



Python for Data Science  
Çalışma Dökümanı  
Recep Aydoğdu

## İçindekiler

Data Science Kullanılan Alanlar .....	5
Data Science Proje Döngüsü.....	5
Python Programlama.....	6
Temel Hareketler.....	6
Integer, Float ve String .....	6
Integer .....	6
Float.....	6
String .....	6
Type .....	6
String Metodları .....	6
len() .....	6
upper() & lower().....	6
isupper() & islower().....	7
replace() .....	7
strip() .....	7
dir() .....	7
capitalize() .....	7
title() .....	7
Substring.....	7
Type Dönüşümleri .....	8
print() fonksiyonu .....	8
Veri Yapıları (Data Types) .....	8
Listeler .....	8
Liste Elemanlarına Ulaşma .....	9
Liste İçeri Type Sorgulama .....	9
Liste elemanlarını değiştirme .....	9
Listeye eleman ekleme .....	9
Listeden eleman silme .....	9
append ve remove metodları.....	10
insert metodu .....	10
pop metodu .....	10
count metodu .....	10
copy metodu .....	10
extend metodu .....	11
index metodu .....	11

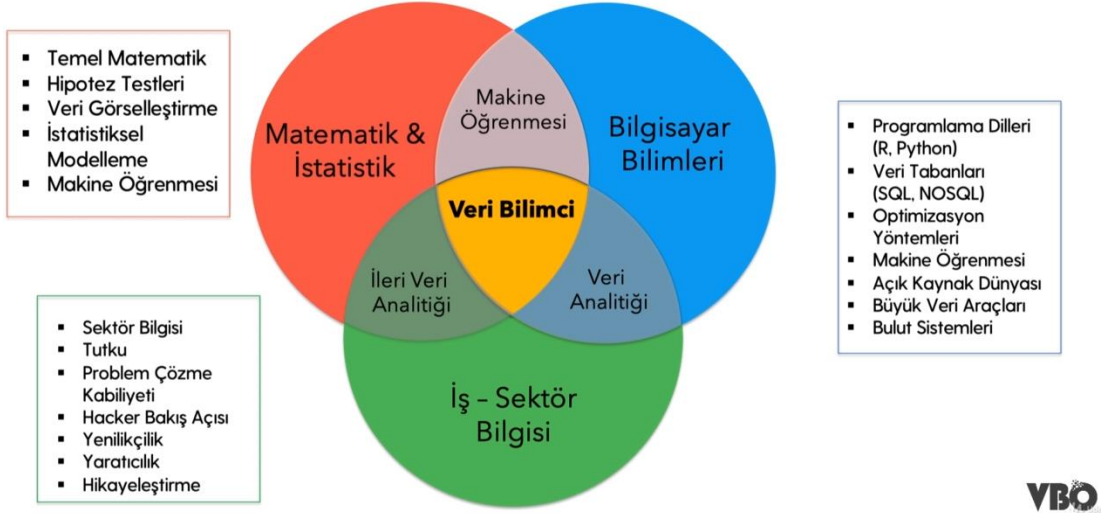
reverse metodu .....	11
sort metodu .....	11
clear metodu .....	11
Tuple (Demet).....	12
Tuple Oluşturma.....	12
Eleman İşlemleri .....	12
Dictionary (Sözlük).....	12
Dictionary Nedir?.....	12
Dictionary Oluşturma .....	12
Eleman Seçme İşlemleri.....	13
Eleman Ekleme & Değişirme .....	13
Sets (Kümeler) .....	14
Set Oluşturma.....	14
Set'lere eleman ekleme ve çıkarma işlemleri.....	15
Set'lerde Fark İşlemleri.....	17
Set'lerde Kesişim ve Birleşim İşlemleri .....	17
Set'lerde Sorgu İşlemleri .....	18
Veri Yapıları Özet .....	18
Fonksiyonlar / Karar-Kontrol Yapıları / Döngüler .....	19
Fonksiyon Nedir?.....	19
Matematiksel İşlemler .....	19
Üs Alma.....	19
Fonksiyon Nasıl Yazılır ?.....	19
Bilgi Notuyla Çıktı Üretmek .....	20
İki Argümanlı Fonksiyon Tanımlamak.....	20
Ön Tanımlı Argümanlar .....	21
Argümanların Sıralaması .....	21
Ne Zaman Fonksiyon Yazılır? .....	21
Fonksiyon Çıktılarını Girdi Olarak Kullanmak .....	21
Local ve Global Değişkenler .....	22
Local Etki Alanından Global Etki Alanını Değiştirme.....	23

# Data Science

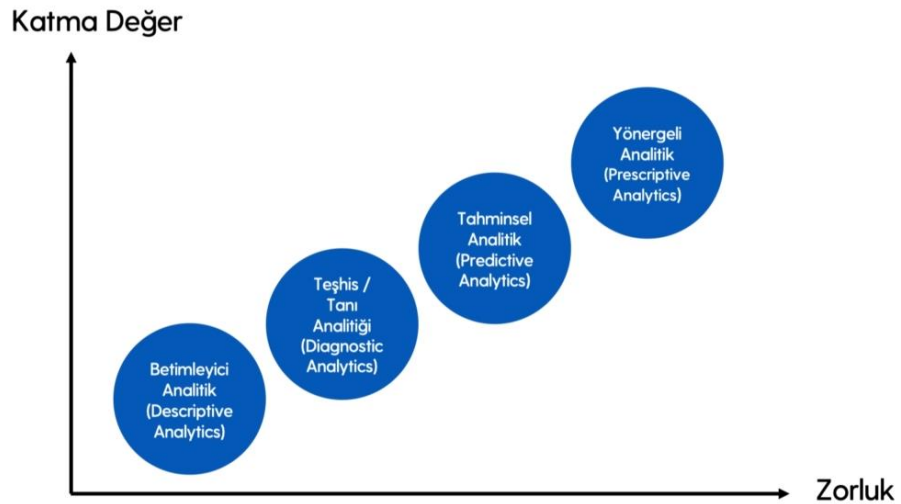
## VERİ BİLİMİNE GİRİŞ



Veri Bilimci, veriden faydalı bilgi çıkarma sürecini yöneten kişidir.



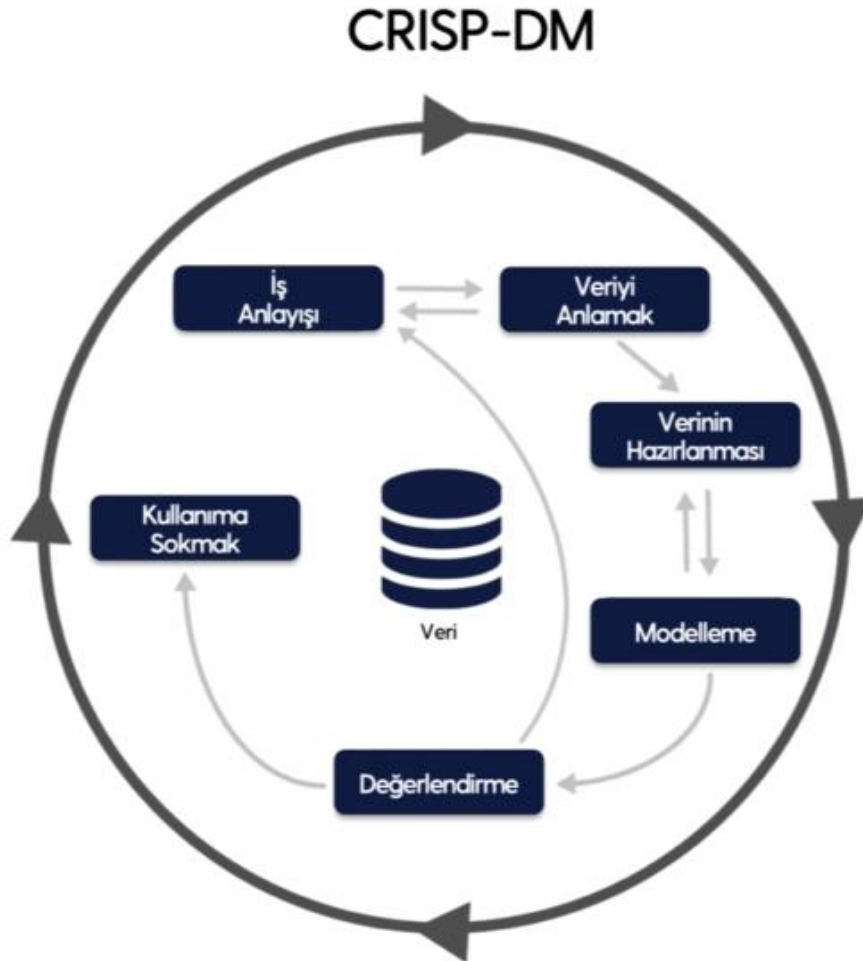
## VERİDEN FAYDALI BİLGİ ÇIKARMAK



## Data Science Kullanılan Alanlar

- Arkadaş önerileri
  - Otomatik fotoğraf etiketlemeleri
  - Hedefli içerik pazarlama
  - Otomatik mesaj tamamlama
  - Hedefli ürün pazarlama
  - Tavsiye sistemleri
  - Müşteri segmentasyonu
  - Kanser/Hastalık teşhisi
  - Şirketlerin gelir tahmini ile strateji belirlemesi
  - Başvuru değerlendirme sistemleri
  - Akıllı portföy yönetimi
  - Doğal afet modelleme çalışmaları
  - E-Spor Analitiği
- 
- Otonom araçlar
  - Nesne tanıma/takip uygulamaları
  - Sahte videolar
  - Eski resimlerin canlandırılması
  - Algoritmaların geliştirdiği resimler/var olmayan kişiler
  - Robotlar!

## Data Science Proje Döngüsü



# Python Programlama

## Temel Hareketler

### Integer, Float ve String

**Integer** = 9 gibi ondalıksız sayılar.

**Float** = 9.2 gibi ondalıklı sayılar.

**String** = Karakter dizileri. "Çift tırnak" veya 'Tek tırnak' içinde yazılır.

**Type** = type() içersine yazılan nesnenin tipini verir.

```
1 print("Hello AI Era")
2
3 #type komutu icerisine yazdigimiz nesnenin tipini verir.
4 type(9) #integer
5 type(9.2) #float
6 type("Recep Aydoğdu") #string
7
8 #####
9
10 type("123") #bunun da ciktisi str olacaktır.
11
12 "a"+"a"
13
14 "a" " a"
15
16 "a"*3
17
18 "a"/3 #type error hatasi
19
20 "a" "*5"
21
```

### String Metodları

**len()**= içersine yazılan değişkenin uzunluğunu verir.

```
1 # STRING METODLARI - len()
2
3 gel_yaz="gelecegi_yazanlar"
4
5 #del mvk #degiskeni silmek icin del kullaniriz. kullandiktan sonra
6 # yorum satiri haline getirilmelidir.
7
8 a=99
9 b=10
10
11 type(a/b) # a/b=9.9 olacagindan tipi float olur.
12
13 len(gel_yaz) # gel_yaz degiskeninin icerisindeki string'in krktr uzunlugunu verir.
14
```

**upper() & lower() =**

```
17 #upper() & lower() fonksiyonlari
18
19 gel_yaz.upper() #stringi buyuk harflere ceviris.
20
21 gel_yaz.lower() #stringi kucuk harflere ceviris.
22
```

isupper() & islower() =

```
23 #isupper() & islower() fonksiyonlari
24
25 gel_yaz.isupper() #buyuk harf mi? sorusu sorar. T or F getirir.
26 gel_yaz.islower() #kucuk harf mi? sorusu sorar.
27
28 B = gel_yaz.upper() #B degiskenine buyuk harfli gel_yaz atadik.
29
30 B.isupper()
31
32 Dnm="AsDfGhGgGgG"
33
34 Dnm.isupper()
35 Dnm.islower() #ikisi de false getirir.
```

replace() =

```
37 # replace() bir karakteri baska bir karakter ile degistirmek icin kullanilir.
38
39 gel_yaz.replace("a","ı")
40
```

replace("eski\_karakter","yeni\_karakter")

gelecegi\_yazanlar → gelecegi\_yızınlr

strip() = Karakter kırpma işlemleri

```
41 # strip() Karakter kırpma işlemleri
42
43 gel_yaz= " gelecegi_yazanlar " #basinda ve sonunda bosluk var
44 gel_yaz.strip() #varsayılan olarak bosluklari siler.
45
46 gel_yaz="*gelecegi_yazanlar*" # basina ve sonuna * ekledik.
47 gel_yaz.strip("*") # *(yıldız) arasındaki ifadeyi kirpar.
48
```

dir() =

```
49 # dir() icersine yazdigimiz veri tipi icin kullanilabilir metodlari verir.
50
51 dir(gel_yaz)
52 #ikisi de ayni sonucu verir.
53 dir(str)
```

capitalize() = İlk harfi büyütür.

gel\_yaz.capitalize()

title() = Her kelimenin ilk harfini büyütür.

gel\_yaz.title()

Substring = Alt küme işlemleri

```
57 # Substring: string ifadeleri ile alt kume işlemleri.
58
59 gel_yaz[0] # 0 index'li ifadeyi getirir.
60
61 gel_yaz[0:3] # 0'dan basla 3'e kadar getir.
62
```

## Type Dönüşümleri

```
63 #TYPE DONUSUMLERI
64
65 toplama_bir=input() #input ile kullanııcıdan veri alırız.
66 toplama_iki=input() #kullanııcıdan aldığımız veri str tipindedir.
67
68 toplama_bir+toplama_iki # 10+20 --> '1020' çıktısı verir.
69 # bunu engellemek için type dönüşümü yapmalıyız.
70 int(toplama_bir)+int(toplama_iki) #tip donusumlerini bu sekilde yaparız.
71
72 int(12.4) #float to int --> 12
73 float(12) #int to float --> 12.0
74 str(12) #int to str --> '12'
```

### print() fonksiyonu

print("gelecegi","yazanlar") → gelecegi yazanlar

print("gelecegi","yazanlar",sep = (" \_")) → gelecegi\_yazanlar

```
76 #Print fonksiyonu
77
78 print("gelecegi","yazanlar")
79
80 print("gelecegi","yazanlar",sep = "_") #sep argumani araya gelecek degeri secmemize olanak saglar.
81
82 ?print #print fonksiyonu ile kullanabilecegimiz argumanlari verir.
83
```

## Veri Yapıları (Data Types)

### Listeler

1. Değiştirilebilir
2. Kapsayıcıdır (Farklı tipte verileri tutabilir.)
3. Sıralıdır

Köşeli parantez [ ] ya da list() fonksiyonu ile liste oluşturabiliriz.

Liste bir üst type'dır içersinde farklı type'da veriler barındırabilir.

```
notlar = [90,80,70,50] #liste olusturma
type(notlar) #--> list

liste=["a",19.5,3] #farkli tipleri barindiran liste

liste_genis=["a",19.5,3,notlar] #kapsayicidir. icersinde farkli veri tipleri hatta liste bile barindirabilir.
len(liste_genis) #boyutu 4 olur.
```



## Liste Elemanlarına Ulaşma

```
#liste elemanlarına ulaşma

liste_genis[0] #-->"a"
liste_genis[1] #-->19.5
liste_genis[2] #-->3
liste_genis[3] #-->[90,80,70,50]

liste_genis[0:2] #0'dan 2 indexli elemana kadar alır
liste_genis[:2] #0'dan 2 indexli elemana kadar alır
liste_genis[2:] # 2 indexli elemandan sona kadar alır

liste_genis

liste_genis[3][1] # liste_genis içersindeki notlar listesinin 1 indexli elemanı
# --> 80

print(liste_genis[3][0]) #--> 90
```

## Liste İçi Type Sorgulama

```
#liste içi type sorgulama

type(liste_genis[0])
type(liste_genis[1])
type(liste_genis[2])
type(liste_genis[3])

tum_liste=[liste,list_genis]
```

**del liste** → liste'yi siler

## Liste elemanlarını değiştirme

```
# Liste elemanlarını değiştirme

liste2=["ali","veli","berkcan","ayse"]
liste2

liste2[1]="velinin babasi" # 1 index'li elemanı değiştirdik

liste2

liste2[1]="veli"
liste2[:3]="alının_babasi","velinin_babasi","berkcanın_babasi" #3 elemanı değiştirdik
liste2
```

## Listeye eleman ekleme

```
#listeye eleman ekleme

liste2 + ["kemal"] # bu şekilde kaydetmez sadece görüntüler.

liste2 = liste2 + ["kemal"]
```

## Listeden eleman silme

**del liste2[5]** → 5 index'li elemanı siler.

#### append ve remove metodlari

liste2.append("berkcan") →sona ekleme yapar

liste2.remove("alinin\_babasi") →silme yapar

liste2.remove("velinin\_babasi")

#### insert metodu

index'e göre ekleme yapar.

```
#insert  
  
liste2.insert(0,"ayca") #0 index'e ayca eklendi  
liste2.insert(2,"recep") #2 index'e recep ekledi  
liste2.insert(8,"asd") #fazla index girdik fakat sona ekledi  
len(liste2)  
liste2.insert(len(liste2),"son_eleman") #listenin sonuna ekledi
```

#### pop metodu

index'e göre silme yapar.

liste2.pop(0) #0 index degerli elemani siler

liste2.pop(1) #1 indexli elemani siler.

#### count metodu

```
#count  
  
liste=["ali","veli","ayca","veli","ali","ali"]  
liste.count("ali") #"ali" elemaninin listede kac kez yer aldigini gosterir.
```

→ 3

#### copy metodu

liste\_yedek=liste.copy() → liste'yi liste\_yedek'e kopyalar.

#### extend metodu

iki farklı listeyi birleştirir.

```
#extend  
  
liste.extend(liste2) #liste ile liste2'yi birleştirir.  
liste  
  
liste2.extend(["a",10]) #liste ile metodun icine yazilan elemanlari birleştirir.  
liste2
```

#### index metodu

```
#index  
  
liste.index("ali") #yazdigimiz elemanın kacinci index oldugunu verir.
```

#### reverse metodu

liste = [1,2,3]

liste.reverse() → liste elemanlarını ters sırayla kaydeder.

liste = [3,2,1]

#### sort metodu

Elemanları küçükten büyüğe sıralar.

```
#sort  
  
liste3=[2,1,5,3,4]  
  
liste3.sort() #liste3'ü küçükten büyüğe sıralayıp kaydeder.  
liste3
```

#### clear metodu

liste'nin içeriğini boşaltır.

```
#clear  
  
liste3.clear() #liste3'ün içeriğini boşaltır  
  
del(liste3) #liste3'ü tamamen siler.
```

## Tuple (Demet)

1. Kapsayıcıdır
2. Sıralıdır
3. Değiştirilemez (Listeden farkı budur.)

### Tuple Oluşturma

```
#Tuple Oluşturma  
t=(1,2,3,"eleman",[1,2,3,4])
```

**NOT=** Tek elemanlı tuple oluştururken sonuna virgül koymalıyız. Aksi takdirde tuple oluşturmak istediğimiz anlaşılamaz.

Örneğin; t = ("eleman",)

### Eleman İşlemleri

Tuple'larda eleman işlemleri listeler ile birebir aynıdır. (index'e göre erişim vs.)

t=(1,2,3,4)

t[0] → 1

t[-1] → 4 (sondan birinci eleman demektir.)

## Dictionary (Sözlük)

1. Kapsayıcıdır
2. Sırasızdır → Listelerden farkı budur.
3. Değiştirilebilirdir.

### Dictionary Nedir?

Key'ler ve bu key'lerin karşılıklarının bir arada tutulduğu veri yapısıdır.

Listelerde olduğu gibi index'leme yapılmaz.

### Dictionary Oluşturma

```
# Sozluk Oluşturma  
  
sozluk={"REG" : "regresyon modeli",  
        "LOJ" : "lojistik regresyon",  
        "CART" : "Classification And Reg"}  
  
sozluk  
len(sozluk) # --> 3
```

{ "key" : "key'in karşılığı" }

**NOT=** Sözlüklerde key'ler sadece sabit veri yapılarından oluşabilir. list gibi yapılardan olamaz. String ve sayılar sabit veri yapılarıdır.

Sabit veri yapısı değiştirilemez demektir. Tuple'da buna dahildir.

t = ("tuple",) → sozluk = { t : "tuple'dan key olur" }

## Eleman Seçme İşlemleri

```
# Eleman secme islemleri

sozluk={"REG" : "regresyon modeli",
        "LOJ" : "lojistik regresyon",
        "CART" : "Classification And Reg"}

sozluk["REG"] #REG key'inin karsiligini bu sekilde getiririz.

sozluk={"REG" : {"ASD" : 10,
                 "XXX" : 20,
                 "ZZZ" : 30},
        "LOJ" : {"ASD" : 10,
                 "XXX" : 20,
                 "ZZZ" : 30},
        "CART" : {"ASD" : 10,
                  "XXX" : 20,
                  "ZZZ" : 30}
        }

sozluk["REG"] # Sozluk icinde sozluk olusturduk. Ic ice yapi.

sozluk["REG"]["XXX"] #ic ice bir yapida elemana erisim.
```

```
In [6]: sozluk["REG"]["XXX"] #ic ice bir yapida elemana erisim.
Out[6]: 20
```

## Eleman Ekleme & Değiştirme

```
In [14]: sozluk={"REG" : "regresyon modeli",
...           "LOJ" : "lojistik regresyon",
...           "CART" : "Classification And Reg"}

In [15]: sozluk["GBM"] = "Gradient Boosting Mac" #sozluk'e eleman ekleme.

In [16]: sozluk
Out[16]:
{'REG': 'regresyon modeli',
 'LOJ': 'lojistik regresyon',
 'CART': 'Classification And Reg',
 'GBM': 'Gradient Boosting Mac'}

In [17]: sozluk["REG"] = "REG'in yeni karsiligi" #REG Key'inin karsiligini degistirme.
...: sozluk
Out[17]:
{'REG': "REG'in yeni karsiligi",
 'LOJ': 'lojistik regresyon',
 'CART': 'Classification And Reg',
 'GBM': 'Gradient Boosting Mac'}
```

REG key'i olmasaydı yeni key oluşturulacaktı.

```

In [22]: t= ("tuple",) # t adında tuple oluşturduk.

In [23]: sozluk[t]="Tuple'dan key olusturuldu."
        ....: sozluk
Out[23]:
{'REG': "REG'in yeni karsiligi",
 'LOJ': 'lojistik regresyon',
 'CART': 'Classification And Reg',
 'GBM': 'Gradient Boosting Mac',
 ('tuple',): "Tuple'dan key olusturuldu."}

```

### Sets (Kümeler)

1. Sırasızdır (Index değerleri yok.)
2. Değerleri eşsizdir. (Tekrar eden değeri olmaz.)
3. Değiştirilebilir.
4. Kapsayıcıdır. Farklı türden veri yapıları barındırabilir.

Set'ler performans odaklı veri tipleridir. Programlama anlamında biraz daha hız istediğimizde kullanılır. Matematiksel anlamda bu veri yapıları kümelere benzer.

### Set Oluşturma

s = set() → s isminde bir set oluşturuldu.

```

In [1]: l= ["ali","ata","bakma","ali","uzaya","git"]

In [2]: s=set(l) # l listesindeki elemanlari birer kez alır.

In [3]: s #set'in elemanlari essiz olacaginden her eleman bir kez alınır.
Out[3]: {'ali', 'ata', 'bakma', 'git', 'uzaya'}

```

```

In [4]: ali="ali_ata_bakma_uzaya_git_lutfen"

In [5]: s=set(ali) #ali cumlesindeki her bir karakteri bir kez alır.

In [6]: s
Out[6]: {'_', 'a', 'b', 'e', 'f', 'g', 'i', 'k', 'l', 'm', 'n', 't', 'u', 'y', 'z'}

```

## Set'lere eleman ekleme ve çıkarma işlemleri

add() fonksiyonu ile ekleme yaparız.

```
In [8]: s.add("ile") #ile stringini set'e ekledi
...: s
Out[8]:
{'_',
 'a',
 'b',
 'e',
 'f',
 'g',
 'i',
 'ile',
 'k',
 'l',
 'm',
 'n',
 't',
 'u',
 'y',
 'z'}
```

```
In [9]: t=("ali","bakma")

In [10]: s.add(t) # t isimli tuple'i set'e ekledi
...: s
Out[10]:
{('ali', 'bakma'),
 '_',
 'a',
 'b',
 'e',
 'f',
 'g',
 'i',
 'ile',
 'k',
 'l',
 'm',
 'n',
 't',
 'u',
 'y',
 'z'}
```

```
In [12]: s.add(ali) # ali elemanini set'e ekledi.
...: s
Out[12]:
{('ali', 'bakma'),
 '_ ',
 'a',
 'ali_ata_bakma_uzaya_git_lutfen',
 'b',
 'e',
 'f',
 'g',
 'i',
 'ile',
 'k',
 'l',
 'm',
 'n',
 't',
 'u',
 'y',
 'z'}
```

remove() fonksiyonu ile set'lerden eleman silebiliriz.

```
In [13]: s.remove(ali) # ali elemanini sildi.
...: s
Out[13]:
{('ali', 'bakma'),
 '_ ',
 'a',
 'b',
 'e',
 'f',
 'g',
 'i',
 'ile',
 'k',
 'l',
 'm',
 'n',
 't',
 'u',
 'y',
 'z'}
```



```
s.remove(ali) # ali'yi tekrar silmek istedigimizde KeyError hatası verir.  
s.discard(t) # discard ile de silme islemi gerceklestirebiliriz  
s  
s.discard(t) # tekrar silmek istedigimizde discard hata uretmez.
```

### Set'lerde Fark İşlemleri

#### difference & symmetric\_difference

**difference** = kümelerin farkını verir.

```
#difference ve symmetric_difference
```

```
set1= set([1,3,5])  
set2= set([1,2,3])
```

```
In [2]: set1.difference(set2) #set1'in set2'den farki  
Out[2]: {5}
```

```
In [3]: set2.difference(set1) #set2'in set1'den farki  
Out[3]: {2}
```

**symmetric\_difference** = ikisinde de ortak olmayan elemanları verir.

```
In [4]: set1.symmetric_difference(set2) #ikisinde de ortak olmayan elemanlari verir  
Out[4]: {2, 5}
```

### Set'lerde Kesişim ve Birleşim İşlemleri

#### intersection & union & intersection\_update

**intersection** = kesişim

```
In [5]: set1.intersection(set2) # set1 ve set2'nin ortak elemanlari  
Out[5]: {1, 3}
```

```
In [6]: set2.intersection(set1)  
Out[6]: {1, 3}
```

**union** = birleşim

```
In [7]: set1.union(set2) # set1 ve set2'nin birlesimi  
Out[7]: {1, 2, 3, 5}
```

**intersection** = set1'in değerini kesişim değerleri olarak değiştirir.

```
In [8]: set1.intersection_update(set2) #set1'in degerini kesisim degerleri olarak degistirir.  
In [9]: set1  
Out[9]: {1, 3}
```

### Set'lerde Sorgu İşlemleri

#### isdisjoint & issubset & issuperset

**isdisjoint** = Ayrık küme mi?

İki kümenin kesişiminin boş olup olmadığını sorgular.

Boş ise True değil ise False döndürür.

```
In [10]: set1.isdisjoint(set2) #set1 ve set2'nin kesisimi bos mu? Ayrık kume mi?  
Out[10]: False
```

**issubset** = subset'i mi? Alt kümesi mi? sorgusunu yapar.

```
In [11]: set1.issubset(set2) #set1 set2'nin subset'i mi?  
Out[11]: True
```

**issuperset** = Kapsar mı?

```
In [13]: set2.issuperset(set1) #set2 set1'in superset'i mi? Kapsar mi?  
Out[13]: True
```

### Veri Yapıları Özet

Listeler	Tuple	Sözlük	Setler
Değiştirilebilir	Değiştirilemez	Değiştirilebilir	Değiştirilebilir
Sıralı	Sıralı	Sırasız	Sırasız + Eşsizdir
Kapsayıcı	Kapsayıcı	Kapsayıcı	Kapsayıcıdır

## Fonksiyonlar / Karar-Kontrol Yapıları / Döngüler

### Fonksiyon Nedir?

Belirli amaçları yerine getiren işleçlerdir.

### Matematiksel İşlemler

```
In [14]: 4*4
Out[14]: 16

In [15]: 4/4
Out[15]: 1.0

In [16]: 4-2
Out[16]: 2

In [17]: 4+2 # bunlar klasik matematiksel operatorlerdir.
Out[17]: 6
```

### Üs Alma

$3^{**}2 \rightarrow 3^2$  anlamına gelir.

```
In [18]: 3**2 # 3'un 2'nci kuvveti
Out[18]: 9

In [19]: 3**3 # 3'un 3'ncü kuvveti
Out[19]: 27
```

### Fonksiyon Nasıl Yazılır ?

**def** ile fonksiyon oluşturacağımızı belirtiriz.

```
# =====
# #Fonksiyon Nasil Yazilir?

def kare_al(x):
    print(x**2) # def ile fonksiyon olusturacagimizi belirtiriz.

kare_al(5) #fonksiyonu bu sekilde calistiririz.

# =====

In [21]: def kare_al(x):
...:     print(x**2) # def ile fonksiyon olusturacagimizi belirtiriz.
...:
...:

In [22]: kare_al(5) #fonksiyonu bu sekilde calistiririz.
25
```

### Bilgi Notuyla Çıktı Üretmek

```
#Bilgi notuyla çıktı üretme
def kare_al(x):
    print("Girilen sayinin karesi : " + x**2) #str + int

kare_al(3) #hata aldık çünkü str ifadeler sadece str ifadeler ile birleştirilebilir.
```

Bu fonksiyonu çalıştırınca aldığımız hata :

```
In [17]: kare_al(3) #hata aldık çünkü str ifadeler sadece str ifadeler ile birleştirilebilir.
Traceback (most recent call last):

  File "<ipython-input-17-31e075573f9a>", line 1, in <module>
    kare_al(3) #hata aldık çünkü str ifadeler sadece str ifadeler ile birleştirilebilir.

  File "<ipython-input-16-4cc719a79d0b>", line 2, in kare_al
    print("Girilen sayinin karesi : " + x**2) #str + int

TypeError: can only concatenate str (not "int") to str
```

str ifadeler ile sadece str ifadeler birleştirilebilir!

type dönüşümü yapmalıyız.:

```
In [19]: def kare_al(x):
...:     print("Girilen sayinin karesi : " + str(x**2)) #str + str(type donusumu)
...:
...:

In [20]: kare_al(3) #bu kez hata almadan çalıştı.
Girilen sayinin karesi : 9
```

Başka bir örnek:

```
In [21]: def kare_al(x):
...:     print("Girilen sayi: " + str(x)
...:           + "\nKaresi: " + str(x**2)) #\n ile alt satıra geçtik.
...:
...:

In [22]: kare_al(4)
Girilen sayi: 4
Karesi: 16
```

### İki Argümanlı Fonksiyon Tanımlamak

```
In [1]: def carpma_yap(x,y):
...:     print("Birinci sayi: " + str(x)
...:           + "\nİkinci sayi: " + str(y)
...:           + "\nCarpimlari: " + str(x*y))
...:
...:

In [2]: carpma_yap(3,4)
Birinci sayi: 3
İkinci sayi: 4
Carpimlari: 12
```

### Ön Tanımlı Argümanlar

Print() fonksiyonundan hatırlayacağımız gibi sep() ve end() gibi argümanlardır.

```
In [8]: def carpma_yap(x,y=1): # y=1 demeseydik iki degeri de girmek zorunda kalirdik.
...:     print(x*y)
...:
...:

In [9]: carpma_yap(3) #Hata vermeden calisacak.
3

In [10]: carpma_yap(3,5) #yeni bir deger girdigimizde eski degeri ezeriz.
15
```

y=1 yazarak ön tanımlı bir argüman oluşturmuş olduk.

### Argümanların Sıralaması

Argümanların sırasını bilmediğimiz fakat isimlerini bildiğimiz zaman aşağıdaki şekilde çalıştırabiliriz.

```
# Argumanların Sıralamasi

def carpma_yap(x,y):
    print(x*y)

carpma_yap(y=2, x=4) # Argumanların sırasını bilmiyorsam ama isimlerini biliyorsam
                     # Bu şekilde çalıştırabiliriz.
```

### Ne Zaman Fonksiyon Yazılır?

Fonksiyonlar programlama dilleri içerisinde tekrar eden görevleri yerine getirmek ve var olan işleri daha programatik bir şekilde gerçekleştirmek için kullanılır.

Örneğin bir şehirde binlerce sokak lambası var ve bu sokak lambaları için ısı, nem, şarj değerlerini kullanarak bir hesaplama yapmamız gerekiyor. Her lamba için tek tek hesap mı yapacağız?

Hayır, fonksiyonu bir kez yazıp her lambada o fonksiyonu kullanacağız.

```
#Fonksiyonlar ne zaman yazılır?
def direk_hesap(isi, nem, sarj):
    print((isi+nem)/sarj)

direk_hesap(25,40,70)

In [14]: direk_hesap(25,40,70)
0.9285714285714286
```

### Fonksiyon Çıktılarını Girdi Olarak Kullanmak

Yazdığımız bir fonksiyonun çıktısını başka bir yerde girdi olarak kullanmak istiyorsak **return** ifadesini kullanmalıyız.

**print()** ekrana çıktı verir. Programlama anlamında kullanılabileceği anlamına gelmez.

Aşağıdaki örnekte görebiliriz.

```
#Fonksiyon Ciktilarini Girdi Olarak Kullanmak
#Fonksiyonun ciktisini baska bir yerde girdi olarak kullanmak icin
# return ifadesini kullanmaliyiz.

def direk_hesap(isi, nem, sarj):
    print((isi+nem)/sarj) #print ekrana cikti verir. Programlama anlaminda
                        #kullanilabilegi anlamina gelmez.

cikti = direk_hesap(25,40,70)
cikti #fonksiyonun sonucunu cikti'ya atayamadik
```

```
In [24]: def direk_hesap(isi, nem, sarj):
...:     return (isi+nem)/sarj #return ifadesini kullanirsak sonucu kullanabiliriz.
...:
...:
In [25]: cikti = direk_hesap(25,40,70)

In [26]: cikti
Out[26]: 0.9285714285714286
```

Fonksiyon **return** ifadesine gelince durur:

```
def direk_hesap(isi, nem, sarj):
    return
    (isi+nem)/sarj # bu sekilde calistirirsak fonksiyon islevini yapmaz.
                # cunku fonksiyon return'un oldugu satira gelince durur.

direk_hesap(25,40,70)
```

### Local ve Global Değişkenler

Ana çalışma alanımızdaki değişkenler **Global** değişkenlerdir.

Her hangi bir fonksiyonun ya da döngünün etkisindeki değişkenler ise **Local** değişkenlerdir.

```
#Local ve Global Degiskenler

x=10
y=10 #Ana calisma alanimizdaki degiskenler Global degiskenlerdir.

def carpma(x,y):
    return x*y #fonksiyon icersindeki degiskenler Local degiskendir.

carpma(2,3)
```

### Local Etki Alanından Global Etki Alanını Değiştirme

Yazmış olduğumuz bir döngü içerisinde ya da tanımlamış olduğumuz bir fonksiyon içerisinde global değişkenlerin değerlerinde değişiklik yapmak istediğimiz zaman ne yapmamız gerekiyor ?

Python öncelikle **local** etki alanındaki değişkenleri tarar, arar ve bulmaya çalışır.

Örneğin bir fonksiyon yazdığımızda değişiklik yapmak istediğimiz değişkeni öncelikle kendi içerisinde (local'de )arar, bulamazsa global alana çıkacak. Global alanda o değişkeni bulursa ona etki edecek (Orada da bulamazsa hata üretecek.). Aşağıdaki örnekte bu durumu gözlemleyebiliriz.

```
In [1]: x=[] #bos bir liste olusturuldu

In [2]: def eleman_ekle(y):
...:     x.append(y) #x'e y'yi ekle.
...:     print(str(y)+" ifadesi eklendi."
...:           +"\nListenin yeni hali: "+str(x))
...:
...:

In [3]: eleman_ekle(4)
4 ifadesi eklendi.
Listenin yeni hali: [4]

In [4]: eleman_ekle(3)
3 ifadesi eklendi.
Listenin yeni hali: [4, 3]
```