

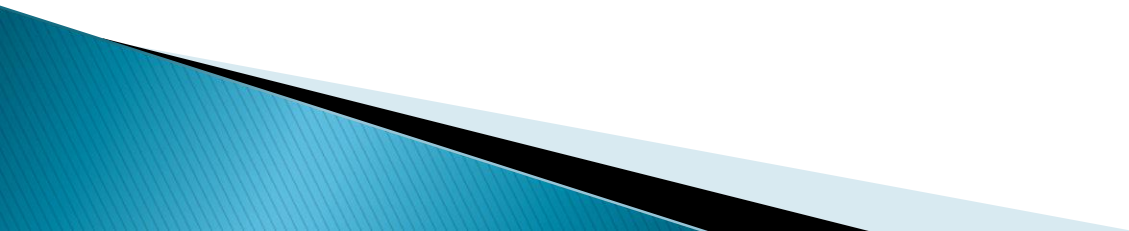
BIL202 Nesneye Yönelik Programlama

Elektrik Elektronik Mühendisliği

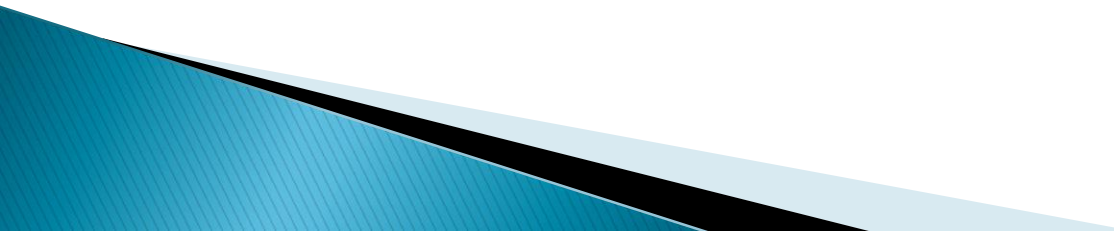
Hafta 12

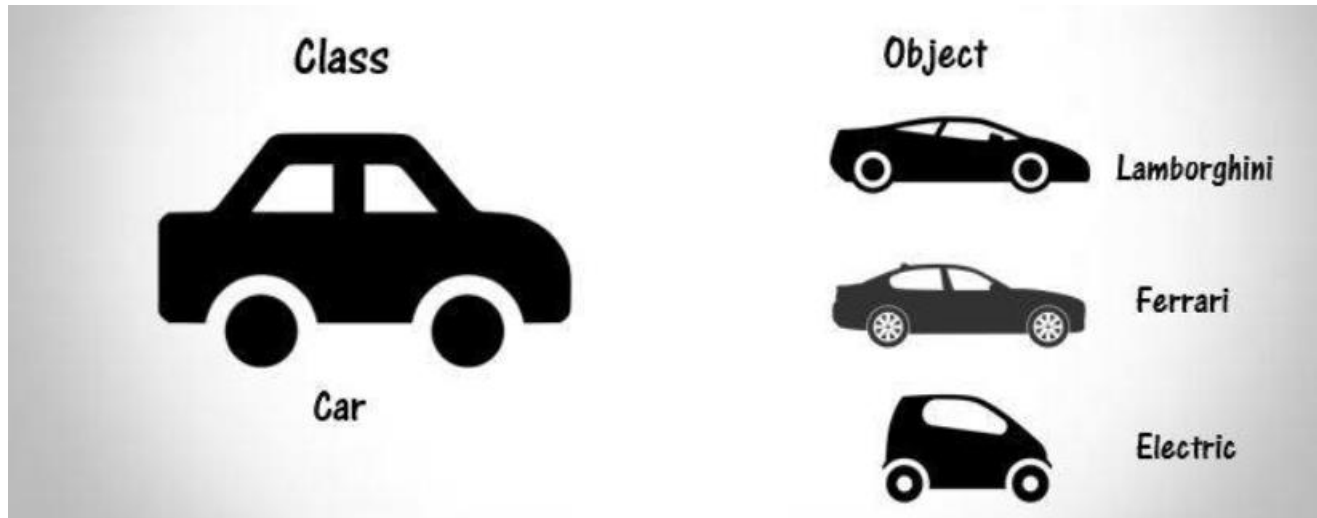
Arş.Gör.Dr.Filiz Gürkan

NESNEYE YÖNELİK PROGRAMLAMA



OOP

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

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
 - Hem sınıf hem örnek ismi ile çağrılabilir

Statik metotlar-@staticmethod

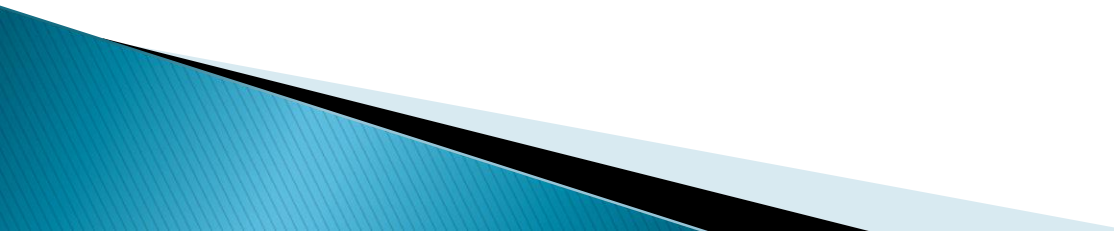
- ▶ Örnek nitelikleri üzerinde işlem → örnek metotları
 - self
- ▶ Sınıf nitelikleri üzerinde işlem → sınıf metotları
 - cls
- ▶ Sınıf içindeki herhangi bir fonksiyonda örnek veya sınıf niteliklerinin hiçbirine erişmeniz gerekmiyorsa → statik metot

- ▶ 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 Matematik:  
    uni="IMU"  
    def __init__(self):  
        self.kredi=5  
    @classmethod  
    def kredi_yazdir(cls):  
        print(cls.uni)  
    @staticmethod  
    def stdm(a,b):  
        print("toplam=",a+b)
```

```
A=Matematik()  
x=Matematik()  
x.stdm(5,2)  
Matematik.stdm(1,2)
```

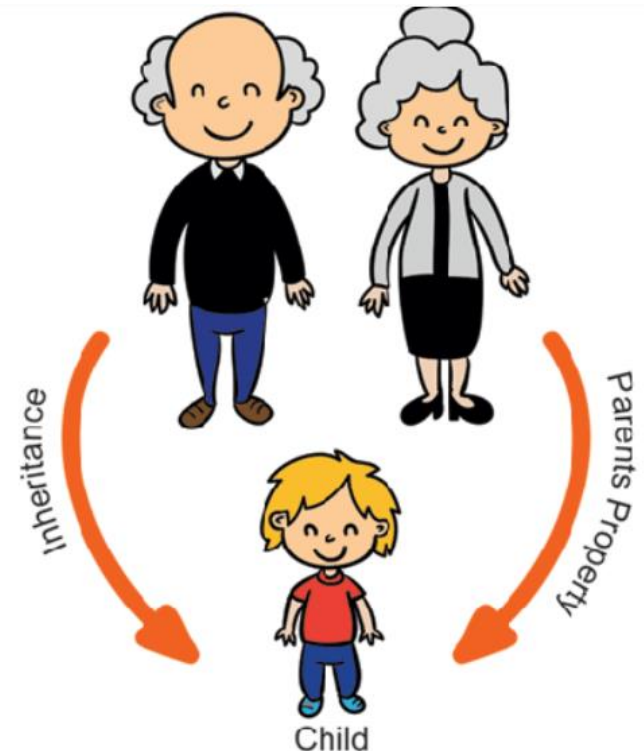
OOP

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

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



Types of Inheritance



```
graph LR; A[Types of Inheritance] --> B[Single]; A --> C[Multiple]; A --> D[Multilevel]; A --> E[Hierarchical];
```

Single

Multiple

Multilevel

Hierarchical

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
- ▶ PARENT CLASS
- ▶ 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,)  
        self.pozisyon=pozisyon  
    def bilgileri_goster(self):  
        print(self.isim, "\n", self.soyisim, "\n", self.ma
```

**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()
```

Eğer alt sınıfa eklenen herhangi bir nitelik veya metot parent sınıfta zaten varsa, alt sınıfa eklenen nitelik ve metotlar taban sınıftaki metot ve niteliklerin yerine geçecektir.

Metotlar için Super()

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

    def bilgileri_goster(self):
        print(self.isim, "\n", self.soyisim, "\n", self.maas)
```

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

    def bilgileri_goster(self):
        print(self.isim, "\n", self.soyisim, "\n", self.maas, "\n", self.pozisyon)
```

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

    def bilgileri_goster(self):
        super().bilgileri_goster()
        print(self.pozisyon)
```


Çoklu kalıtım (multiple)

```
class Anne:
    def __init__(self, isim):
        self.isim = isim
        print("AnneSınıfı")
    def yuru(self):
        print("yuru methodu")
class Baba:
    def __init__(self, soyad):
        self.soyad = soyad
        print("BabaSınıfı")
    def kos(self):
        print("kos methodu")
class Cocuk(Arne, Baba):
    def dur(self):
        print("dur methodu")

a = Cocuk("Yıldırım")
a.kos()
a.yuru()
```

Cocuk için __init__ methodu nasıl belirlenir?

Anne mi öncelikli baba mı?

Öncelik değiştirmek istenirse

```
class Anne:
    def __init__(self, isim):
        self.isim=isim;
        print("AnneSınıfı")
    def yuru(self):
        print("yuru methodu-Anne")

class Baba():
    def __init__(self, soyad):
        self.soyad=soyad
        print("BabaSınıfı")
    def kos(self):
        print("kos methodu")
    def yuru(self):
        print("yuru methodu-Baba")

class Cocuk(Arne, Baba):
    def __init__(self, soyad):
        Baba.__init__(self, soyad)
    def dur(self):
        print("dur methodu")

a=Cocuk("Yıldırım")
print(a.soyad)
a.yuru()
```

Doğrudan
Cocuk(Baba, Anne) neden
yapmadık?

Öncelik değiştirmek istenirse

```
class Cocuk(Anne,Baba):  
    def __init__(self,isim,soyad):  
        Anne.__init__(self, isim)  
        Baba.__init__(self,soyad)  
    def dur(self):  
        print("dur methodu")  
    def yuru(self):  
        Baba.yuru(self)  
  
a=Cocuk("Ahmet","Burak")  
print(a.isim)  
print(a.soyad)
```

Hem anne hem baba'nın
öznitelikleri istenirse?

Çok seviyeli (Multilevel) kalıtım

```
class AnneAnne:
    def __init__(self, isim):
        self.isim = isim
        print("AnneAnneSınıfı")
    def yuru(self):
        print("yuru methodu")
class Anne(ArneAnne):
    def __init__(self, soyad):
        self.soyad = soyad
        print("AnneSınıfı")
    def kos(self):
        print("kos methodu")

class Cocuk(Arne):
    def dur(self):
        print("dur methodu")

a = Cocuk("Yıldırım")
print(a.soyad)
a.yuru()
a.kos()
```

Hiyerarşik (Hierarchical) kalıtım

- ▶ Bir üst sınıftan birden çok sınıf türetilmesini sağlar

PARENT CLASS = BASE(BAZ-TABAN) CLASS

```
class Isci:
    def __init__(self, isim, soyisim, memleket):
        self.isim=isim;
        self.soyisim=soyisim;
        self.memleket=memleket;
    def calis(self):
        print("calıs methodu")
    def yemek(self):
        print("yemek methodu")

class Yonetici:
    def __init__(self, isim, soyisim, memleket, maas):
        self.isim=isim;
        self.soyisim=soyisim;
        self.memleket=memleket;
        self.maas=maas;
    def calis(self):
        print("calıs methodu")
    def toplantı(self):
        print("toplantı methodu")

class Muhendis():
    def __init__(self, isim, soyisim, memleket):
        self.isim=isim;
        self.soyisim=soyisim;
        self.memleket=memleket;
    def calis(self):
        print("calıs methodu")
    def mesai(self):
        print("mesai methodu")
```

Bu üç sınıfın bir çok ortak nitelik ve methodu var-
Bunların temelini (bazını) oluşturan tek bir sınıf tanımlanabilir = parent=base class

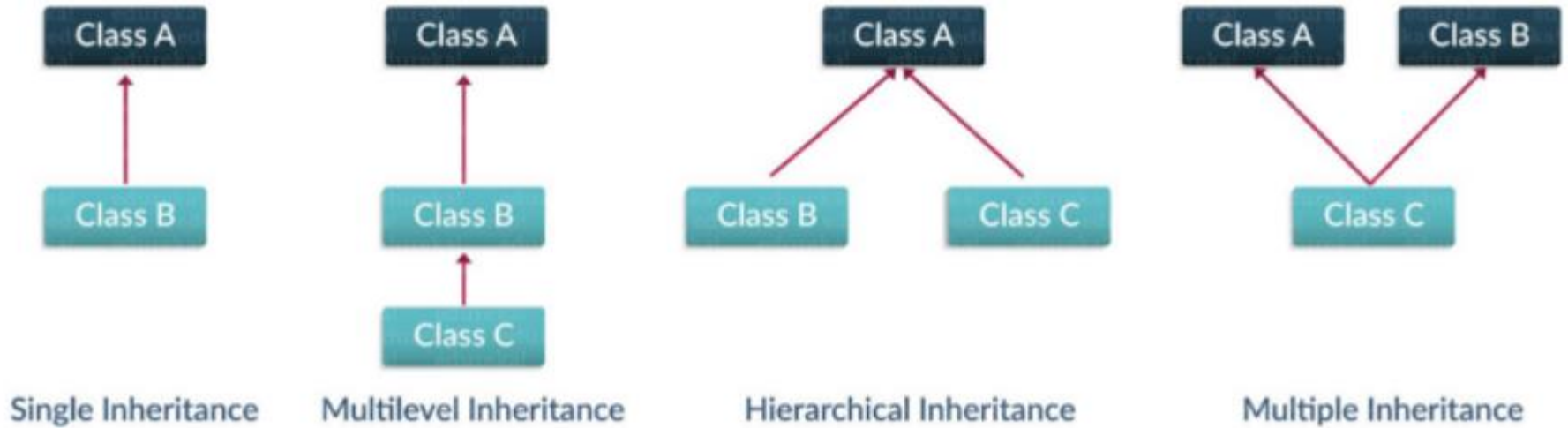
```
class Calisan:
    def __init__(self, isim, soyisim, memleket):
        self.isim=isim;
        self.soyisim=soyisim;
        self.memleket=memleket;
    def calis(self):
        print("calıs methodu")
```

```
class Isci(Calisan):
    def yemek(self):
        print("yemek methodu")
```

```
class Yonetici(Calisan):
    def __init__(self, isim, soyisim, memleket, maas):
        super().__init__(isim, soyisim, memleket,)
        self.memleket=maas;
    def toplanti(self):
        print("toplantı methodu")
```

```
class Muhendis(Calisan):
    def mesai(self):
        print("mesai methodu")
```

Kısaca Kalıtım



Koyu mavi: base class
Açık mavi : child class

- Miras alınan sınıfın bütün nitelik ve metotları alt sınıfa olduğu gibi devredilir.
- Miras alınan sınıfın bazı nitelik ve metotları alt sınıfta yeniden tanımlanır.
- Miras alınan sınıfın bazı nitelik ve metotları alt sınıfta değişikliğe uğratılır.

ÖRNEK:

- Bir çokgen base sınıfından üçgen ve kare alt sınıfları elde edelim.

```
class Cokgen:
    def __init__(self,n):
        self.kenar=n
        self.uzunluk=[]
    def kenar_gir(self):
        for i in range(self.kenar):
            self.uzunluk.append(float(input("{} kenar uzunluğu:".format(i+1))))
    def kenar_goster(self):
        if self.uzunluk==[]:
            print("uzunluk girilmedi")
        else:
            for i in range(self.kenar):
                print("{} kenar uzunluğu:{}".format(i+1,self.uzunluk[i]))
```

```
class Ucgen(Cokgen):
    def __init__(self):
        super().__init__(3)
```

Kare???

```
class Kare(Cokgen):
    def __init__(self):
        super().__init__(4)
    def kenar_gir(self):
        a=float(input("Kenar uzunluğu:"))
        for i in range(self.kenar):
            self.uzunluk.append(a)
```

Dahil etme (composition)

- ▶ Bir sınıftaki nitelik ve metotları başka bir sınıf içinde kullanmanın miras almak dışındaki yolu.
- ▶ Özellikle çok seviyeli miras alma yöntemini kullanmak yerine, dahil etme (*composition*) kullanılabilir
- ▶ Bir sınıf içerisinde farklı bir sınıfa ait nitelik ve metotlar miras alınmak yerine, alt sınıf içine dahil edilir.
- ▶ Kısaca sınıf içerisinde, farklı bir sınıfa ait nesne oluşturulmasıdır.

► Genel tercih yaklaşımı:

- Eğer yazdığımız program, bir başka sınıfın **türevi** ise, o sınıf **miras** alınmalı. ‘olma ilişkisi’ (*is-a relationship*)
- Eğer bir sınıf, yazdığımız programın bir **parçası** ise o sınıf programa **dahil** edilmeli. ‘sahiplik ilişkisi’ (*has-a relationship*)

Kompozisyon oluşturulurken, diğer sınıfların nesnelerini barındıran sınıflara Bileşik sınıflar (**Composite**), daha karmaşık türler oluşturulmak üzere kullanılan sınıflara ise Bileşenler (**Components**) adı verilir.

```

class Mat:
    def __init__(self,x,y):
        self.x=x
        self.y=y
    def topla(self):
        return self.x+self.y
    def cikar(self):
        return self.x-self.y
class Mat2:
    def __init__(self,x,y):
        self.x=x
        self.y=y
    def carp(self):
        return self.x*self.y
    def bol(self):
        return self.x/self.y
class Mat3(Mat,Mat2):
    pass
print(Mat3(2,3).topla())

```



- ▶ Mat3 sınıfı, Mat ve Mat2 sınıflarına SAHİP
- ▶ Miras yerine DAHİL ETME KULLANILMALI

```

class Mat3():
    def __init__(self,x,y):
        self.x=x
        self.y=y
        self.m1=Mat(x,y)
        self.m2=Mat(x,y)
    def us(self):
        return self.x**self.y
    def cikar(self):
        return self.m1.cikar()
    def carp(self):
        return self.m2.carp()

```

```
class Maas:
    def __init__(self, aylik):
        self.aylik = aylik

    def yillik(self):
        return (self.aylik*12)

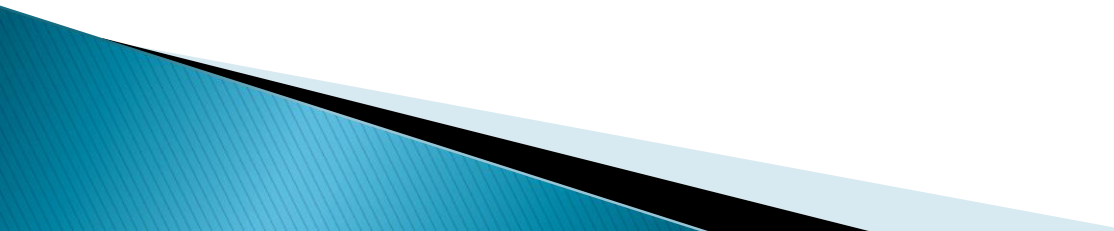
class Calisan:
    def __init__(self, aylik, bonus):
        self.aylik = aylik
        self.bonus = bonus
        self.obj_Maas = Maas(self.aylik)

    def YillikToplam(self):
        return "Toplam: " + str(self.obj_Maas.yillik() + self.bonus)

obj_cal = Calisan(600, 500)
print(obj_cal.YillikToplam())
```



OOP

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

Sarmalama/paketleme (encapsulation)

- ▶ Kapsülleme, metotların ve niteliklerin erişimlerini kısıtlamak anlamına geliyor.
- ▶ Kodların değiştirilmemesi ya da değerlerin kontrollü olarak değiştirilmesi için

```
class Matematik:  
    def __init__(self,a,b,c,d):  
        self.isim = a  
        self.soyisim = b  
        self.vize = c  
        self.final = d
```

__degiskenadi:
Sadece sınıf içinde erişilebilir

```
kayit1 = Matematik("Filiz","Gurkan",67,83)
```

```
print("İsim : ",kayit1.isim)  
print("Soyisim : ",kayit1.soyisim)  
print("Vize : ",kayit1.vize)  
print("Final :",kayit1.final)
```

```
kayit1.vize=51  
print("Vize : ",kayit1.vize)
```

```
class Matematik:
    def __init__(self,a,b,c,d):
        self.isim = a
        self.soyisim = b
        self.__vize = c
        self.__final = d
```

Gizli değişken (Private Variable)

```
kayit1 = Matematik("Filiz","Gurkan",67,83)

print("İsim : ",kayit1.isim)
print("Soyisim : ",kayit1.soyisim)
```

```
class Matematik:
    def __init__(self,a,b,c,d):
        self.isim = a
        self.soyisim = b
        self.__vize = c
        self.__final = d
        self.not_son=0
    def __yilsonu(self):
        self.not_son=self.__vize*0.4+self.__final*0.6
```

ULAŞMAK/DEĞİŞTİRMEK
İSTERSEM??

```
kayit1 = Matematik("Filiz","Gurkan",67,83)

print("İsim : ",kayit1.isim)
print("Soyisim : ",kayit1.soyisim)
kayit1.yilsonu()
print(kayit1.not_son)
```



```

class Matematik:
    def __init__(self,a,b,c,d):
        self.isim=a
        self.soyisim=b
        self.__vize=c
        self.__final=d

kayit1=Matematik("Filiz","Gurkan",67,83)
kayit1.vize=55      #Hata vermez, neden??

print(kayit1.vize)

```

Nesnenin (kayit1) sahip olduğu niteliklere (Attribute) bakarsak, iki farklı vize değişkeni görülür. Nesneye ait asıl değişken, `_Matematik__vize` → gizli olan, değişmiyor. `kayit1.vize=55` ile yeni bir değişken tanımlamış oluyoruz. Asıl değişkeni

DEĞİŞTİREMİYORUZ

▼ kayit1	Matematik	1 Matematik object of __main__ module
> _Matematik__final	int	1 83
> <u>_Matematik__vize</u>	int	1 67
> isim	str	5 Filiz
> soyisim	str	6 Gurkan
> <u>vize</u>	int	1 55

```
class Matematik:
    def __init__(self,a,b,c,d):
        self.isim = a
        self.soyisim = b
        self.__vize = c
        self.__final = d
        self.not_son=0
    def __yilsonu(self):
        self.not_son=self.__vize*0.4+self.__final*0.6
    def notuGoruntulu(self):
        print("Vize:",self.__vize)
        print("Final:",self.__final)
    def VnotuDegistir(self,v):
        self.__vize=v
```

```
kayit1 = Matematik("Filiz","Gurkan",67,83)
```

```
print("İsim : ",kayit1.isim)
print("Soyisim : ",kayit1.soyisim)
kayit1.notuGoruntulu()
kayit1.VnotuDegistir(51)
kayit1.notuGoruntulu()
```

```

class Matematik:
    def __init__(self,a,b,c,d):
        self.isim = a
        self.soyisim = b
        self.__vize = c
        self.__final = d
        self.not_son=self.__yilsonu()
    def __yilsonu(self):
        return self.__vize*0.4+self.__final*0.6
    def notuGoruntulu(self):
        print("Vize:", self._vize)
        print("Final:", self.__final)
    def VnotuDegistir(self,v):
        self._vize=v

```

```

class Matematik:
    def __init__(self,a,b,c,d):
        self.isim = a
        self.soyisim = b
        self.__vize = c
        self.__final = d
        self.not_son=0
        self.__yilsonu()
    def __yilsonu(self):
        self.not_son=self.__vize*0.4+self.__final*0.6
    def notuGoruntulu(self):
        print("Vize:",self._vize)
        print("Final:",self.__final)
    def VnotuDegistir(self,v):
        self._vize=v

```

Python değişkenleri tam anlamıyla korumaz

```
class Matematik:
    def __init__(self,a,b,c,d):
        self.isim = a
        self.soyisim = b
        self.__vize = c
        self.__final = d
        self.not_son=0
        self.__yilsonu()
    def __yilsonu(self):
        self.not_son=self.__vize*0.4+self.__final*0.6
        print("yilsonu metotu")
    def notuGoruntulu(self):
        print("Vize:",self._vize)
        print("Final:",self.__final)
    def VnotuDegistir(self,v):
        self._vize=v
```

```
kayit1 = Matematik("Filiz","Gurkan",67,83)
```

```
print("Vize : ",kayit1._Matematik__vize)
kayit1._Matematik__yilsonu()
```