

Python for Data Science

Çalışma Dökümanı

Recep Aydoğdu

İçindekiler

[Data Science Kullanılan Alanlar 5](#_Toc37026004)

[Data Science Proje Döngüsü 5](#_Toc37026005)

[Python Programlama 6](#_Toc37026006)

[Temel Hareketler 6](#_Toc37026007)

[Integer, Float ve String 6](#_Toc37026008)

[Integer 6](#_Toc37026009)

[Float 6](#_Toc37026010)

[String 6](#_Toc37026011)

[Type 6](#_Toc37026012)

[String Metodları 6](#_Toc37026013)

[len() 6](#_Toc37026014)

[upper() & lower() 6](#_Toc37026015)

[isupper() & islower() 7](#_Toc37026016)

[replace() 7](#_Toc37026017)

[strip() 7](#_Toc37026018)

[dir() 7](#_Toc37026019)

[capitalize() 7](#_Toc37026020)

[title() 7](#_Toc37026021)

[Substring 7](#_Toc37026022)

[Type Dönüşümleri 8](#_Toc37026023)

[print() fonksiyonu 8](#_Toc37026024)

[Veri Yapıları (Data Types) 8](#_Toc37026025)

[Listeler 8](#_Toc37026026)

[Liste Elemanlarına Ulaşma 9](#_Toc37026027)

[Liste İçi Type Sorgulama 9](#_Toc37026028)

[Liste elemanlarını değiştirme 9](#_Toc37026029)

[Listeye eleman ekleme 9](#_Toc37026030)

[Listeden eleman silme 9](#_Toc37026031)

[append ve remove metodlari 10](#_Toc37026032)

[insert metodu 10](#_Toc37026033)

[pop metodu 10](#_Toc37026034)

[count metodu 10](#_Toc37026035)

[copy metodu 10](#_Toc37026036)

[extend metodu 11](#_Toc37026037)

[index metodu 11](#_Toc37026038)

[reverse metodu 11](#_Toc37026039)

[sort metodu 11](#_Toc37026040)

[clear metodu 11](#_Toc37026041)

[Tuple (Demet) 12](#_Toc37026042)

[Tuple Oluşturma 12](#_Toc37026043)

[Eleman İşlemleri 12](#_Toc37026044)

[Dictionary (Sözlük) 12](#_Toc37026045)

[Dictionary Nedir? 12](#_Toc37026046)

[Dictionary Oluşturma 12](#_Toc37026047)

[Eleman Seçme İşlemleri 13](#_Toc37026048)

[Eleman Ekleme & Değiştirme 13](#_Toc37026049)

[Sets (Kümeler) 14](#_Toc37026050)

[Set Oluşturma 14](#_Toc37026051)

[Set’lere eleman ekleme ve çıkarma işlemleri 15](#_Toc37026052)

[Set’lerde Fark İşlemleri 17](#_Toc37026053)

[Set’lerde Kesişim ve Birleşim İşlemleri 17](#_Toc37026054)

[Set’lerde Sorgu İşlemleri 18](#_Toc37026055)

[Veri Yapıları Özet 18](#_Toc37026056)

[Fonksiyonlar / Karar-Kontrol Yapıları / Döngüler 19](#_Toc37026057)

[Fonksiyon Nedir? 19](#_Toc37026058)

[Matematiksel İşlemler 19](#_Toc37026059)

[Üs Alma 19](#_Toc37026060)

[Fonksiyon Nasıl Yazılır ? 19](#_Toc37026061)

[Bilgi Notuyla Çıktı Üretmek 20](#_Toc37026062)

[İki Argümanlı Fonksiyon Tanımlamak 20](#_Toc37026063)

[Ön Tanımlı Argümanlar 21](#_Toc37026064)

[Argümanların Sıralaması 21](#_Toc37026065)

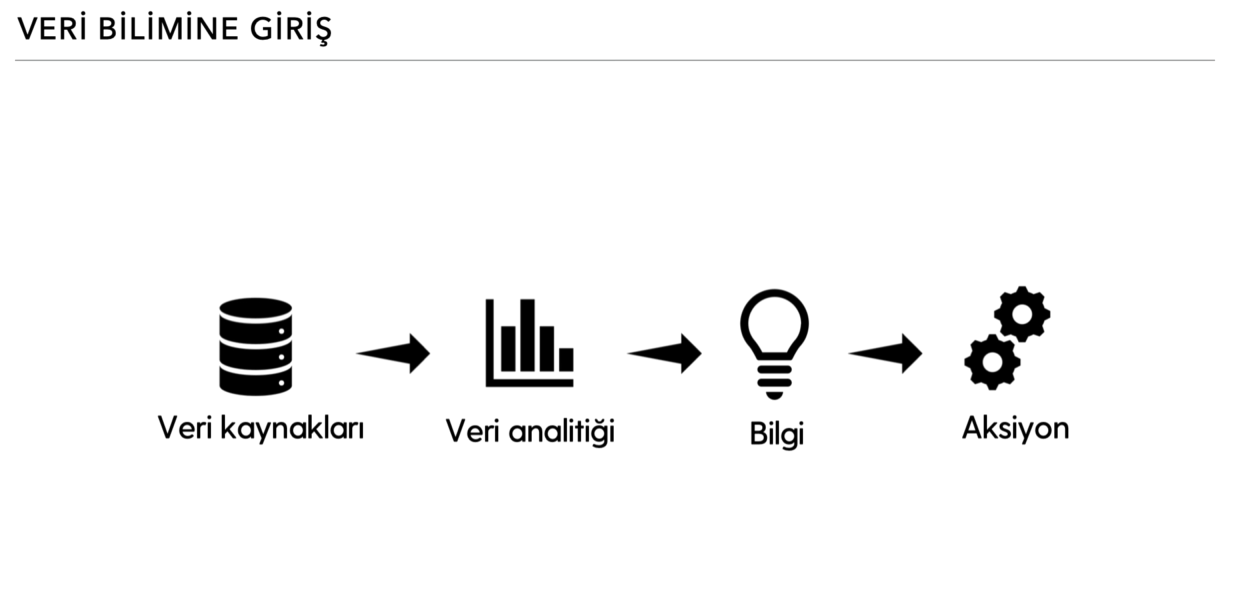
[Ne Zaman Fonksiyon Yazılır? 21](#_Toc37026066)

[Fonksiyon Çıktılarını Girdi Olarak Kullanmak 21](#_Toc37026067)

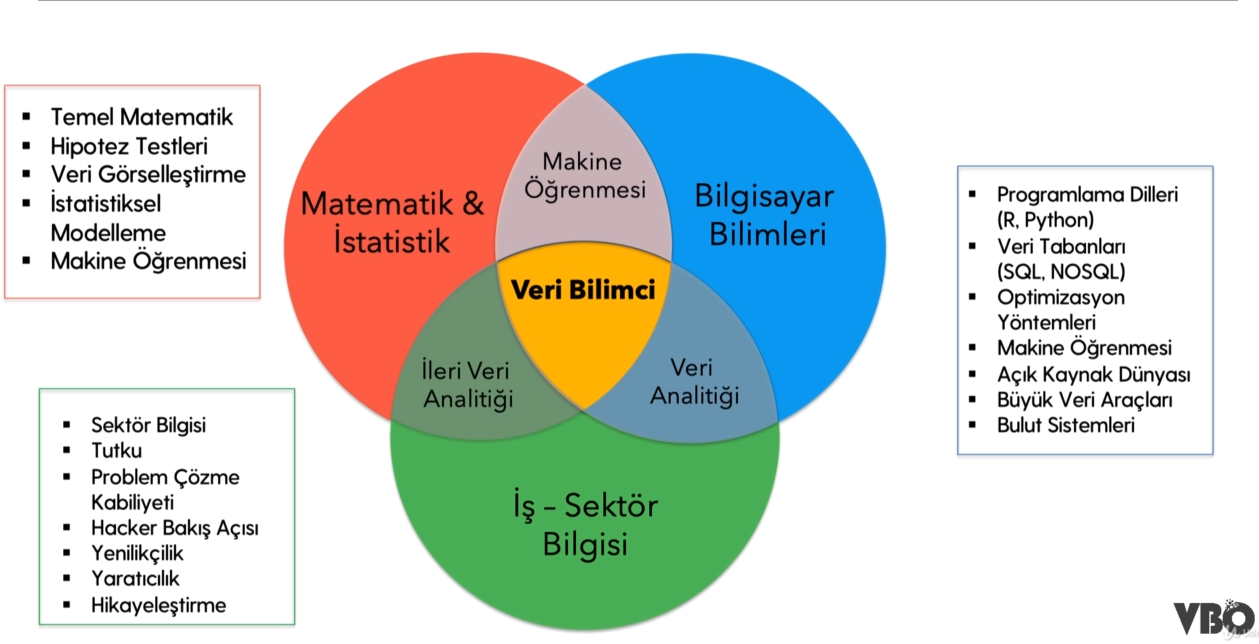
[Local ve Global Değişkenler 22](#_Toc37026068)

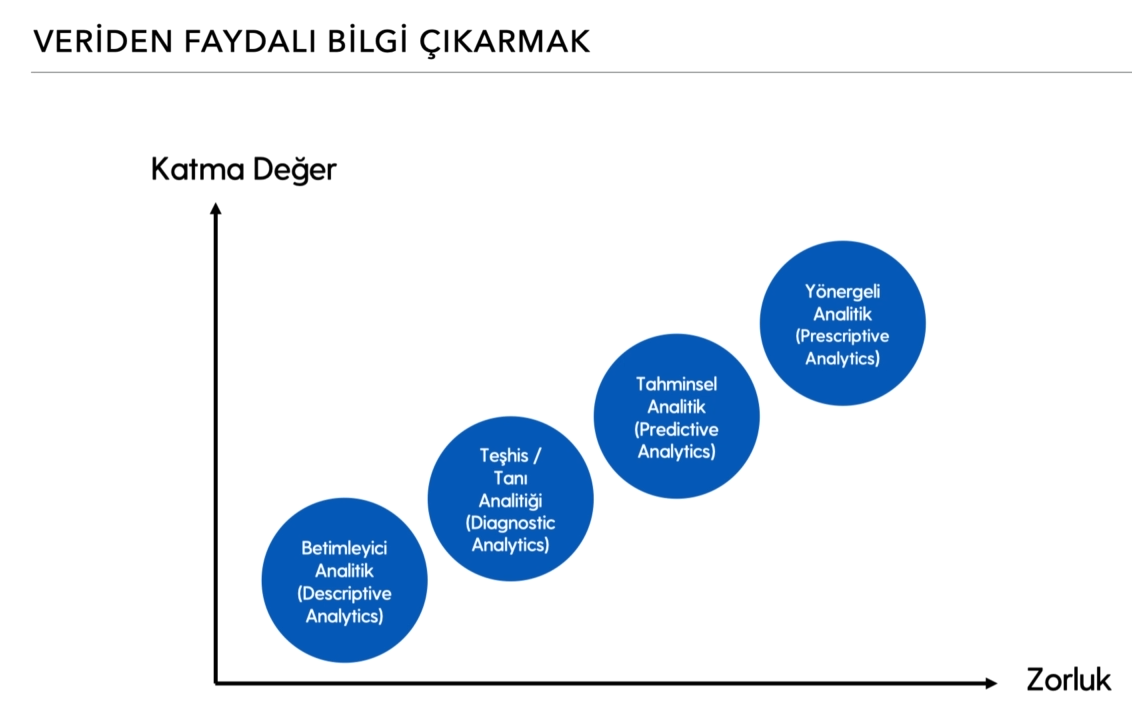
[Local Etki Alanından Global Etki Alanını Değiştirme 23](#_Toc37026069)

**Data Science**

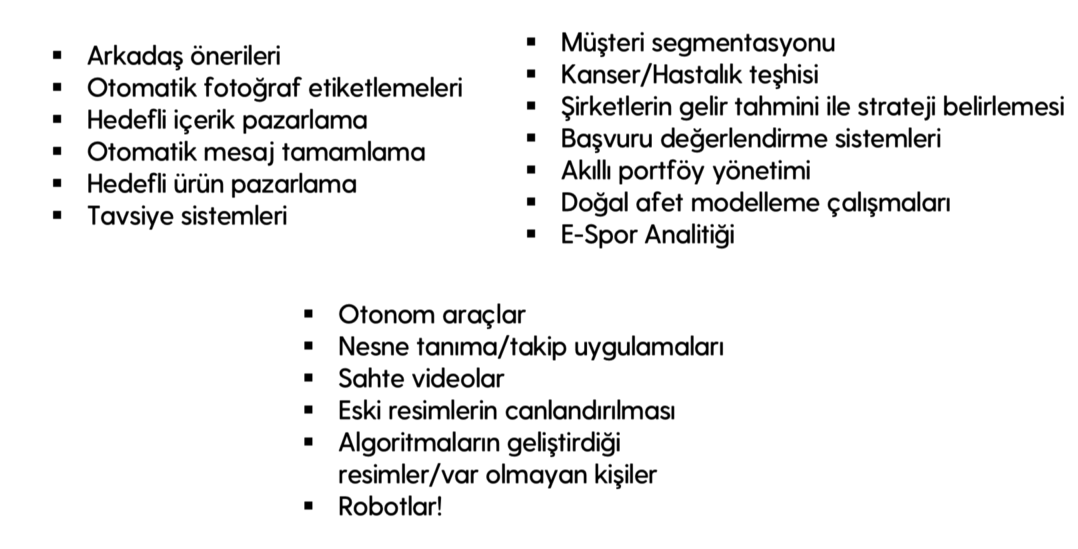


Veri Bilimci, veriden faydalı bilgi çıkarma sürecini yöneten kişidir.

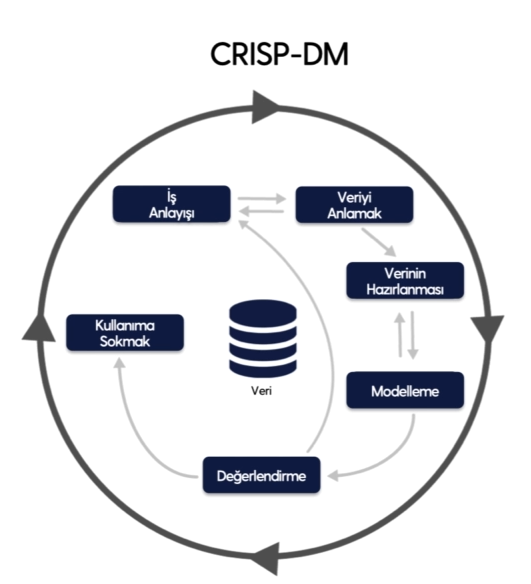




# Data Science Kullanılan Alanlar



# Data Science Proje Döngüsü



# Python Programlama

## Temel Hareketler

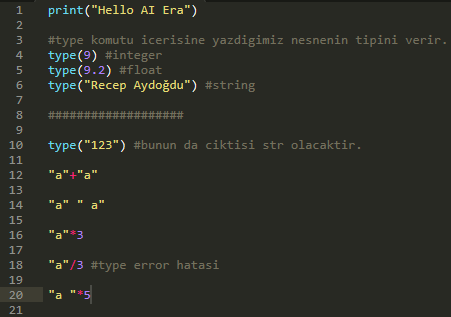
### Integer, Float ve String

Integer **=** 9 gibi ondalıksız sayılar.

Float **=** 9.2 gibi ondalıklı sayılar.

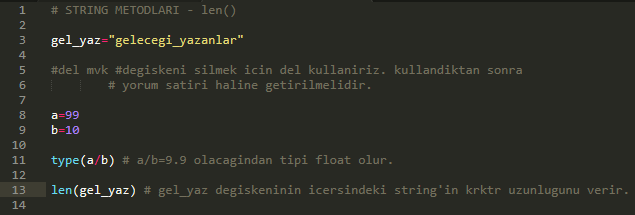
String **=** Karakter dizileri. “Çift tırnak” veya ‘Tek tırnak’ içinde yazılır.

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

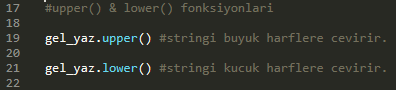


### String Metodları

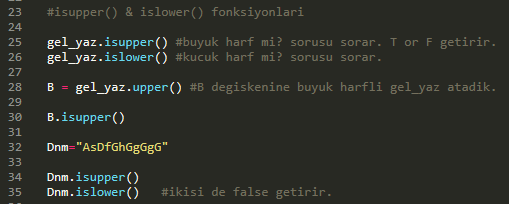
len()**=** içerisine yazılan değişkenin uzunluğunu verir.



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



isupper() & islower() **=**



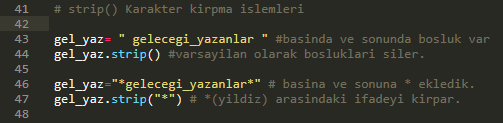
replace() **=**



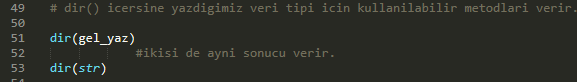
replace(“eski\_karakter”,”yeni\_karakter”)

gelecegi\_yazanlar 🡪gelecegi\_yızınlır

strip() **=** Karakter kırpma işlemleri



dir() **=**



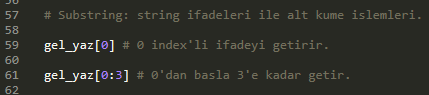
capitalize() **=** İlk harfi büyütür.

gel\_yaz.capitalized()

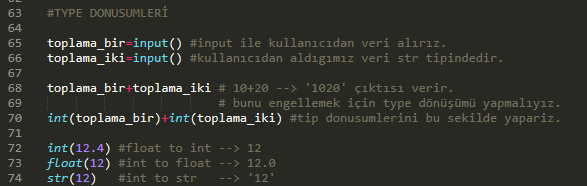
title() **=** Her kelimenin ilk harfini büyütür.

gel\_yaz.title()

Substring **=** Alt küme işlemleri



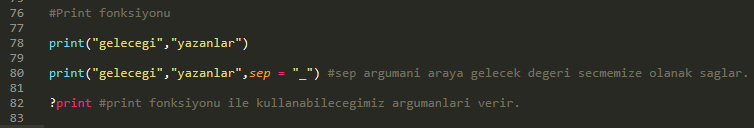
#### Type Dönüşümleri



#### print() fonksiyonu

print(“gelecegi”,”yazanlar”) 🡪 gelecegi yazanlar

print(“gelecegi”,”yazanlar”,sep = (“\_”)) 🡪 gelecegi\_yazanlar



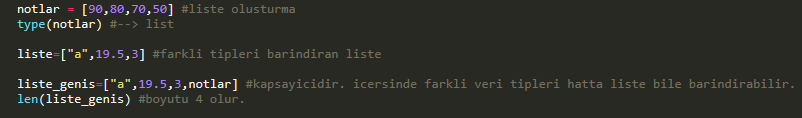
## Veri Yapıları (Data Types)

### Listeler

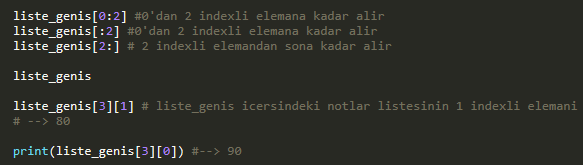
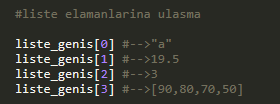
1. Değiştirilebilir
2. Kapsayıcıdır (Farklı tipte verileri tutabilir.)
3. Sıralıdır

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

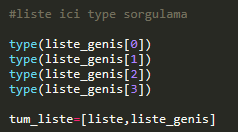
Liste bir üst type’dır içersinde farklı type’da veriler barındırabilir.



#### Liste Elemanlarına Ulaşma

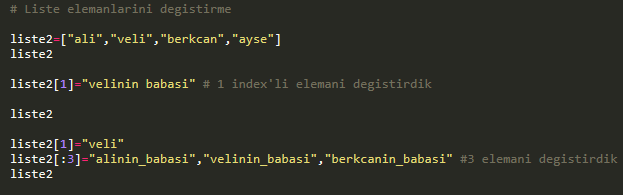


#### Liste İçi Type Sorgulama

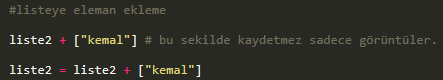


**del liste 🡪 liste’yi siler**

#### Liste elemanlarını değiştirme



#### Listeye eleman ekleme



#### Listeden eleman silme

del liste2[5] 🡪 5 index’li elamanı siler.

#### append ve remove metodlari

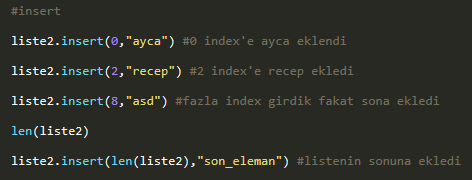
liste2.append("berkcan") 🡪sona ekleme yapar

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

liste2.remove("velinin\_babasi")

#### insert metodu

index’e göre ekleme yapar.



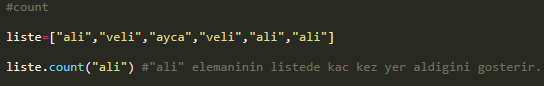
#### pop metodu

index’e göre silme yapar.

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

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

#### count metodu

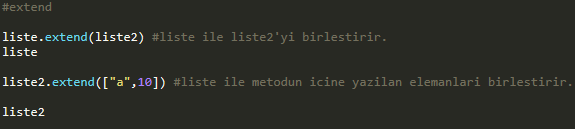


🡪 3

#### copy metodu

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

#### extend metodu

İki farklı listeyi birleştirir.

#### index metodu



#### reverse metodu

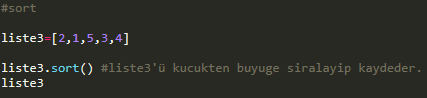
liste = [1,2,3]

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

liste = [3,2,1]

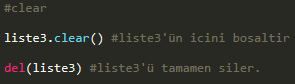
#### sort metodu

Elemanları küçükten büyüğe sıralar.



#### clear metodu

liste’nin içini boşaltır.



### Tuple (Demet)

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

#### Tuple Oluşturma



**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)

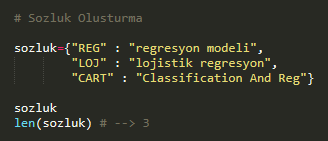
1. Kapsayıcıdır
2. Sırasızdır 🡪 Listelerden farkı budur.
3. Değiştirilebilirdir.

#### Dictionary Nedir?

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

Listelerde olduğu gibi index’leme yapılmaz.

#### Dictionary Oluşturma



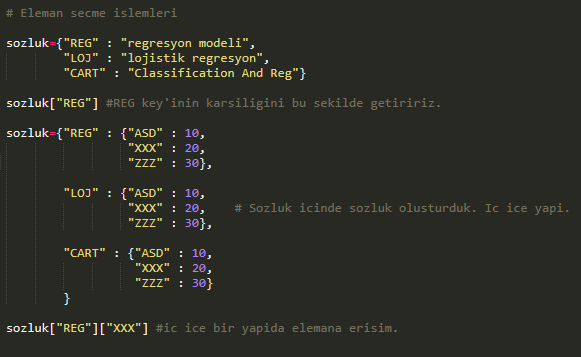
{“key” : “key’in karşılığı”}

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

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

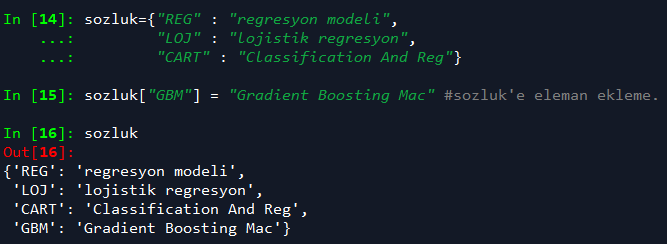
t = (“tuple”,) 🡪 sozluk = { t : “tuple’dan key olur” }

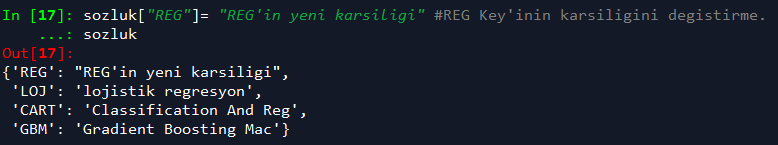
#### Eleman Seçme İşlemleri



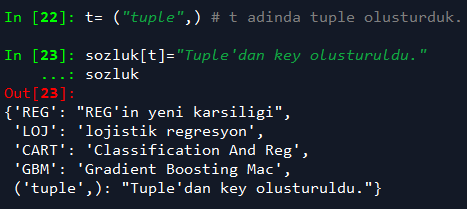


#### Eleman Ekleme & Değiştirme





REG key’i olmasaydı yeni key oluşturulacaktı.



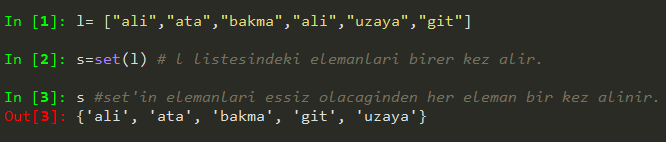
### Sets (Kümeler)

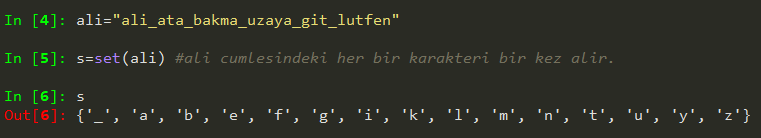
1. Sırasızdır (Index değerleri yok.)
2. Değerleri eşsizdir. (Tekrar eden değeri olmaz.)
3. Değiştirilebilir.
4. Kapsayıcıdır. Farklı türden veri yapıları barındırabilir.

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

#### Set Oluşturma

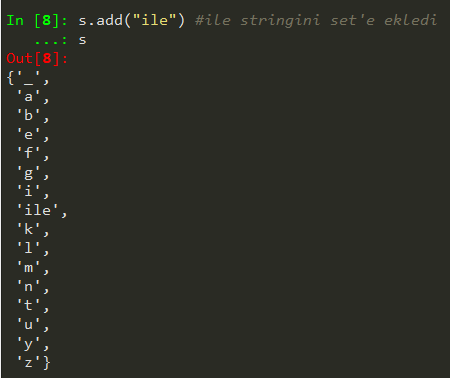
s = set() 🡪 s isminde bir set oluşturuldu.

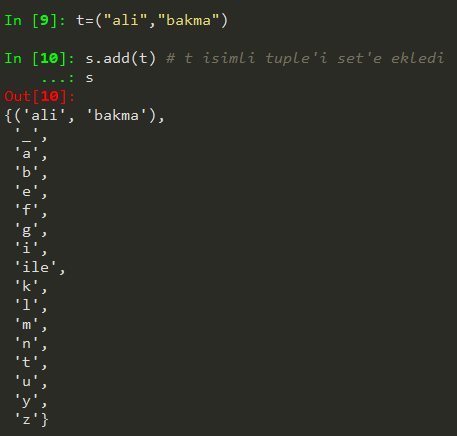


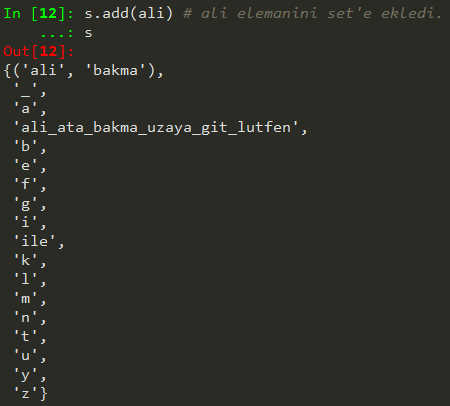


#### Set’lere eleman ekleme ve çıkarma işlemleri

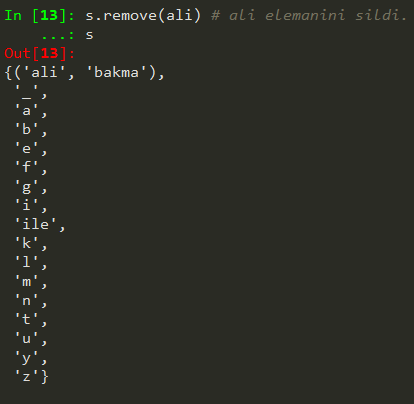
add() fonksiyonu ile ekleme yaparız.

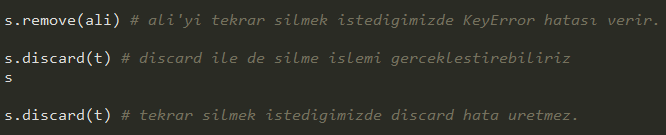






remove() fonksiyonu ile set’lerden eleman silebiliriz.

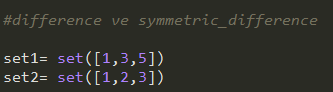




#### Set’lerde Fark İşlemleri

**difference & symmetric\_difference**

**difference =** kümelerin farkını verir.



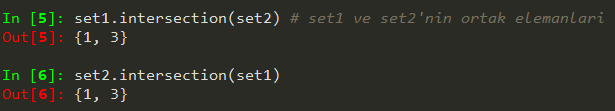
**symmetric\_difference =** ikisinde de ortak olmayan elemanları verir.



#### Set’lerde Kesişim ve Birleşim İşlemleri

**intersection & union & intersection\_update**

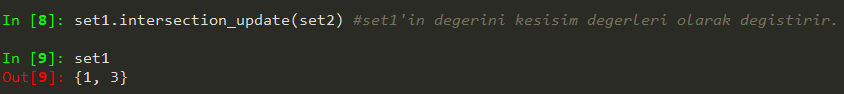
**intersection =** kesişim



**union =** birleşim



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



#### 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.



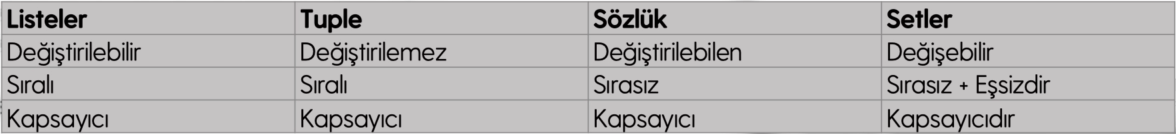
**issubset =** subset’i mi? Alt kümesi mi? sorgusunu yapar.



**issuperset =** Kapsar mı?



### Veri Yapıları Özet

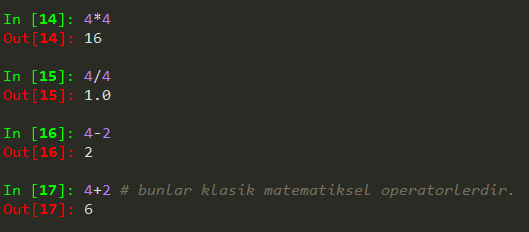


## Fonksiyonlar / Karar-Kontrol Yapıları / Döngüler

### Fonksiyon Nedir?

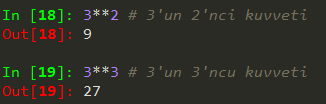
Belirli amaçları yerine getiren işleçlerdir.

### Matematiksel İşlemler



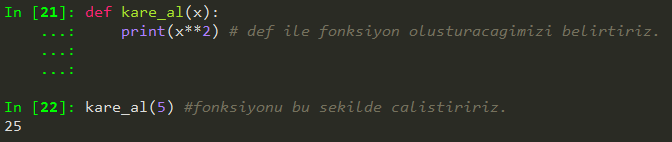
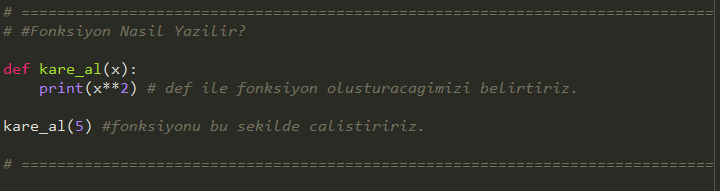
#### Üs Alma

3\*\*2 🡪 32 anlamına gelir.

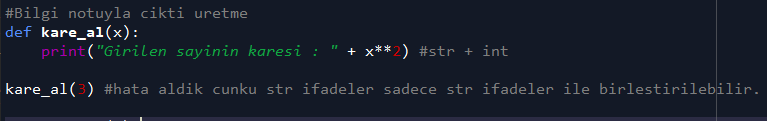


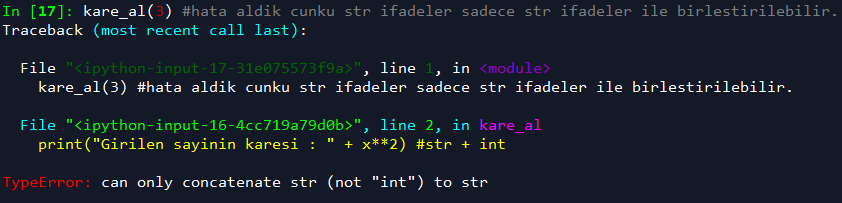
### Fonksiyon Nasıl Yazılır ?

def ile fonksiyon oluşturacağımızı belirtiriz.



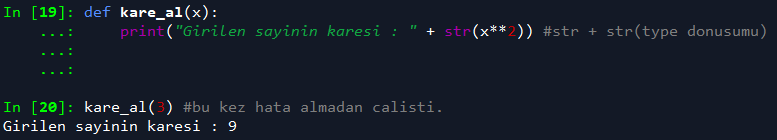
### Bilgi Notuyla Çıktı Üretmek

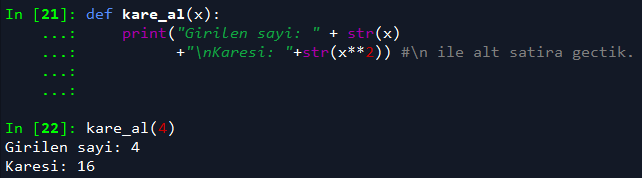


Bu fonksiyonu çalıştırınca aldığımız hata : 

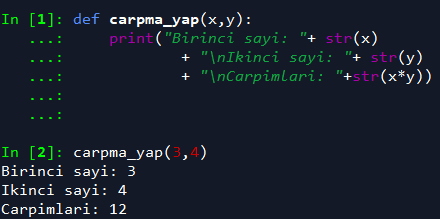
str ifadeler ile sadece str ifadeler birleştirilebilir!

type dönüşümü yapmalıyız.:



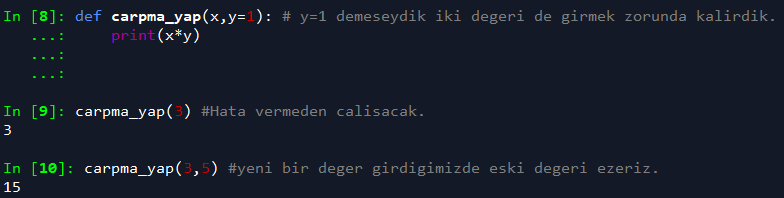
Başka bir örnek: 

### İki Argümanlı Fonksiyon Tanımlamak



### Ön Tanımlı Argümanlar

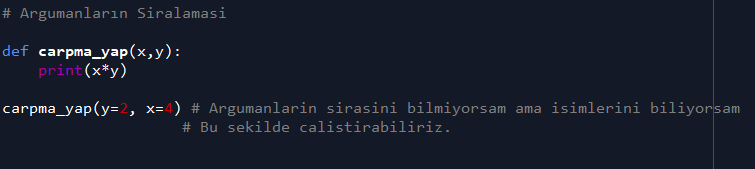
Print() fonksiyonundan hatırlayacağımız gibi sep() ve end() gibi argümanlardır.



y=1 yazarak ön tanımlı bir argüman oluşturmuş olduk.

#### Argümanların Sıralaması

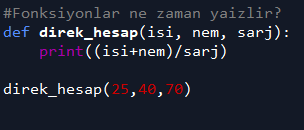
Argümanların sırasını bilmediğimiz fakat isimlerini bildiğimiz zaman aşağıdaki şekilde çalıştırabiliriz.



### 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?

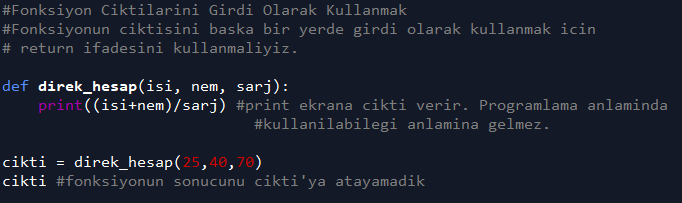
Hayır, fonksiyonu bir kez yazıp her lambada o fonksiyonu kullanacağız. 

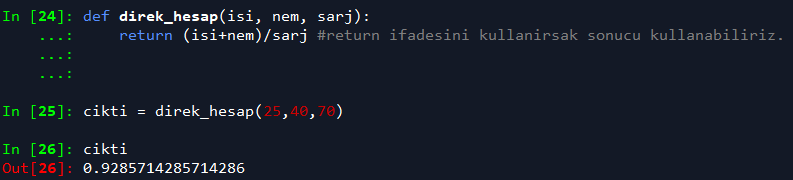
### Fonksiyon Çıktılarını Girdi Olarak Kullanmak

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

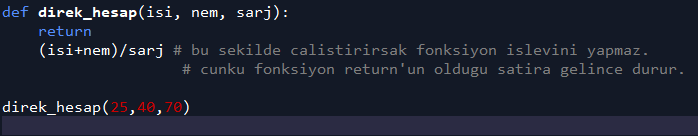
print() ekrana çıktı verir. Programlama anlamında kullanılabileceği anlamına gelmez.

Aşağıdaki örnekte görebiliriz.





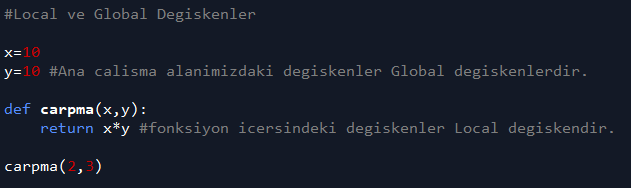
Fonksiyon return ifadesine gelince durur:



### Local ve Global Değişkenler

Ana çalışma alanımızdaki değişkenler Global değişkenlerdir.

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

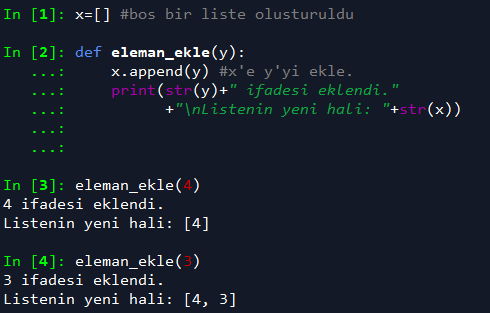


### Local Etki Alanından Global Etki Alanını Değiştirme

Yazmış olduğumuz bir döngü içerisinde ya da tanımlamış olduğumuz bir fonksiyon içerisinden 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 içersinde (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.



**NOT=**

Argüman sayısı bilinmiyorsa argüman isminden önce \* ekleyin

