
2     l.append(i)
```

```
1 l
[0, 1, 2, 3, 4, 5]
```

```
1 list(range(6))
```

```
 [0, 1, 2, 3, 4, 5]
```

- Bunun yerine [list comprehension] şeklinde listemizi oluşturabiliriz.
- Aslında for döngüsünün çıktısını direkt list comprehension ile liste tanımlarken alabiliriz.
- List Comprehension içindeki ilk kısım çıktımızı nasıl return edeceğini gösterir. **Örneğin:** `i, i**i` vb.
- List Comprehension içindeki diğer kısım ne işlem yapılacağını gösterir. **Örneğin:** `for, range(), if` vb.
- List Comprehension ile tanımlanan bir listeyi başka bir listedeki list comprehension içindeki for döngüsünde iteratif bir yapı olarak kullanabiliriz.
- Yeni listedeki list comprehensiondaki return edilecek değerde, önceki listedeki her bir eleman için kullanılacak metotları kullanarak return edebiliriz. **Örneğin:** `i.upper()` vb.

```
1 l2 = [i for i in range(6)]
```

```
1 l2
[0, 1, 2, 3, 4, 5]
```

```
1 l2 = [i**i for i in range(6)]
```

```
1 l2
[1, 1, 4, 27, 256, 3125]
```

```
1 l2 = [i**i for i in range(10) if i%2 == 0]
```

```
1 l2
```

```
[1, 4, 256, 46656, 16777216]
```

```
1 l3 = ["deneme", "merhaba", "ben"]
```

```
1 l4 = [i for i in l3]
```

```
1 l4
```

```
['deneme', 'merhaba', 'ben']
```

## ▼ .upper() metodu

Bu metot yazıdaki harflerin hepsini büyük harfe dönüştürür.

```
1 l4 = [i.upper() for i in l3]
```

```
1 l4
```

```
['DENEME', 'MERHABA', 'BEN']
```

## ▼ Generator Object

- dir() olarak baktığımızda listeden ve diğer veri yapılarından farklı olduğunu, farklı sınıfları ve fonksiyonları geriye döndürdüğünü görebiliriz.
- Elemanlarına listede olduğu gibi for ile erişilip yazdırılabilir.

```
1 l5 = (i**i for i in range(6))
```

```
1 for i in l5:  
2     print(i)
```

```
1  
1  
4  
27  
256  
3125
```

```
1 l5
```

```
<generator object <genexpr> at 0x0000021CDA05E580>
```

```
1 type(l5)
```

```
generator
```

```
1 dir(l5)
```

```
['__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']
```

```
1 l5[0]
```

**ERROR SEBEBİ:** İndis çağırılmadı diyor çünkü hafızada fazla yer kaplanmasın diye fonksiyonu biz ihtiyaç duyduğumuzda çağırıyor.

▼ List Comprehension ile Generator Object arasındaki fark:

- List Comprehension'da greedy yaklaşımı vardır. Bu yaklaşım, o esnada hafızada yer edinin elemanları çalıştırıyor eğer çok büyük bir veriyse hafızada çok büyük bir alan kaplıyor. Yani biz onu çağırdığımızda onu hafızadan çağırıyor ve çıktısını alıyoruz.
- Generator Object'te ise lazy evaluation tekniği kullanılıyor. Çok büyük verilerde hafızada fazla yer barındırmamasın, daha hızlı çalışsın diye birnevi (zipliyor) sıkıştırıyor. Hafızada fonksiyonu çağırıyor. Fonksiyonu biz ihtiyaç duyduğumuzda çağırıyor.
- List comprehension'da if yapıları kullanılıyor olması çok kullanışlı fakat büyük verilerle çalışırken generator object kullanmak daha faydalı olabilir.

```
1 l4 = [i**i for i in range(6)]
2 l5 = (i**i for i in range(6))
```

```
1 def deneme(x):
2     print(f"{x} karesi: ")
3     return x**2
```

```
1 sayi = [1,5,8]
```

```
1 l = [deneme(i) for i in sayi]
2 print("Liste")
3 for i in l:
4     print(i)
```

```
1 karesi:
5 karesi:
8 karesi:
Liste
1
25
64
```

```
1 g = (deneme(i) for i in sayi)
2 print("Generator")
3 for i in g:
4     print(i)
```

```
Generator
1 karesi:
1
5 karesi:
25
8 karesi:
64
```

```
1 [i**2 for i in range(100) if i%2 == 0]
```

```
[0,
4,
16,
36,
64,
```

```
100,  
144,  
196,  
256,  
324,  
400,  
484,  
576,  
676,  
784,  
900,  
1024,  
1156,  
1296,  
1444,  
1600,  
1764,  
1936,  
2116,  
2304,  
2500,  
2704,  
2916,  
3136,  
3364,  
3600,  
3844,  
4096,  
4356,  
4624,  
4900,  
5184,  
5476,  
5776,  
6084,  
6400,  
6724,  
7056,  
7396,  
7744,  
8100,  
8464,  
8836,  
9216,  
9604]
```

## ▼ yield keywordu:

Bir fonksiyonda yield keywordu kullanarak iteratif bir yapı tanımladığımızı ifade ediyoruz.

```
1 def f():  
2     yield 10  
3     yield 20  
4     yield 30
```

```
1 a = f()
```

```
1 a
```

```
<generator object f at 0x0000021CDA0655F0>
```

## ▼ next() generic fonksiyonu:

- Fonksiyondaki her bir elemanı sırayla çağırır.
- Her çağırışımızda bir sonraki elemanı verir.
- İçindeki eleman sayısı bittiğinde StopIteration ile karşılaşırız. Bu da eleman sayısının bittiği anlamına gelir.
- Bir for döngüsüyle her elemana ulaşabiliriz ve sonra next() olarak çağırıldığında StopIteration ile karşılaşırız. Çünkü her eleman çıktı olarak geri döndürülmüştür ve eleman sayısının bittiğini bildirir.

```
1 next(a)
```

```
10
```

```
1 next(a)
```

```
20
```

```
1 next(a)
```

```
30
```

```
1 next(a)
```

**ERROR SEBEBİ:** StopIteration eleman sayısının bittiği anlamına gelir.

```
1 a1 = f()  
2  
3 for i in a1:  
4     print(i)
```

```
10
```

```
20
```

```
30
```

```
1 next(a1)
```

**ERROR SEBEBİ:** StopIteration eleman sayısının bittiği anlamına gelir.

## ▼ .getsizeof() yöntemi:

- Sisteme özgü bir yöntemdir ve onu kullanmak için sys modülünü import etmemiz gerekmektedir.
- Ne kadar alan kapladığını gösterir.
- byte türünden bir değeri döndürür.
- List Comprehension ile Generator Object'in hafızada kapladığı alanlara bakarsak neden generator object kullanmanın daha mantıklı olduğunu anlayabiliriz.

```
1 import sys
```

```
1 l = [i*2 for i in range(1000)]  
2 g = (i*2 for i in range(1000))
```

```
1 sys.getsizeof(l)
```

```
8856
```

```
1 sys.getsizeof(g)
```

```
112
```

```
1 l[5]
```

```
10
```

```
1 g[5]
```

**ERROR SEBEBİ:** Yapı tanımlı değil. Önce o yapıyı tanımlayacağız ki çağırabilelim.

```
1 l2 = [1,2,3,4,5]
```

```
1 l2
```

```
[1, 2, 3, 4, 5]
```

```
1 def f(x):  
2     return x%2 != 0
```

## ▼ filter() fonksiyonu:

- Generic bir fonksiyondur.

- Her bir elemanı alıp onu bir süzgeçten geçirip ilgili değerleri return etmek için kullanılıyor.
- İki parametre alır:
  - Birinci parametreye filtreleme yapacağı fonksiyondur.
  - İkinci parametre ise bu fonksiyonu çalıştıracığı iteratiftir.

```
1 list(filter(f, l1))
```

**ERROR SEBEBİ:** filter() fonksiyonunda ikinci parametreye iteratiftir yapılmadığı için hata verir.

```
1 list(filter(f, l2))
```

```
[1, 3, 5]
```

```
1 [i for i in l2 if f(i)]
```

```
[1, 3, 5]
```

**Yukarıdaki gibi** filter() gibi bir fonksiyon ile List Comprehension'daki if ile aynı işlemi gerçekleştirebiliriz.

## ▼ lambda kullanımı:

def() şeklinde tanımladığımız fonksiyonumuzu lambda notasyonu ile daha kısa bir şekilde tanımlayabiliriz.

def adı(parametre):

```
    return ifade
```

lambda parametreler:ifade

```
1 list(filter(lambda x: x%2 != 0, l2))
```

```
[1, 3, 5]
```

## ▼ map fonksiyonu:

- Generic bir fonksiyondur.
- Her bir elemanı alıp her bir eleman üzerinde işlem yaparak kullanılıyor.
- İlk parametresi bir fonksiyon olmak zorundadır.
- İkinci parametresi de iterasyon olmalıdır.



```
1 tuple(map(lambda x: x**2 , l2))  
  
(1, 4, 9, 16, 25)
```

```
1 [i**2 for i in l2]  
  
[1, 4, 9, 16, 25]
```

**Bu aslında** list comprehension ile yapılan birnevi map işlemidir.

```
1 [i**2 for i in l2 if i%2 != 0]  
  
[1, 9, 25]
```

**iften sonraki kısım** filter işlemidir.

```
1 tuple(map(lambda x: x**2, filter(lambda x: x%2 != 0, l2)))  
  
(1, 9, 25)
```

## ▼ ÖRNEK BİR SORU:

Aşağıdaki gibi tanımlanan bir l2 listesinin tek sayıların karesinin alındığı, çift sayıların ise aynı kalıp l2 listesindeki her değerin karşılığının yazıldığı bir kod yazınız.

```
1 l2 = [1,2,3,4,5,6]
```

```
1 l2  
  
[1, 2, 3, 4, 5, 6]
```

## ▼ Örnek sorunun çözümü:

```
1 [i**2 if i%2!=0 else i for i in l2]  
  
[1, 2, 9, 4, 25, 6]
```

- [i\*\*2 if i%2!=0 else i for i in l2 ] map kısmı, filter() gibi gözükebilir ama değildir.
- for i in l2 filter kısmı (filtreleme)

```
1 l3 = [i**2 if i%2!=0 else i for i in l2]
```

```
1 l2  
  
[1, 2, 3, 4, 5, 6]
```

```
1 l3
```

```
[1, 2, 9, 4, 25, 6]
```

## ▼ zip metodu:

- Bu metot, elimizde iki vektör varsa her iki vektörün de ilk elemanını alıp diğer elamanlara ata ve bunları kullan demektir.
- Yeni atanan değerlerle istediğimiz işlemi yapabiliriz.

```
1 for i,j in l2,l3:  
2     print(i,j)
```

**ERROR SEBEBİ:** Bu hata, değişken sayısı değer sayısı ile eşleşmediğinde ortaya çıkar. Eşitsizliğin bir sonucu olarak, Python hangi değerlerin hangi değişkenlere atanacağını bilmez.

```
1 for i,j in zip(l2,l3):  
2     print(i,j)
```

```
1 1  
2 2  
3 9  
4 4  
5 25  
6 6
```

```
1 [i+j for i,j in zip(l2,l3)]
```

```
[2, 4, 12, 8, 30, 12]
```

## ▼ İki boyutlu liste tanımlama:

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

```
1 l
```

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

**Aşağıdaki gibi** listenin satırlarını matris şeklinde yazdırabiliriz.

```
1 for i in l:  
2     for j in i:
```

```
3     print(j, end=' ')
4     print()
```

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

## ▼ dict kullanımı:

- Yani sözlükler
- Python'daki sözlük, bir öge olarak yalnızca tek bir değer tutan diğer veri türlerinin aksine, bir harita gibi veri değerlerini depolamak için kullanılan bir anahtar değerler koleksiyonudur.
- key:value pair dediğimiz bir yapıya sahiptir.
- unordered yani sırasızdır, içindeki elemanlar sıralı değildir. İçindeki değerlerin sırasına göre elemanlara erişmek mümkün değildir.
- Slicing uygulanmıyor!
- **Slicing:** Bu sözdizimini kullanarak bir dizi karakter döndürebilirsiniz. Dizenin bir bölümünü döndürmek için başlangıç dizinini ve bitiş dizinini iki nokta üst üste ile ayırarak belirtin.
- key değerleri immutable veri yapılarında olmalıdır. Yani int, float, str, tuple olabilir ama list olamaz.
- generic kendine ait fonksiyonları vardır.
- value değerinin tekrar etmesi problem değildir yani farklı key'lere aynı value'lar atanabilir.
- key değerleri aynı olmamalıdır. Eğer aynı olursa override eder. Bu durumda ilk key değerini kaldırıp yeni key değerini ekleyecektir.

```
1 a = {}
```

```
1 a
```

```
{}
```

```
1 type(a)
```

```
dict
```

```
1 dir(a)
```

```
['__class__',
 '__class_getitem__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattr__',
 '__getitem__',
```

```
__gt__',
__hash__',
__init__',
__init_subclass__',
__ior__',
__iter__',
__le__',
__len__',
__lt__',
__ne__',
__new__',
__or__',
__reduce__',
__reduce_ex__',
__repr__',
__reversed__',
__ror__',
__setattr__',
__setitem__',
__sizeof__',
__str__',
__subclasshook__',
'clear',
'copy',
'fromkeys',
'get',
'items',
'keys',
'pop',
'popitem',
'setdefault',
'update',
'values']
```

```
1 a={1:'A', 2:'B', 3:'C'}
```

```
1 a
```

```
{1: 'A', 2: 'B', 3: 'C'}
```

```
1 a={1.5:'A', 2:'B', 3:'C'}
```

```
1 a
```

```
{1.5: 'A', 2: 'B', 3: 'C'}
```

```
1 a={'deneme':'A', 2:'B', 3:'C'}
```

```
1 a
```

```
{'deneme': 'A', 2: 'B', 3: 'C'}
```

```
1 a={'key':'B', 2:'B', 3:'C'}
```

```
1 a
```

```
{'key': 'B', 2: 'B', 3: 'C'}
```

```
1 a={1:'A', 2:'B', 1:'C'}
```

```
1 a
```

```
{1: 'C', 2: 'B'}
```

```
1 a={1:'A', 2:'B', 3:[1,2,3]}
```

```
1 a
```

```
{1: 'A', 2: 'B', 3: [1, 2, 3]}
```

```
1 a={1:'A', 2:{1:'a',2:'b'}, 3:'C'}
```

```
1 a
```

```
{1: 'A', 2: {1: 'a', 2: 'b'}, 3: 'C'}
```

```
1 a= {(1,2):'A', 2:'B', 3:'C'}
```

```
1 a
```

```
{(1, 2): 'A', 2: 'B', 3: 'C'}
```

```
1 a= {[1,2]:'A', 2:'B', 3:'C'}
```

**ERROR SEBEBİ:** List yapısının immutable olmamasıdır.

```
1 a
```

```
{(1, 2): 'A', 2: 'B', 3: 'C'}
```

```
1 len(a)
```

```
3
```

▼ true-false return:

**true return:**

```
1 if a:
2     print("eleman var")
3 else:
4     print("eleman yok")
```

eleman var

## ▼ .clear() kullanımı:

İçindeki bütün değerleri boşaltır.

```
1 a.clear()
```

```
1 a
```

```
{}
```

## false return:

```
1 if a:
2     print("eleman var")
3 else:
4     print("eleman yok")
```

eleman yok

```
1 a={1:'A', 2:'B', 3:'C'}
```

```
1 a
```

```
{1: 'A', 2: 'B', 3: 'C'}
```

```
1 for i in a:
2     print(i)
```

```
1
2
3
```

## ▼ .items() metodu:

- Her bir elemanı görüntülemek için kullanılır.
- Her bir elemanı tuple olarak return eder.
- Hem key hem value değerlerini çıktı olarak alabiliriz.
- Key değerleri de string değerler olabilir hem key hem de value değerler yine çıktı olarak görünür.

```
1 for i in a.items():
2     print(i)
```

```
(1, 'A')
(2, 'B')
(3, 'C')
```

```
1 for i,j in a.items():
2     print(i,j)
```

```
1 A
2 B
3 C
```

```
1 a={'1':'A', '2': 'B','3':'C'}
```

```
1 a
```

```
{'1': 'A', '2': 'B', '3': 'C'}
```

```
1 for i,j in a.items():
2     print(i,j)
```

```
1 A
2 B
3 C
```

```
1 a['1']
```

```
'A'
```

```
1 a['1'] = 100
```

```
1 a
```

```
{'1': 100, '2': 'B', '3': 'C'}
```

```
1 a['deneme']
```

**ERROR SEBEBİ:** Hiç olmayan bir değeri çıktı olarak alamayız.

```
1 a['deneme'] = 1234
```

```
1 a
```

```
{'1': 100, '2': 'B', '3': 'C', 'deneme': 1234}
```

**Ama** hiç olmayan bir değere at ama yapılabilir.

## ▼ del işlemi:

Girilen değeri siler.

```
1 del a['1']
```

```
1 a
```

```
{'2': 'B', '3': 'C', 'deneme': 1234}
```

## ▼ .pop() metodu:

Bu metot la yazılan değeri çıkarır.

```
1 a.pop('deneme')
```

```
1234
```

```
1 a
```

```
{'2': 'B', '3': 'C'}
```

## ▼ .get() metodu:

- Bu metot la değeri alabiliriz.
- Eğer olmayan bir değeri almaya kalkarsak hat a ile karşılaşmayız değer olmadığı için çıktı almayız.

```
1 a.get('3')
```

```
'C'
```

```
1 a.get('45')
```

## ▼ true-false dönüşü alma:

```
1 '3' in a
```

```
True
```



```
1 '55' in a
```

```
False
```

## ▼ .keys() metodu:

Sadece key değerlerini döndürebiliriz.

```
1 for i in a.keys():  
2     print(i)
```

```
2  
3
```

## ▼ .values() metodu:

Sadece value değerlerini döndürebiliriz.

```
1 for i in a.values():  
2     print(i)
```

```
B  
C
```

## ▼ shallow copy (sığ kopyalama)

- Atama a ile b arasındaki ilişkiyi koparmaz.
- a üzerinde yapılan değişiklikler b üzerine yansır.
- b farklı bir yerde tutulmasına rağmen a'nın bir temsilini gösterir.
- b üzerinde yapılan değişiklikler hataya sebep olabilir çünkü a üzerinde etkilidir.
- Riskli bir durumdur.

```
1 b=a.keys()
```

```
1 type(b)
```

```
dict_keys
```

```
1 id(b)
```

```
2322939154000
```

```
1 id(a)
```

```
2322940254144
```

```
1 for i in b:  
2     print(i)
```

```
2  
3
```

```
1 a['3']=123
```

```
1 a
```

```
{'2': 'B', '3': 123}
```

```
1 b
```

```
dict_keys(['2', '3'])
```

```
1 a['5']=123
```

```
1 a
```

```
{'2': 'B', '3': 123, '5': 123}
```

```
1 b
```

```
dict_keys(['2', '3', '5'])
```

## ▼ deep copy (derin kopyalama) işlemi

- a ve b ilişkisi kopmuş olur.
- a üzerinde yapılan değişiklikler b'yi etkilemez.

```
1 b = a.copy()
```

```
1 b
```

```
{'2': 'B', '3': 123, '5': 123}
```

```
1 a['1234'] = 123
```

```
1 a
```

```
{'2': 'B', '3': 123, '5': 123, '1234': 123}
```

```
1 b
```

```
{'2': 'B', '3': 123, '5': 123}
```

## ▼ eşittir veya eşit değildir operatörleri

```
1 a == b
```

```
False
```

```
1 a != b
```

```
True
```

## ▼ küçüktür ve büyüktür operatörleri

- Sözlükte desteklenmez.
- İçindeki değerin ne olduğu önemsenmeksizin sözlük tarafından desteklenmezler.

```
1 a < b
```

```
1 a={1:1,2:2,3:4}
```

```
2 b={1:1,2:2,3:4}
```

```
1 a
```

```
{1: 1, 2: 2, 3: 4}
```

```
1 b
```

```
{1: 1, 2: 2, 3: 4}
```

```
1 a < b
```

## ▼ GÜZEL BİR ÖRNEK

**.split()** Bir string değeri, her kelimenin bir list e ögesi olduğu bir listeye ayırır.

- metini {} şeklinde tanımlarsak set tipinde olur ve kod hat a verir.
- metin () şeklinde tanımlarsak string olur ve hat a vermez.

```
1 metin = ("Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ip  
2  
3 a={}  
4  
5 for kelime in metin.split():  
6     if kelime in a:  
7         a[kelime] += 1  
8     else:  
9         a[kelime] = 1
```

1 a

```
{'Lorem': 4,  
'Ipsum': 3,  
'is': 1,  
'simply': 1,  
'dummy': 2,  
'text': 2,  
'of': 4,  
'the': 6,  
'printing': 1,  
'and': 3,  
'typesetting': 1,  
'industry.': 1,  
'has': 2,  
'been': 1,  
'industry's': 1,  
'standard': 1,  
'ever': 1,  
'since': 1,  
'1500s.': 1,  
'when': 1,  
'an': 1,  
'unknown': 1,  
'printer': 1,  
'took': 1,  
'a': 2,  
'galley': 1,  
'type': 2,  
'scrambled': 1,  
'it': 1,  
'to': 1,  
'make': 1,  
'specimen': 1,  
'book.': 1,  
'It': 2,  
'survived': 1,  
'not': 1,  
'only': 1,  
'five': 1,  
'centuries.': 1,  
'but': 1,  
'also': 1,  
'leap': 1,  
'into': 1,  
'electronic': 1,  
'typesetting.': 1,  
'remaining': 1,  
'essentially': 1,  
'unchanged.': 1,
```

```
'was': 1,  
'popularised': 1,  
'in': 1,  
'1960s': 1,  
'with': 2,  
'release': 1,  
'Letraset': 1,  
'sheets': 1,  
'containing': 1,  
'massages': 1
```

```
1 len(a)
```

```
69
```

## ▼ .update() kullanımı:

- Kullanılan stringin kaç kere kullanıldığı değiştirilebilir.
- Hiç kullanılmayan bir stringi de ekleyebiliriz.

```
1 a.update('Lorem': 6)
```

**ERROR SEBEBİ:** ({}) şeklinde tanımlanmalıdır ya da atama işlemi (...=...) bu şekilde gerçekleşmelidir.

```
1 a.update({'Lorem': 6})
```

```
1 a
```

```
{'Lorem': 6,  
'Ipsum': 3,  
'is': 1,  
'simply': 1,  
'dummy': 2,  
'text': 2,  
'of': 4,  
'the': 6,  
'printing': 1,  
'and': 3,  
'typesetting': 1,  
'industry.': 1,  
'has': 2,  
'been': 1,  
'industry's': 1,  
'standard': 1,  
'ever': 1,  
'since': 1,  
'1500s.': 1,  
'when': 1,  
'an': 1,  
'unknown': 1,  
'printer': 1,  
'took': 1,
```

```
'a': 2,  
'galley': 1,  
'type': 2,  
'scrambled': 1,  
'it': 1,  
'to': 1,  
'make': 1,  
'specimen': 1,  
'book.': 1,  
'It': 2,  
'survived': 1,  
'not': 1,  
'only': 1,  
'five': 1,  
'centuries.': 1,  
'but': 1,  
'also': 1,  
'leap': 1,  
'into': 1,  
'electronic': 1,  
'typesetting.': 1,  
'remaining': 1,  
'essentially': 1,  
'unchanged.': 1,  
'was': 1,  
'popularised': 1,  
'in': 1,  
'1960s': 1,  
'with': 2,  
'release': 1,  
'Letraset': 1,  
'sheets': 1,  
'containing': 1,  
''': 1
```

```
1 a.update({'deneme':4})
```

```
1 a
```

```
{'Lorem': 6,  
'Ipsum': 3,  
'is': 1,  
'simply': 1,  
'dummy': 2,  
'text': 2,  
'of': 4,  
'the': 6,  
'printing': 1,  
'and': 3,  
'typesetting': 1,  
'industry.': 1,  
'has': 2,  
'been': 1,  
'industry's': 1,  
'standard': 1,  
'ever': 1,  
'since': 1,  
'1500s.': 1,  
'when': 1,  
'an': 1,  
'unknown': 1,
```

```
'printer': 1,  
'took': 1,  
'a': 2,  
'galley': 1,  
'type': 2,  
'scrambled': 1,  
'it': 1,  
'to': 1,  
'make': 1,  
'specimen': 1,  
'book.': 1,  
'It': 2,  
'survived': 1,  
'not': 1,  
'only': 1,  
'five': 1,  
'centuries,': 1,  
'but': 1,  
'also': 1,  
'leap': 1,  
'into': 1,  
'electronic': 1,  
'typesetting,': 1,  
'remaining': 1,  
'essentially': 1,  
'unchanged.': 1,  
'was': 1,  
'popularised': 1,  
'in': 1,  
'1960s': 1,  
'with': 2,  
'release': 1,  
'Letraset': 1,  
'sheets': 1,  
'containing': 1,  
'passages': 1
```

```
1 a.update(denemee=5)
```

```
1 a
```

```
{'Lorem': 6,  
'Ipsum': 3,  
'is': 1,  
'simply': 1,  
'dummy': 2,  
'text': 2,  
'of': 4,  
'the': 6,  
'printing': 1,  
'and': 3,  
'typesetting': 1,  
'industry.': 1,  
'has': 2,  
'been': 1,  
'industry's': 1,  
'standard': 1,  
'ever': 1,  
'since': 1,  
'1500s,': 1,  
'when': 1,
```

```
'an': 1,  
'unknown': 1,  
'printer': 1,  
'took': 1,  
'a': 2,  
'galley': 1,  
'type': 2,  
'scrambled': 1,  
'it': 1,  
'to': 1,  
'make': 1,  
'specimen': 1,  
'book.': 1,  
'It': 2,  
'survived': 1,  
'not': 1,  
'only': 1,  
'five': 1,  
'centuries.': 1,  
'but': 1,  
'also': 1,  
'leap': 1,  
'into': 1,  
'electronic': 1,  
'typesetting.': 1,  
'remaining': 1,  
'essentially': 1,  
'unchanged.': 1,  
'was': 1,  
'popularised': 1,  
'in': 1,  
'1960s': 1,  
'with': 2,  
'release': 1,  
'Letraset': 1,  
'sheets': 1,  
'containing': 1,  
'passages.': 1.
```

```
1 c={i:j for i, j in a.items()}
```

```
1 c
```

```
{'Lorem': 6,  
'Ipsum': 3,  
'is': 1,  
'simply': 1,  
'dummy': 2,  
'text': 2,  
'of': 4,  
'the': 6,  
'printing': 1,  
'and': 3,  
'typesetting': 1,  
'industry.': 1,  
'has': 2,  
'been': 1,  
'industry's': 1,  
'standard': 1,  
'ever': 1,  
'since': 1,
```



```
'1500s.': 1,  
'when': 1,  
'an': 1,  
'unknown': 1,  
'printer': 1,  
'took': 1,  
'a': 2,  
'galley': 1,  
'type': 2,  
'scrambled': 1,  
'it': 1,  
'to': 1,  
'make': 1,  
'specimen': 1,  
'book.': 1,  
'It': 2,  
'survived': 1,  
'not': 1,  
'only': 1,  
'five': 1,  
'centuries.': 1,  
'but': 1,  
'also': 1,  
'leap': 1,  
'into': 1,  
'electronic': 1,  
'typesetting.': 1,  
'remaining': 1,  
'essentially': 1,  
'unchanged.': 1,  
'was': 1,  
'popularised': 1,  
'in': 1,  
'1960s': 1,  
'with': 2,  
'release': 1,  
'Letraset': 1,  
'sheets': 1,  
'containing': 1,  
'passages.': 1.
```

```
1 d={i: j**2 for i, j in a.items()}
```

```
1 d
```

```
{'Lorem': 36,  
'Ipsum': 9,  
'is': 1,  
'simply': 1,  
'dummy': 4,  
'text': 4,  
'of': 16,  
'the': 36,  
'printing': 1,  
'and': 9,  
'typesetting': 1,  
'industry.': 1,  
'has': 4,  
'been': 1,  
'industry's': 1,  
'standard': 1,
```

```
'ever': 1,  
'since': 1,  
'1500s,': 1,  
'when': 1,  
'an': 1,  
'unknown': 1,  
'printer': 1,  
'took': 1,  
'a': 4,  
'galley': 1,  
'type': 4,  
'scrambled': 1,  
'it': 1,  
'to': 1,  
'make': 1,  
'specimen': 1,  
'book.': 1,  
'It': 4,  
'survived': 1,  
'not': 1,  
'only': 1,  
'five': 1,  
'centuries,': 1,  
'but': 1,  
'also': 1,  
'leap': 1,  
'into': 1,  
'electronic': 1,  
'typesetting,': 1,  
'remaining': 1,  
'essentially': 1,  
'unchanged.': 1,  
'was': 1,  
'popularised': 1,  
'in': 1,  
'1960s': 1,  
'with': 4,  
'release': 1,  
'Letraset': 1,  
'sheets': 1,  
'containing': 1,  
'passages,': 1,
```

```
1 x={'A':[1,2,3,4], 'B':[5,6,8,7]}
```

```
1 x
```

```
    {'A': [1, 2, 3, 4], 'B': [5, 6, 8, 7]}
```

```
1 {i: sum(j)/len(j) for i,j in x.items()}
```

```
    {'A': 2.5, 'B': 6.5}
```

## ▼ set kullanımı:

- Kümelerdir.

- Sırasızdır.
- unique değerler barındırır yani veri tekrarlamaz.
- Sadece immutable veri yapılarını destekler. (int, str, float, tuple gibi)
- Bu yüzden liste kullanılamaz.
- indexing, slicing mekanizmalarını desteklemez.
- { ' ' } şeklinde tanımlanmalıdır.
- Kümelerle ilgili tüm işlemler ve fonksiyonlar kullanılabilir.

```
1 s={}
```

```
1 type(s)
```

```
dict
```

```
1 s={'a'}
```

```
1 type(s)
```

```
set
```

```
1 dir(s)
```

```
['_and__',  
 '_class__',  
 '_class_getitem__',  
 '_contains__',  
 '_delattr__',  
 '_dir__',  
 '_doc__',  
 '_eq__',  
 '_format__',  
 '_ge__',  
 '_getattribute__',  
 '_gt__',  
 '_hash__',  
 '_iand__',  
 '_init__',  
 '_init_subclass__',  
 '_ior__',  
 '_isub__',  
 '_iter__',  
 '_ixor__',  
 '_le__',  
 '_len__',  
 '_lt__',  
 '_ne__',  
 '_new__',  
 '_or__',  
 '_rand__',  
 '_reduce__',  
 '_reduce_ex__',  
 '_repr__',  
 '_ror__',  
 '_rsub__']
```

```
'__rxor__',
'__setattr__',
'__sizeof__',
'__str__',
'__sub__',
'__subclasshook__',
'__xor__',
'add',
'clear',
'copy',
'difference',
'difference_update',
'discard',
'intersection',
'intersection_update',
'isdisjoint',
'issubset',
'issuperset',
'pop',
'remove',
'symmetric_difference',
'symmetric_difference_update',
'union',
'update']
```

```
1 s={'a', 1, 2.5, (1,2)}
```

```
1 s
```

```
{(1, 2), 1, 2.5, 'a'}
```

```
1 s={'a', 1, 2.5, (1,2), [1,2,3]}
```

**ERROR SEBEBİ:** Sadece immutable veri yapılarını destekler. (int, str, float, tuple gibi) Bu yüzden liste kullanılamaz.

```
1 s={'a', 1, 2.5, (1,2), {1:1,2:2}}
```

**ERROR SEBEBİ:** Sadece immutable veri yapılarını destekler. (int, str, float, tuple gibi) Bu yüzden sözlük veri yapısı desteklenmediği için kullanılamaz.

```
1 s={'a', 1, 2.5, (1,2), {1,2,3}}
```

**ERROR SEBEBİ:** set kendi veri yapısını da içerisinde kullanımlarını desteklemez. Sadece immutable veri yapılarını destekler.

```
1 s={1,2,3,4,5}
```

```
1 len(s)
```

```
5
```

```
1 3 in s
```

```
True
```

```
1 l = list(range(10)) + list(range(5))
```

## ▼ Listenin set yapısına dönüşümü:

- Otomatik olarak dönüştürebiliriz.
- Kümeye dönüşüm gerçekleşeceğinden tekrarlanan elemanlar yalnızca bir defa yazılır.

```
1 l
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4]
```

```
1 s1 = set(l)
```

```
1 s1
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
1 s[0]
```

**ERROR SEBEBİ:** indexing kullanımı yoktur.

```
1 s[0:1]
```

**ERROR SEBEBİ:** slicing kullanımı yoktur.

```
1 s
```

```
{1, 2, 3, 4, 5}
```

```
1 s1
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

## ▼ Karşılaştırma operatörleriyle true-false döndürme

```
1 s == s1
```

False

```
1 s != s1
```

True

```
1 s < s1
```

True

```
1 s > s1
```

False

## ▼ .issubset() kullanımı:

- s kümesi s1'in bir alt kümesi midir? şeklinde kullanılır.
- true-false return edilir.

```
1 s.issubset(s1)
```

True

## ▼ .issuperset() kullanımı:

- set(string1).issuperset(string2) şeklindeki bir gösterimde string2, string1'de bulunuyor mu? şeklinde kullanılır.
- true-false return edilir.

```
1 set('ben devam ederken okula python dersine girerken konuyu dinlerken').issuperset('ders')
```

True

## ▼ Birleştirme işlemi:

```
1 s | s1
```

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

## ▼ .union() yani birleştirme işlemi

```
1 s.union(s1)

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

## ▼ Kesişim işlemi:

```
1 s & s1

{1, 2, 3, 4, 5}
```

## ▼ .intersection() yani kesişim işlemi

```
1 s.intersection(s1)

{1, 2, 3, 4, 5}
```

## ▼ .difference() fark alma işlemi

```
1 s1.difference(s)

{0, 6, 7, 8, 9}
```

## ▼ .add() veri ekleme işlemi

Sırasız gerçekleşir.

```
1 s.add(10)
```

```
1 s

{1, 2, 3, 4, 5, 10}
```

## ▼ .pop() işlemi

Sırasız olarak çalıştığı için pop() istediği elemanı alabilir.

```
1 s.pop()
```

```
1 s
```

```
{2, 3, 4, 5, 10}
```

```
1 s.pop()
```

```
2
```

```
1 s
```

```
{3, 4, 5, 10}
```

```
1 s.pop()
```

```
3
```

```
1 s
```

```
{4, 5, 10}
```

## ▼ .clear() işlemi

Bütün set i boşaltabiliriz.

```
1 s.clear()
```

```
1 s
```

```
set()
```

## ▼ .isdisjoint() metodu

.isdisjoint() yöntemi, öğelerin hiçbirisi her iki kümede de yoksa True döndürür, aksi takdirde False döndürür.

```
1 {1,2,3,4}.isdisjoint({0,5,7})
```

```
True
```

## ▼ Fark alma işlemi

Kümelerde olduğu gibi, ikinci kümede olmayıp ilk kümede olan elemanları yazdırır.

```
1 {1,2,3,4} - {0,5,7,2}
```



```
{1, 3, 4}
```

## ▼ Birleşim işlemi

İki kümedeki elemanların hepsini yazdırır, tekrar eden elemanlar bir kez yazılır.

```
1 {1,2,3,4} ^ {0,5,7,2}
```

```
{0, 1, 3, 4, 5, 7}
```

## ▼ .discard() işlemi

- Belirtilen öğeyi kümeden kaldırır.
- Olmayan bir öğeyi kaldırmak isterseniz hata oluşmaz.
- Bu yöntem `.remove()` yönteminden farklıdır. Çünkü belirtilen öğe mevcut değilse `.remove()` yöntemi bir hataya neden olur.

```
1 kume = {1,2,3,4}
```

```
1 kume.discard(4)
2 kume
```

```
{1, 2, 3}
```

```
1 kume.discard(5)
2 kume
```

```
{1, 2, 3}
```

```
1 kume.remove(4)
```

**ERROR SEBEBİ:** Belirtilen öğe mevcut değilse `.remove()` yöntemi bir hataya neden olur.

## ▼ set ile dict kullanımı arasındaki fark:

```
1 l=[1,2,3,4,5]
2 c={i for i in l if i%2 == 0}
```

```
1 type(c)
```

```
set
```

```
1 l=[1,2,3,4,5]
2 c={i:i for i in l if i%2 == 0}
```

```
1 type(c)
```

```
dict
```

