

Data Science

Çalışma Dökümanı

Recep Aydoğdu

İçindekiler

[Data Science Kullanılan Alanlar 7](#_Toc37552064)

[Data Science Proje Döngüsü 7](#_Toc37552065)

[Veri Bilimine Giriş Alıştırmalar – 1 8](#_Toc37552066)

[Veri Bilimine Giriş Alıştırmalar – 2 11](#_Toc37552067)

[Python Programlama 15](#_Toc37552068)

[Temel Hareketler 15](#_Toc37552069)

[Integer, Float ve String 15](#_Toc37552070)

[Integer 15](#_Toc37552071)

[Float 15](#_Toc37552072)

[String 15](#_Toc37552073)

[Type 15](#_Toc37552074)

[String Metodları 16](#_Toc37552075)

[len() 16](#_Toc37552076)

[upper() & lower() 16](#_Toc37552077)

[isupper() & islower() 16](#_Toc37552078)

[replace() 16](#_Toc37552079)

[strip() 16](#_Toc37552080)

[dir() 17](#_Toc37552081)

[capitalize() 17](#_Toc37552082)

[title() 17](#_Toc37552083)

[Substring 17](#_Toc37552084)

[Type Dönüşümleri 17](#_Toc37552085)

[print() fonksiyonu 17](#_Toc37552086)

[Python Programlama Alıştırmalar – 1 18](#_Toc37552087)

[Python Programlama Alıştırmalar – 2 21](#_Toc37552088)

[Python Programlama Alıştırmalar – 3 26](#_Toc37552089)

[Veri Yapıları (Data Types) 30](#_Toc37552090)

[Listeler 30](#_Toc37552091)

[Liste Elemanlarına Ulaşma 30](#_Toc37552092)

[Liste İçi Type Sorgulama 30](#_Toc37552093)

[Liste elemanlarını değiştirme 31](#_Toc37552094)

[Listeye eleman ekleme 31](#_Toc37552095)

[Listeden eleman silme 31](#_Toc37552096)

[append ve remove metodlari 31](#_Toc37552097)

[insert metodu 31](#_Toc37552098)

[pop metodu 31](#_Toc37552099)

[count metodu 32](#_Toc37552100)

[copy metodu 32](#_Toc37552101)

[extend metodu 32](#_Toc37552102)

[index metodu 32](#_Toc37552103)

[reverse metodu 32](#_Toc37552104)

[sort metodu 32](#_Toc37552105)

[clear metodu 33](#_Toc37552106)

[Tuple (Demet) 33](#_Toc37552107)

[Tuple Oluşturma 33](#_Toc37552108)

[Eleman İşlemleri 33](#_Toc37552109)

[Dictionary (Sözlük) 33](#_Toc37552110)

[Dictionary Nedir? 33](#_Toc37552111)

[Dictionary Oluşturma 33](#_Toc37552112)

[Eleman Seçme İşlemleri 34](#_Toc37552113)

[Eleman Ekleme & Değiştirme 34](#_Toc37552114)

[Sets (Kümeler) 35](#_Toc37552115)

[Set Oluşturma 35](#_Toc37552116)

[Set’lere eleman ekleme ve çıkarma işlemleri 36](#_Toc37552117)

[Set’lerde Fark İşlemleri 38](#_Toc37552118)

[Set’lerde Kesişim ve Birleşim İşlemleri 38](#_Toc37552119)

[Set’lerde Sorgu İşlemleri 39](#_Toc37552120)

[Veri Yapıları Özet 39](#_Toc37552121)

[Python Programlama Alıştırmalar – 4 40](#_Toc37552122)

[Python Programlama Alıştırmalar – 5 43](#_Toc37552123)

[Python Programlama Alıştırmalar – 6 48](#_Toc37552124)

[Fonksiyonlar 53](#_Toc37552125)

[Fonksiyon Nedir? 53](#_Toc37552126)

[Matematiksel İşlemler 53](#_Toc37552127)

[Üs Alma 53](#_Toc37552128)

[Fonksiyon Nasıl Yazılır ? 53](#_Toc37552129)

[Bilgi Notuyla Çıktı Üretmek 54](#_Toc37552130)

[İki Argümanlı Fonksiyon Tanımlamak 54](#_Toc37552131)

[Ön Tanımlı Argümanlar 55](#_Toc37552132)

[Argümanların Sıralaması 55](#_Toc37552133)

[Ne Zaman Fonksiyon Yazılır? 55](#_Toc37552134)

[Fonksiyon Çıktılarını Girdi Olarak Kullanmak 55](#_Toc37552135)

[Local ve Global Değişkenler 56](#_Toc37552136)

[Local Etki Alanından Global Etki Alanını Değiştirme 57](#_Toc37552137)

[Karar-Kontrol Yapıları (Koşullar) 58](#_Toc37552138)

[Koşul Nedir? 58](#_Toc37552139)

[True – False Sorgulamaları (Boolean) 58](#_Toc37552140)

[if – else – elif 58](#_Toc37552141)

[**Uygulama:** if ve input ile kullanıcı etkileşimli program 60](#_Toc37552142)

[Döngüler 60](#_Toc37552143)

[For Döngüsü 60](#_Toc37552144)

[Döngü ve Fonksiyonların Birlikte Kullanımı 61](#_Toc37552145)

[**Uygulama:** if, for ve fonksiyonların birlikte kullanımı 61](#_Toc37552146)

[break & continue 61](#_Toc37552147)

[while 62](#_Toc37552148)

[Python Programlama Alıştırmalar - 7 63](#_Toc37552149)

[Python Programlama Alıştırmalar - 8 67](#_Toc37552150)

[Python Programlama Alıştırmalar – 9 72](#_Toc37552151)

[Object Oriented Programming 78](#_Toc37552152)

[Class’lara Giriş ve Class Tanımlamak 78](#_Toc37552153)

[Class Nedir? 78](#_Toc37552154)

[Class Özellikleri 78](#_Toc37552155)

[Class Özelliklerine Erişmek 78](#_Toc37552156)

[Class Özelliklerini Değiştirmek 78](#_Toc37552157)

[Class Örneklendirmesi (instantiniation) 78](#_Toc37552158)

[Örnek Özellikleri 79](#_Toc37552159)

[Örnek Metodları 80](#_Toc37552160)

[Miras Yapıları (inheritance) 80](#_Toc37552161)

[Functional Programming 81](#_Toc37552162)

[Fonksiyonel Programlamaya Giriş 81](#_Toc37552163)

[Yan Etkisiz Fonksiyonlar (Pure Functions) 81](#_Toc37552164)

[Örnek-1: Bağımsızlık 81](#_Toc37552165)

[Örnek-2: Ölümcül Yan Etkiler 82](#_Toc37552166)

[İsimsiz Fonksiyonlar (Lambda) (Anonymous Functions) 83](#_Toc37552167)

[Vektörel Operasyonlar (Vectorel Operations) 83](#_Toc37552168)

[OOP ile iki listeyi çarpmak 83](#_Toc37552169)

[Functionel Programming ile 83](#_Toc37552170)

[Map & Filter & Reduce 84](#_Toc37552171)

[Map 84](#_Toc37552172)

[Filter 84](#_Toc37552173)

[Reduce 84](#_Toc37552174)

[Modül Oluşturma 84](#_Toc37552175)

[Hatalar/İstisnalar (exception) 85](#_Toc37552176)

[Python Programlama Alıştırmalar – 10 86](#_Toc37552177)

[Python Programlama Alıştırmalar – 11 90](#_Toc37552178)

[Python Programlama Alıştırmalar – 12 95](#_Toc37552179)

[Python ile Veri Manipülasyonu: NumPy & Pandas 100](#_Toc37552180)

[NumPy (Numerical Python) 100](#_Toc37552181)

[NumPy Giriş 100](#_Toc37552182)

[Neden NumPy? 101](#_Toc37552183)

[NumPy Array’i Oluşturmak 101](#_Toc37552184)

[zeros, ones, full, random, arange, linspace, random.normal, random.randint 102](#_Toc37552185)

[NumPy Array Özellikleri 103](#_Toc37552186)

[Matris Oluşturma 103](#_Toc37552187)

[Reshaping (Array’i Yeniden Şekillendirme) 104](#_Toc37552188)

[Concantenation (Array Birleştirme) 105](#_Toc37552189)

[Splitting (Array Ayırma) 106](#_Toc37552190)

[İki Boyutlu Array Ayırma 106](#_Toc37552191)

[Sorting (Sıralama) 107](#_Toc37552192)

[Matris sıralama 107](#_Toc37552193)

[Index ile Elemana Erişmek 108](#_Toc37552194)

[Matrislerde elemana erişme işlemleri 108](#_Toc37552195)

[Slicing (Array Alt Küme İşlemleri) 109](#_Toc37552196)

[Matrislerde Slicing İşlemleri 109](#_Toc37552197)

[Alt Küme Üzerinde İşlem Yapmak 110](#_Toc37552198)

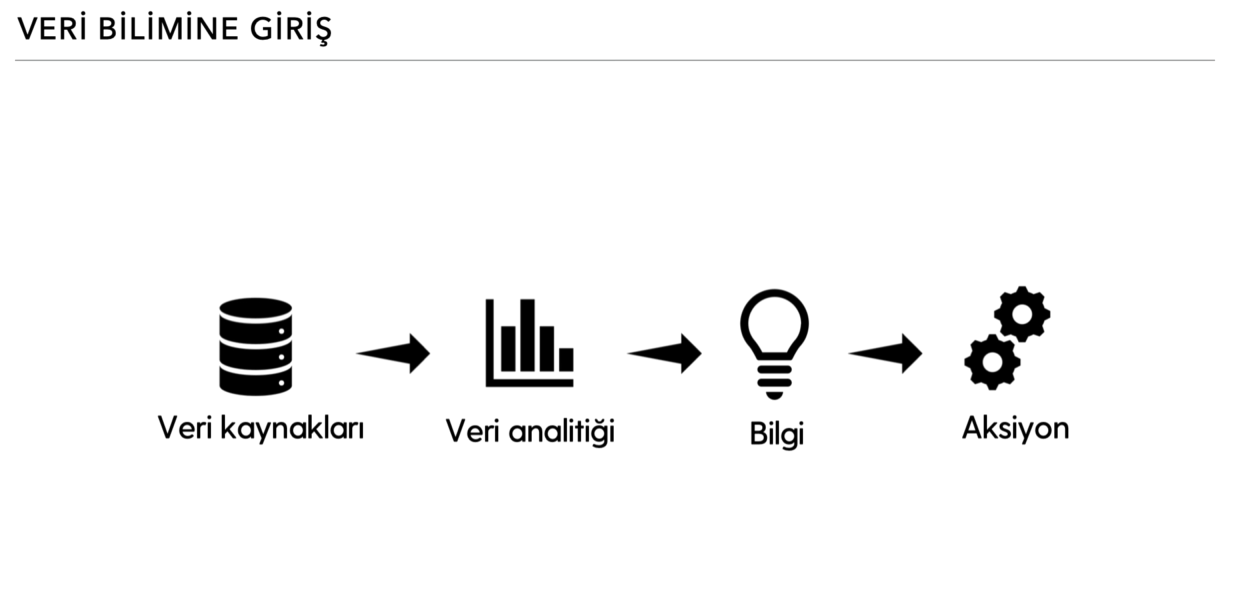
[Fancy Index ile Elemanlara Erişmek 111](#_Toc37552199)

[Matrislerde Fancy Index Kullanımı 111](#_Toc37552200)

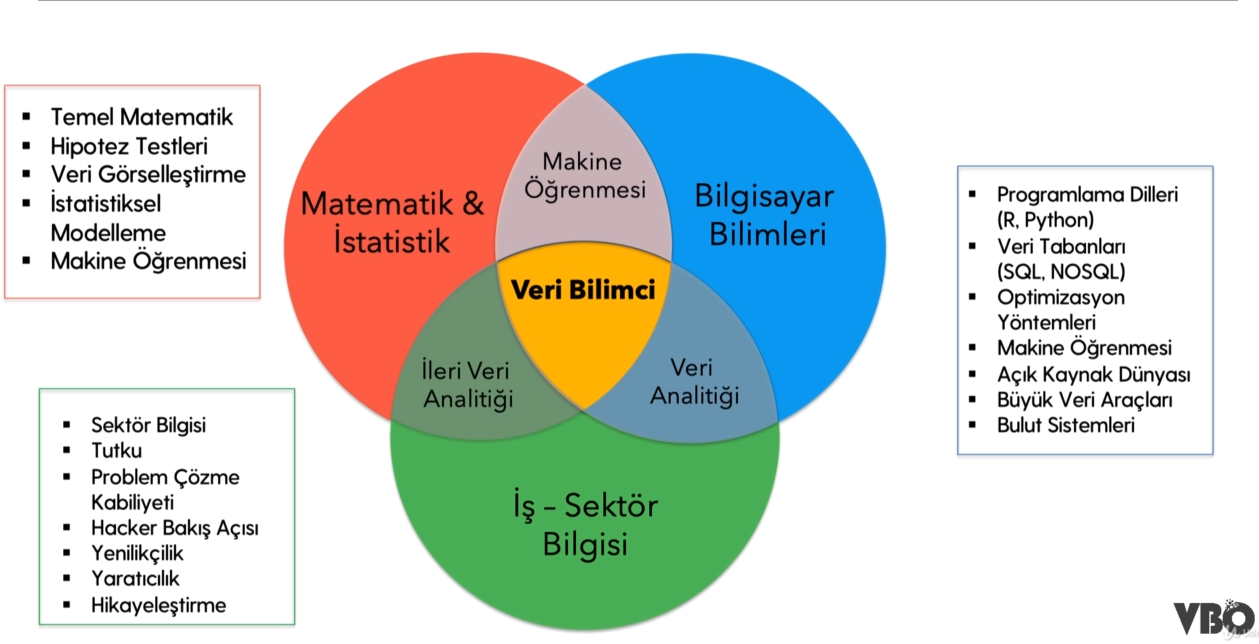
[Basit Index ile Fancy kullanımı 112](#_Toc37552201)

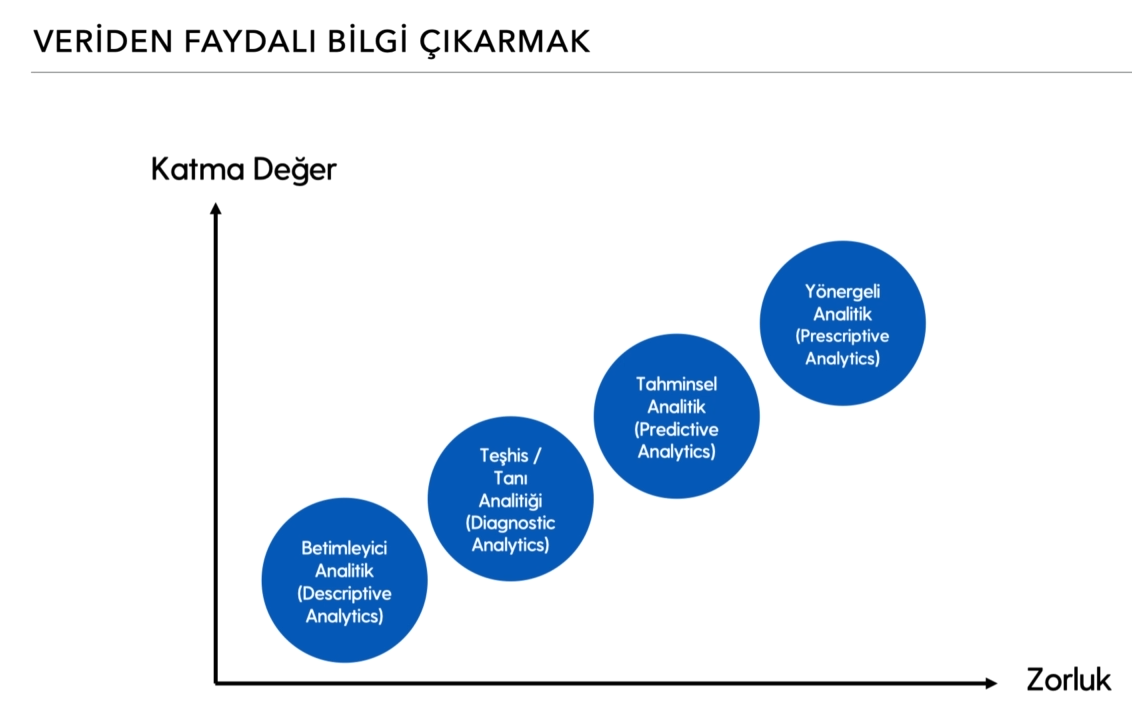
[Slice ile Fancy kullanımı 112](#_Toc37552202)

**Data Science**

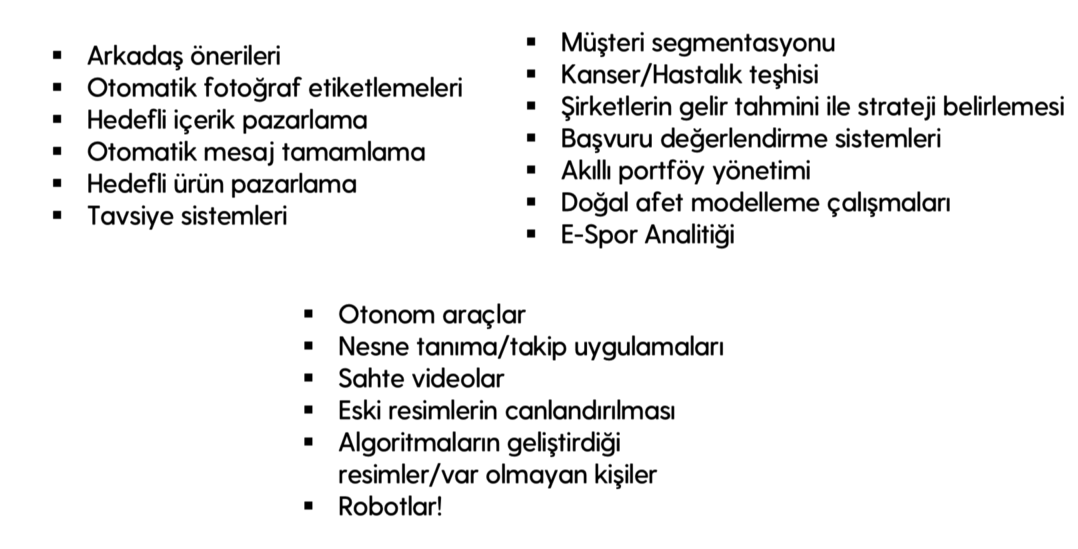


Veri Bilimci, veriden faydalı bilgi çıkarma sürecini yöneten kişidir.

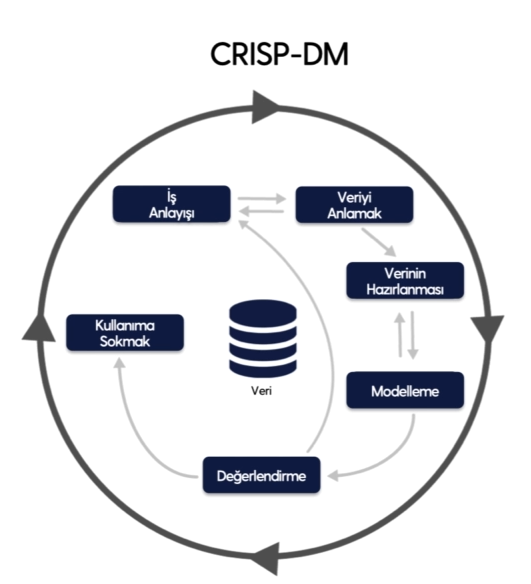




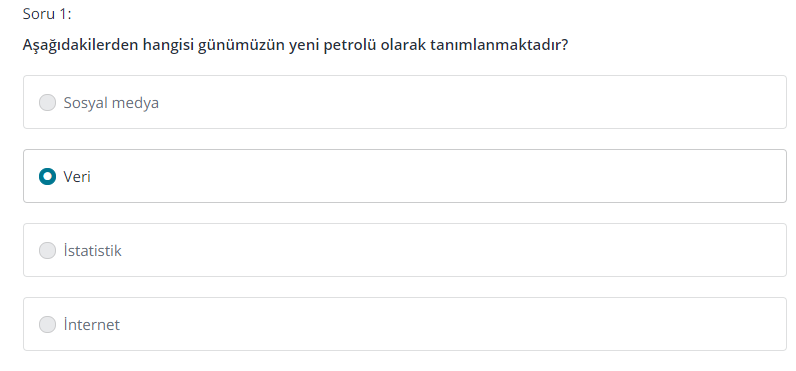
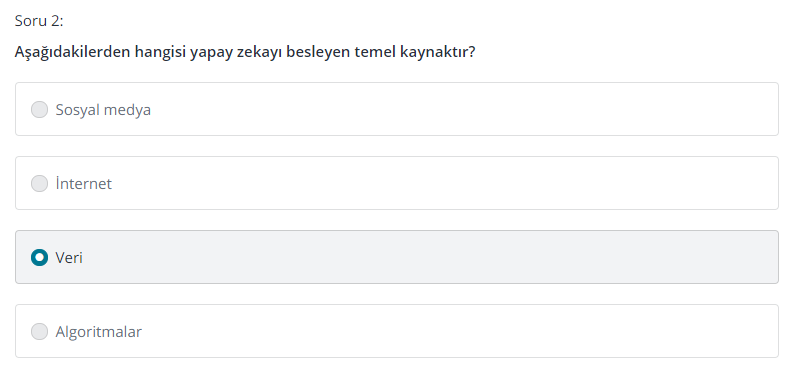
# Data Science Kullanılan Alanlar

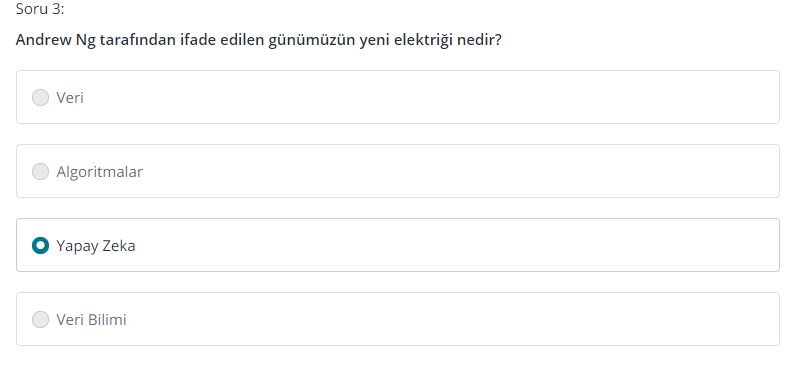


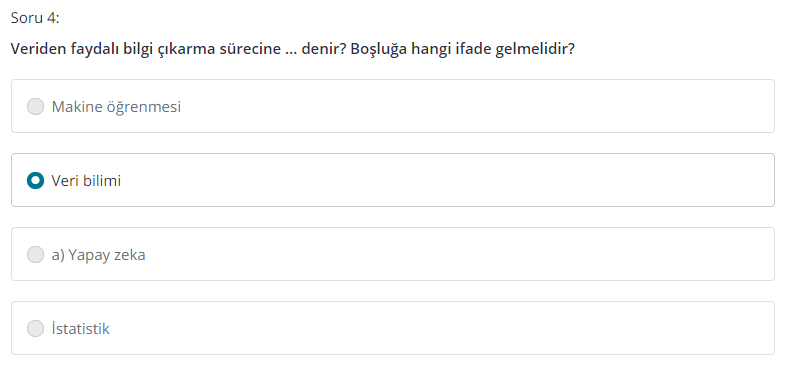
# Data Science Proje Döngüsü

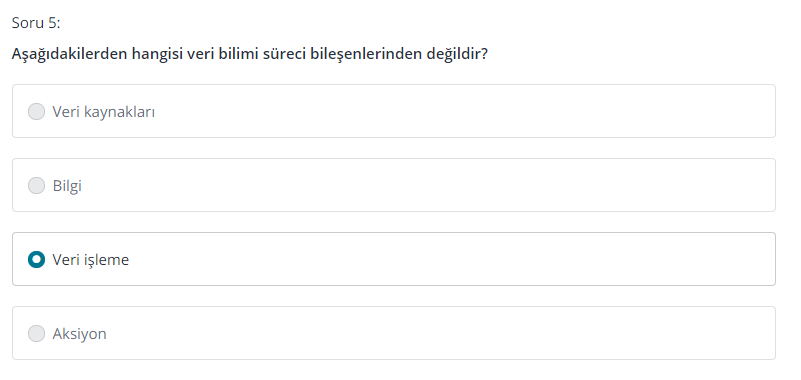


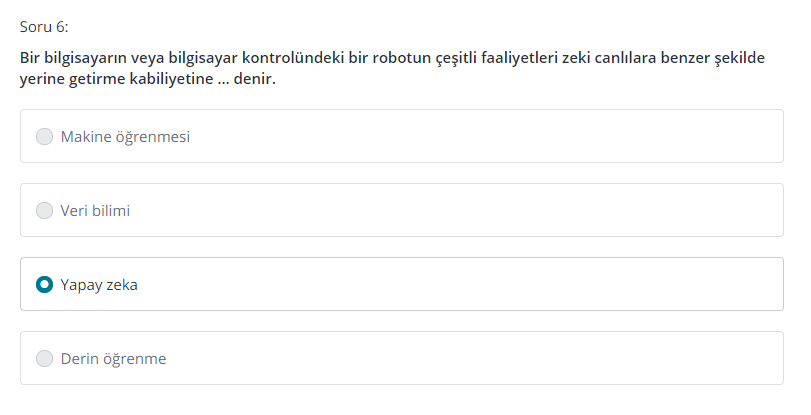
### Veri Bilimine Giriş Alıştırmalar – 1

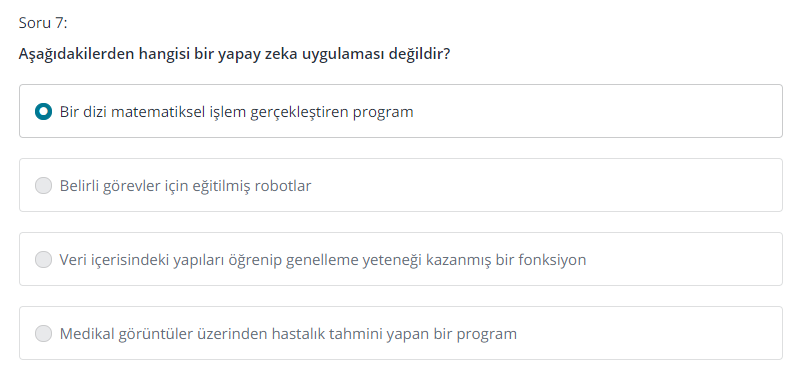
 

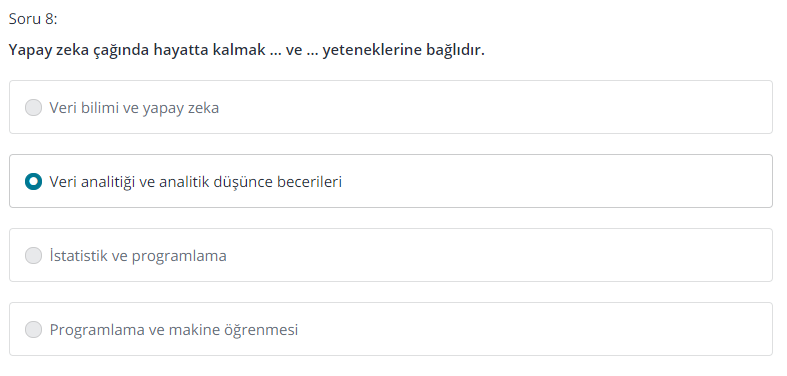


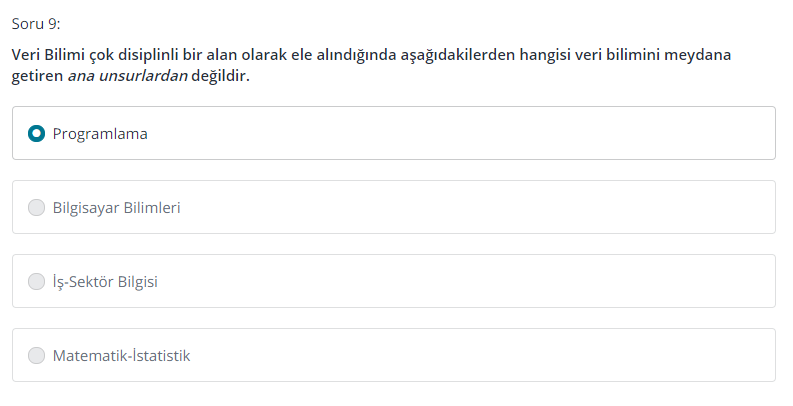




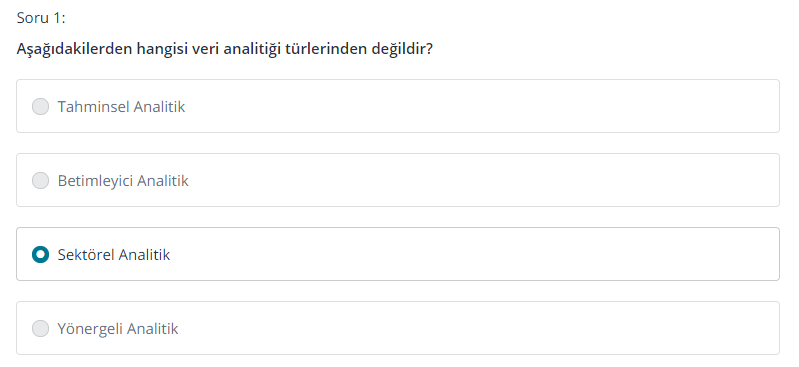


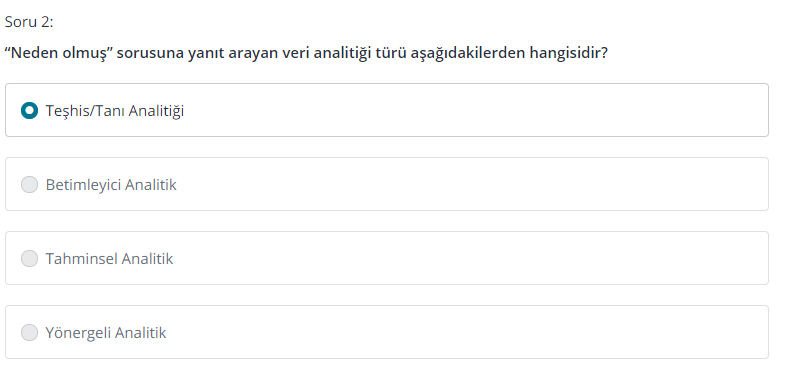


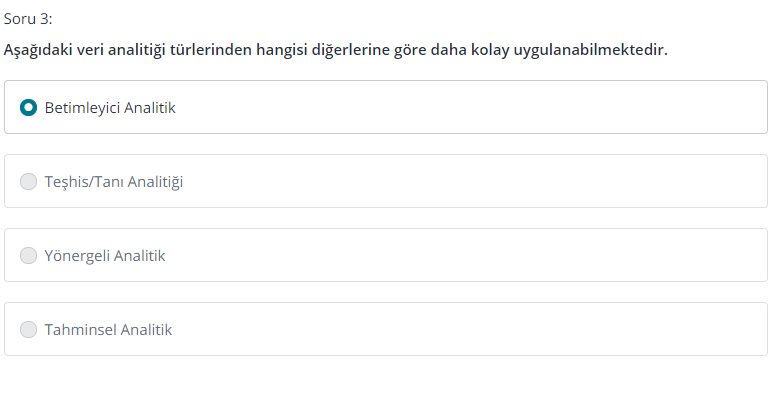


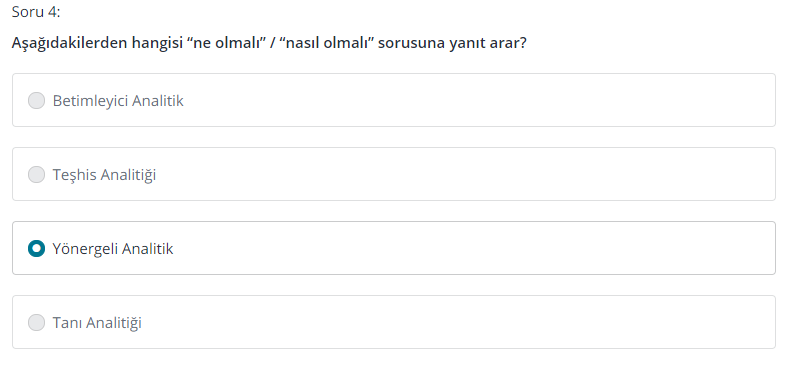
 

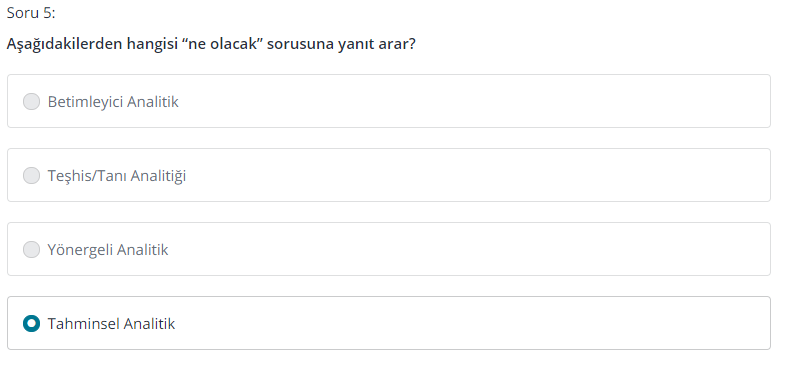
### Veri Bilimine Giriş Alıştırmalar – 2

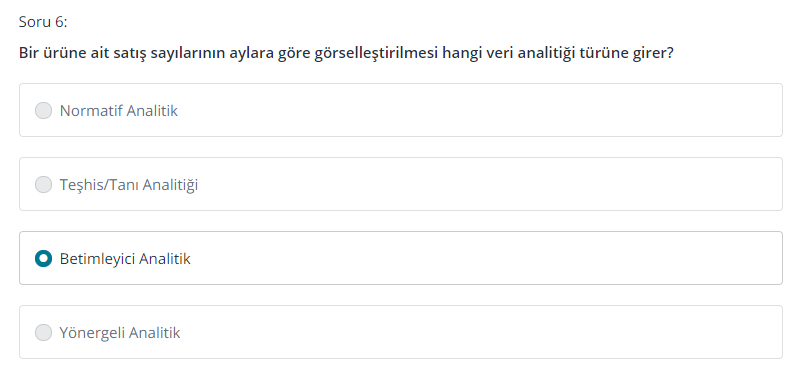


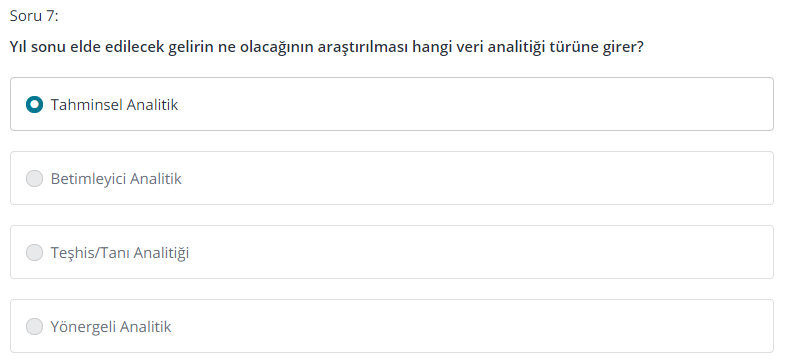


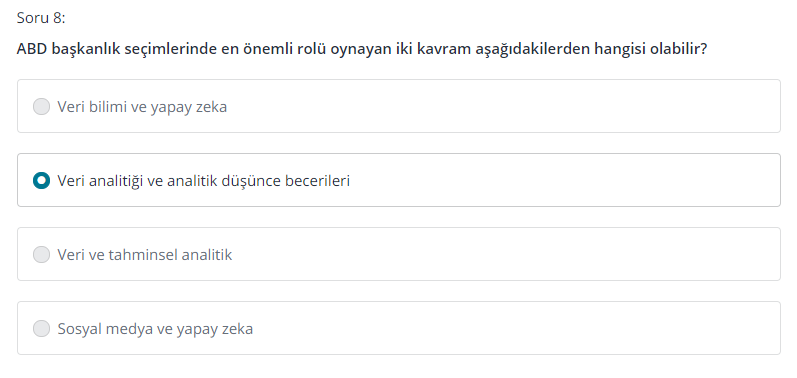


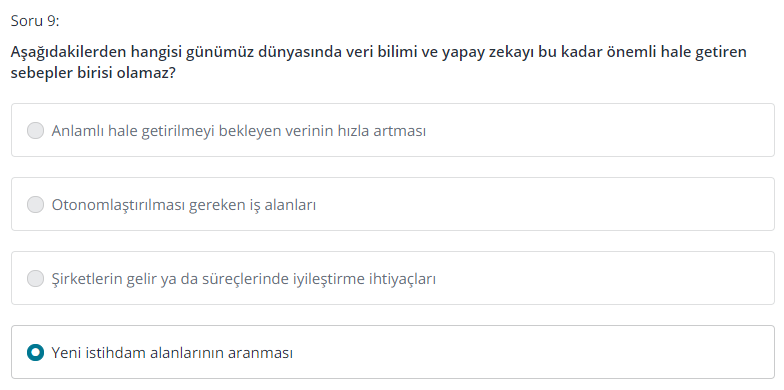


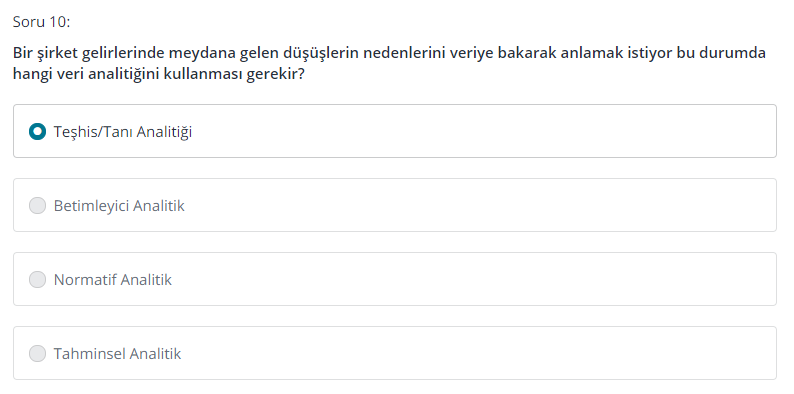












# Python Programlama

* Python, Google tarafından destekleniyor.
* Python’ın yorumlayıcı özelliği vardır. Etkileşim özelliğine sahiptir. (Soru-cevap mantığıyla çalışır.)
* High Level bir programlama dili.
* OPP (nesneye dayalı) ve FP(Fonksiyonel programlama).

## Temel Hareketler

* Seçili alanı F9 tuşu ile çalıştırabiliriz.
* Python programlama dilinde oluşturulan her şey bir nesnedir.
* Yorum satırı oluşturmak için satır başına # koyarız.

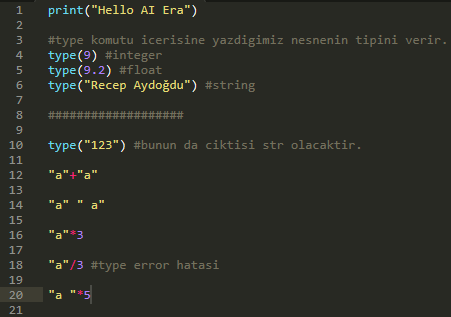
### Integer, Float ve String

Integer **=** 9 gibi ondalıksız sayılar.

Float **=** 9.2 gibi ondalıklı sayılar.

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

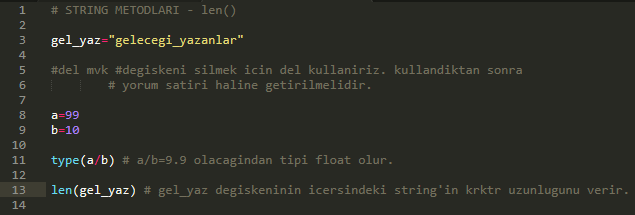
Type **=** type() içersine yazılan nesnenin tipini verir.



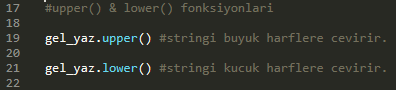
* “a”+”a” 🡪 aa
* “a””a” 🡪 aa
* “a”\*3 🡪 aaa
* “a”-“b” 🡪 TypeError alırız. Bu operatör sadece numeric ifadelerde kullanılır.
* “a”/3 à TypeError

### String Metodları

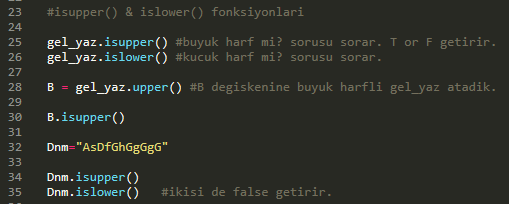
len()**=** içerisine yazılan değişkenin uzunluğunu verir.



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



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



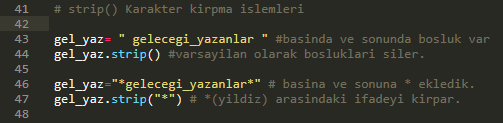
replace() **=**



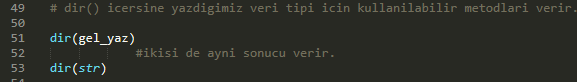
replace(“eski\_karakter”,”yeni\_karakter”)

gelecegi\_yazanlar 🡪gelecegi\_yızınlır

strip() **=** Karakter kırpma işlemleri



dir() **=**



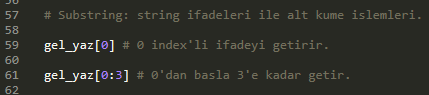
capitalize() **=** İlk harfi büyütür.

gel\_yaz.capitalized()

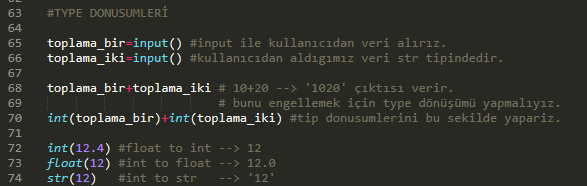
title() **=** Her kelimenin ilk harfini büyütür.

gel\_yaz.title()

Substring **=** Alt küme işlemleri



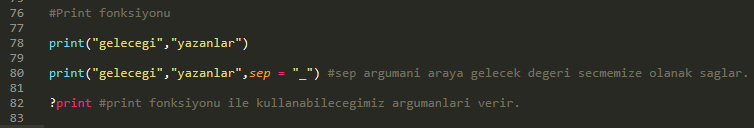
#### Type Dönüşümleri



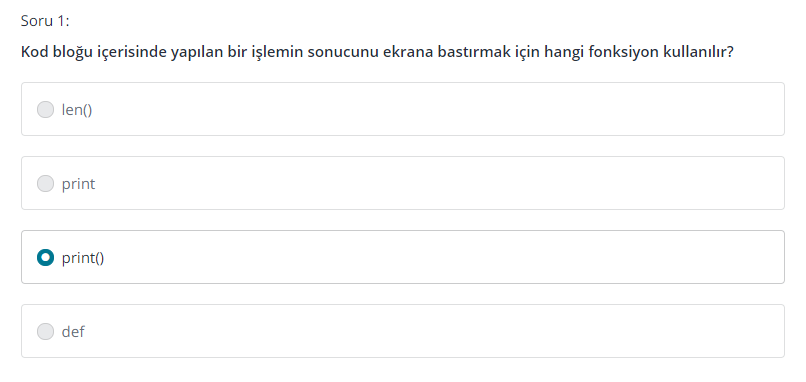
#### print() fonksiyonu

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

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

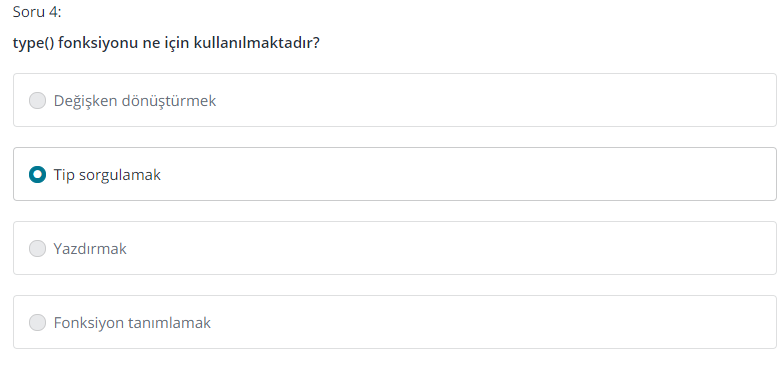


### Python Programlama Alıştırmalar – 1







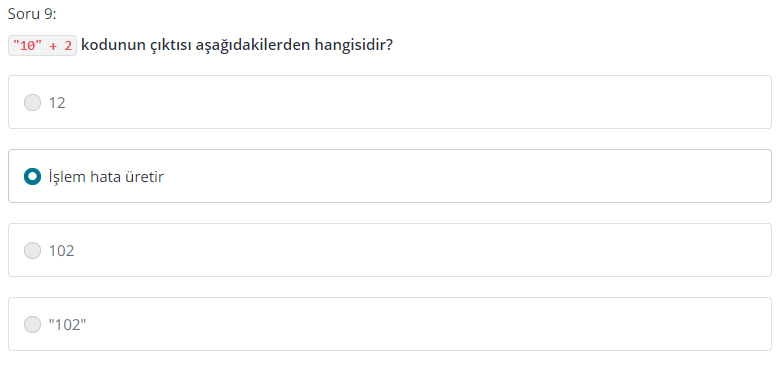








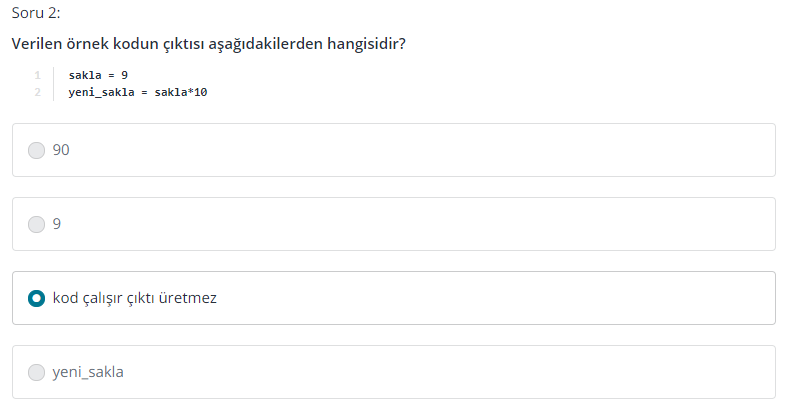


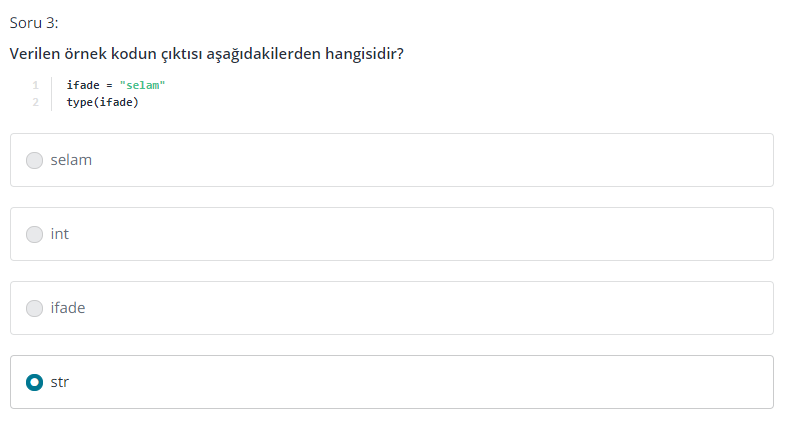


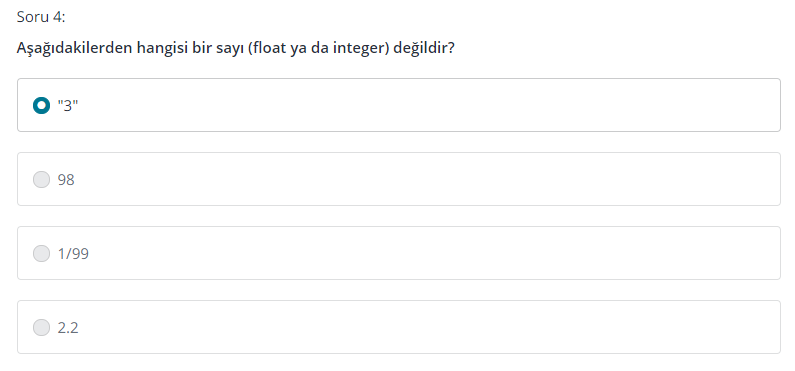


### Python Programlama Alıştırmalar – 2

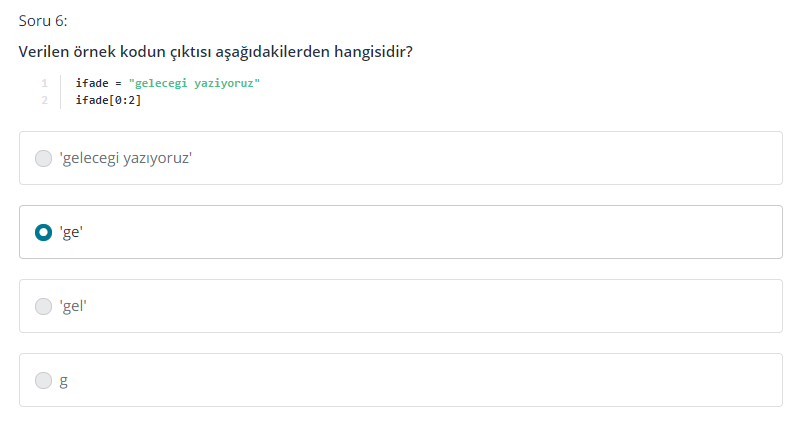




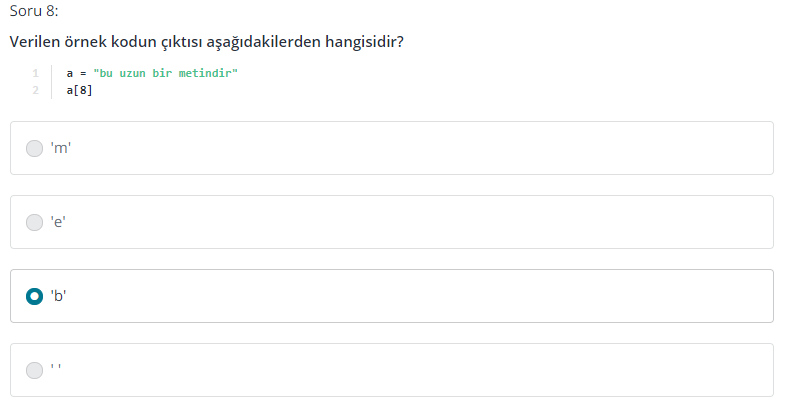


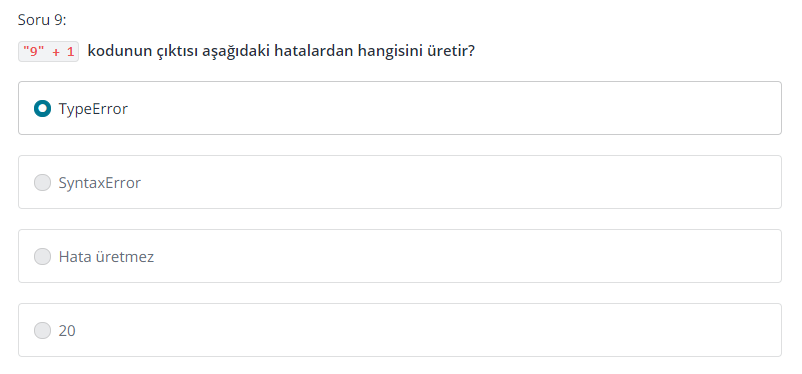


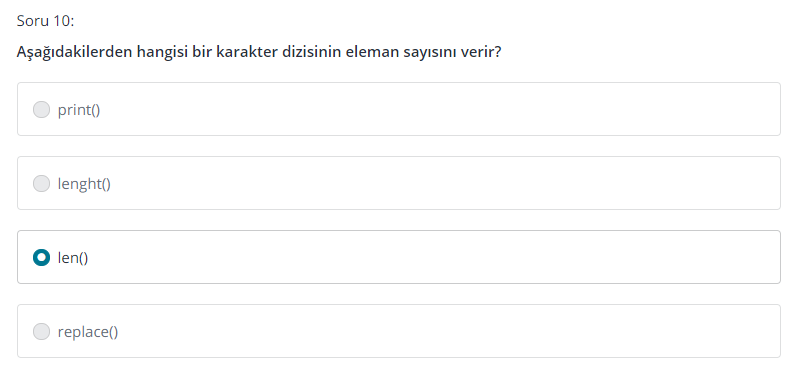




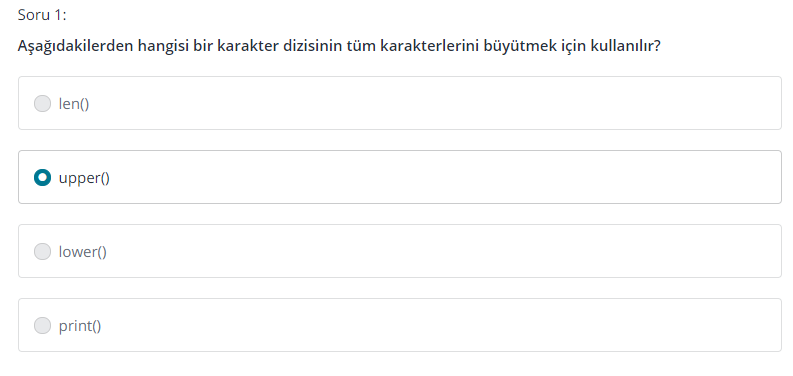


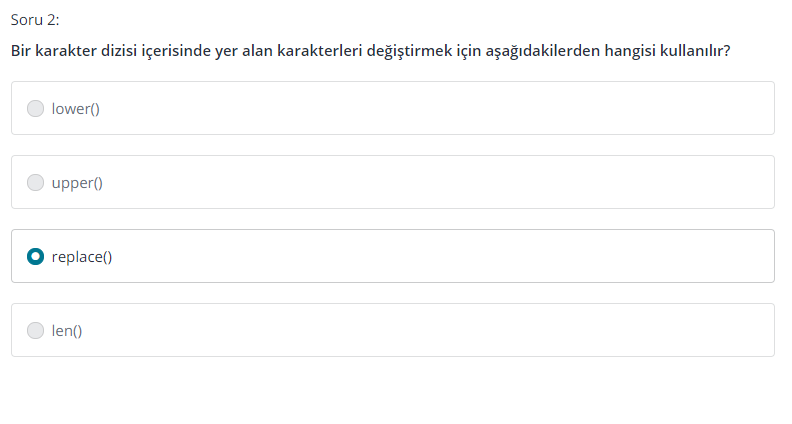




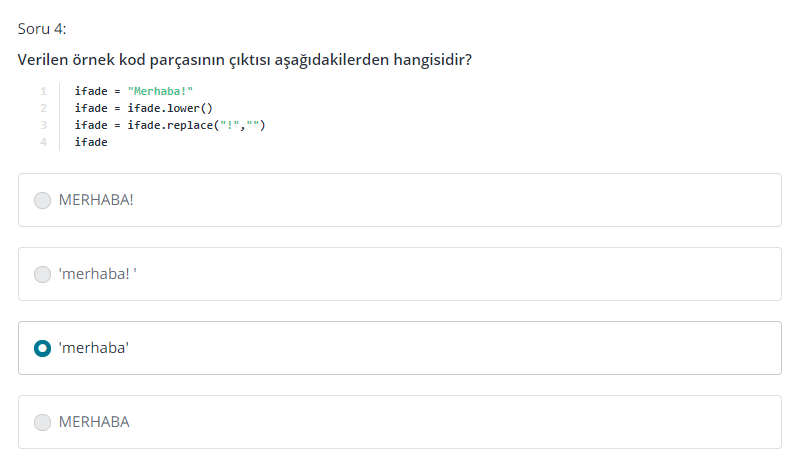


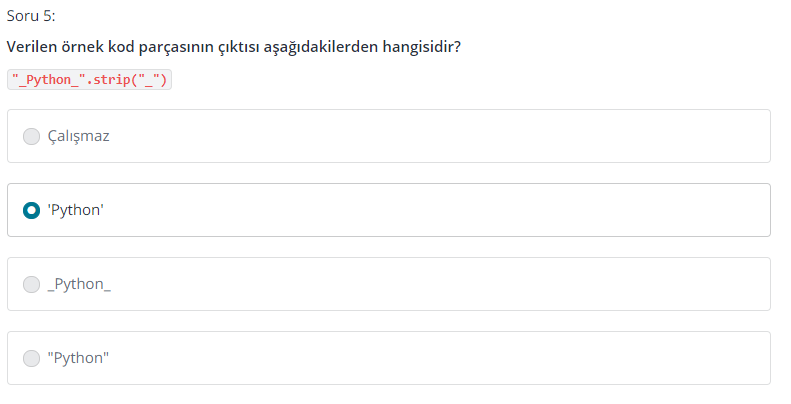
### Python Programlama Alıştırmalar – 3

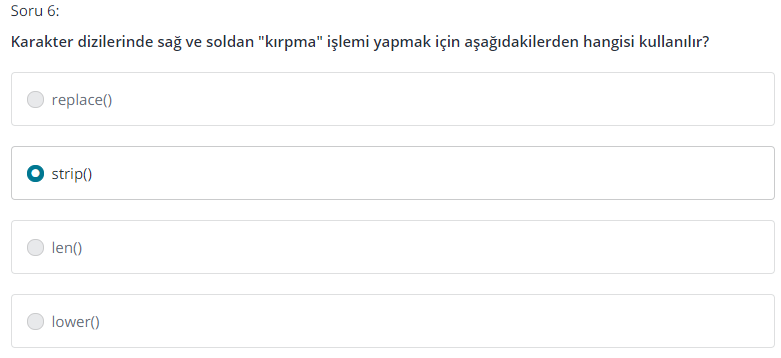


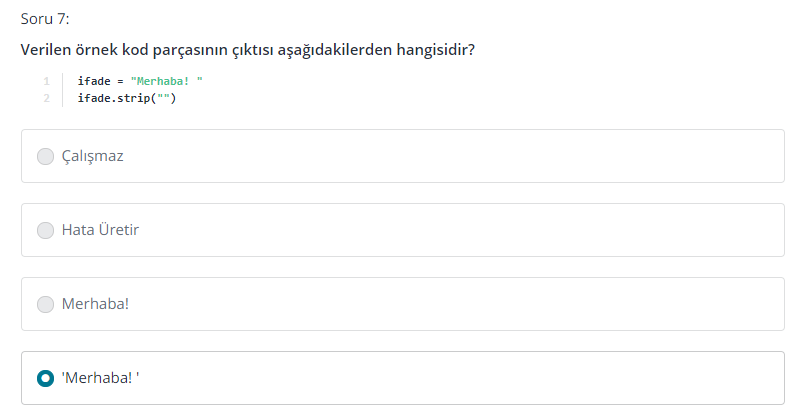


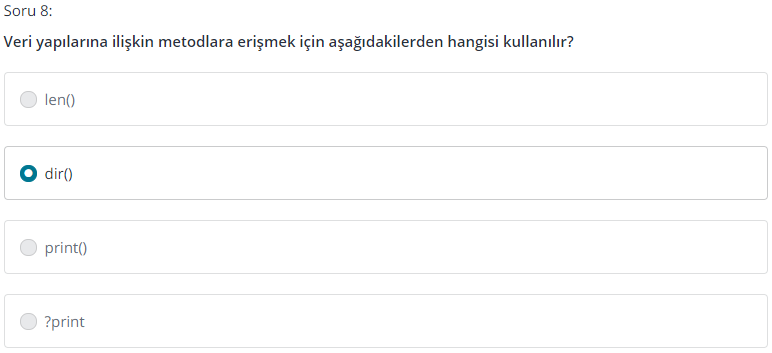


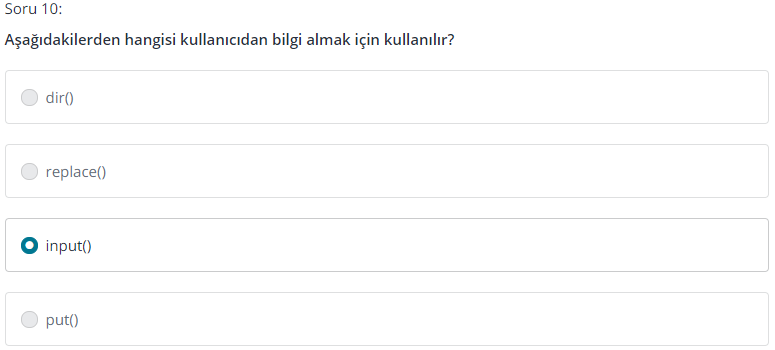










## Veri Yapıları (Data Types)

### Listeler

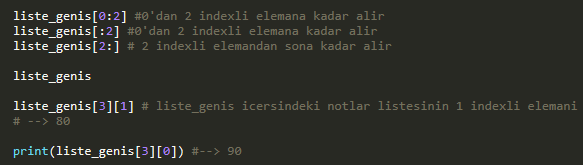
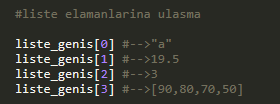
1. Değiştirilebilir
2. Kapsayıcıdır (Farklı tipte verileri tutabilir.)
3. Sıralıdır

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

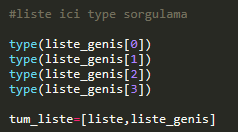
Liste bir üst type’dır içersinde farklı type’da veriler barındırabilir.



#### Liste Elemanlarına Ulaşma

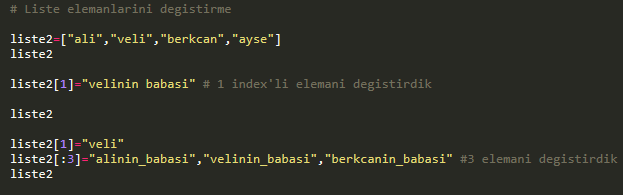


#### Liste İçi Type Sorgulama

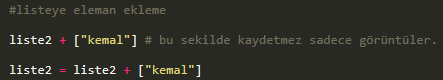


**del liste 🡪 liste’yi siler**

#### Liste elemanlarını değiştirme



#### Listeye eleman ekleme



#### Listeden eleman silme

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

#### append ve remove metodlari

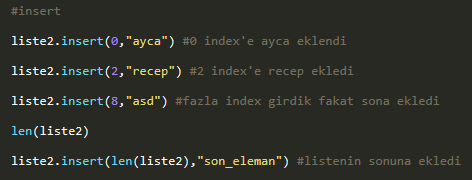
liste2.append("berkcan") 🡪sona ekleme yapar

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

liste2.remove("velinin\_babasi")

#### insert metodu

index’e göre ekleme yapar.



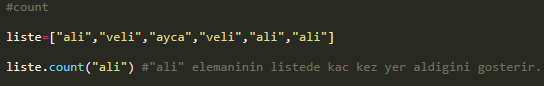
#### pop metodu

index’e göre silme yapar.

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

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

#### count metodu

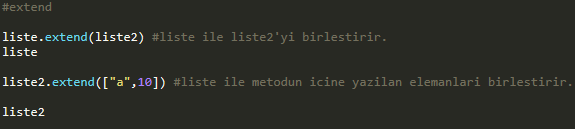


🡪 3

#### copy metodu

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

#### extend metodu

İki farklı listeyi birleştirir.

#### index metodu



#### reverse metodu

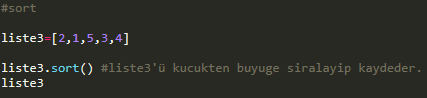
liste = [1,2,3]

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

liste = [3,2,1]

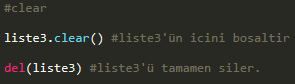
#### sort metodu

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)

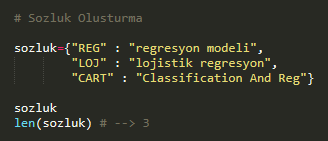
1. Kapsayıcıdır
2. Sırasızdır 🡪 Listelerden farkı budur.
3. Değiştirilebilirdir.

#### Dictionary Nedir?

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

Listelerde olduğu gibi index’leme yapılmaz.

#### Dictionary Oluşturma



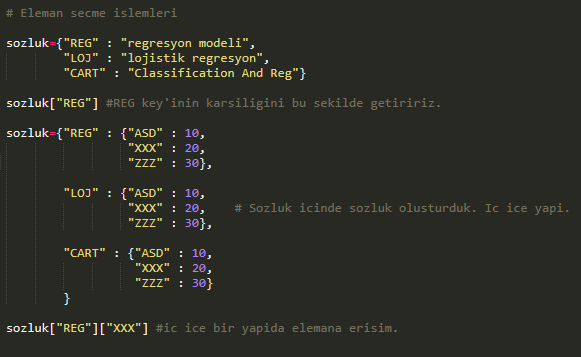
{“key” : “key’in karşılığı”}

**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.

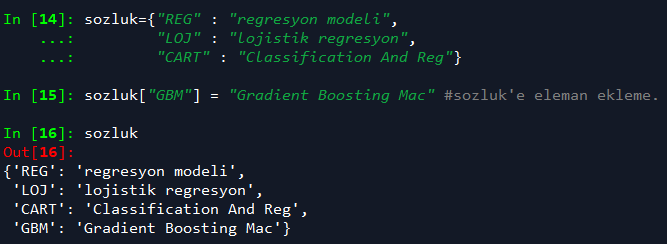
t = (“tuple”,) 🡪 sozluk = { t : “tuple’dan key olur” }

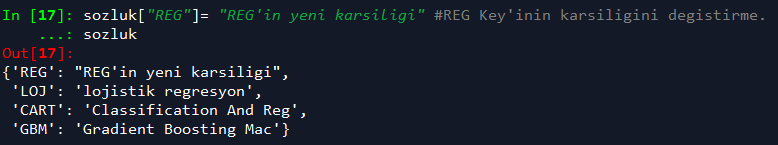
#### Eleman Seçme İşlemleri



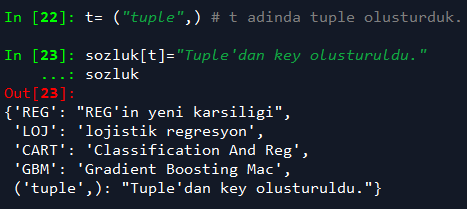


#### Eleman Ekleme & Değiştirme





REG key’i olmasaydı yeni key oluşturulacaktı.



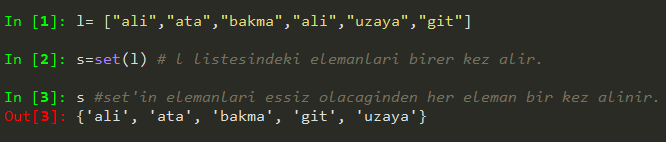
### Sets (Kümeler)

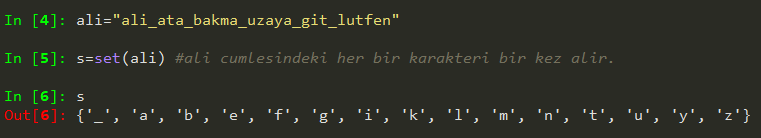
1. Sırasızdır (Index değerleri yok.)
2. Değerleri eşsizdir. (Tekrar eden değeri olmaz.)
3. Değiştirilebilir.
4. Kapsayıcıdır. Farklı türden veri yapıları barındırabilir.

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

#### Set Oluşturma

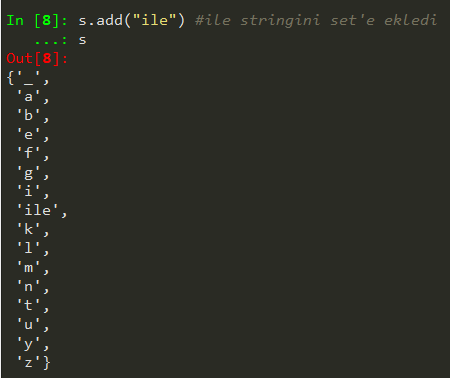
s = set() 🡪 s isminde bir set oluşturuldu.

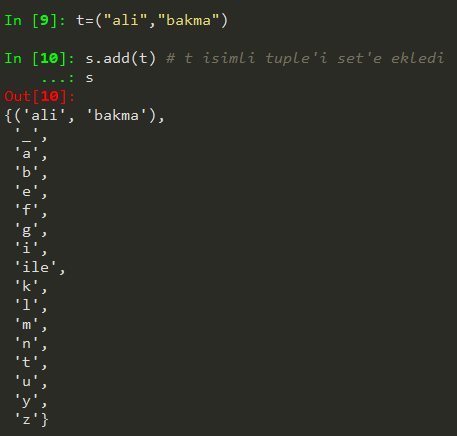


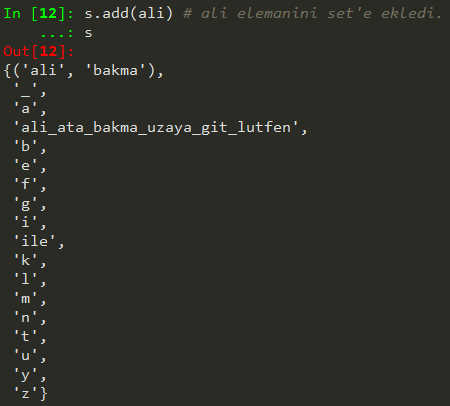


#### Set’lere eleman ekleme ve çıkarma işlemleri

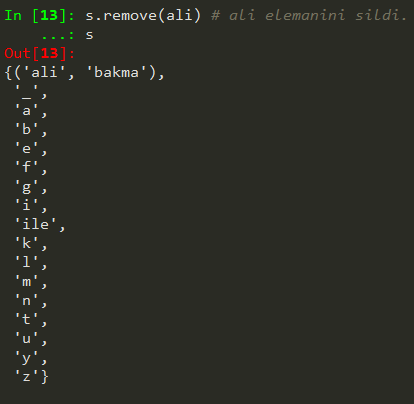
add() fonksiyonu ile ekleme yaparız.

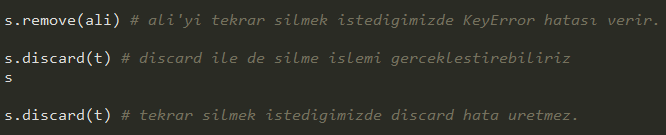






remove() fonksiyonu ile set’lerden eleman silebiliriz.

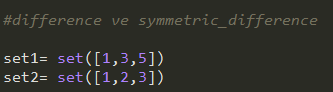




#### Set’lerde Fark İşlemleri

**difference & symmetric\_difference**

**difference =** kümelerin farkını verir.



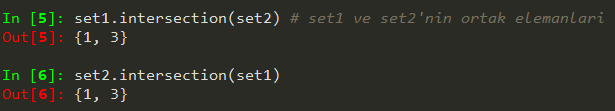
**symmetric\_difference =** ikisinde de ortak olmayan elemanları verir.



#### Set’lerde Kesişim ve Birleşim İşlemleri

**intersection & union & intersection\_update**

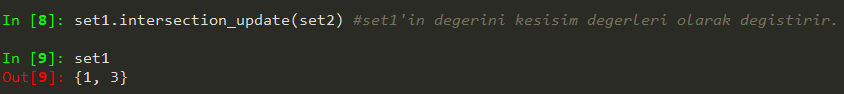
**intersection =** kesişim



**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.



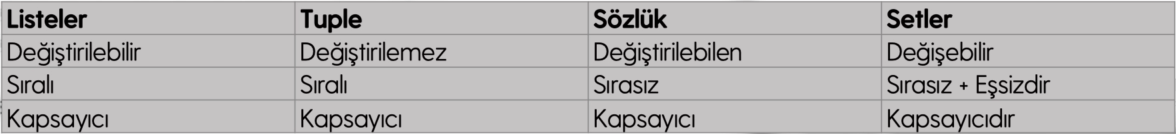
**issubset =** subset’i mi? Alt kümesi mi? sorgusunu yapar.



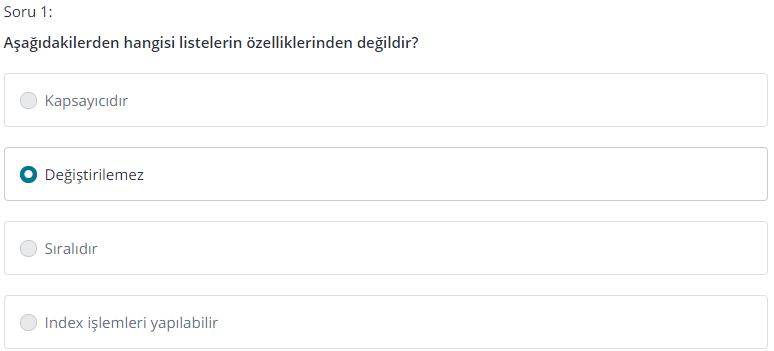
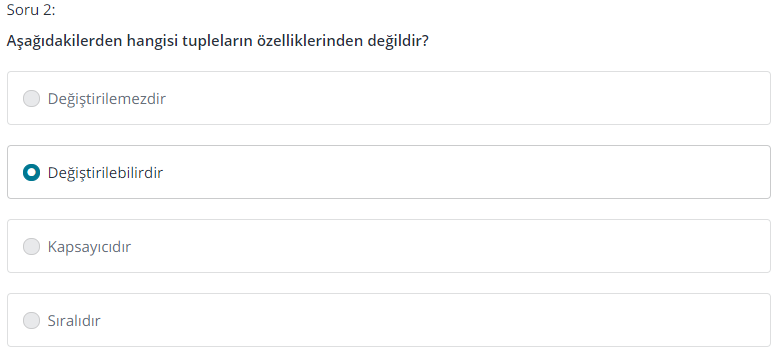
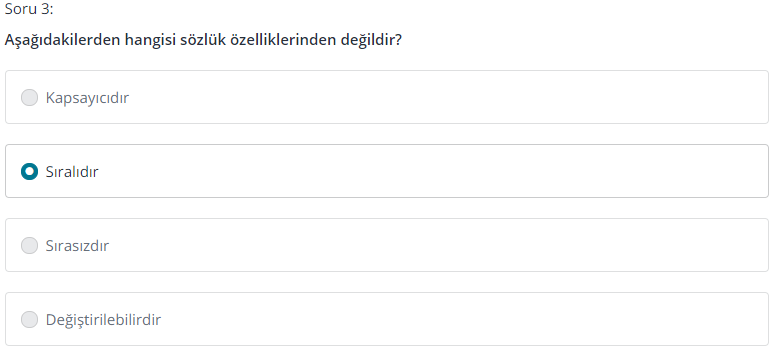
**issuperset =** Kapsar mı?

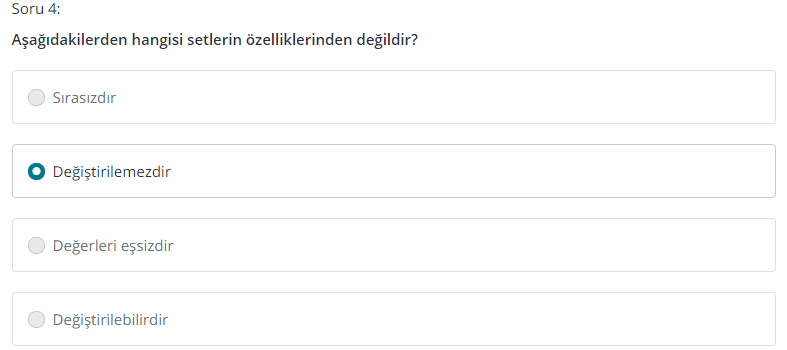
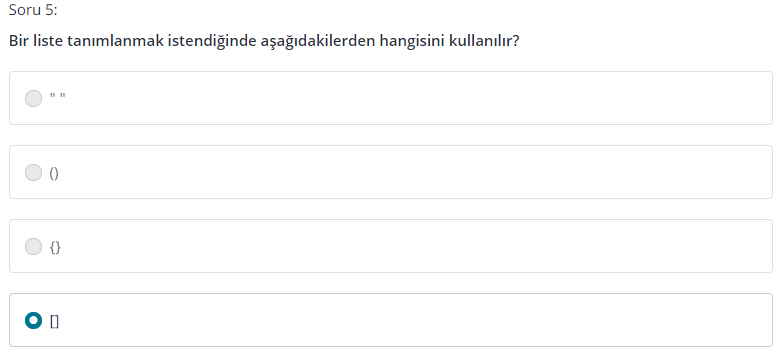
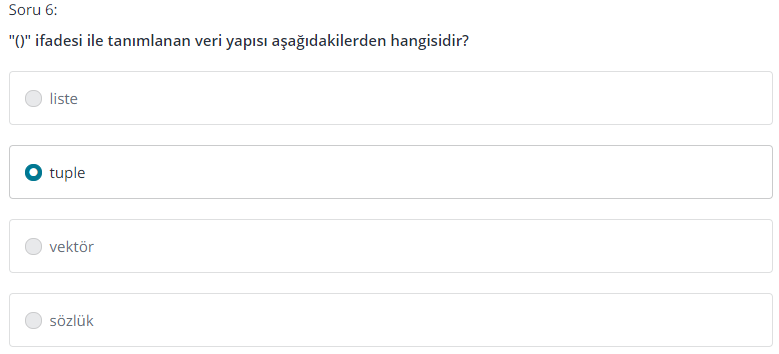


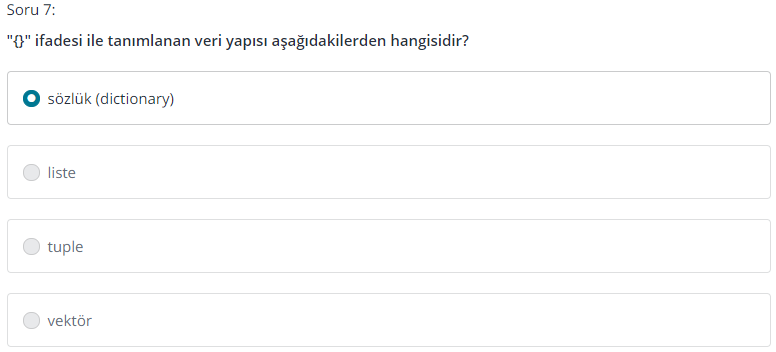
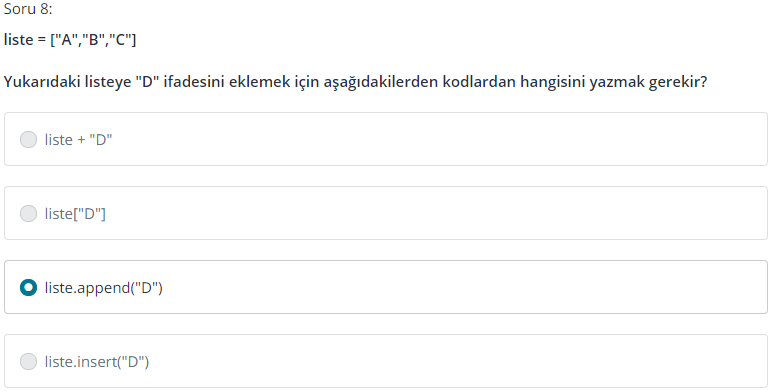
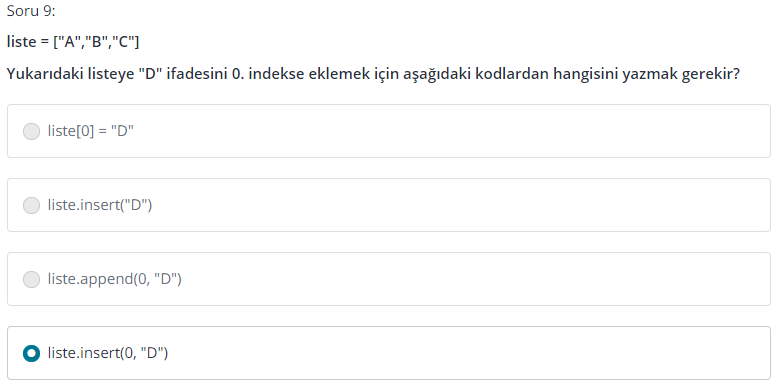
### Veri Yapıları Özet

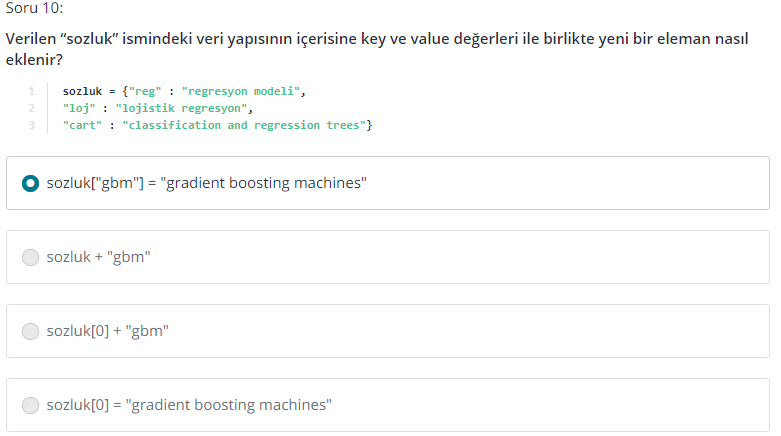


### Python Programlama Alıştırmalar – 4

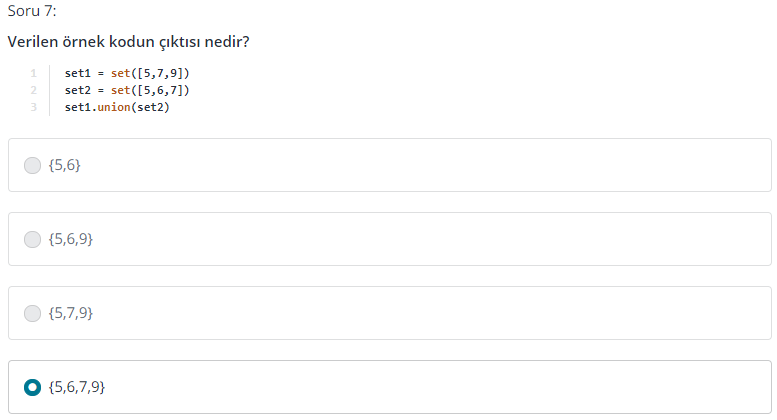


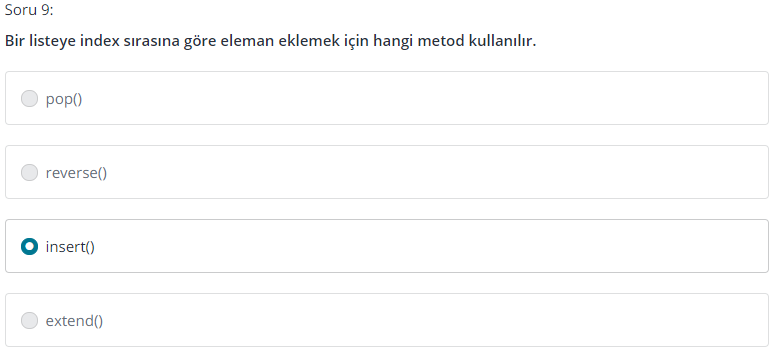
### Python Programlama Alıştırmalar – 5

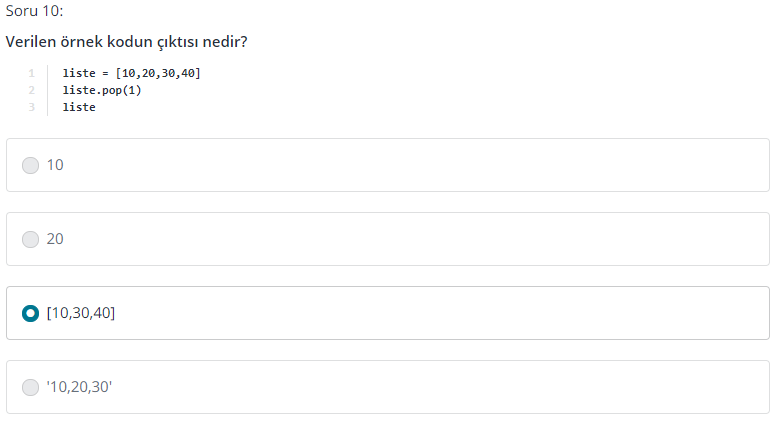


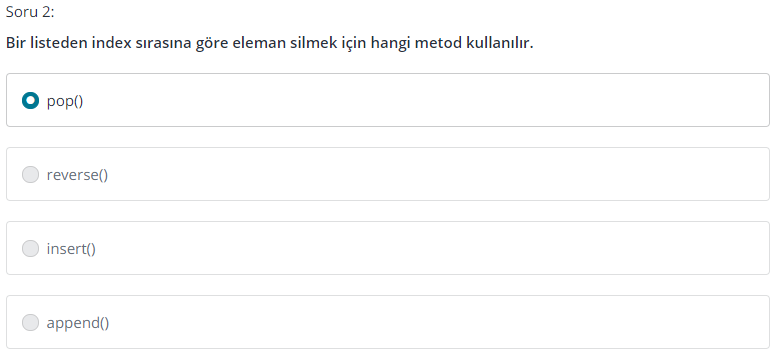
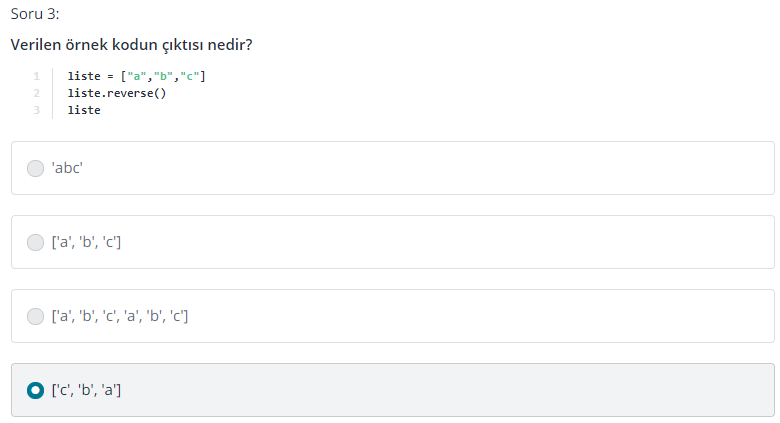
 

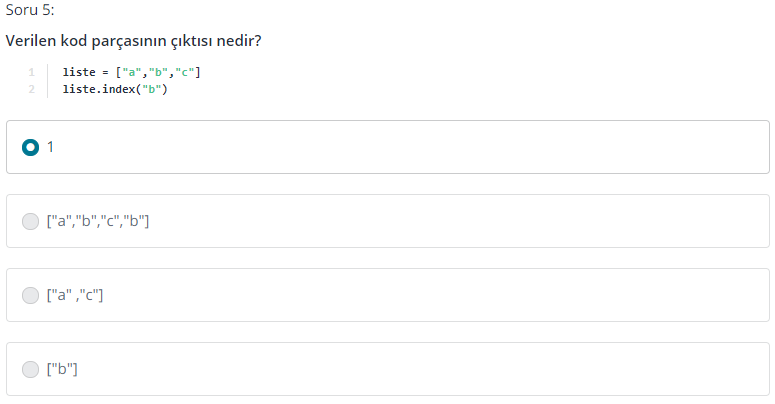


### Python Programlama Alıştırmalar – 6

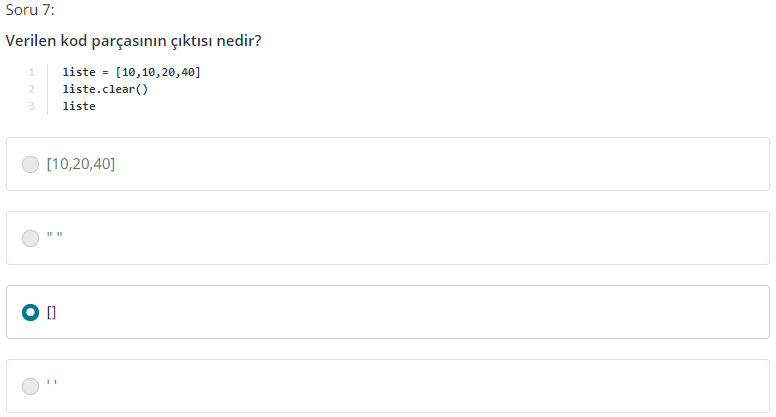
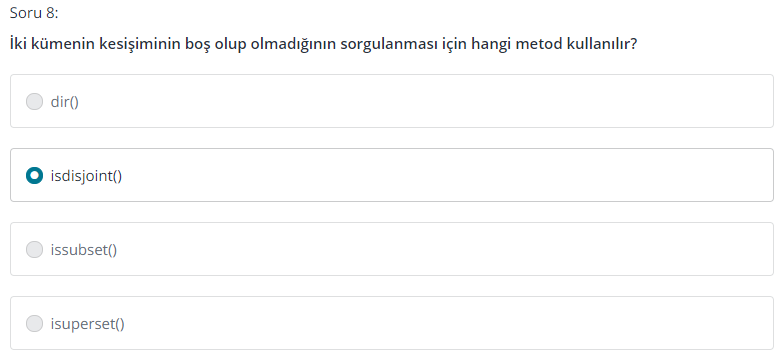
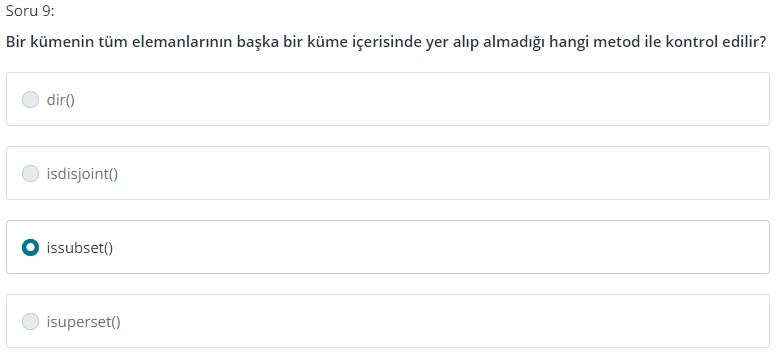


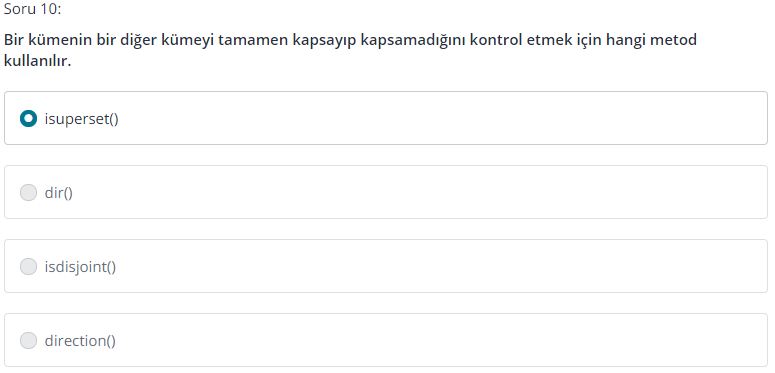
 







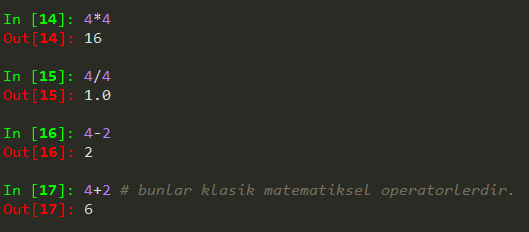


## Fonksiyonlar

### Fonksiyon Nedir?

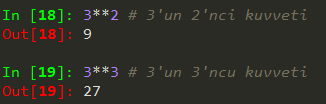
Belirli amaçları yerine getiren işleçlerdir.

### Matematiksel İşlemler



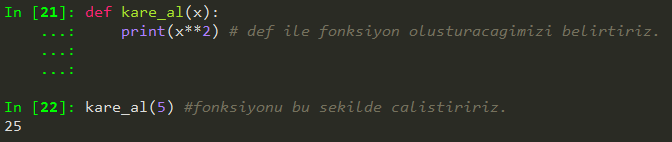
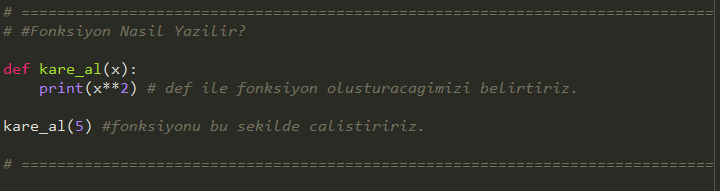
#### Üs Alma

3\*\*2 🡪 32 anlamına gelir.

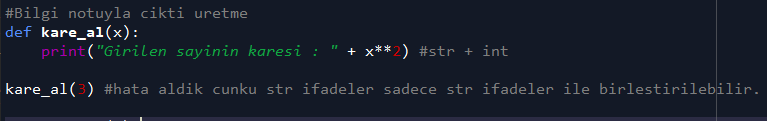


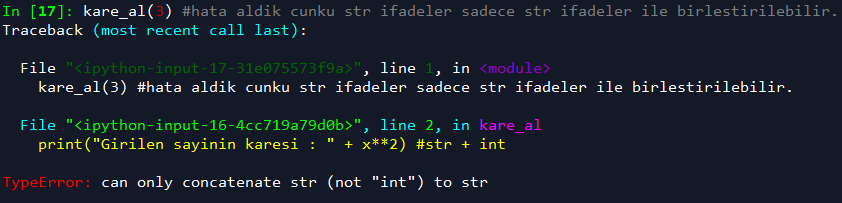
### Fonksiyon Nasıl Yazılır ?

def ile fonksiyon oluşturacağımızı belirtiriz.



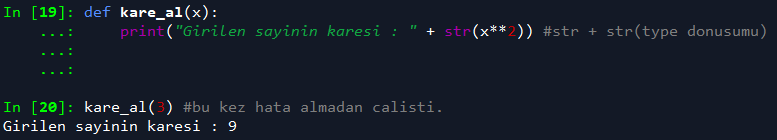
### Bilgi Notuyla Çıktı Üretmek

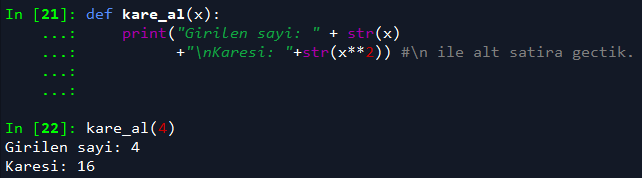


Bu fonksiyonu çalıştırınca aldığımız hata : 

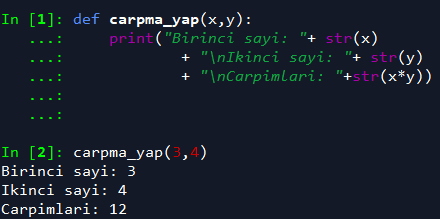
str ifadeler ile sadece str ifadeler birleştirilebilir!

type dönüşümü yapmalıyız.:



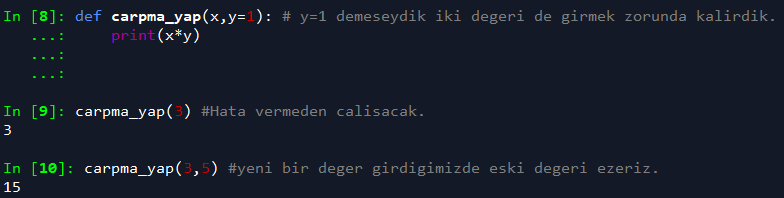
Başka bir örnek: 

### İki Argümanlı Fonksiyon Tanımlamak



### Ön Tanımlı Argümanlar

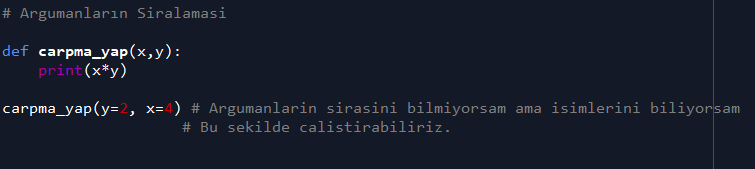
Print() fonksiyonundan hatırlayacağımız gibi sep() ve end() gibi argümanlardır.



y=1 yazarak ön tanımlı bir argüman oluşturmuş olduk.

#### Argümanların Sıralaması

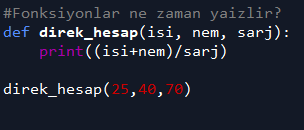
Argümanların sırasını bilmediğimiz fakat isimlerini bildiğimiz zaman aşağıdaki şekilde çalıştırabiliriz.



### 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?

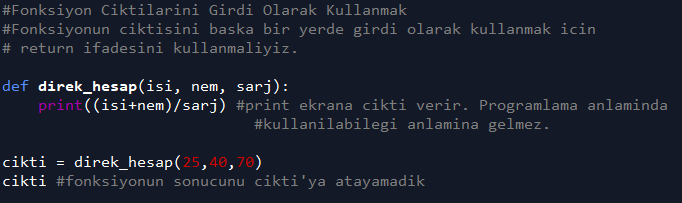
Hayır, fonksiyonu bir kez yazıp her lambada o fonksiyonu kullanacağız. 

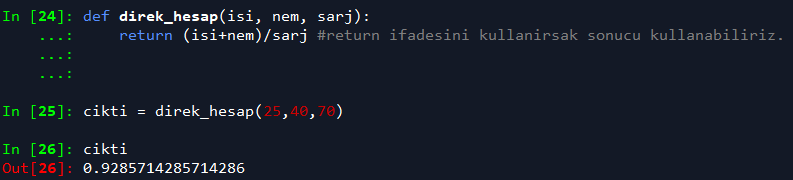
### Fonksiyon Çıktılarını Girdi Olarak Kullanmak

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.





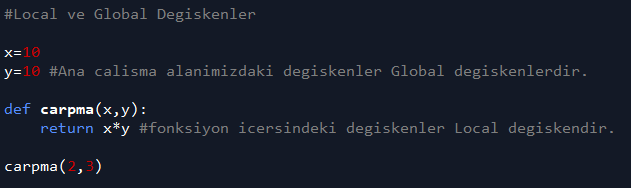
Fonksiyon return ifadesine gelince durur:



### Local ve Global Değişkenler

Ana çalışma alanımızdaki değişkenler Global değişkenlerdir.

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

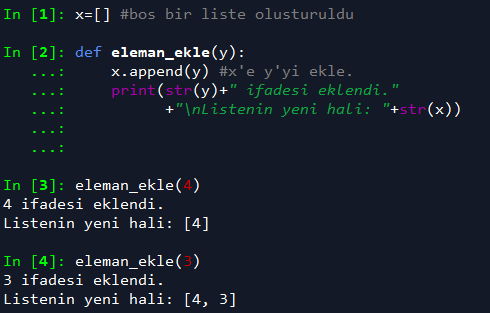


### Local Etki Alanından Global Etki Alanını Değiştirme

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

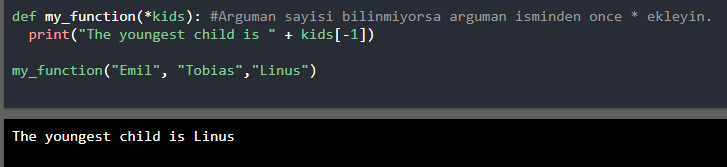
Python öncelikle local etki alanındaki değişkenleri tarar, arar ve bulmaya çalışır.

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



## Karar-Kontrol Yapıları (Koşullar)

### Koşul Nedir?

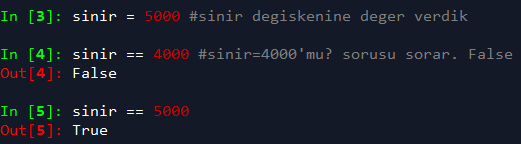
Örneğin günlük hayatta da kullandığımız gibi;

* Yağmur yağarsa şemsiye al
* Kar yağarsa zincir tak

gibi bazı olaylar gerçekleştiğinde bazı olayların gerçekleşmesi gerektiğini programlama diline ifade etmenin yollarıdır.

### True – False Sorgulamaları (Boolean)

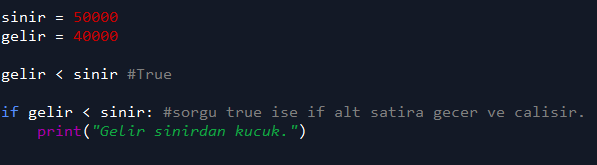
Doğru mu? sorusu sorar. == ile kullanırız.



### if – else – elif

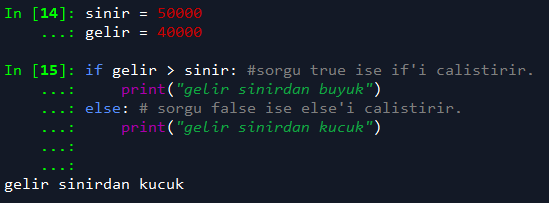
if eğer anlamındaki koşuldur.

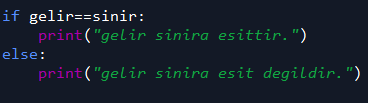
Eğer yazdığımız sorgu true ise alt satıra geçer ve çalışır.



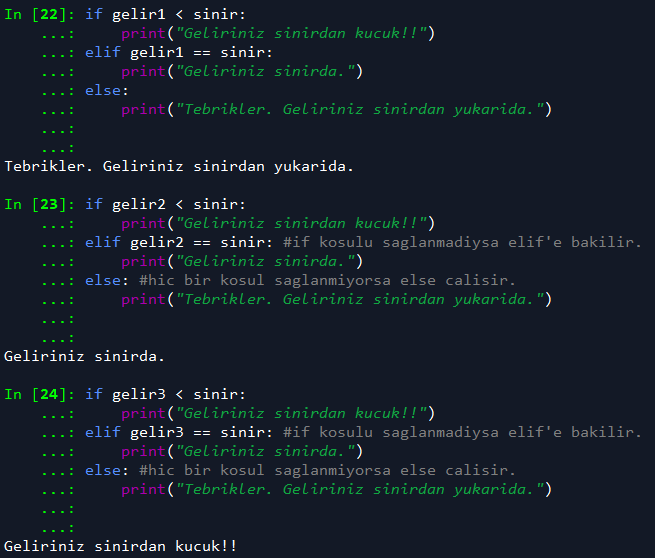
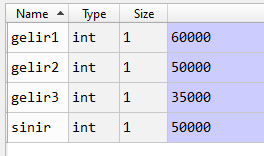
if = eğer true ise if çalışır.

else= değilse else çalışır.



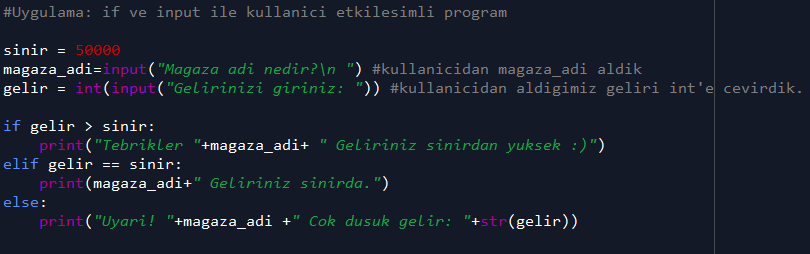


elif= if koşulu sağlanmazsa elif’e bakılır. elif koşulu da sağlanmazsa else çalışır.

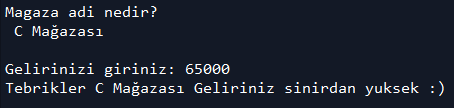
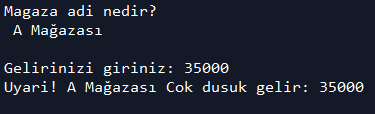


### **Uygulama:** if ve input ile kullanıcı etkileşimli program

Kullanıcıdan mağaza adı ve gelir bilgilerini alalım. Sınır değeri ile gelir değerini karşılaştıralım. Düşük, eşit, yüksek seviyelerine göre 3 farklı sonuç üretelim.



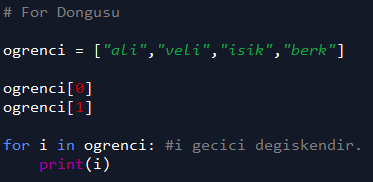
Program çıktıları:



## Döngüler

### For Döngüsü

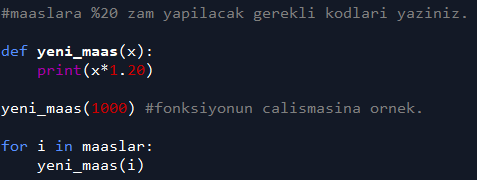
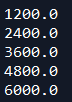
Örneğin bir liste içerisindeki elemanlara işlem yapmak istediğimizde o elemanlara tek tek gitme işlemini gerçekleştiren yapılara döngüler denir.

à

### Döngü ve Fonksiyonların Birlikte Kullanımı

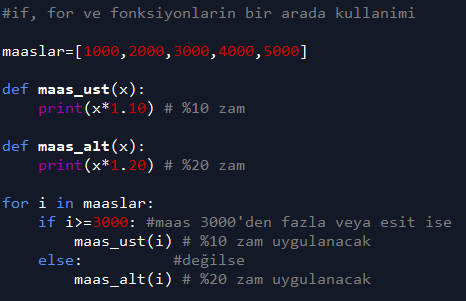


Maaşlara %20 zam yapılacak. Gerekli kodlar nelerdir?

à

### **Uygulama:** if, for ve fonksiyonların birlikte kullanımı

Az önceki uygulamadaki maaş listesi kullanılarak; maaşı 3000 tl’den yüksek olanlara %10 zam, maaşı 3000 tl’den az olanlara ise %20 zam yapılacak.

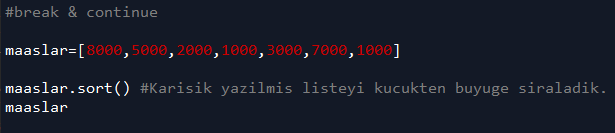
à

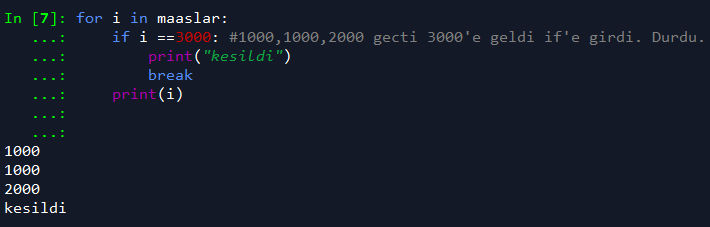
### break & continue

Döngüler içersinde belirli bir şartı sağlayan ifadeler yakalandığında (if döngüsü ile yakalıyorduk.) döngü bitirilmek istenebilir. Ya da bu şartı sağlayan eleman görmezden gelinmek istenebilir.

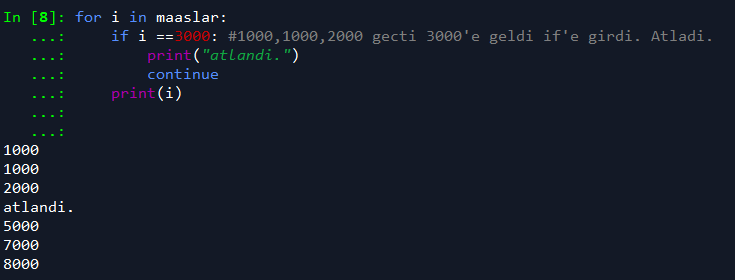
Bu gibi durumlarda break ve continue ifadeleri kullanılır.

Örneğin; maaşı 3000 tl’ye kadar olanlarla ilgilendiğimizi düşünelim.



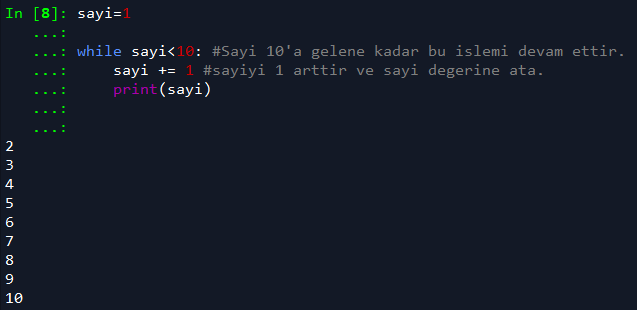


Örneğin; 3000’i atlayıp devam etsin.

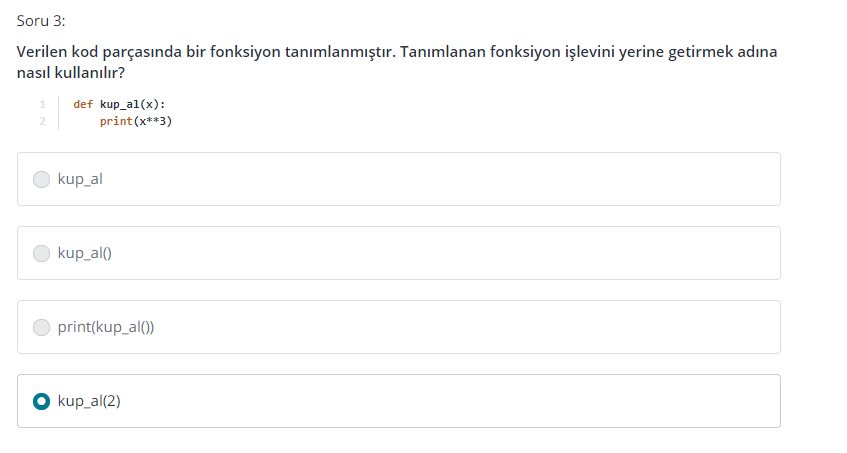
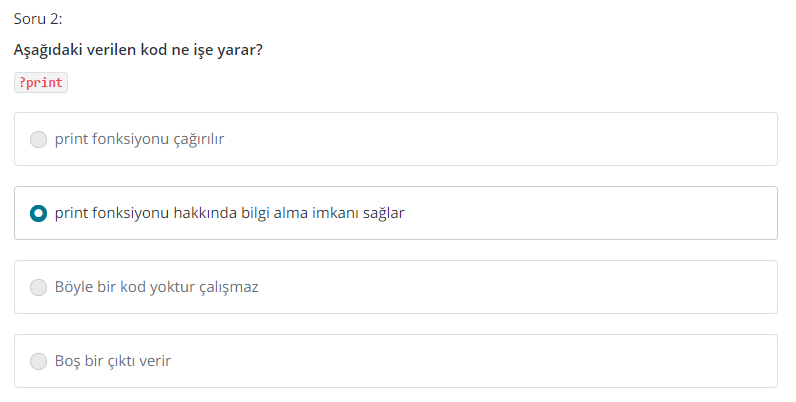
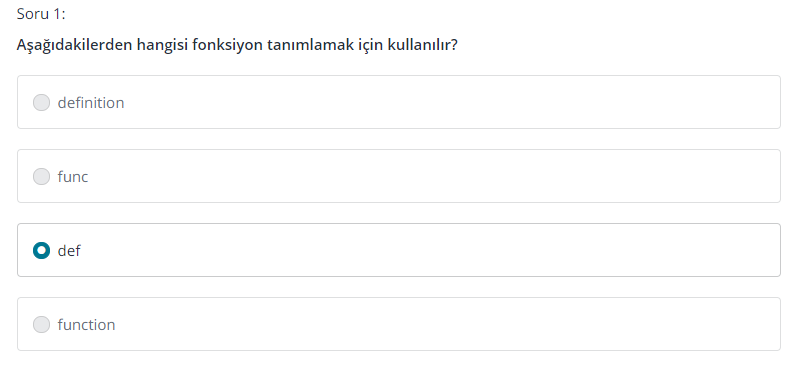


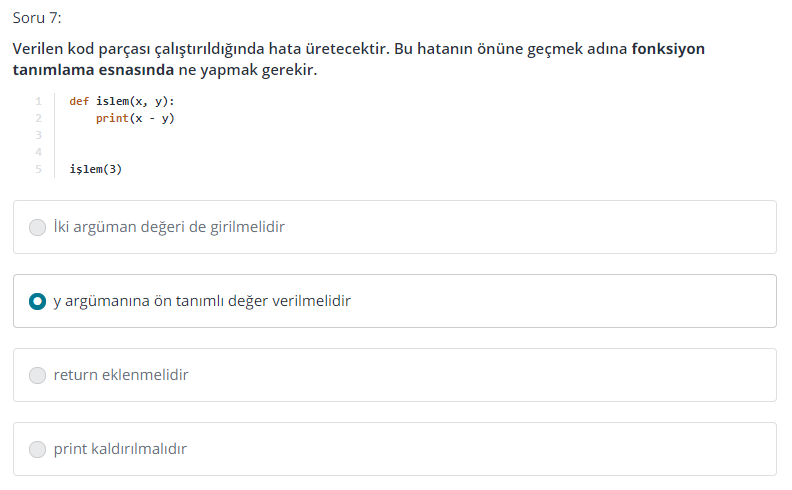
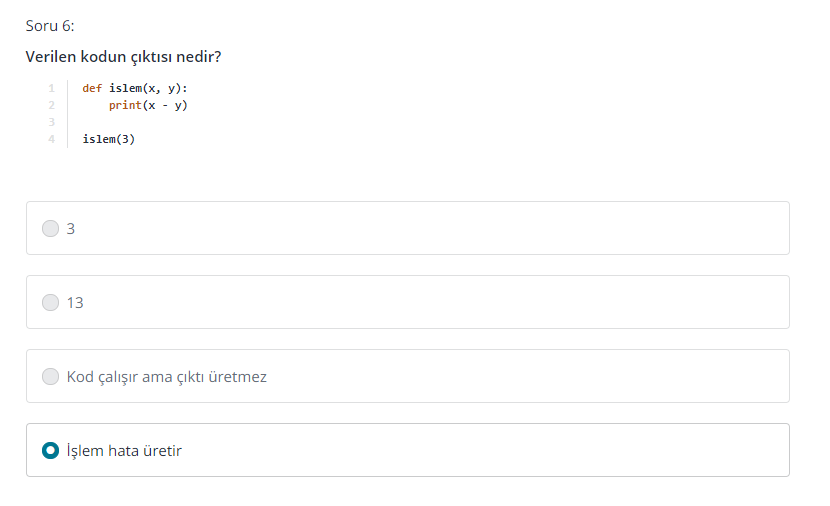
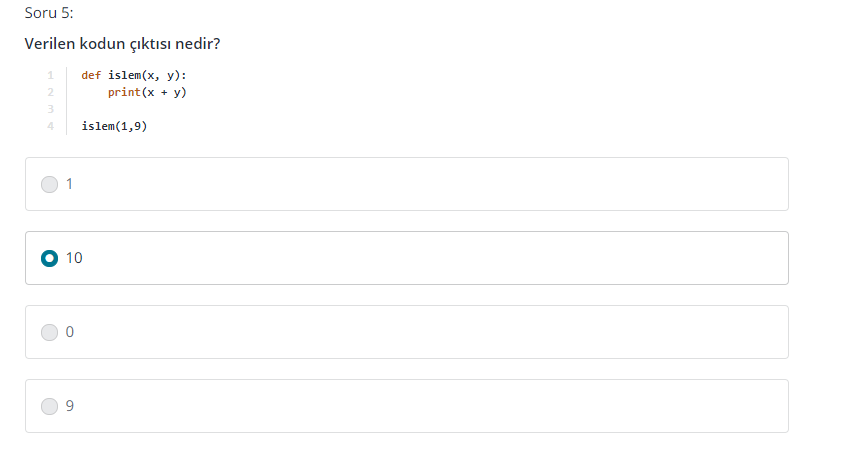
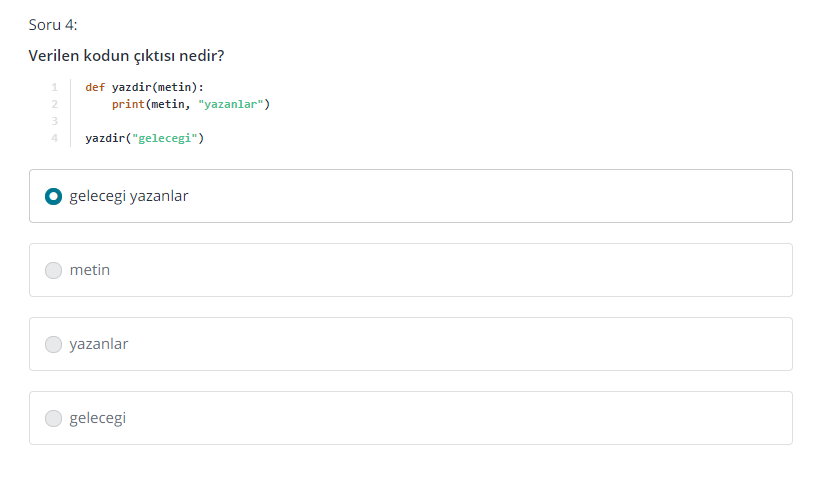
### while

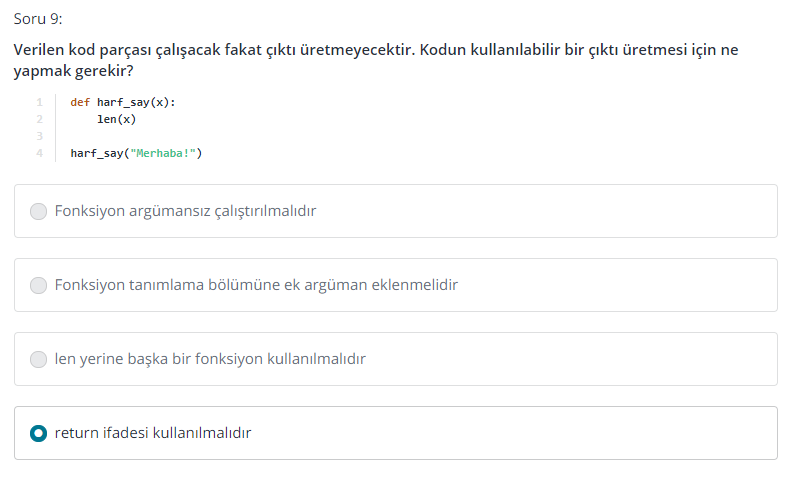
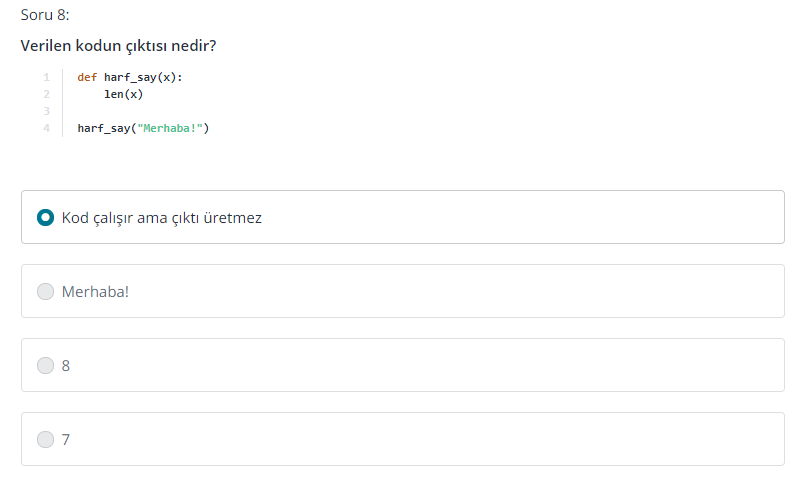
Şart sağlandığı sürece devam eden bir döngüdür.

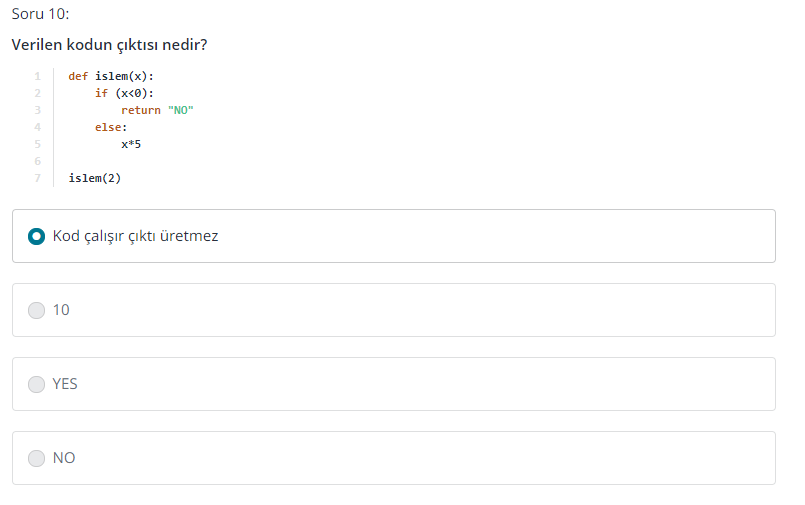


### Python Programlama Alıştırmalar - 7

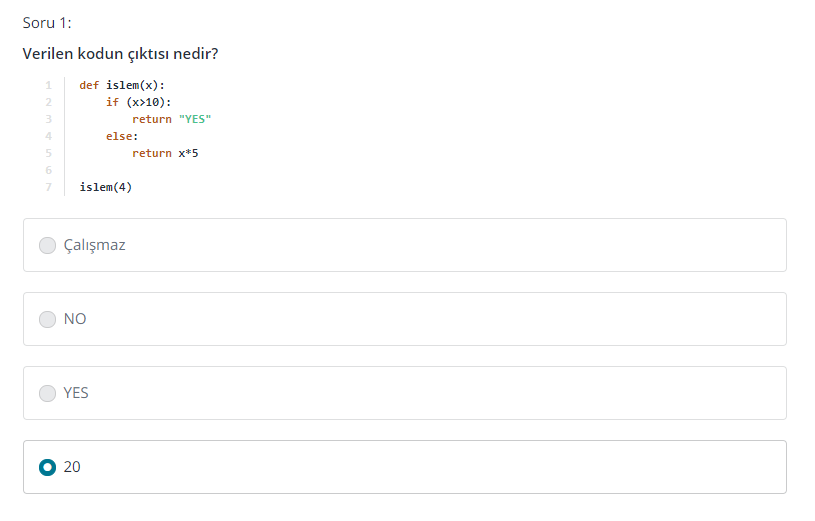


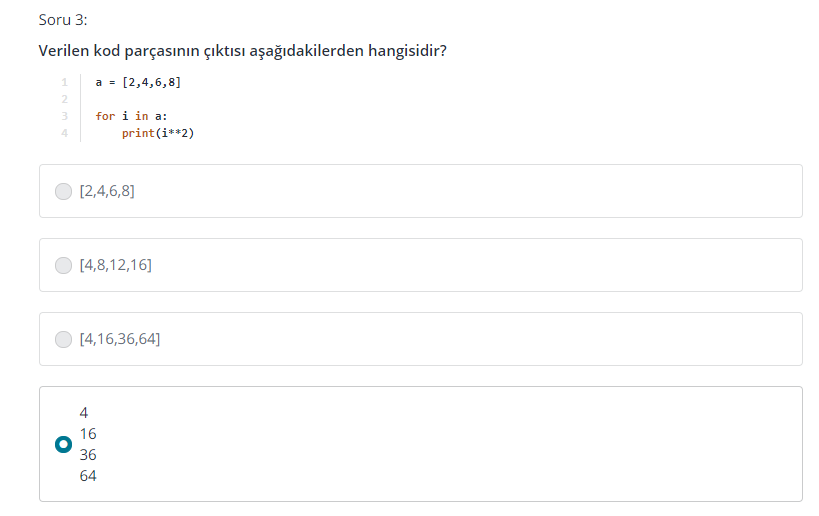
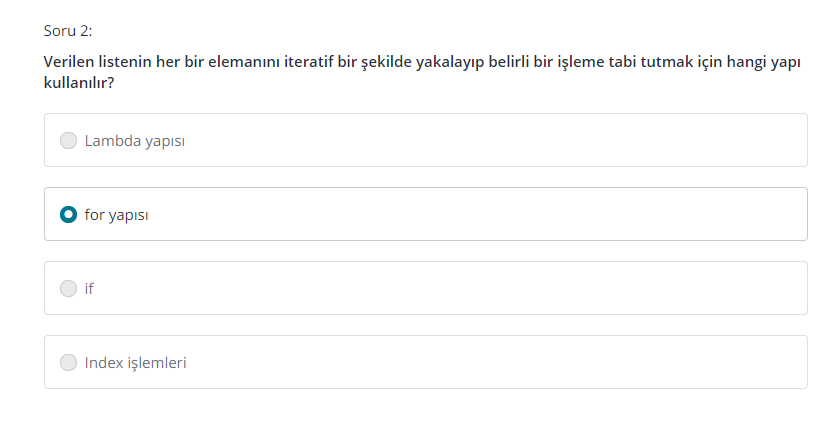






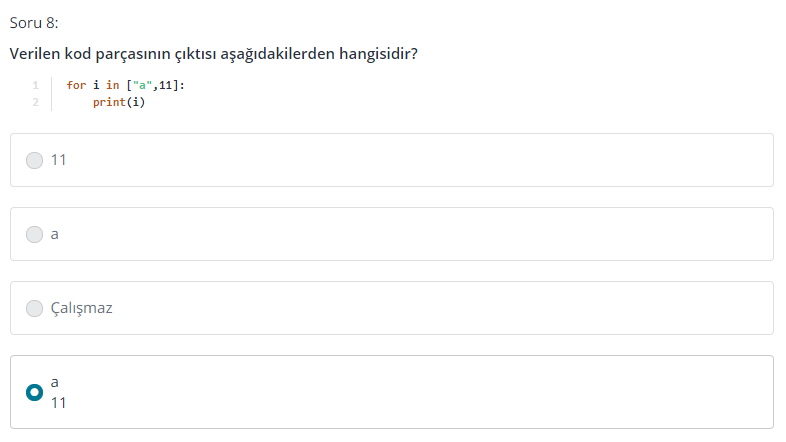
### Python Programlama Alıştırmalar - 8

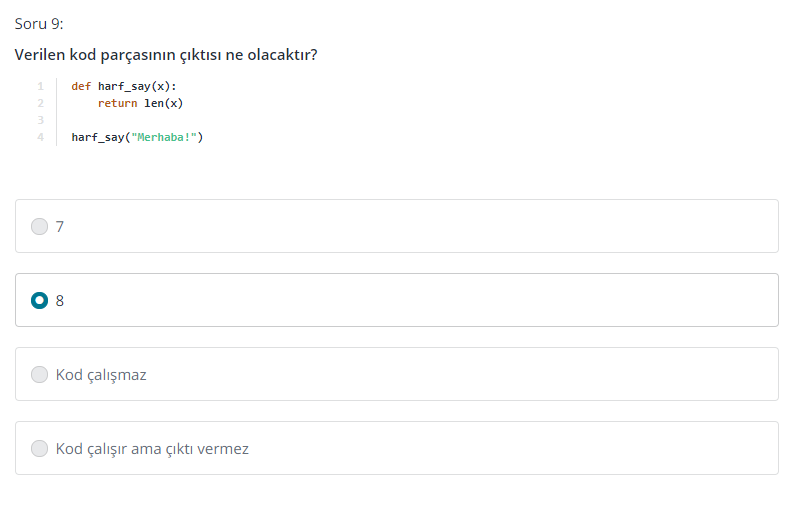


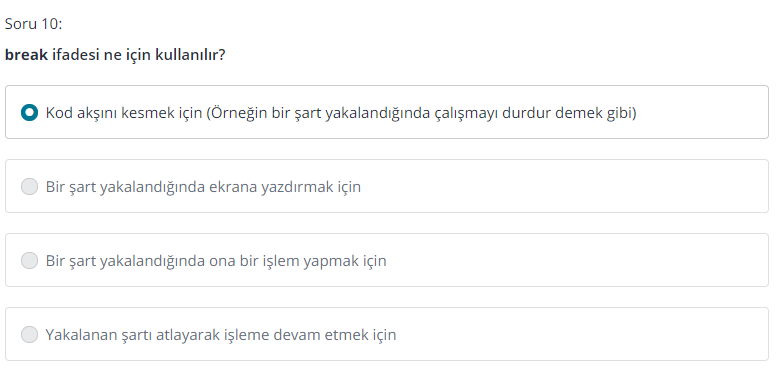




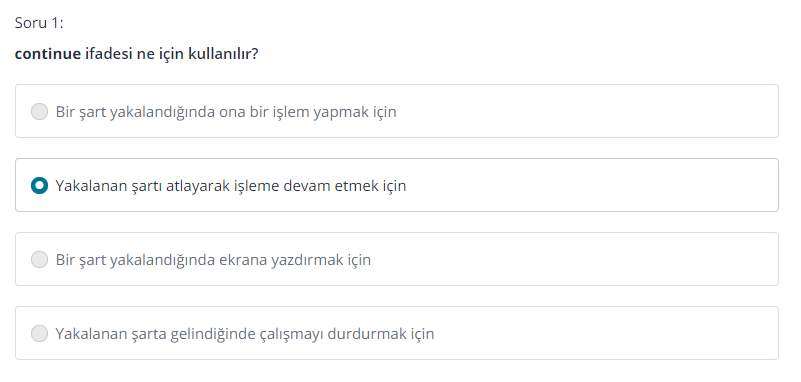


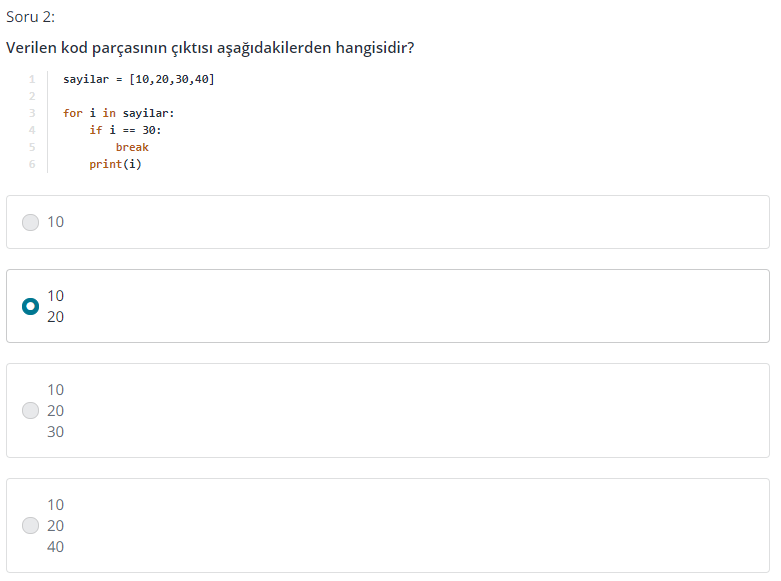
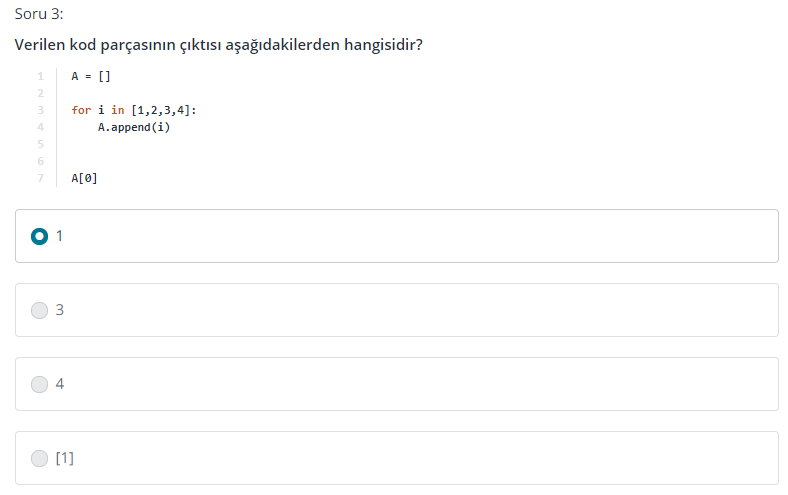


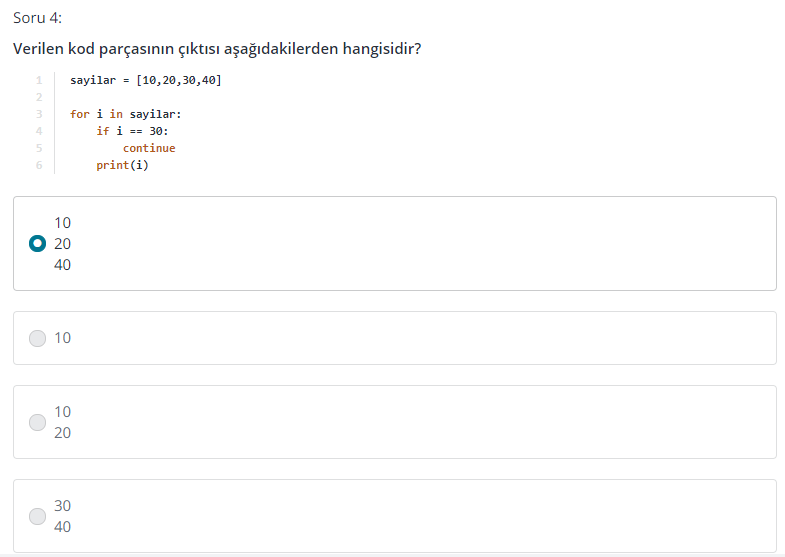
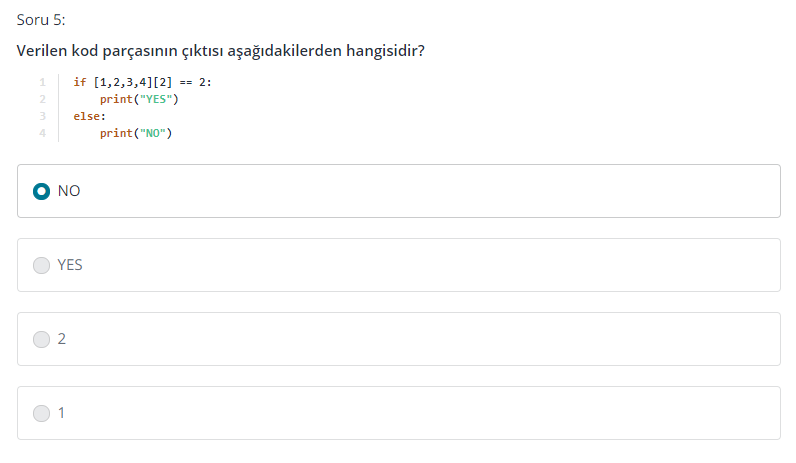


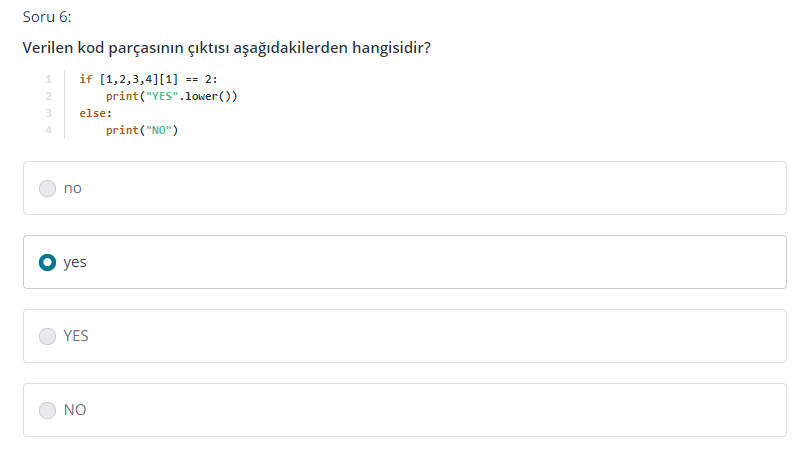
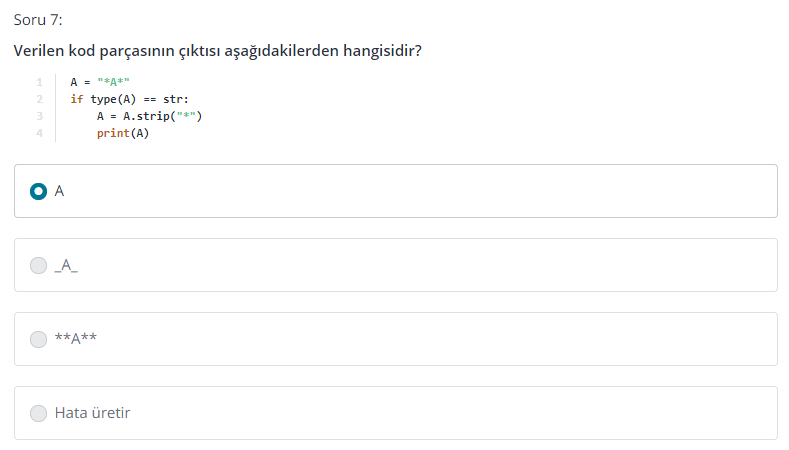


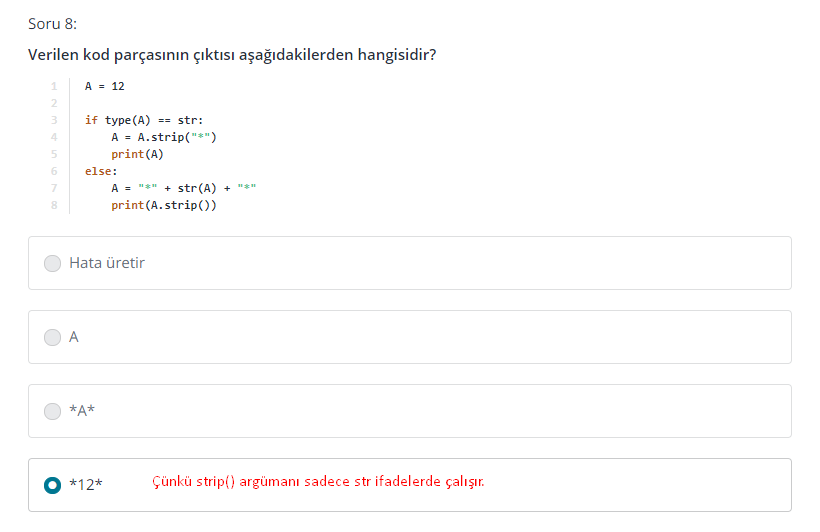
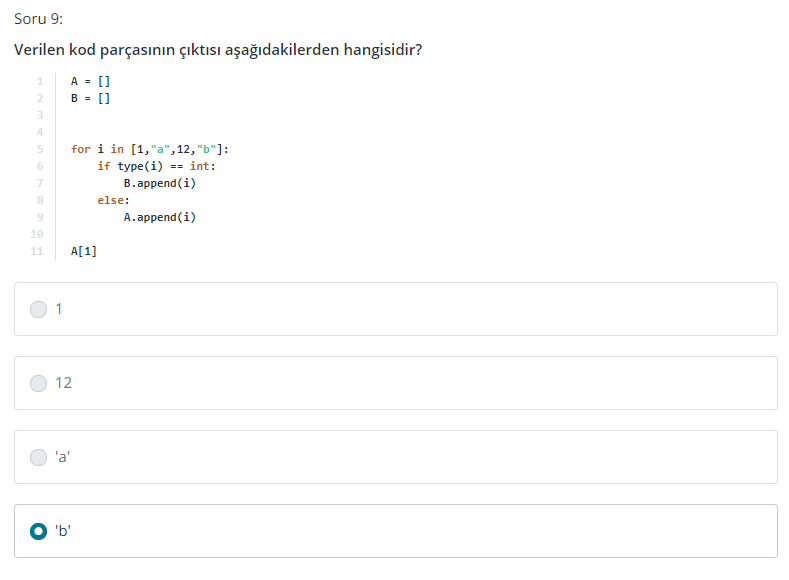
### Python Programlama Alıştırmalar – 9

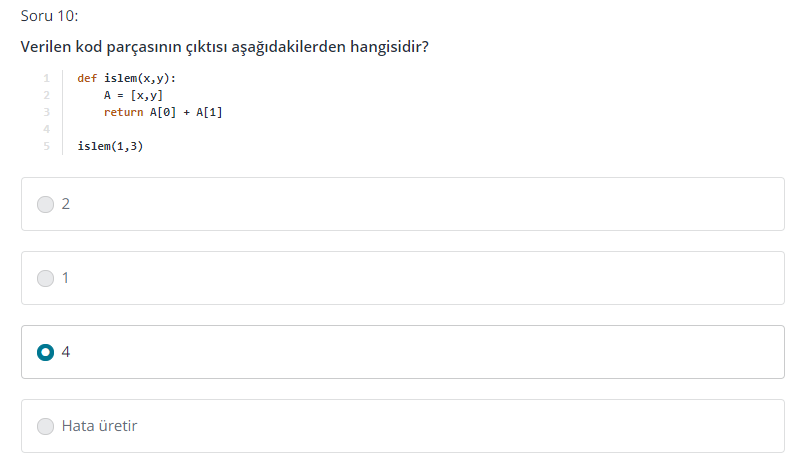


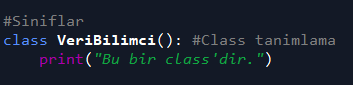


## Object Oriented Programming

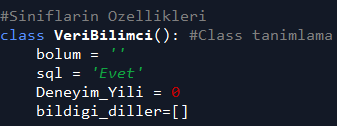
### Class’lara Giriş ve Class Tanımlamak

#### Class Nedir?

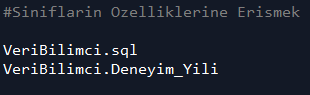
Sınıflar; benzer özellikler, ortak amaçlar taşıyan, içerisinde metod ve değişkenler olan yapılardır.



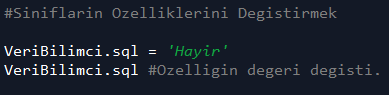
### Class Özellikleri



#### Class Özelliklerine Erişmek

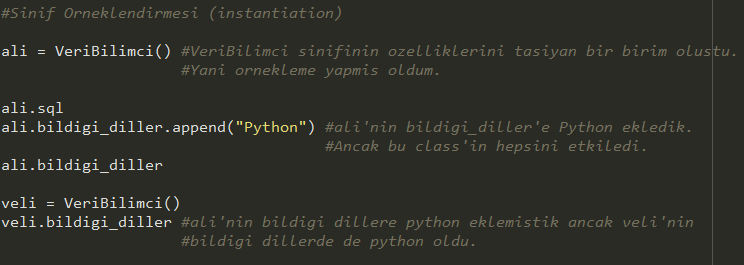


#### Class Özelliklerini Değiştirmek



### Class Örneklendirmesi (instantiniation)

Sınıfın özelliklerini barındıran alt kümeler oluşturma işlemine sınıf örneklendirmesi denir.



### Örnek Özellikleri

Şuan yapmış olduğumuz işlem her bir örneğin kendi içinde değişebilen özelliklerden oluşabildiği bilgisini vermek. Yani her bir ayrı örneklendirme için özellik tutma bilgisini sağlıyor.

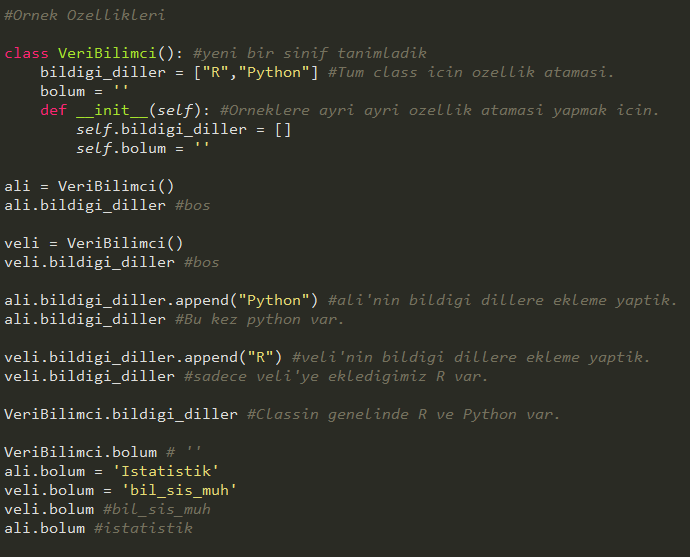
Sınıflar için tanımlanan özellikler örnekler içn değişebilir bir formata getirilmedikçe bir örnekte yapılan değişiklik tüm örneklere etki ediyor.

def \_\_init\_\_(self):

self.bildigi\_diller = ‘’

self.bolum = ‘’ 🡪 fonksiyonunu kullanacağız. Buradaki self temsilci anlamındadır. Her bir örneklemi temsil eder (ali, veli gibi).

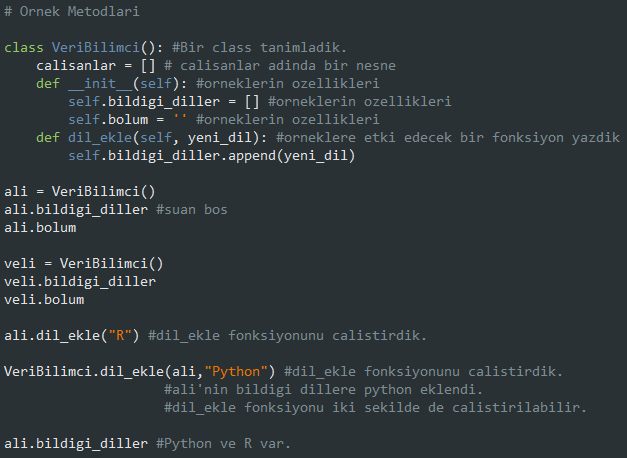
Genelde sınıf özelliklerinin isimleri ve örnek niteliklerinin isimleri aynı olmamalıdır. Örneğimizde anlaşılır olması açısından aynı kullandık.



### Örnek Metodları

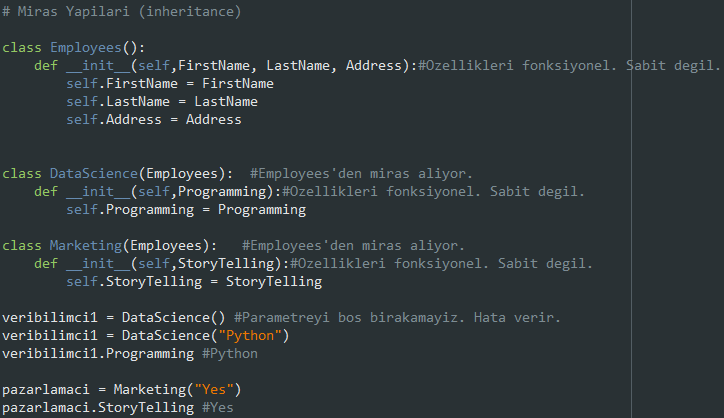
Mesela her bir veri bilimci için bir yeni öğrenilen dili o veri bilimcinin bildiği dillere ekleme işlemi yapsın.

Örnekler üzerinde çalışan fonksiyonlar yazmak istiyoruz.



### Miras Yapıları (inheritance)

Başka yerde başka bir class tanımlarken, tanımlayacak olduğumuz bu class daha önceden tanımlamış olduğumuz başka bir class’ın özelliklerini barındırıyorsa ve biz bunları kullanmak istiyorsak eski class’ın özelliklerini miras olarak kullanabiliyoruz.



## Functional Programming

### Fonksiyonel Programlamaya Giriş

Python dili ile bir program yazmak istediğimizde bunu OOP(Nesneye Dayalı Programlama) özellikleri ile de yazabiliriz FP(Fonksiyonel Programlama) özellikleri ile de yazabiliriz.

Fonksiyonlar dilin baştacıdır. (Birinci sınıf nesnelerdir.)

Yan etkisiz fonksiyonlar. (stateless(durumsuz), girdi-çıktı 🡪Ancak bir girdi verdiğimde çıktı üretir. Ve bu çıktı hep aynı olur. Dışarıdan etkilenemez.)

Yüksek seviye fonksiyonlar.

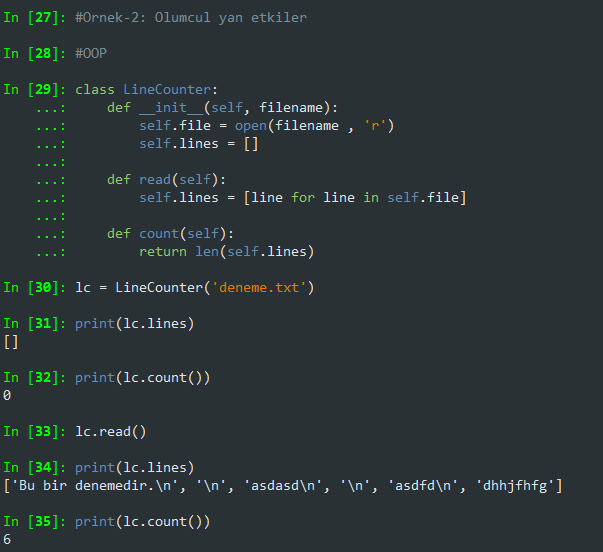
### Yan Etkisiz Fonksiyonlar (Pure Functions)

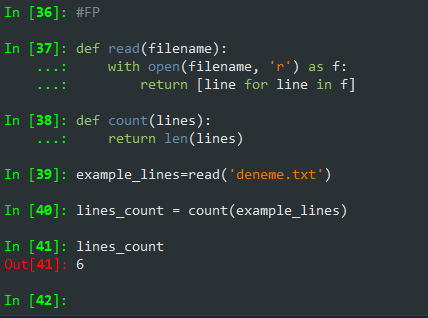
Fonksiyonun bir şekilde dışarı bağımlı olduğu durumlara yan etkili yani impure(saf olmayan) fonksiyon denir.

#### Örnek-1: Bağımsızlık

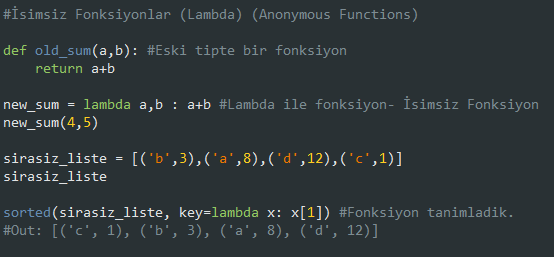


#### Örnek-2: Ölümcül Yan Etkiler





### İsimsiz Fonksiyonlar (Lambda) (Anonymous Functions)



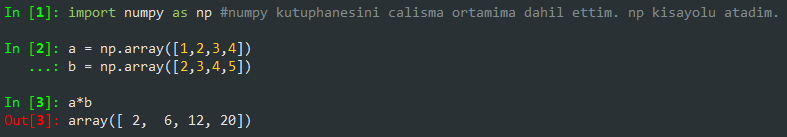
Sorted bir fonksiyondu. Birinci argümanı bir nesneydi, listeydi. Elemanları da tuple idi. Bu listeye bir fonksiyon uygulamak istiyoruz. x’e bağlı bir fonksiyon, x olarak kendi içine girilen değerin 1. indexli elemanına ulaşsın.

### Vektörel Operasyonlar (Vectorel Operations)

#### OOP ile iki listeyi çarpmak

#### Functionel Programming ile

Fakat söz konusu matematik, istatistik, veri bilimi, makine öğrenmesi gibi konular olduğunda asla bu tip döngülere vs. girmiyoruz. Vektörel operasyonlara giriyoruz.



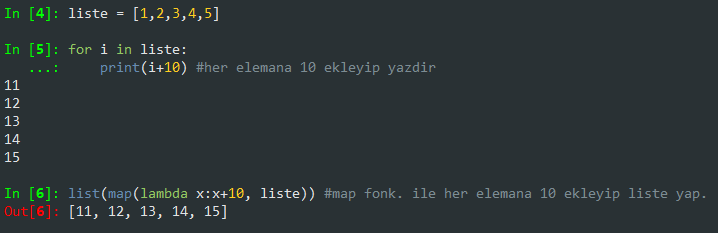
Fonksiyonel Programlama ile daha az çaba ile aynı sonuca ulaşmış olduk.

### Map & Filter & Reduce

Fonksiyona argüman olarak fonksiyon yazmamıza izin veren fonksiyonlara First Class fonksiyon denir.

#### Map

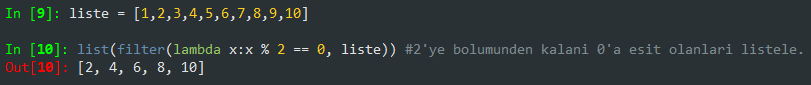
Verilen bir nesne üzerinde tanımlanacak bir fonksiyonu çalıştırma imkanı verir.(lambda yani isimsiz fonksiyonu)



#### Filter

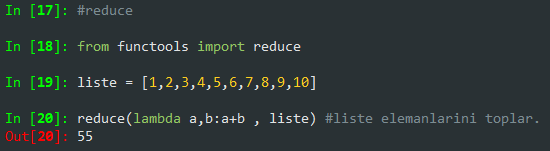
filter fonksiyonu iteratif bir nesne alır bu nesne üzerinden başka bir iteratif nesne oluşturulur. Ve iteratif nesne içerisinde aradığı şartın sağlandığı tüm elemanlar listelenir.

Çift sayıları bulan fonksiyonu yazalım.



#### Reduce

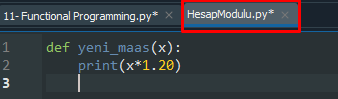
Az önceki filter fonksiyonu bize aradığımız değerleri bulup getirdi. Yani değerler ile ilgili bir işlem yapmadı. Reduce fonksiyonu yine map ve filter’a benzerdir fakat indirgeme işlemi yapar.



### Modül Oluşturma

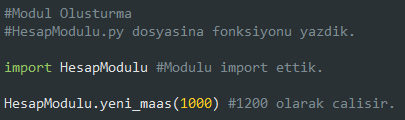
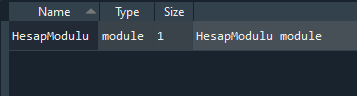
Bazen modül, bazen kütüphane, bazen de paket dendiğini görebiliriz, bunların üçü de doğrudur. Modüller belirli amaçları yerine getirmek için bir arada bulunan fonksiyonlar topluluğudur.

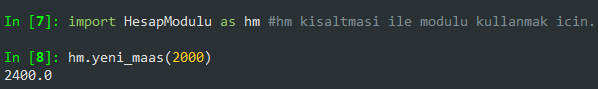
Maaşlarla ilgili işlemler gerçekleştiren birkaç tane fonksiyonumuz olduğunu düşünelim ve bunu paketleyip bir modül haline getirelim.

Yeni bir .py dosyası açalım ve ismi HesapModulu.py olsun. 

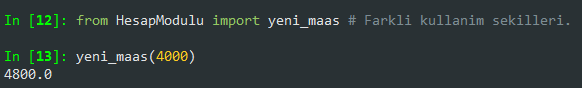
Modülün içine fonksiyonu yazıp kaydettik. Modülümüz kullanıma hazır.

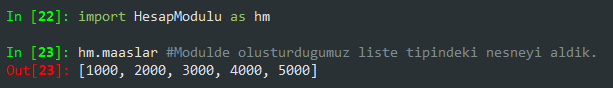
Başka bir .py dosyasından modüle erişmek:



Daha da kısa kullanımı için:



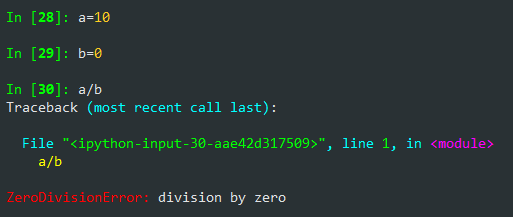


### Hatalar/İstisnalar (exception)

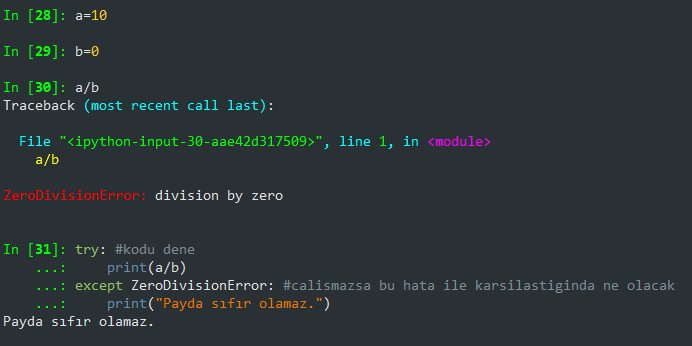
1-Programcı hataları: Bunlar basit hatalardır. Syntax hatası gibi.

2-Program hataları / bug: Bunlar kritik hatalardır çünkü program çalışmaya devam eder ancak çıktılar probremlidir. Çıktıların hatalı olmasının tespiti bile bazen zorlayıcı olabilir.

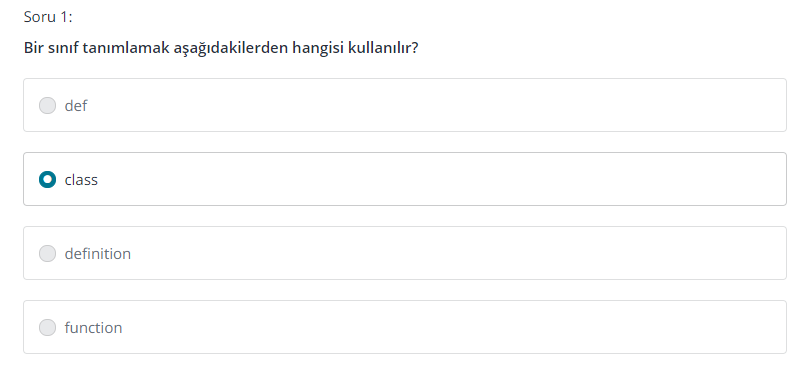
3-İstisnalar (exceptions): Programda bildiğimiz bazı hatalardır fakat bu hatalar gerçekleştiğinde programı durdurma, çalışmaya devam et demenin yoludur. Bunu try except yapısı ile sağlarız.

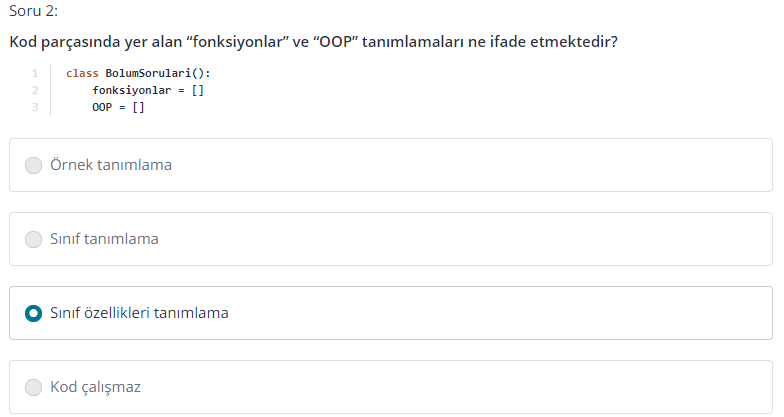


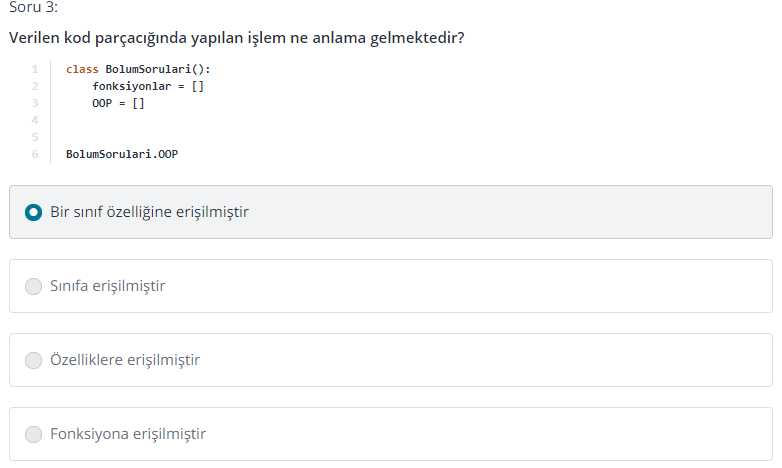
Gördüğümüz üzere ZeroDivisionError hatası ile karşılaştık. 0’a bölünemez.

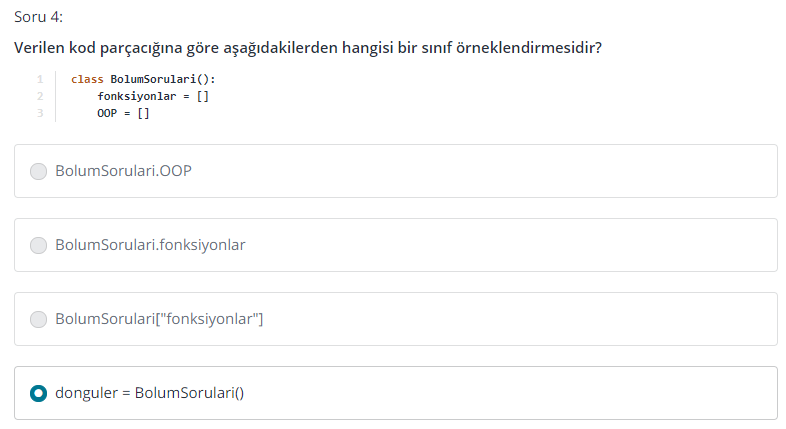
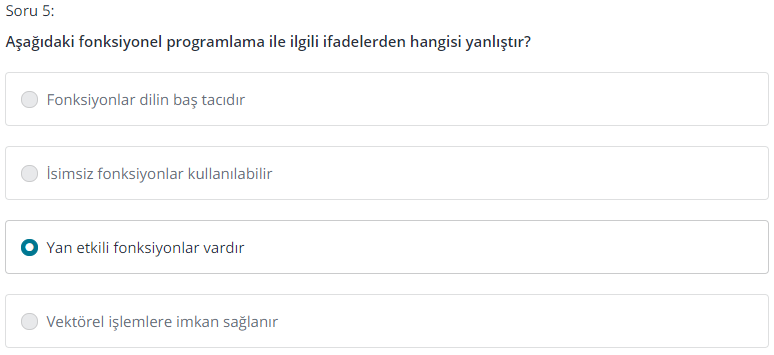


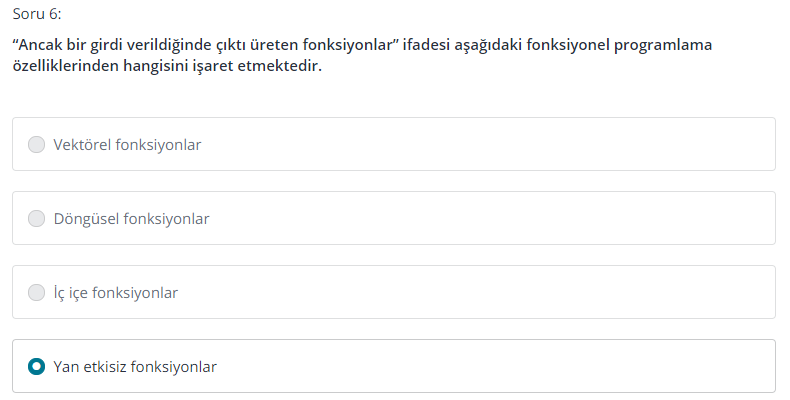
### Python Programlama Alıştırmalar – 10

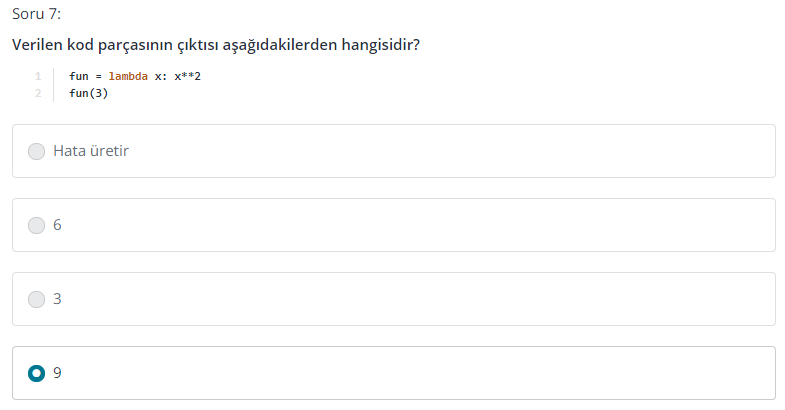
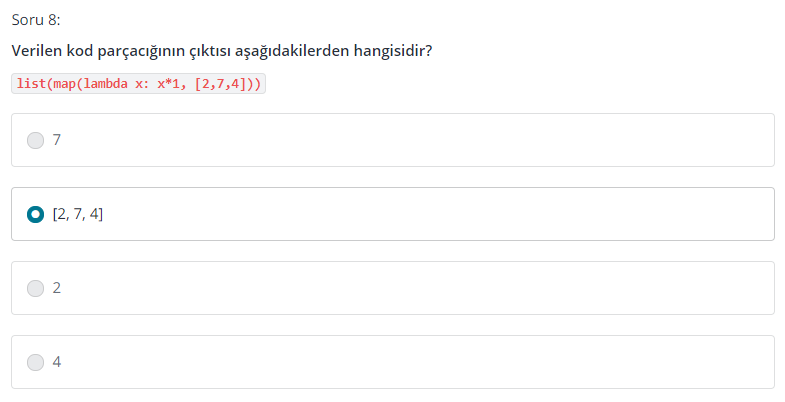
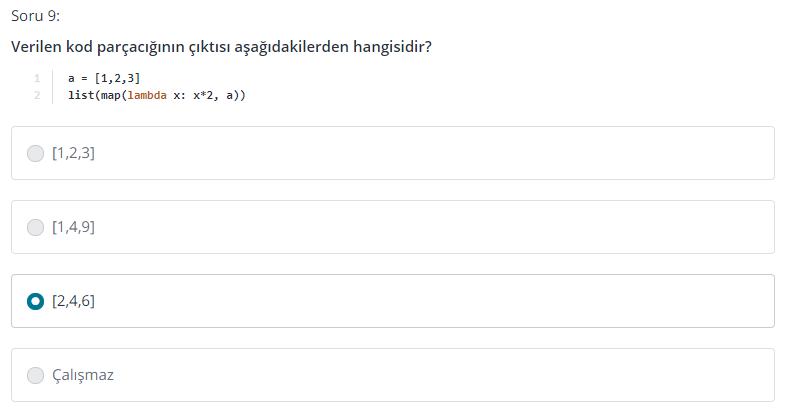


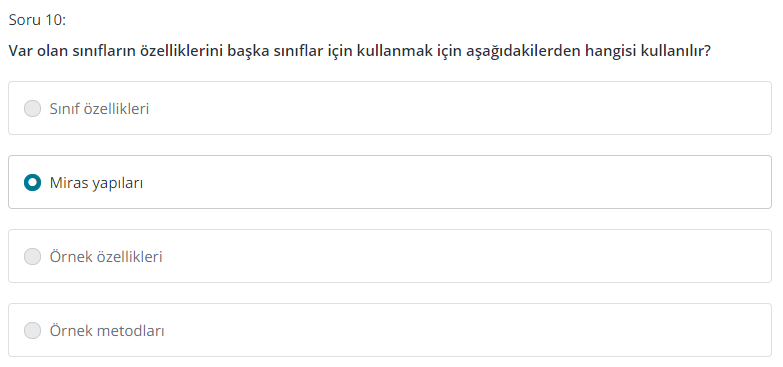




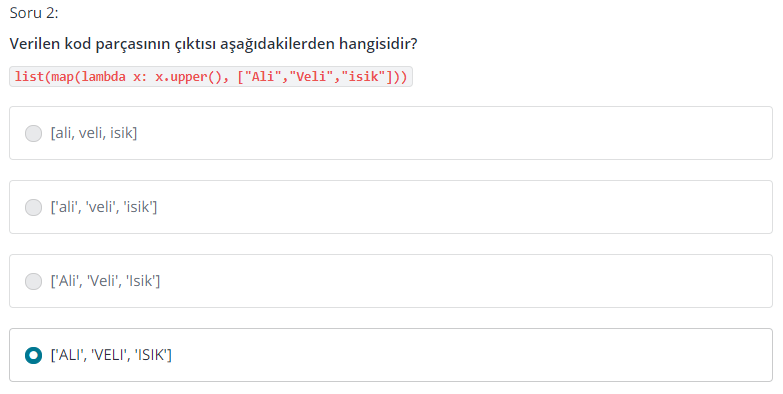
 

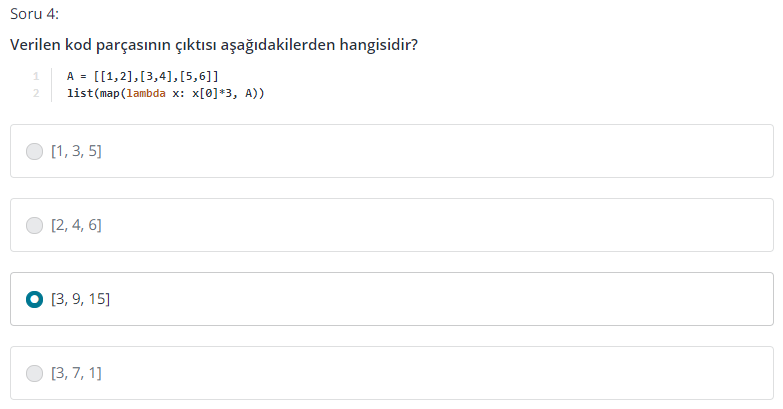


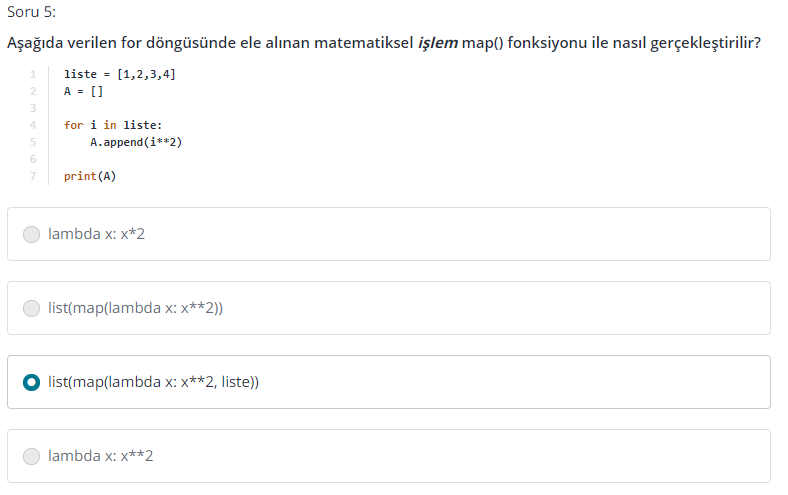
  

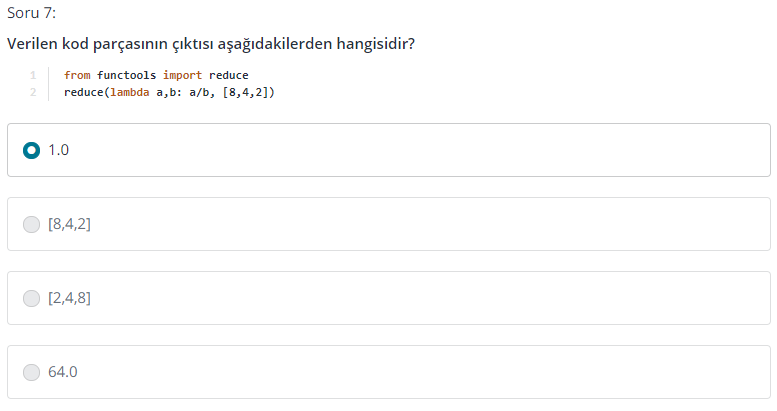


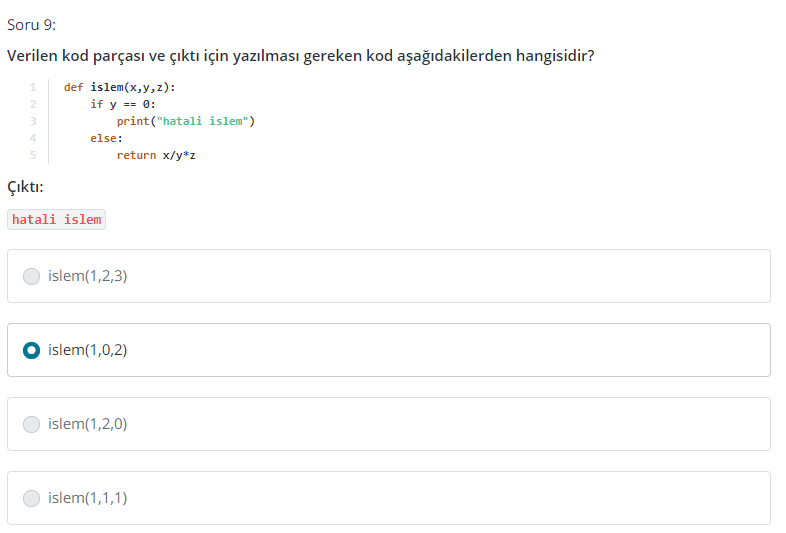
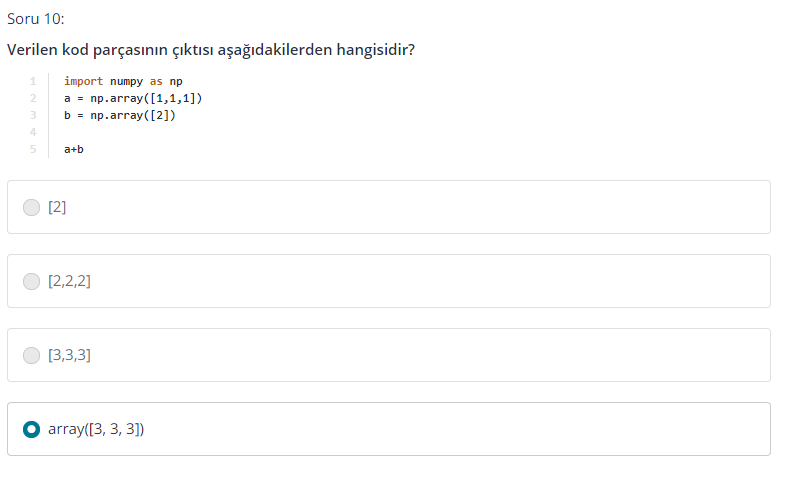
### Python Programlama Alıştırmalar – 11

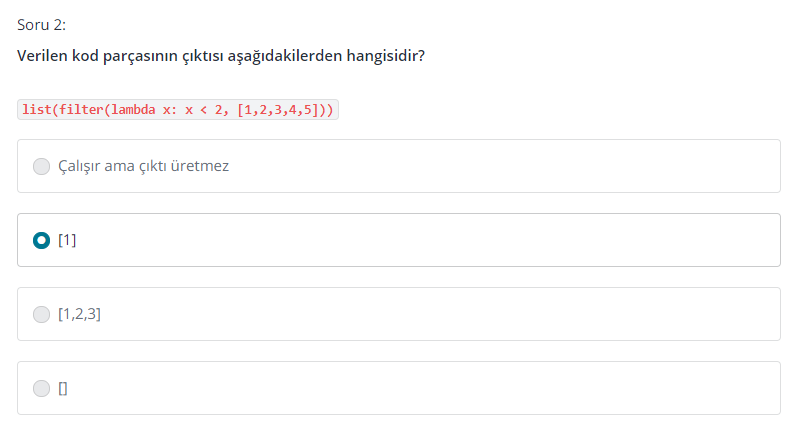
 

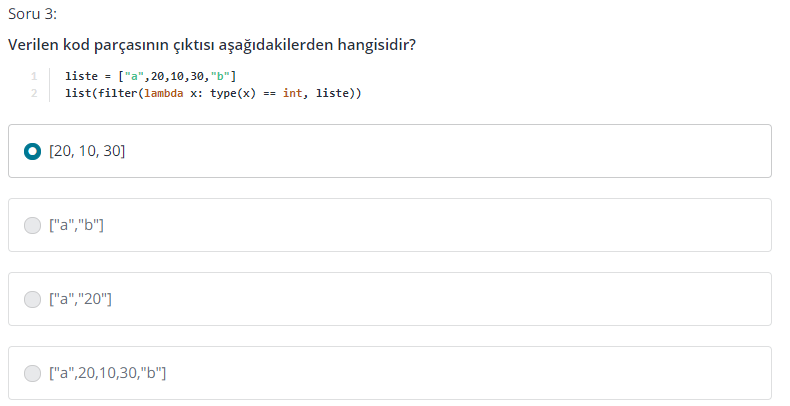
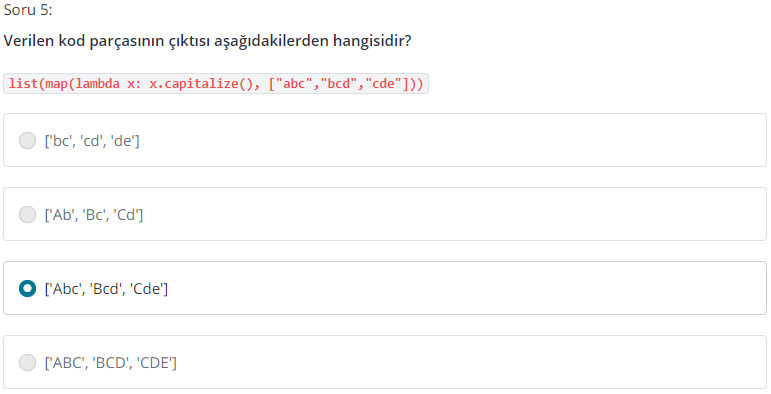
 

### Python Programlama Alıştırmalar – 12

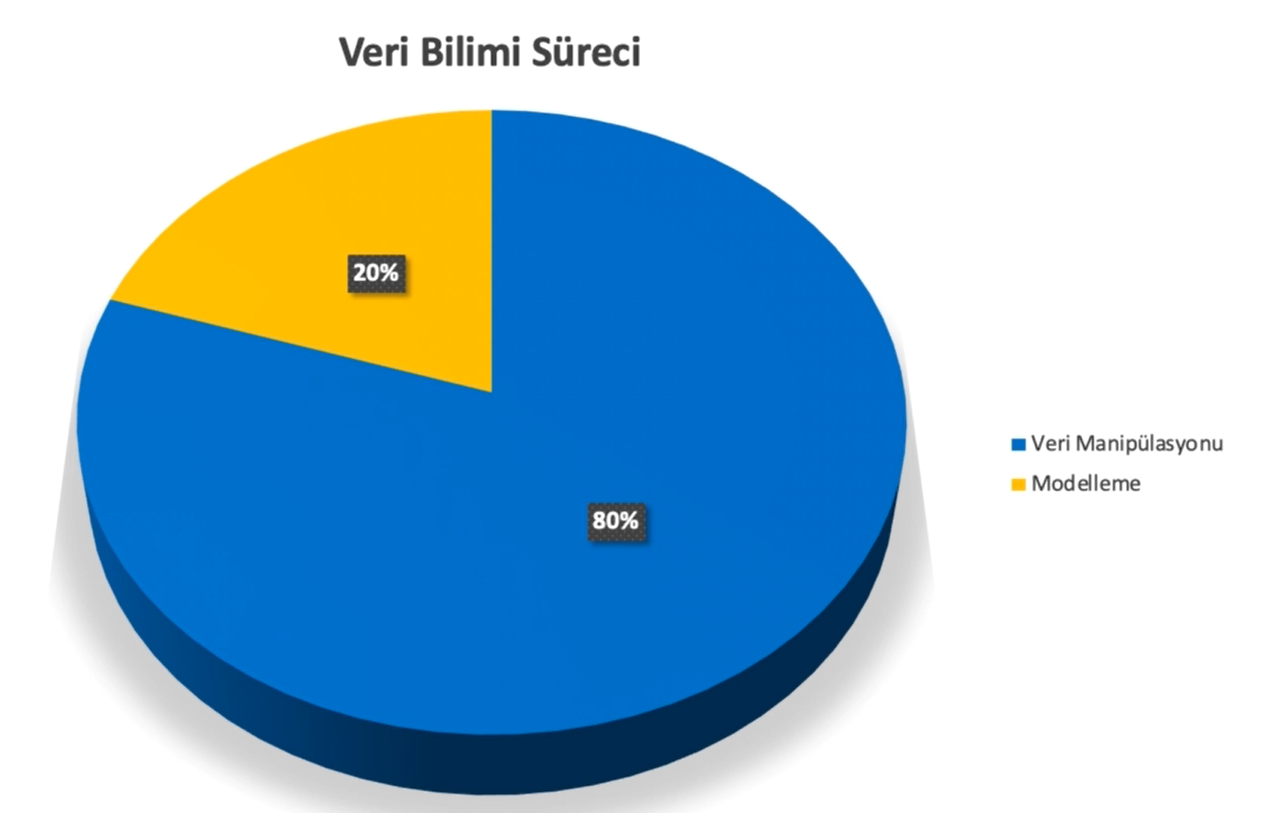
 



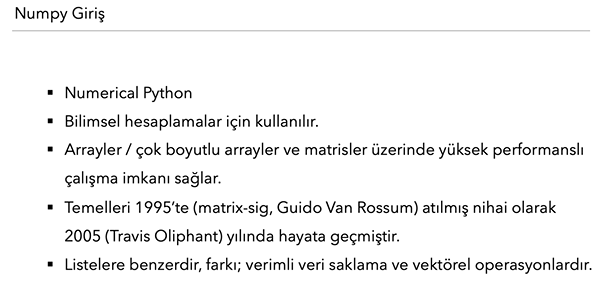
# Python ile Veri Manipülasyonu: NumPy & Pandas



# NumPy (Numerical Python)

## NumPy Giriş

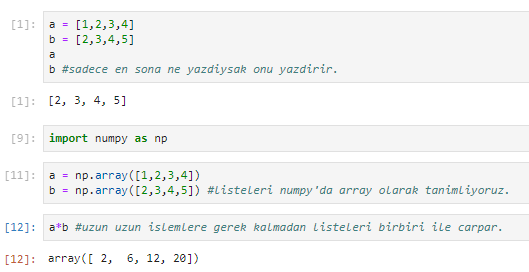
NumPy Python’ın bazı numerik işlemlerde yetersiz kaldığı noktalarda ihtiyaçlarımızı gidermek için ortaya çıkmış bir kütüphanedir/modüldür.



## Neden NumPy?

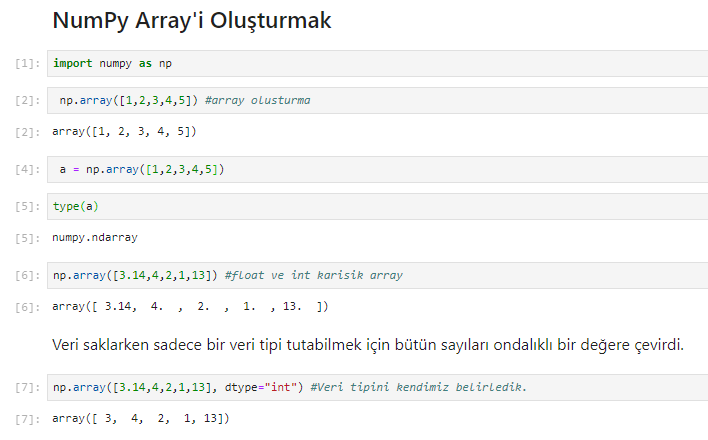
Daha üst seviyeden, daha az çabayla daha büyük işler yapma olanağı sağladığından dolayı kullanıyor olacağız.

Neden NumPy? sorusunun ikinci ve önemli yanı yer tutma maliyetlerini numpy çok azaltmaktadır. Örneğin listede 4 elemanın her biri için type=int bilgisi 4 kez tutulur. Numpy array’inde ise sadece bir kez array’in kendisi için tutulur.

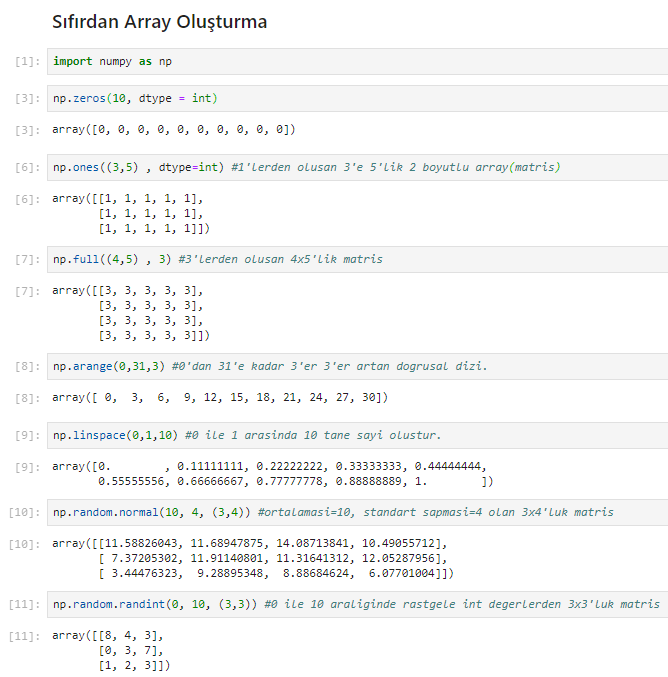


## NumPy Array’i Oluşturmak

NumPy Array tıpkı sözlükler gibi listeler gibi bir veri tipidir.



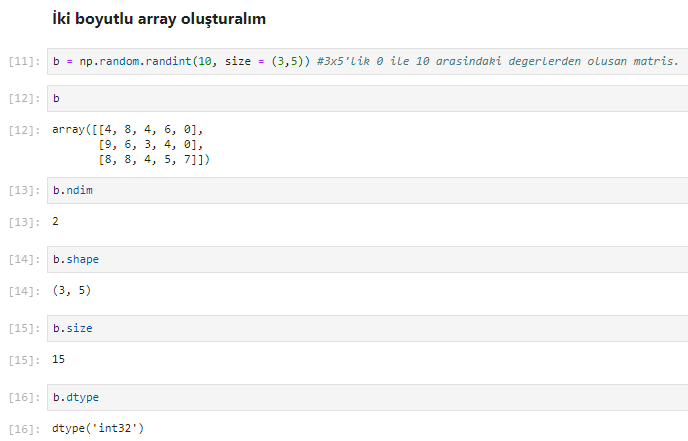
### zeros, ones, full, random, arange, linspace, random.normal, random.randint



## NumPy Array Özellikleri



### Matris Oluşturma



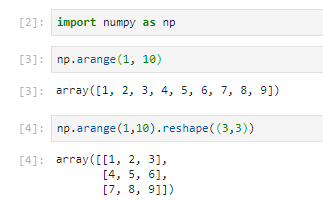
## Reshaping (Array’i Yeniden Şekillendirme)

Elimizde var olan bir array'i yeniden şekillendirme işlemi yapacağız. Örneğin elimizde bir array olsun ve bunu yeniden boyutlandıralım.

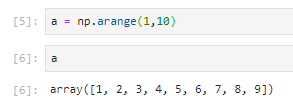
Fonskiyonlarımızın ürettiği çıktılar tek bir boyutta, tek bir array formunda gerçekleşebiliyor.

Bunları bazen tek boyuttan 2 boyuta ya da 2 boyuttan tek boyuta indirgeme işlemi gerekebiliyor.

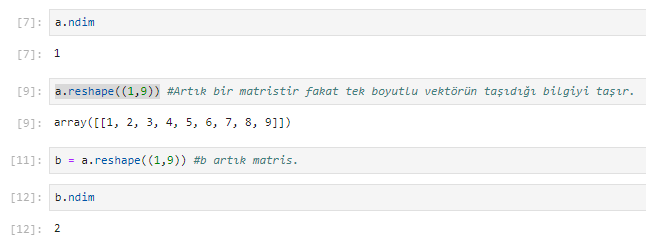
Bu ihtiyaçlarla Reshape fonksiyonu ile başa çıkmış oluyoruz.



**Not:** Tek boyutlu array: Vektör, 2 boyutlu array: Matris.

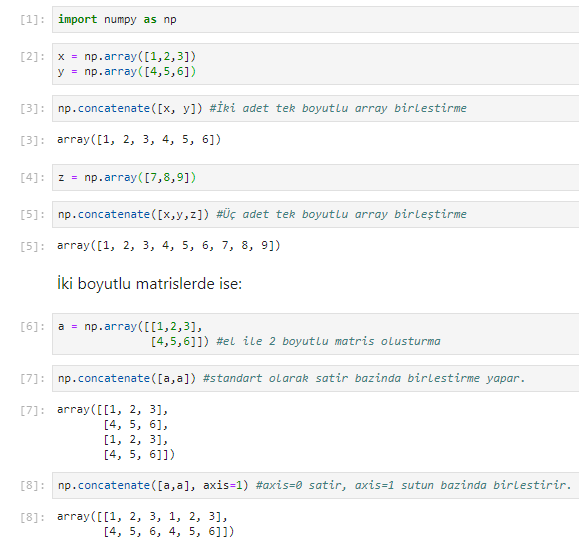


Elimizdeki tek boyutlu array'i 2 boyutlu matris'e çevirmek istiyoruz ama tek boyuttaki bilgisi de olduğu şekilde kalsın.



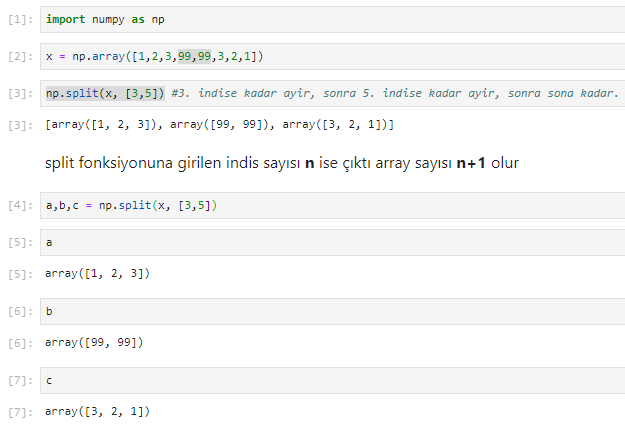
## Concantenation (Array Birleştirme)

concatenate() fonksiyonu ile array’leri birleştirebiliriz.



## Splitting (Array Ayırma)

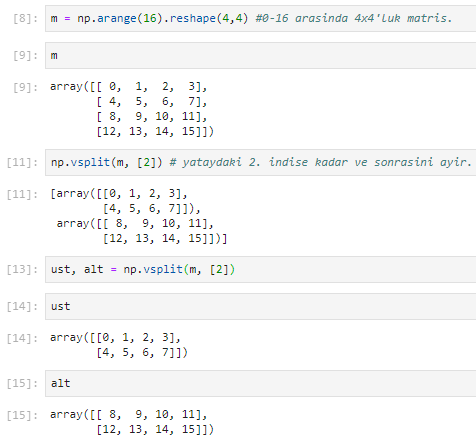
**split()** fonksiyonu kullanılır.

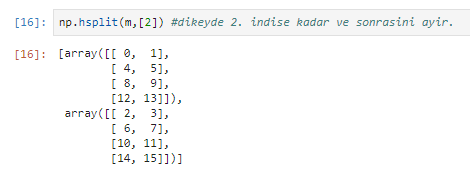


### İki Boyutlu Array Ayırma

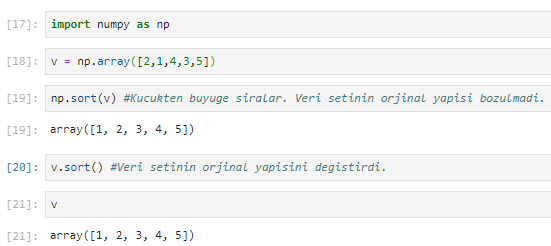
**vsplit() :** dikey olarak ayırmak için kullanılır.

**hsplit() :** yatay olarak ayırmak için kullanılır.

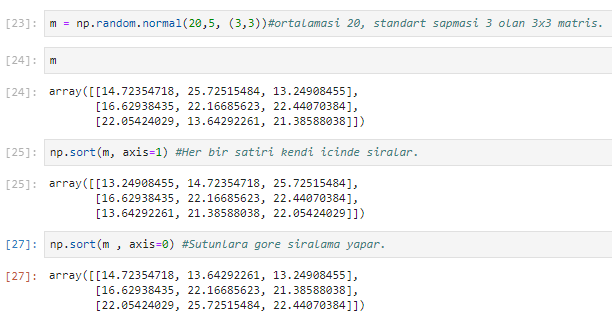




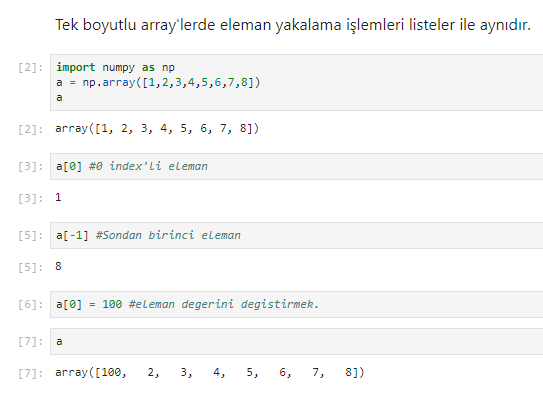
## Sorting (Sıralama)



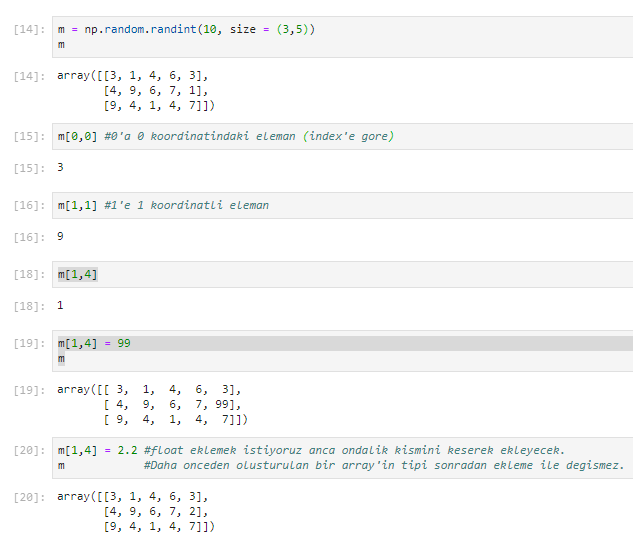
### Matris sıralama



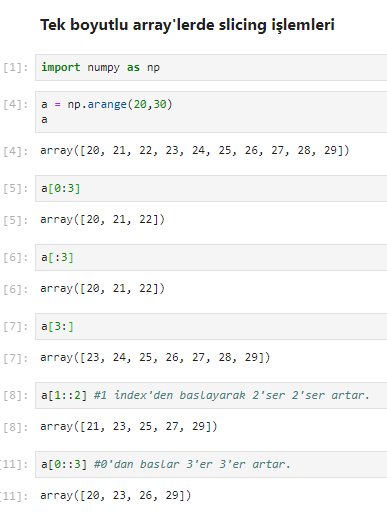
## Index ile Elemana Erişmek



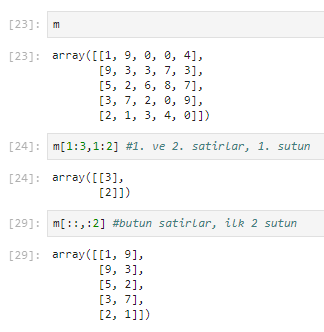
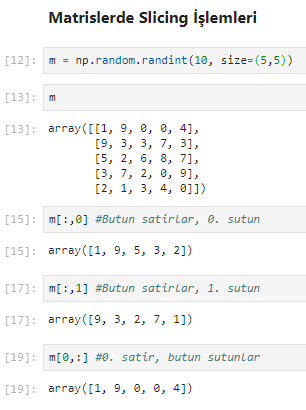
### Matrislerde elemana erişme işlemleri



## Slicing (Array Alt Küme İşlemleri)



### Matrislerde Slicing İşlemleri



## Alt Küme Üzerinde İşlem Yapmak

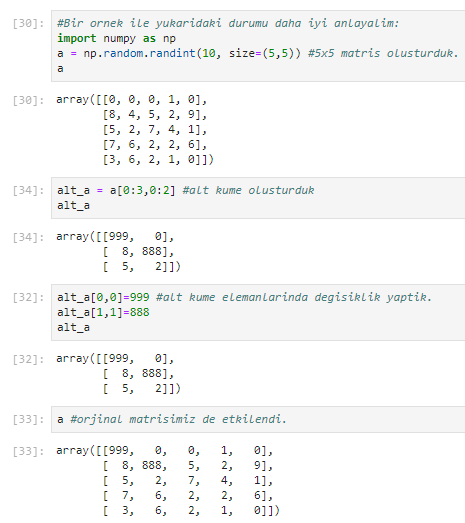
Önceki bölümde array'lerin alt kümelerine eriştik fakat burada şöyle bir durum söz konusu;

Örneğin bir array'in alt kümesine eriştikten sonra bunu isimlendirip kaydettiğimizi düşünelim.

Bu kaydetmiş olduğumuz isimlendirme üzerinde bir değişiklik yaptığımızda array'in orjinali de değişiyordu.

Fakat bazen seçilen array'in alt kümesinde o alt kümeye özel işlemler yapılmak istenebilir.

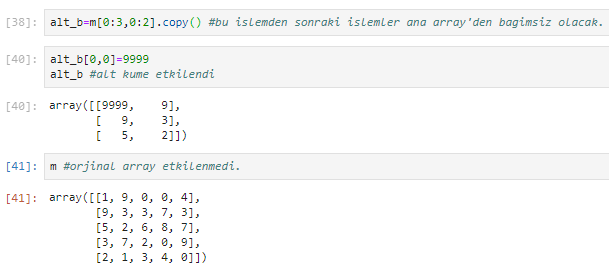
İşte bu yüzden alt kümeleri bağımsızlaştırmak isimli bir işlem yapılması gerekiyor.



Bu durum bazen çok iş görebilmekte.

Çok büyük boyutta array'ler elimizde olduğunda onların bazı parçalarını seçip spesifik olarak onların üzerinde çalışıp ana parçanın üzerinde değişiklik yapmak açısından çok işe yarar.

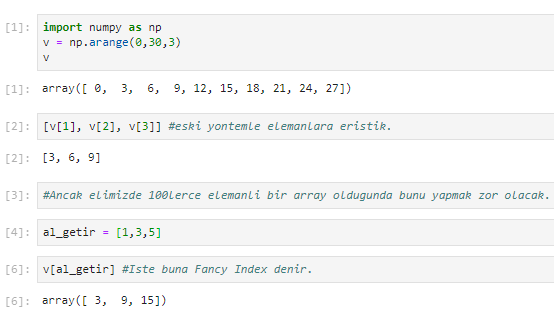
**copy()** metodunu kullanarak bu durumdan vazgeçebiliriz.



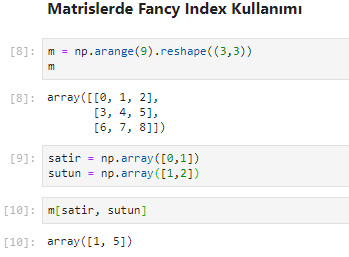
## Fancy Index ile Elemanlara Erişmek

**Fancy Index** kavramı ilerleyen bölümlerde bizim için en önemli kavramlardan birisi olacak.

Bize hem Pandas data frame'lerinde hem de NumPy array'lerinde ileri düzey eleman seçme imkanları vermektedir.



### Matrislerde Fancy Index Kullanımı



#### Basit Index ile Fancy kullanımı



#### Slice ile Fancy kullanımı

