

# BIL202 Nesneye Yönelik Programlama

Elektrik Elektronik Mühendisliği

Hafta1

Arş.Gör.Dr.Filiz Gürkan

# PYTHON VERİ TİPLERİ

Çıktılar  
Jupyter ile  
elde  
edilmiştir

```
a=5  
print(type(a))  
  
<class 'int'>
```

```
b="Medeniyet"  
print(type(b))  
  
<class 'str'>
```

```
c=(1,"2","bir")  
type(c)  
  
tuple
```

```
d=["a","2",3,"hello"]  
type(d)  
  
list
```

```
e={"bir":1,"iki":2}  
type(e)  
  
dict
```

```
f={"1",2,"IMU"}  
type(f)  
  
set
```

— Immutable  
— Mutable

- Koşullara bağlı olarak birden fazla deyim yerine getirilir.

Eğer **KOŞUL1** ise **İŞLEM1**

Eğer **KOŞUL2** ise **İŞLEM2**

...

Eğer **notun 50 üstü** ise **geçtin**

Eğer **notun 20–50 arası** ise koşullu **geçtin**

```
if koşul1:  
    işlem1  
elif koşul2:  
    işlem2  
else:  
    işlem3
```

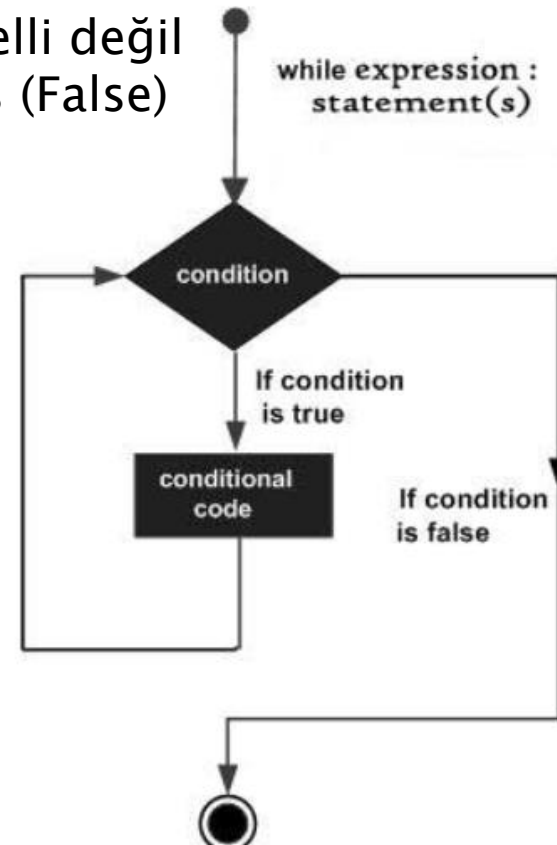
En son **else** kullanımı  
zorunlu olmasada  
kullanılması  
mantıklıdır

# While döngüsü

- İşlemlerin ne kadar tekrarlanacağı belli değil
- Bir koşulun doğru (True) ya da yanlış (False) olma durumu kontrol edilir.

```
i=1  
while i<5:  
    print(i)
```

```
i=1  
while True:  
    print(i)  
    i+=1
```



# For döngüsü

- İşlemlerin ne kadar tekrarlanacağı bellidir

```
for val in sequence:  
    loop body
```

```
for i in [1,2,3,4]:  
    print(i)
```

```
for i in ["a","b","c"]:  
    print(i)
```



## ► Fonksiyon tanımı

```
def fonksiyon_ismi (p1,p2...): #parametre varsa  
#def fonksiyon_ismi (): #parametre yoksa  
    İşlem_1  
    İşlem_2  
    return d1,d2... #değer döndürüyorsa
```

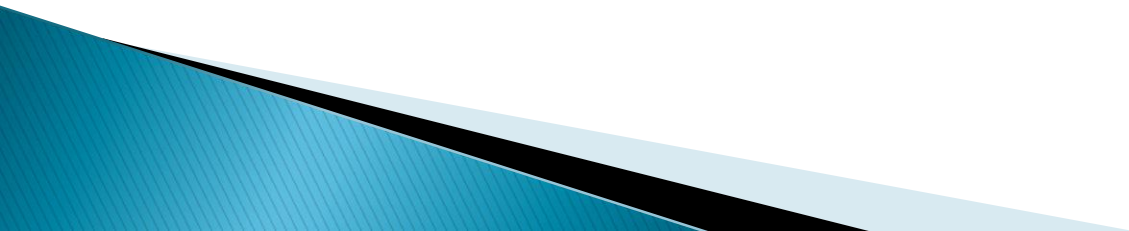
## ► Fonksiyonun çağırılması

```
fonksiyon_ismi(g1,g2) # veya fonksiyon_ismi()
```

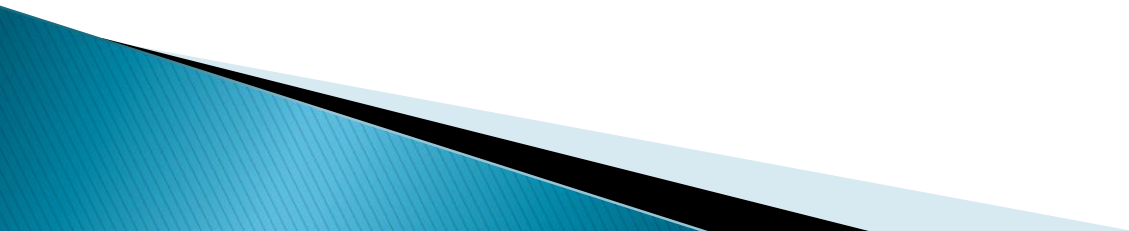
veya

```
s1,s2=fonksiyon_ismi(g1,g2) # veya s1,s2=fonksiyon_ismi()
```

vize

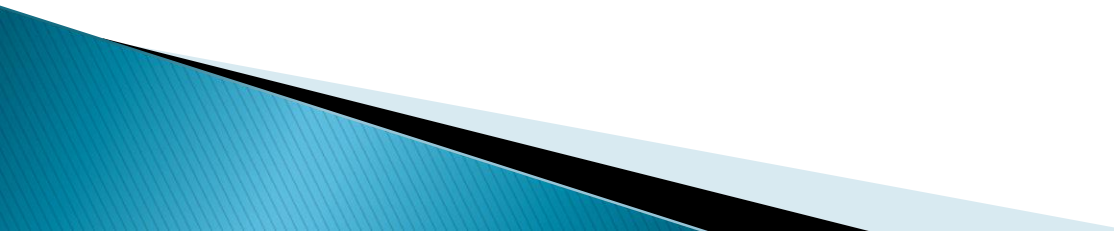


# NESNEYE YÖNELİK PROGRAMLAMA





# Avantajlar

- ▶ Okunabilir kodlar
  - ▶ Modülerlik – iş parçacıklarına bölme
  - ▶ Kodları modifiye etme kolaylığı
  - ▶ Programı kolay geliştirme
  - ▶ Modüler programlama yaptığımız için, her bir kod parçacığı başka bir projede rahatlıkla kullanılabilir
- 

```
list1=[1,2,3]  
print(type(list1))
```

```
string1="Medeniyet"  
print(type(string1))
```

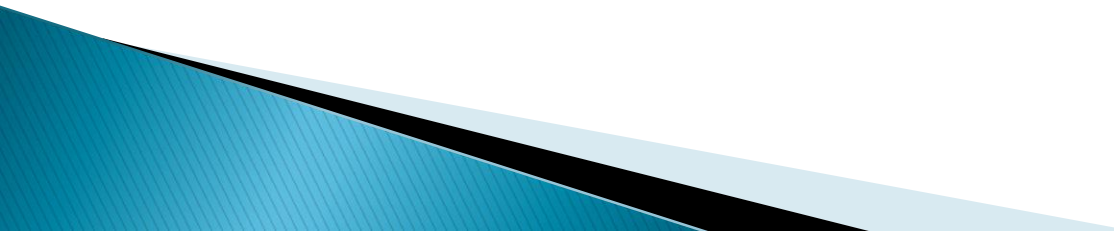
```
<class 'list'>  
<class 'str'>
```

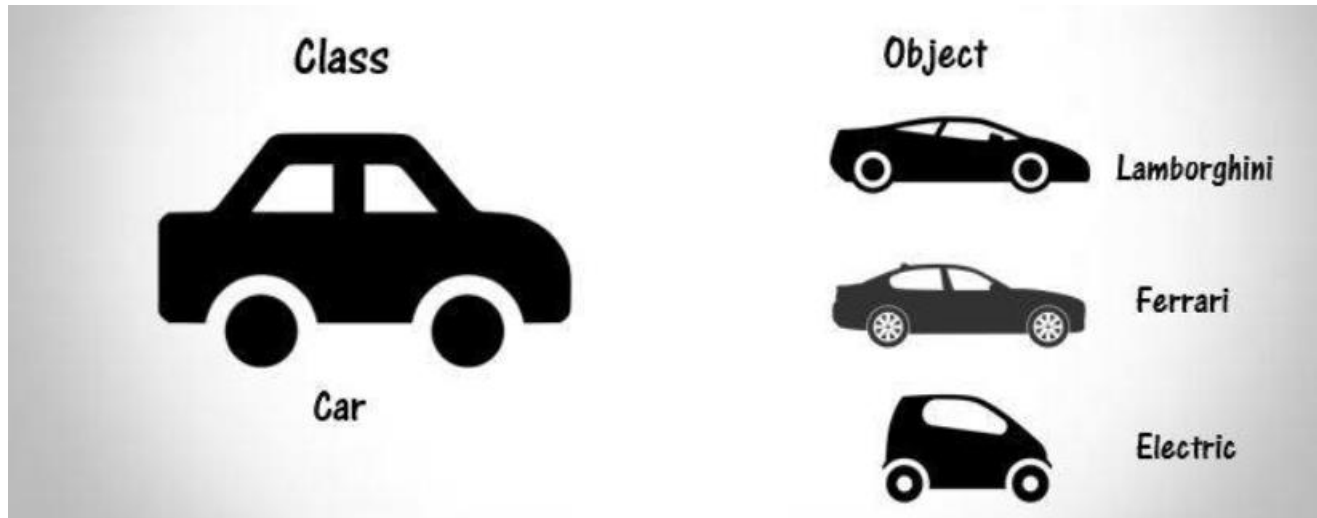
```
list1=[1,2,3]  
print(type(list1))  
  
string1="Medeniyet"  
print(type(string1))
```

list1.

F	append(object)	function	^
F	clear()	function	
F	copy()	function	
F	count(object)	function	
F	extend(iterable)	function	
F	index(object, start, stop)	function	
F	insert(index, object)	function	
F	pop(index)	function	
F	remove(object)	function	v

# OOP

- ▶ Sınıf/Nesne ilişkisi
  - ▶ Soyutlama (Abstraction)
  - ▶ Sarmalama/paketleme (encapsulation)
  - ▶ Kalıtım/Miras alma (Inheritance)
  - ▶ Çok biçimlilik (Polymorphism)
- 



Methods:  
.yavasla()  
.hizlan()  
.dur()...

```
class Araba:
    def yavasla(self):
        print ("arac yavasladi")
    def hizlan(self):
        print ("arac hizlandi")
```

```
Ferrari = Araba()
Ferrari.yavasla()
```

```
Electric = Araba()
Electric.hizlan()
```

Sınıf bir tasarım/şablondur  
Benzer özellikleri taşıyan  
objeleri oluşturmak için  
kullanılır

Class = attributes (2çеşit)+  
methods

# Sınıf nedir?

- ▶ İçerisinde sınıfa ait değişkenler ve fonksiyonların bulunduğu bir kalıp–kod bloğu olarak düşünülebilir. “class” anahtar kelimesi ile oluşturulmaktadır
- ▶ Nesne üretmemizi sağlayan veri tipidir.

```
class Araba:  
    def yavasla(self):  
        print ("arac yavasladi")  
    def hizlan(self):  
        print ("arac hizlandi")
```

```
Ferrari = Araba()  
Ferrari.yavasla()
```

```
Electric = Araba()  
Electric.hizlan()
```

# Sınıf vs Fonksiyon

- Fonksiyonlar çağırıldıklarında, sınıflar ise çağırılmadan çalışmaya başlar

```
class siparis():  
    firma="trendyol"  
    adet=0  
    tarih=""  
    stok_adet=0  
    print("Deneme")
```

```
def siparis_fonk():  
    print("Bu bir fonksiyondur")
```



# Temel kavramlar

- ▶ Sınıflar (*classes*)
- ▶ Sınıf nitelikleri (*class attributes*)
- ▶ Örnekler (*instances*)
- ▶ Örnek nitelikleri (*instance attributes*)
- ▶ Örnek metotları (*instance methods*)

```
class Asker():  
    rutbe=""  
    guc=50  
    birlik=""
```

Class attribute– sınıf nitelikleri

```
class Calisan():  
    kabiliyet=[]  
    unvan="ıscı"  
    maas=5000  
    memleket=""  
    dogum_yili=""
```

```
class siparis():  
    firma=""  
    adet=0  
    tarih=""  
    stok_adet=0
```

```
Asker.rutbe="Er"  
Calisan.memleket="İzmir"  
  
print(Asker.rutbe)  
print(Calisan.maas)  
print(siparis.firma)
```

Doğrudan sınıf adi ile niteliklere erişirsek/değiştirirsek, bu sınıf tek kullanımlık olur



# Sınıfların örneklenmesi – nesne üretme → instantiation

- ▶ Örnek = instance → sınıf'ın bir sureti, bir kopyası

```
bilgisayar=siparis()  
mehmet=Asker()  
fatma=Calisan()  
kerem=Calisan()
```

- ▶ Örnek → sınıfın tüm özelliklerini taşıyan birer üye
- ▶ Sınıflar çağrılmadan çalıştığı için, sınıf niteliklerine örneklemeden erişmek mümkündür

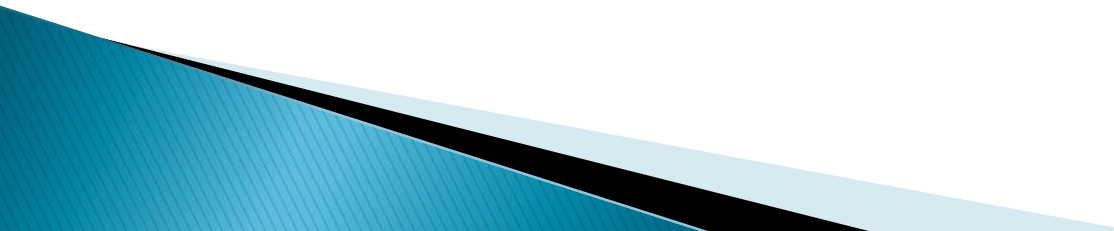
# dir(NesneAdi)-nesnenin içeriği

```
class siparis():  
    firma="trendyol"  
    adet=0  
    tarih=""  
    stok_adet=0
```

```
bilgisayar=siparis()
```

```
print(dir(bilgisayar))  
['__class__', '__delattr__', '__dict__', '__dir__',  
'__doc__', '__eq__', '__format__', '__ge__',  
'__getattribute__', '__gt__', '__hash__', '__init__',  
'__init_subclass__', '__le__', '__lt__', '__module__',  
'__ne__', '__new__', '__reduce__', '__reduce_ex__',  
'__repr__', '__setattr__', '__sizeof__', '__str__',  
'__subclasshook__', '__weakref__', 'adet', 'firma',  
'stok_adet', 'tarih']
```

# Sınıfların gömülü (built-in) özellikleri

- ▶ `__dict__` : Sınıfın ad alanını içerir
  - ▶ `__doc__` : Sınıfın belgelendirme yorumlarını içerir
  - ▶ `__name__`: Sınıfın adını içerir
  - ▶ `__module__` : Sınıfın tanımlandığı modül adı. Ana kodda ise `__main__` şeklindedir.
- 

```
class Araba :  
    """Bu bir Araba sinifidir  
    Bu araba sınıfına ait dokumantasyondur"""  
    # def yavasla(self):  
    #     print("Yavasladı")  
    # def hizlan(self):  
    #     print("Hizla")  
  
print(Araba.__dict__)  
#print(Araba.__doc__)  
#print(Araba.__name__)  
#print(Araba.__module__)
```

# Her bir örnek için sınıf özelliğini kendi içinde değiştirebiliriz

```
fatma=Calisan()  
kerem=Calisan()  
  
fatma.dogum_yili="1990"  
kerem.dogum_yili="1985"  
  
print(fatma.dogum_yili)  
print(kerem.dogum_yili)
```

```
class Calisan():  
    kabiliyet=[]  
    unvan="iscı"  
    maas=5000  
    memleket=""  
    dogum_yili=""
```

Sizce bunun  
çıktısı ne olur?



```
fatma.dogum_yili="1990"  
kerem.dogum_yili="1985"  
Calisan.dogum_yili="1970"  
print(fatma.dogum_yili)  
print(kerem.dogum_yili)  
ali=Calisan()  
print(ali.dogum_yili)
```

# Değiştirilebilen ve Değiştirilemeyen türde sınıf özelliği

```
class Calisan():  
    kabiliyet=[]  
    unvan="İscı"  
    maas=5000  
    memleket=""  
    dogum_yili=""
```

Değiştirilebilen: liste, sözlük, küme  
Değiştirilemeyen: karakter, sayı, demet

**DEĞİŞTİRİLEBİLİR BİR VERİ TİPİNDE  
YAPILACAK DEĞİŞİKLİKLER TÜM  
NESNELERE YANSIR**

```
fatma=Calisan()  
kerem=Calisan()  
  
fatma.kabiliyet.append("caliskan")  
kerem.kabiliyet.append("GrupCalismasi")  
#Calisan.dogum_yili="1970"  
print(fatma.kabiliyet)  
print(kerem.kabiliyet)  
ali=Calisan()  
print(ali.kabiliyet)
```

Sizce neden?

```
class Calisan():  
    kabiliyet=[]  
    unvan="isci"  
    maas=5000  
    memleket="aydin"  
    dogum_yili=""
```

```
ahmet=Calisan()  
fatma=Calisan()  
print(id(ahmet.kabiliyet))  
print(id(fatma.kabiliyet))  
print(id(ahmet.maas))  
print(id(fatma.maas))  
ahmet.maas=7000  
fatma.maas=8000  
print(id(ahmet.maas))  
print(id(fatma.maas))
```

```
ahmet.kabiliyet.append("konuskan")  
fatma.kabiliyet.append("caliskan")
```

```
print(id(ahmet.kabiliyet))  
print(id(fatma.kabiliyet))
```

# Örnek nitelikleri-instance attributes

`__init__` fonksiyonu ve `self`

`__init__` fonksiyonu, sınıfımızı örneklediğimiz anda oluşacak nitelik ve işlemleri tanımlar.

**YAPICI FONKSİYON (constructors)**

Örnek nitelikleri yapıcı fonk. içinde tanımlanır

```
class Sınıf():  
    sinif_a1=0  
    sinif_a2=0  
    print("sinif")  
    def __init__(self):  
        self.ornek_a1=0  
        self.ornek_a2=0  
        print("örnek")
```

`__init__` fonksiyonunun ilk parametresi **self** olmalıdır

```
deneme=Sınıf()
```

Bu noktalarda çıkış olarak ne görünür?



# Örnek nitelikleri

- Sınıf niteliklerine, sınıf adları üzerinden erişebilirken, örnek niteliklerine sadece örnek adları üzerinden erişebiliriz.

```
class Sınıf():  
    sinif_a1=1  
    sinif_a2=2  
    #print("sinif")  
    def __init__(self):  
        self.ornek_a1=3  
        self.ornek_a2=4  
        #print("örnek")
```

```
print(Sınıf.sinif_a1)  
print(Sınıf.ornek_a1)  
deneme=Sınıf()  
print(deneme.sinif_a1)  
print(deneme.ornek_a1)
```



```
print(Sınıf.sinif_a1)  
print(Sınıf().ornek_a1)  
deneme=Sınıf()  
print(deneme.sinif_a1)  
print(deneme.ornek_a1)
```

# self

- ▶ self, bir sınıfın örneklerini temsil eder ??

```
class Calisan():  
    def __init__(self):  
        self.kabiliyet=[]  
        self.unvan="isci"  
        self.maas=5000  
        self.memleket=""  
        self.dogum_yili=""  
  
fatma=Calisan()  
kerem=Calisan()
```

**KURAL:** init fonk. ilk parametresi ne ise, örnek niteliklerini temsil eden kelime odur.

**ANCAK** kodun okunurluğu açısından self kullanılması anlamlı olacaktır

self kelimesi, fatma,  
ahmet kelimesini  
(nesnesini) temsil eder

**self kelimesini kullanmak  
zorunlu değildir**

```
class Calisan():  
    def __init__(xywh):  
        xywh.unvan="Muhendis"  
        xywh.maas="5000"  
  
fatma=Calisan()  
print(fatma.maas)  
print(fatma.unvan)
```

# Sınıf ve Örnek nitelikleri

```
class Calisan():  
    maas="3000"  
    def __init__(self):  
        self.unvan="Muhendis"  
  
fatma=Calisan()  
print(fatma.maas)  
print(fatma.unvan)
```

```
class Calisan():  
    maas="3000"  
    def __init__(self):  
        self.unvan="Muhendis"  
        self.maas="5000"  
  
fatma=Calisan()  
print(fatma.maas)  
print(fatma.unvan)
```

Yine de aynı adı taşıyan örnek ve sınıf niteliği çok mantıklı değildir

# KISACA:

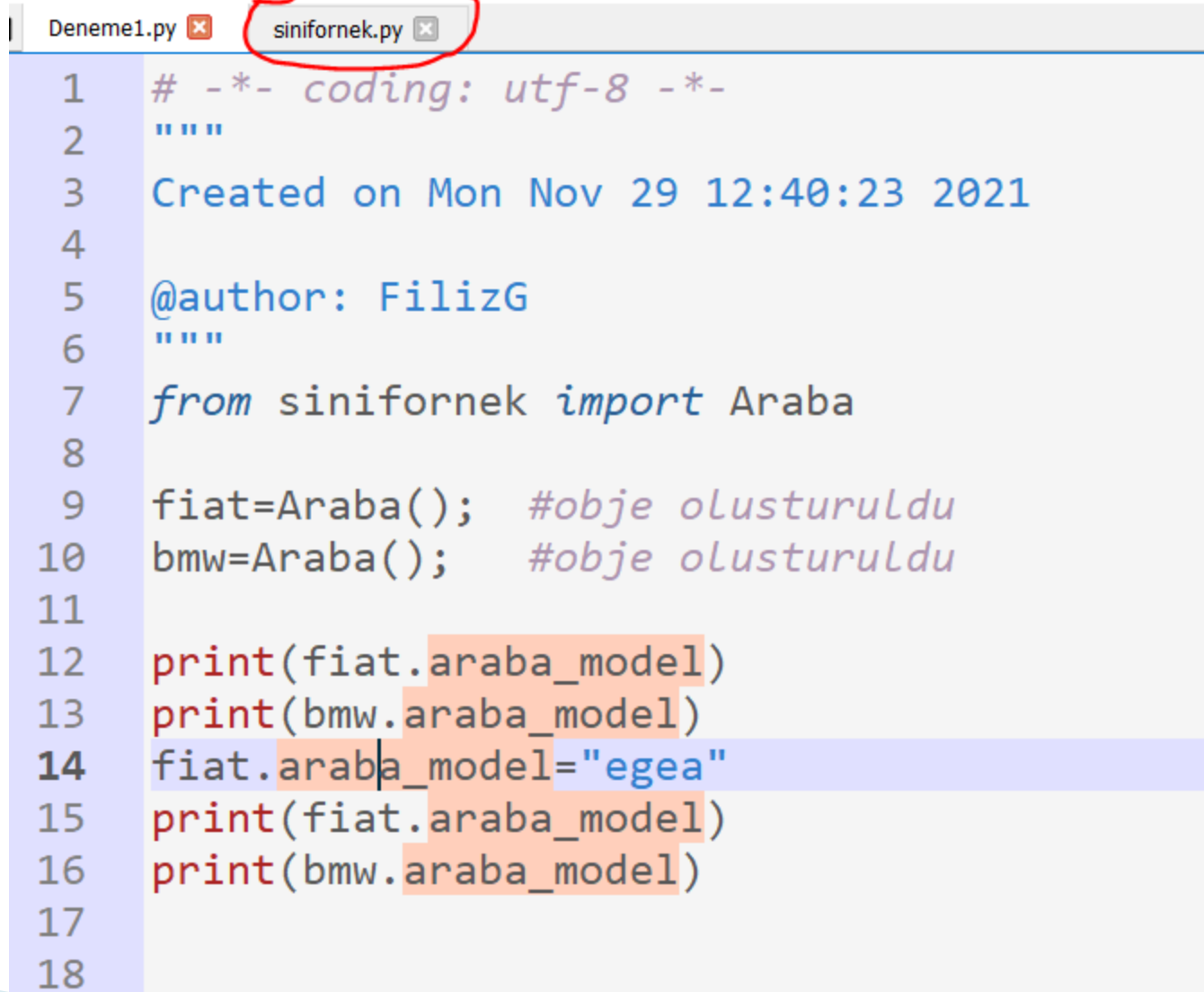
```
class Calisan():  
    kabiliyet=[]  
    def __init__(self):  
        self.hobi=[]
```

```
fatma=Calisan()  
ahmet=Calisan()  
ahmet.kabiliyet.append("caliskan")  
print("fatma kabiliyet=",fatma.kabiliyet)  
print("ahmet kabiliyet=",ahmet.kabiliyet)  
ali=Calisan()  
print("ali kabiliyet=",ali.kabiliyet)
```

```
ahmet.hobi.append("kosu")  
print("fatma hobi=",fatma.hobi)  
print("ahmet hobi=",ahmet.hobi)  
ali=Calisan()  
print("ali hobi=",ali.hobi)
```

```
fatma kabiliyet= ['caliskan']  
ahmet kabiliyet= ['caliskan']  
ali kabiliyet= ['caliskan']  
fatma hobi= []  
ahmet hobi= ['kosu']  
ali hobi= []
```

Araba sınıfını içeren kod

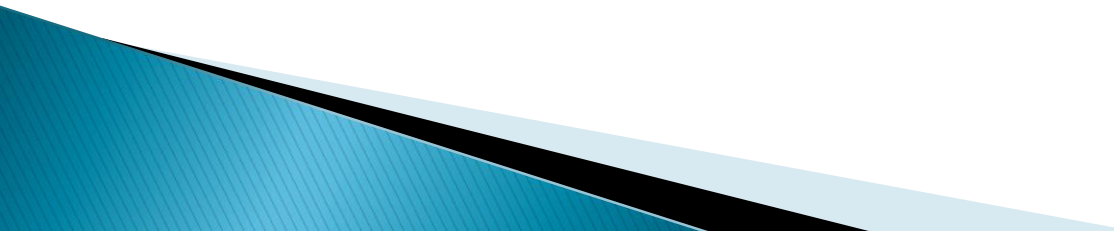


```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Nov 29 12:40:23 2021
4
5  @author: FilizG
6  """
7  from sinifornek import Araba
8
9  fiat=Araba(); #obje olusturuldu
10 bmw=Araba(); #obje olusturuldu
11
12 print(fiat.araba_model)
13 print(bmw.araba_model)
14 fiat.araba_model="egea"
15 print(fiat.araba_model)
16 print(bmw.araba_model)
17
18
```

```
class Calisan():  
    def __init__(self, isim, dogum):  
        self.isim=isim  
        self.d_yili=dogum  
  
calisan1=Calisan("Ahmet",1990)  
  
print(calisan1.isim)  
print(calisan1.d_yili)
```

# Örnek Methodları

Method, bir **Nesnenin** (object) davranışını temsil eder. Bir **Sınıf** (class) içinde tanımlanmalıdır. Nokta (.) kullanılarak çağrılır. Method ilişkilendirdikleri nesnenin verileri üzerinde çalışabilirler. Ait oldukları sınıfa bağımlıdır.



```
class Araba :  
    """Bu bir Araba sinifidir  
    Bu araba sınıfına ait dokumantasyondur"""  
    def yavasla(self):  
        print("Yavasladi")  
    def hizlan(self):  
        print("Hizlandi")  
  
fiat=Araba();  #obje olusturuldu  
bmw=Araba();  #obje olusturuldu  
  
fiat.yavasla()  
bmw.hizlan()
```



# Temel kavramlar

- ▶ Sınıflar (*classes*)
- ▶ Örnekler (*instances*)
- ▶ Sınıf nitelikleri (*class attributes*)
  - Hem sınıf hem örnek ismi ile çağrılabilir
- ▶ Örnek nitelikleri (*instance attributes*)
  - Sadece örnek ismi ile çağrılabilir
- ▶ Örnek metotları (*instance methods*)
  - Sadece örnek ismi ile çağrılabilir

## ▶ SINIF METOTLARI

- ▶ Sınıf niteliklerine hem örnekler hem de doğrudan sınıf adları üzerinden erişilebilir.
- ▶ Örnek niteliklerine ve örnek metotlarına yalnızca örnekler üzerinden erişebiliriz.

*Ders* isimli;

Dersin bilgilerini (isim,kredi, gün,saat) yazdıran,

Ders katılımcılarını gösteren ,

Derse katılımcı ekleyen sınıfı yazınız.

```
class Ders:
    uni="Istanbul Medeniyet Universitesi"
    def __init__(self,i,k,g,s):
        self.isim=i
        self.kredi=k
        self.gun=g
        self.saat=s
        self.ogrenci=[]
    def ders_bilgileri(self):
        print("{} kredili {} dersinin günü {}, saati {} olarak kayıtlıdır".
              format(self.kredi, self.isim,self.gun,self.saat))
    def katilimci_goster(self):
        for i in self.ogrenci:
            print(i)
    def katilimci_ekle(self,ogr):
        self.ogrenci.append(ogr)

Mat=Ders("Matematik",5,"Pazartesi","13.30")
Mat.ders_bilgileri()
print(Mat.uni)
Mat.katilimci_goster()
Mat.katilimci_ekle("Filiz Gürkan")
Mat.katilimci_goster()
```

```
Mat=Ders("Matematik",5,"Pazartesi","13.30")
Mat.ders_bilgileri()
print(Mat.uni)
Mat.katilimci_goster()
Mat.katilimci_ekle("Filiz Gürkan")
Mat.katilimci_goster()
#Ders.uni="Marmara Universitesi"
#Mat.uni="Marmara Universitesi"
```

```
Fiz=Ders("Fizik",5,"Salı", "13.30")
Fiz.ders_bilgileri()
print(Fiz.uni)
Fiz.katilimci_ekle("Tarkan")
Fiz.katilimci_goster()
```

→ Katılımcı yoksa,  
katılımcı yoktur  
yazsın

# Sınıf methotlarına neden ihtiyaç duyarız

```
class Matematik:
    ogrenci=[]
    def __init__(self,i,s):
        self.isim=i
        self.sinif=s
        self.katilimci_ekle()
    def katilimci_ekle(self):
        self.ogrenci.append(self.isim)
        print("{} isimli ogrenci eklendi.".format(self.isim))
    def katilimci_goster(self):
        for i in self.ogrenci:
            print(i)
    def katilimci_sayisi(self):
        print(len(self.ogrenci))

ahmet=Matematik("Ahmet",2)
mehmet=Matematik("Mehmet",1)
mehmet.katilimci_sayisi()
```

Hatırlatma: Sınıflar çağırılmadan önce (örneklenmeden önce) işlerler, ancak methotlar – fonksiyonlar için bu geçerli değildir.

```
class Matematik:
    ogrenci=[]
    def __init__(self,i,s):
        self.isim=i
        self.sinif=s
        self.katilimci_ekle()
    def katilimci_ekle(self):
        self.ogrenci.append(self.isim)
        print("{} isimli ogrenci eklendi.".format(self.isim))
    def katilimci_goster(self):
        for i in self.ogrenci:
            print(i)
    def katilimci_sayisi(self):
        print(len(self.ogrenci))

ahmet=Matematik("Ahmet",2)
mehmet=Matematik("Mehmet",1)
mehmet.katilimci_sayisi()
```

Katılımcı sayısı, katılımcı gösterme metotları için:  
oluşturulmuş nesneler kullanılır.

Bu metotlar, sınıfı ilgilendirir

Katılımcı sayısı 0 ise ?

```
Matematik.katilimci_sayisi()
Matematik.katilimci_goster()
```

## ► Çözüm1: Sınıftan ayırık bir yazdırma

```
print(len(Matematik.ogrenci))
```

Kod bütünlüğü bozuldu.

## ► Çözüm: Sınıf metotları

@classmethod ve cls

```
class Sınıf():
    sınıf_ozelligi = 0

    def __init__(self, param1):
        self.param1 = param1
        self.ornek_ozelligi = 0

    def örnek_metodu(self):
        pass

    def sınıf_metodu(cls):
        pass
```

self kelimesi, python geleneği için önemli olsada kod için bir anlam ifade etmiyordu?

## ► Çözüm: Sınıf metotları

@classmethod ve cls

```
class Sınıf():  
    sınıf_ozelligi = 0  
  
    def __init__(self, param1):  
        self.param1 = param1  
        self.ornek_ozelligi = 0  
  
    def örnek_metodu(self):  
        pass  
  
    @classmethod  
    def sınıf_metodu(cls):  
        pass
```



- ▶ Sınıf niteliklerine ve **örnek metotlarına** hem örnekler hem de doğrudan sınıf adları üzerinden erişilebilir.
- ▶ Örnek niteliklerine ve örnek metotlarına yalnızca örnekler üzerinden erişebiliriz.

```
class Matematik:
    ogrenci=[]
    def __init__(self,i,s):
        self.isim=i
        self.sinif=s
        self.katilimci_ekle()
    def katilimci_ekle(self):
        self.ogrenci.append(self.isim)
        print("{} isimli ogrenci eklendi."
              .format(self.isim))
    def katilimci_goster(self):
        for i in self.ogrenci:
            print(i)
    @classmethod
    def katilimci_sayisi(cls):
        print(len(cls.ogrenci))

ahmet=Matematik("Ahmet",2)
ahmet.katilimci_sayisi()
Matematik.katilimci_sayisi()
```

# Örnek:

- ▶ Bir banka sistemine giriş yapmak için `Giris()` sınıfı tanımlayalım. Ön tanımlı olarak Müşteri numarası ile giriş sağlansın, ancak isteğe bağlı Parola veya TC Kimlik Numarası ile de giriş imkanı sağlanılsın.

# Cozum1

```
class Giris():  
    def __init__(self, grs):  
        if grs=="Musteri":  
            mesaj="Musteri numaranizi giriniz"  
            sifre="123456"  
        elif grs=="parola":  
            mesaj="Lütfen parolanızı giriniz"  
            sifre="654321"  
        else:  
            mesaj="Lütfen TCK giriniz"  
            sifre="11223344556"  
        cevap = input(mesaj)  
        if cevap==sifre:  
            print('Hoşgeldiniz!')  
        else:  
            print("Yanlış giris")  
  
Giris("parola")
```

# Cozum2 (daha güzel)

```
class Giris():  
    def __init__(self, mesaj='Müşteri numaranız: ',sifre="123456"):  
        cevap = input(mesaj)  
        if cevap==sifre:  
            print('Hoşgeldiniz!')  
        else:  
            print("Yanlış giris")  
  
    @classmethod  
    def paroladan(cls):  
        mesaj = 'Lütfen parolanızı giriniz: '  
        cls(mesaj,"7654321")  
  
    @classmethod  
    def tcknden(cls):  
        mesaj = 'Lütfen TC kimlik numaranızı giriniz: '  
        cls(mesaj,"11223344556")  
  
Giris.tcknden()
```

# Statik metotlar-@staticmethod

- ▶ Örnek nitelikleri üzerinde işlem → örnek metotları
  - self
- ▶ Sınıf nitelikleri üzerinde işlem → sınıf metotları
  - cls
- ▶ Sınıf içindeki herhangi bir fonksiyonda örnek veya sınıf niteliklerinin hiçbirine erişmeniz gerekmiyorsa → statik metot

- ▶ Anlamsal olarak sınıfla bağlantılı, ancak sınıfın/örneğin herhangi bir niteliğine erişmesine gerek olmayan fonksiyonlar
- ▶ Bütünlük bozulmaması açısından ayırık yazılmaz
- ▶ Örnek/sınıf adı üzerinden erişilebilir

```
class Sınıf():
    sınıf_niteliği = 0

    def __init__(self, deger):
        self.ornek_n = deger

    def ornek_metodu(self):
        return self.ornek_n

    @classmethod
    def sınıf_metodu(cls):
        print(cls.sınıf_niteliği)
        return cls.sınıf_niteliği

    @staticmethod
    def statik_metot():
        print("Bu bir statik metottur")

Sınıf(5).statik_metot()
```



# Hatırlatma–Nesneler

- ▶ Belli birtakım metotlara ve/veya niteliklere sahip olan öğelere nesne adı verilir.
- ▶ Nesne üretilmesi

```
class Sınıf():  
    pass
```

```
nesne=Sınıf()      #Örnekleme yaparak nesne üretildi
```

# ÖRNEK–Oyun

- ▶ Oyuncu sınıfı oluşturulacak ve bu sınıftan iki adet oyuncu örneklenecek (nesneler)
- ▶ Kurallar: Her oyuncunun başlangıçta 3 canı ve 0 darbesi olacak, her 3 darbede bir can gidecek
- ▶ Ana oyuncu (siz) saldırı/kaçış durumlarından birini seçecek
  - Saldırı durumunda berabere/darbe alma/kazanma durumları random belirlenecek
  - Kaçma durumunda kaçma/darbe alma durumu random belirlenecek

```
class Oyuncu():  
    def __init__(self, isim, can=3):  
        self.isim = isim  
        self.darbe = 0  
        self.can = can  
  
    def oyuncu_durumu(self):  
        print('darbe: ', self.darbe)  
        print('can: ', self.can)
```

Şimdi saldır ve kaç metotlarını tanımlayalım

```
def saldır(self, rakip):  
    print('Bir saldırı gerçekleştirdiniz.')  
    print('Saldırı sürüyor. Bekleyiniz.')  
  
    for i in range(10):  
        time.sleep(.3)  
        print('.', end='')  
  
    sonuç = random.randint(0, 2)  
  
    if sonuç == 0:  
        print('\nSONUÇ: kazanan taraf yok')  
  
    if sonuç == 1:  
        print('\nSONUÇ: rakibinizi darbelediniz')  
        self.darbele(rakip)  
  
    if sonuç == 2:  
        print('\nSONUÇ: rakibinizden darbe aldınız')  
        rakip.darbele(self)
```

```
def kaç(self):  
    print('Kaçılıyor...')  
    for i in range(10):  
        time.sleep(.3)  
        print('\n')  
    sonuc=random.randint(0, 1)  
    if sonuc==0:  
        print("Tebrikler kaçtınız")  
    else:  
        print('Rakibiniz sizi yakaladı')  
        rakip.darbele(self)
```

darbele methodu???

```
def darbele(self, darbelenen):  
    darbelenen.darbe += 1  
    if (darbelenen.darbe % 3) == 0:  
        darbelenen.can -= 1  
    if darbelenen.can < 1:  
        print('Oyunu {} kazandı!'.format(self.isim))  
        print('Çıkılıyor...')  
        sys.exit()
```

# Ana kodumuz:

```
siz = Oyuncu("Filiz")
rakip = Oyuncu('Mert')

while True:
    print("""Şu anda rakibinizle karşı karşıyasınız.
    Yapmak istediğiniz hamle:
    Saldır:  s
    Kaç:     k""")

    hamle = input('')
    if hamle == 's':
        siz.saldır(rakip)

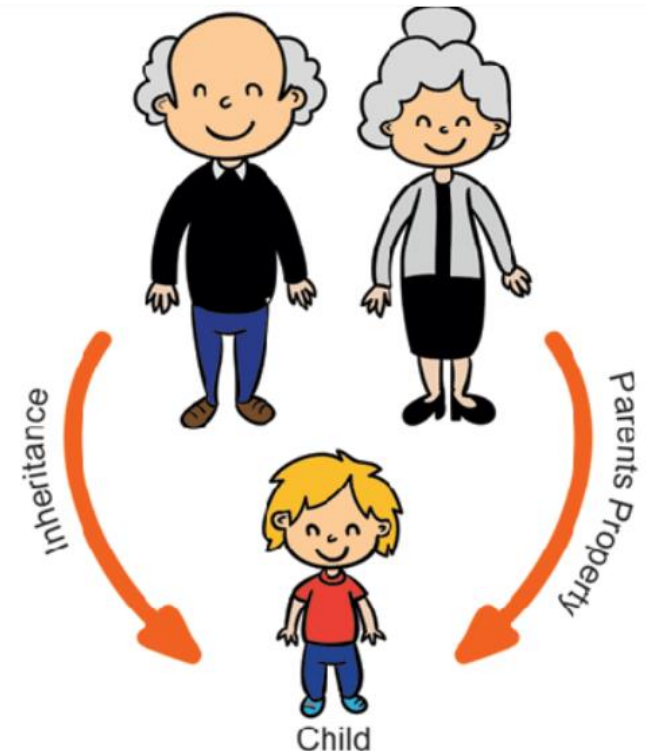
        print('Rakibinizin durumu')
        rakip.oyuncu_durumu()
        print('Sizin durumunuz')
        siz.oyuncu_durumu()

    if hamle == 'k':
        siz.kaç()
        print('Rakibinizin durumu')
        rakip.oyuncu_durumu()
        print('Sizin durumunuz')
        siz.oyuncu_durumu()
```

# Miras alma/Kalıtım/Inheritance

- ▶ Yazmış olduğumuz bir **class**'a ait özellikleri başka bir **class**'ta kullanabiliriz ve bu özellikleri **kalıtım** yoluyla aktarabiliriz
- ▶ Kalıtım OOP dillerinin temel bir özelliğidir.

Kalıtım tek bir üst sınıftan olabileceğiz gibi (**single**), birden fazla üst sınıftan da (**multiple**) olabilir





# Örnek

```
class Calisan():  
    def __init__(self, isim, soyisim, maas):  
        self.isim = isim  
        self.soyisim = soyisim  
        self.maas = maas  
  
    def bilgileri_goster(self):  
        print(" İsim:", self.isim, "\n", "Soyisim:", self.soyisim)  
  
class Yonetici():  
    def __init__(self, isim, soyisim, maas, pozisyon):  
        self.isim = isim  
        self.soyisim = soyisim  
        self.maas = maas  
        self.pozisyon = pozisyon  
  
    def bilgileri_goster(self):  
        print(" İsim:", self.isim, "\n", "Soyisim:", self.soyisim)
```

# Örnek

```
class Calisan():  
    def __init__(self, isim, soyisim, maas):  
        self.isim = isim  
        self.soyisim = soyisim  
        self.maas = maas  
  
    def bilgileri_goster(self):  
        print(" İsim:", self.isim, "\n", "Soyisim:", self.soyisim)  
  
class Yonetici():  
    def __init__(self, isim, soyisim, maas, pozisyon):  
        self.isim = isim  
        self.soyisim = soyisim  
        self.maas = maas  
        self.pozisyon = pozisyon  
  
    def bilgileri_goster(self):  
        print(" İsim:", self.isim, "\n", "Soyisim:", self.soyisim)
```

```
class Yonetici(Calisan):  
    pass
```

- ▶ Yonetici sınıfı (Child), Calisan sınıfının (parent) tüm özelliklerine sahip
- ▶ Yonetici pozisyon özelliği??

# Overriding– super() methodu

```
class Yonetici(Calisan):  
    def __init__(self, isim, soyisim, maas, pozisyon):  
        self.isim = isim  
        self.soyisim = soyisim  
        self.maas = maas  
        self.pozisyon=pozisyon
```

**super().\_\_init\_\_(ust\_sinif\_parametreleri)**

```
class Yonetici(Calisan):  
    def __init__(self, isim, soyisim, maas, pozisyon):  
        super().__init__(isim, soyisim, maas,)  
    def bilgileri_goster(self):  
        print(" İsim:", self.isim, "\n", "Soyisim:", self.soy
```

**DOĞRUDAN KULLANILAN METHODLARI TEKRAR YAZMAYA  
GEREK YOKTUR**

```
class Calisan():
    def __init__(self, isim, soyisim, maas):
        self.isim = isim
        self.soyisim = soyisim
        self.maas = maas
    def isim_goster(self):
        print(" İsim:", self.isim, "\n", "Soyisim:")
    def bilgileri_goster(self):
        print(" İsim:", self.isim, "\n", "Soyisim:", self.soyisim, "\n", "Maaş:", s

class Yonetici(Calisan):
    def __init__(self, isim, soyisim, maas, pozisyon):
        super().__init__(isim, soyisim, maas,)
        self.pozisyon=pozisyon
    def bilgileri_goster(self):
        print(" İsim:", self.isim, "\n", "Soyisim:", self.soyisim, "\n", "Maaş:", s

calisan_1=Calisan("Filiz", "Gurkan", 50000)
yonetici_1=Yonetici("Elon", "Musk", 100000,"CEO")
calisan_1.isim_goster()
yonetici_1.isim_goster()
```