Nesne Tabanlı Programlama Mantığı

Bu konuyla beraber Nesne Tabanlı Programlamaya giriş yapıyoruz ve bu konuda biraz Nesne Tabanlı programlama hakkında konuşacağız. **Nesne Tabanlı Programlama** veya ingilizce ismiyle **Object Oriented Programming** en basit anlamıyla gerçek hayatı programlamaya uyarlamak olarak düşünülebilir. Örneğin bir tane öğrenci otomasyon sistemi yazmak istiyoruz. Bunun için öğretmenleri , öğrencileri ve kursları aslında birer nesne olarak oluşturmamız gerekiyor. Böyle bir sistemi programlamayla gerçekleştirmek için aslında her bir nesnenin yapısını tanımlayıp, daha sonra bu yapılardan nesneler üretmemiz gerekiyor. İşte **Nesne Tabanlı Programlama** en basit anlamıyla bu şekildedir. Şimdi isterseniz **obje veya nesne** nedir anlamaya çalışalım.

Obje nedir ?

Etrafımıza baktığımızda aslında her bir eşyanın bir obje olduğunu görüyoruz. Örneğin bir tane televizyon kumandasını düşünelim. Bu kumandanın kendi içinde değişik özellikleri (**attribute**) ve fonksiyonları(**metod**) bulunuyor. Örneğin, kumandanın markası, tuşları aslında bu kumandanın özellikleridir(**attribute**). Kumandanın kırmızı tuşuna bastığımızda televizyonun kapanması ve sesi kapatma tuşuna bastığımızda televizyonun sesinin kapanması bu kumandanın metodlarıdır. Bunun gibi Pythondaki aslında her şey bir **objedir.** Örneğin, listelere bakacak olursak bu liste objelerinin aslında birçok metodu ve özelliği bulunur.

In [1]:
```
liste = [1,2,3,4,5]  # Liste objesi oluşturmak
```

In [2]:
```
liste.append(6) # Append metodu

print(liste)
```

```
[1, 2, 3, 4, 5, 6]
```

In [9]:
```
type(liste) # liste objesi
```

Out[9]:
```
list
```

In [5]:
```
sözlük = dict()
```

In [8]:
```
type(sözlük) # dictionary objesi
```

Out[8]:
```
dict
```

In [7]:
```
type((1,2,3,4)) # tuple objesi
```

Out[7]:
```
tuple
```

In [16]:
```
def toplama(a,b):
    return a + b
```

In [18]:
```
type(toplama) # Fonksiyon objesi
```

Out[18]:
```
function
```

e

Nesne Tabanlı Programlama - Sınıflar

Kendi veri tiplerimizi oluşturmak ve bu veri tiplerinden objeler üretmemiz için öncelikle objeleri üreteceğimiz yapıyı tanımlamamız gerekiyor. Bunu tasarladığımız yapıya da **sınıf** veya ingilizce ismiyle **class diyoruz.** Şimdi class yapılarını öğrenerek konumuza başlayalım.

Class Anahtar Kelimesi

Sınıflar veya Classlar objelerimizi oluştururken objelerin özelliklerini ve metodlarını tanımladığımız bir yapıdır ve biz herbir objeyi bu yapıya göre üretiriz. bir Araba **classı** tanımlayarak yapımızı kurmaya başlayalım.

In [2]:
```
#  Yeni bir Araba veri tipi oluşturuyoruz.
class Araba():
    model =  "Renault Megane"
    renk = "Gümüş"            # Sınıfımızın özellikleri (attributes)
    beygir_gücü = 110
    silindir = 4
```

Sınıfımızı Pythonda tanımladık. Peki bu sınıftan obje nasıl oluşturacağız ? Bunu da şu şekilde yapabiliyoruz.

```
obje_ismi = Sınıf_İsmi(parametreler(opsiyonel))
```

In [25]:
```
araba1 =  Araba() # Araba veri tipinden bir "araba1" isminde bir obje oluşturduk.
```

In [5]:
```
araba1 # Objemizin veri tipi Araba
```

Out[5]:
```
<__main__.Araba at 0x8b9f76f860>
```

In [6]:
```
type(araba1)
```

Out[6]:
```
__main__.Araba
```

araba1 objesi artık sınıfta tanımladığımız bütün özelliklere (attributes) sahip olmuş oldu. İşte sınıf ve obje üretmek bu şekilde olmaktadır. Peki bu araba objesinin özelliklerinin nasıl görebiliriz ?

```
obje_ismi.özellik_ismi
```

In [13]:
```
araba1.model
```

Out[13]:
```
'Renault Megane'
```

In [14]:
```
araba1.renk
```

Out[14]:
```
'Gümüş'
```

a/Ne

In [15]:
```
araba1.beygir_gücü
```

Out[15]:
```
110
```

In [16]:
```
araba1.silindir
```

Out[16]:
```
4
```

Şimdi de başka bir **Araba** objesi oluşturalım.

In [17]:
```
araba2 =  Araba()
```

In [18]:
```
araba2.model
```

Out[18]:
```
'Renault Megane'
```

In [19]:
```
araba2.renk
```

Out[19]:
```
'Gümüş'
```

In [20]:
```
araba2.beygir_gücü
```

Out[20]:
```
110
```

In [21]:
```
araba2.silindir
```

Out[21]:
```
4
```

Burda gördüğümüz gibi oluşturduğumuz **objelerin** buradaki model,renk vs. gibi özelliklerinin değeri **aynıdır.** Çünkü aslında burada tanımladığımız **özellikler** birer sınıf **özelliğidir.** Yani biz bir obje oluşturduğumuzda bu özelliklerin değerleri varsayılan olarak gelir. Bu özelliklerin değerlerine , herhangi bir **obje oluşturmadan** da erişebiliriz. Bunu da şu şekilde yapabiliriz.

In [30]:
```
Araba.renk # Class_İsmi.özellik_ismi
```

Out[30]:
```
'Gümüş'
```

In [31]:
```
Araba.beygir_gücü
```

Out[31]:
```
110
```

Bizim her bir objeyi başlangıçta farklı değerlerle oluşturmamız için her bir objeyi oluştururken objelerin değerlerini göndermemiz gerekiyor. Bunun için de özel bir metodu kullanmamız gerekmektedir.

```
__init__()
```

Peki bu metod ne anlama geliyor ? İsterseniz ilk olarak ***dir()*** fonksiyonu yardımıyla **araba1** objemizde neler var bakalım.

In [32]:
```
dir(araba1)
```

Out[32]:
```
['__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__',
 'beygir_gücü',
 'model',
 'renk',
 'silindir']
```

Burada objemizin tüm özelliklerini ve metodlarını görüyoruz. Ancak biz herhangi bir metod tanımlamamıştır. Buradaki metodlar Python tarafından bir obje oluşturulduğunda özel olarak tanımlanan metodlardır ve biz eğer özel olarak bunları tanımlamazsak Python kendisi bunları varsayılan tanımlıyor. Burada aynı zamanda **init** metodunu da görüyoruz. Eğer biz bu metodu kendimiz tanımlarsak objelerimizi farklı değerlerle başlatabiliriz.

Aslında **init** metodu Pythonda **yapıcı(constructor) fonksiyon** olarak tanımlanmaktadır. Bu metod **objelerimiz oluşturulurken otomatik olarak ilk çağrılan fonksiyondur.** Bu metodu özel olarak tanımlayarak objelerimizi farklı değerlerle oluşturabiliriz.

Peki bu metodu nasıl tanımlayacağız ? Direk örnek üzerinden görmeye çalışalım.

In [33]:
```
# Araba Veri tipi

class Araba():
    # Şimdilik Class özelliklerine ihtiyacımız yok.

    def __init__(self):
        print("init fonksiyonu çağrıldı.")
```

In [34]:
```
araba1 = Araba() # araba1 objesi oluşurken otomatik olarak __init__ metodumuz çağrılıyor.
```

```
init fonksiyonu çağrıldı.
```

Peki burada **self** ne anlama geliyor ? **self** anahtar kelimesi objeyi oluşturduğumuz zaman o objeyi gösteren bir referanstır ve metodlarımızda en başta bulunması gereken bir parametredir. Yani biz bir objenin bütün özelliklerini ve metodlarını bu referans üzerinden kullanabiliriz.

Objeler oluşturulurken, Python bu referansı metodlara otomatik olarak kendisi gönderir. Özel olarak self referansını göndermemize gerek yoktur.

init metodunu ve self'i iyi anlamak için objelerimize özellikler ekleyelim.

In [36]:
```
class Araba():

    def __init__(self,model,renk,beygir_gücü,silindir): # Parametrelerimizin değerlerini objelerimizi
        self.model =  model # self.özellik_ismi = parametre değeri şeklinde objemizin model özelliğin
        self.renk = renk # self.özellik_ismi = parametre değeri şeklinde objemizin renk özelliğine de
        self.beygir_gücü = beygir_gücü # self.özellik_ismi = parametre değeri şeklinde objemizin beyg
        self.silindir = silindir # self.özellik_ismi = parametre değeri şeklinde objemizin silndir öz
```

In [45]:
```
# araba1 objesini oluşturalım.
# Artık değerlerimizi göndererek objelerimizin özelliklerini istediğimiz değerle başlatabiliriz.
araba1 = Araba("Peugeot 301","Beyaz",90,4)
```

In [46]:
```
# araba2 objesini oluşturalım.
araba2 = Araba("Renault Megane","Gümüş",110,4)
```

In [47]:
```
araba1.model
```

Out[47]:
```
'Peugeot 301'
```

In [48]:
```
araba1.renk
```

Out[48]:
```
'Beyaz'
```

In [49]:
```
araba2.model
```

Out[49]:
```
'Renault Megane'
```

In [50]:
```
araba2.renk
```

Out[50]:
```
'Gümüş'
```

İstersek init metodunu varsayılan değerlerle de yazabiliriz.

In [51]:
```
class Araba():

    def __init__(self , model = "Bilgi Yok",renk = "Bilgi Yok",beygir_gücü = 75 ,silindir = 4):
        self.model =  model
        self.renk = renk
        self.beygir_gücü = beygir_gücü
        self.silindir = silindir
```

In [52]:
```
araba1 = Araba(beygir_gücü = 85, renk = "Siyah")
```

In [53]:
```
araba1.renk
```

Out[53]:
```
'Siyah'
```

In [54]:
```
araba1.model
```

Out[54]:
```
'Bilgi Yok'
```

İşte burada gördüğümüz gibi bir objeyi **init** metodunu kendimiz yazarak farklı değerlerle oluşturabiliyoruz

Nesne Tabanlı Programlama - Metodlar

ilk olarak bir **Yazılımcı** sınıfı tanımlayalım.

In [1]:
```
class Yazılımcı():

    def __init__(self,isim,soyisim,numara,maaş,diller):
        self.isim = isim
        self.soyisim = soyisim
        self.numara = numara    # Yazılımcı objelerinin özellikleri
        self.maaş = maaş
        self.diller = diller
```

In [2]:
```
# yazılımcı1 objesi
yazılımcı1 =  Yazılımcı("             ","      ",12345,3000,["Python","C","Java"])
```

In [3]:
```
yazılımcı2 = Yazılımcı("Serhat","Say",23456,3500,["Matlab","R","SmallTalk"])
```

In [5]:
```
yazılımcı1.diller
```

Out[5]:
```
['Python', 'C', 'Java']
```

In [6]:
```
yazılımcı2.soyisim
```

Out[6]:
```
'Say'
```

bu class'a metodlarımızı nasıl tanımlayabiliriz ? Aynı **init metodunu** tanımladığımız gibi bir class'a da istediğimiz kadar metod tanımlayabiliriz. Örneğin ,**Yazılımcı** classına **bilgilerigöster** isimli bir metod tanımlayalım.

In [35]:
```
class Yazılımcı():

    def __init__(self,isim,soyisim,numara,maaş,diller):
        self.isim = isim
        self.soyisim = soyisim
        self.numara = numara    # Yazılımcı objelerinin özellikleri
        self.maaş = maaş
        self.diller = diller
    def bilgilerigöster(self):
        print("""
        Çalışan Bilgisi:

        İsim :  {}

        Soyisim : {}

        Şirket Numarası: {}

        Maaş :  {}

        Diller: {}
        """.format(self.isim,self.soyisim,self.numara,self.maaş,self.diller))
```

In [36]:
```
yazılımcı1 =  Yazılımcı("             ","      ",12345,3000,["Python","C","Java"])
```

In [37]:
```
yazılımcı1.bilgilerigöster()
```

```
        Çalışan Bilgisi:

        İsim :

        Soyisim :

        Şirket Numarası: 12345

        Maaş :  3000

        Diller: ['Python', 'C', 'Java']
```

Burada **bilgilerigöster** isimli metod tanımlayarak her bir özelliğimizin değeri ekrana derli toplu bir şekilde yazdırmış olduk. *Metodlarımızı yazarken dikkat etmemiz gerek nokta, her metodun birinci parametresinin self referansı olması gerektiğidir. Ayrıca objelerimizin özelliklerine mutlaka self referansıyla erişmemiz gerekiyor.* İsterseniz bu classa 2 tane daha metod yazalım.

In [38]:
```
class Yazılımcı():

    def __init__(self,isim,soyisim,numara,maaş,diller):
        self.isim = isim
        self.soyisim = soyisim
        self.numara = numara    # Yazılımcı objelerinin özellikleri
        self.maaş = maaş
        self.diller = diller
    def bilgilerigöster(self):
        print("""
        Çalışan Bilgisi:

        İsim :  {}

        Soyisim : {}

        Şirket Numarası: {}

        Maaş :  {}

        Diller: {}
        """.format(self.isim,self.soyisim,self.numara,self.maaş,self.diller))
    def dil_ekle(self,yeni_dil):
        print("Dil ekleniyor.")
        self.diller.append(yeni_dil)
    def maas_yukselt(self,zam_miktarı):
        print("Maaş yükseliyor...")

        self.maaş += 250
```

In [39]:
```
yazılımcı1 =  Yazılımcı(              ","      ",12345,3000,["Python","C","Java"])
```

In [41]:
```
yazılımcı1.bilgilerigöster()
```

```
        Çalışan Bilgisi:

        İsim :

        Soyisim :

        Şirket Numarası: 12345

        Maaş :  3000

        Diller: ['Python', 'C', 'Java']
```

In [42]:
```
yazılımcı1.maas_yukselt(500)
```

```
Maaş yükseliyor...
```

In [43]:
```
yazılımcı1.bilgilerigöster()
```

```
        Çalışan Bilgisi:

        İsim :

        Soyisim :

        Şirket Numarası: 12345

        Maaş :  3250

        Diller: ['Python', 'C', 'Java']
```

In [44]:
```
yazılımcı1.dil_ekle("Javascript")
```

```
Dil ekleniyor.
```

In [45]:
```
yazılımcı1.bilgilerigöster()
```

```
        Çalışan Bilgisi:

        İsim :

        Soyisim :

        Şirket Numarası: 12345

        Maaş :  3250

        Diller: ['Python', 'C', 'Java', 'Javascript']
```

Nesne Tabanlı Programlama - Kalıtım (Inheritance)

Inheritance veya kalıtım bir sınıfın başka bir sınıftan özelliklerini(**attribute**) ve metodlarını miras almasıdır.

Peki inheritance nerede işimize yarar ? Örneğin, bir şirketin çalışanlarını tasarlamak için sınıflar oluşturuyoruz. Bunun için Yönetici, Proje Direktörü, İşçi gibi sınıflar oluşturmamız gerekiyor. Aslında baktığımız zaman bu sınıfların hepsinin belli ortak metodları ve özellikleri bulunuyor. O zaman bu ortak özellikleri ve metodları tekrar tekrar bu sınıfların içinde tanımlamak yerine, bir tane ana class tanımlayıp bu classların bu classın özelliklerini ve metodlarını almalarını sağlayabiliriz.
Inheritance'ın veya Kalıtım'ın temel mantığı budur.

İsterseniz inheritance yapısını kurmak için öncelikle bir tane çalışan sınıfı oluşturalım.

In [63]:
```
class Çalışan():
    def __init__(self,isim,maaş,departman):
        print("Çalışan sınıfının init fonksiyonu")
        self.isim = isim
        self.maaş = maaş
        self.departman = departman
    def bilgilerigoster(self):

        print("Çalışan sınıfının bilgileri.....")

        print("İsim : {} \nMaaş: {} \nDepartman: {}\n".format(self.isim,self.maaş,self.departman))
    def departman_degistir(self,yeni_departman):
        print("Departman değişiyor....")
        self.departman = yeni_departman
```

Çalışan sınıfını oluşturduk şimdi de Yönetici sınıfını bu Çalışan sınıfından türetmeye çalışalım.

In [64]:
```
class Yönetici(Çalışan): # Çalışan sınıfından miras alıyoruz.
    pass # Pass Deyimi bir bloğu sonradan tanımlamak istediğimizde kullanılan bir deyimdir.
```

Burada, yönetici sınıfında herhangi bir şey tanımlamadık ancak Çalışan sınıfından bütün özellikleri ve metodları miras aldık. Bakalım burada Çalışan sınıfının metodlarını kullanabilecek miyiz ?

In [65]:
```
yönetici1 = Yönetici("Mehmet Baltacı",3000,"İnsan Kaynakları") # yönetici objesi
```

```
Çalışan sınıfının init fonksiyonu
```

In [66]:
```
yönetici1.bilgilerigoster()
```

```
Çalışan sınıfının bilgileri.....
İsim : Mehmet Baltacı
Maaş: 3000
Departman: İnsan Kaynakları
```

In [67]:
```
yönetici1.departman_degistir("Halkla İlişkiler")
```

```
Departman değişiyor....
```

In [68]:
```
yönetici1.bilgilerigoster()
```

```
Çalışan sınıfının bilgileri.....
İsim : Mehmet Baltacı
Maaş: 3000
Departman: Halkla İlişkiler
```

Burada gördüğümüz gibi bütün özellikleri ve metodları Çalışan sınıfından miras aldığımız için kullanabiliyoruz. Bunu ***dir()*** fonksiyonu ile de görebiliriz.

In [69]:
```
dir(yönetici1)
```

Out[69]:
```
['__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__',
 'bilgilerigoster',
 'departman',
 'departman_degistir',
 'isim',
 'maaş']
```

Peki biz Yönetici sınıfına ekstra fonksiyonlar ve özellikler ekleyebiliyor muyuz ? Örnek olması açısından **zam_yap** isimli bir metod ekleyelim.

In [73]:
```
class Yönetici(Çalışan):
    def zam_yap(self,zam_miktarı):
        print("Maaşa zam yapılıyor....")
        self.maaş += zam_miktarı
```

In [74]:
```
yönetici2 = Yönetici("                    ",3000,"Bilişim") # yönetici objesi
```

```
Çalışan sınıfının init fonksiyonu
```

In [75]:
```
yönetici2.bilgilerigoster()
```

```
Çalışan sınıfının bilgileri.....
İsim :
Maaş: 3000
Departman: Bilişim
```

In [76]:
```
yönetici2.zam_yap(500) # Ekstra eklediğimiz fonksiyonu da kullanabiliyoruz.
```

```
Maaşa zam yapılıyor....
```

In [77]:
```
yönetici2.bilgilerigoster()
```

```
Çalışa                        ...
İsim :
Maaş: 3500
Departman: Bilişim
```

İşte biz bir sınıftan miras alarak oluşturduğumuz sınıflara ekstra metodlar ve özellikler de ekleyebiliyoruz.

I

Nesne Tabanlı Programlama - Özel Metodlar

Nesne tabanlı programlamada son olarak sınıfların özel metodlarını nasıl kendimiz yazarız öğrenmeye çalışalım. Özel metodlar, daha önceden de bahsettiğimiz gibi bizim özel olarak çağırmadığımız ancak her classa ait metodlardır. Bunların **çoğu** biz tanımlamasak bile Python tarafından varsayılan olarak tanımlanır. Ancak bu metodların bazılarını da **özel olarak bizim tanımlamamız** gerekmektedir. Daha önceden gördüğümüz **init** metodu bu metodlara bir örnektir.

In [64]:
```
class Kitap():
    pass
```

In [65]:
```
kitap1 = Kitap() # __init__ metodu çağrılıyor.
```

In [66]:
```
len(kitap1) # __len__ metodu çağrılacak ancak tanımlı değil. Bunu özellikle bizim tanımlamamız gerekiy
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-66-5f5391921a08> in <module>()
----> 1 len(kitap1) # __len__ metodu çağrılacak ancak tanımlı değil. Bunu özellikle bizim tanımlam

TypeError: object of type 'Kitap' has no len()
```

In [67]:
```
print(kitap1) # __str__ metodu çağrılır.
```

```
<__main__.Kitap object at 0x000000CEE4E93390>
```

In [68]:
```
del kitap1 # del anahtar kelimesi bir objeyi siler ve __del__ metodu çağrılır.
```

In [69]:
```
kitap1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-69-361ccf2f9d6d> in <module>()
----> 1 kitap1

NameError: name 'kitap1' is not defined
```

Şimdi buradaki metodları kendimiz nasıl tanımlayacağız öğrenmeye çalışalım.

init metodu

init metodunu kendimiz tanımlarsak artık kendi init fonksiyonumuz çalışacaktır.

In [71]:
```
class Kitap():
    def __init__(self,isim,yazar,sayfa_sayısı,tür):
        print("Kitap Objesi oluşuyor....")
        self.isim = isim
        self.yazar = yazar
        self.sayfa_sayısı = sayfa_sayısı
        self.tür = tür
```

In [72]:
```
kitap1 = Kitap("İstanbul Hatırası","Ahmet Ümit",561,"Polisiye") # Kendi metodumuz
```

```
Kitap Objesi oluşuyor....
```

str metodu

Normalde **print(kitap1)** ifadesi ekrana şöyle bir yazı yazdırıyor.

In [73]:
```
print(kitap1)
```

```
<__main__.Kitap object at 0x000000CEE886EAC8>
```

Ancak eğer **str** metodunu kendimiz tanımlarsak artık ekrana **kitap1** in içeriğini daha anlaşılır yazabileceğiz.

In [74]:
```
class Kitap():
    def __init__(self,isim,yazar,sayfa_sayısı,tür):
        print("Kitap Objesi oluşuyor....")
        self.isim = isim
        self.yazar = yazar
        self.sayfa_sayısı = sayfa_sayısı
        self.tür = tür
    def __str__(self):
        # Return kullanmamız gerekli
        return "İsim: {}\nYazar: {}\nSayfa Sayısı: {}\nTür: {}".format(self.isim,self.yazar,self.sayf
```

In [75]:
```
kitap1 = Kitap("İstanbul Hatırası","Ahmet Ümit",561,"Polisiye")
```

```
Kitap Objesi oluşuyor....
```

In [76]:
```
print(kitap1)
```

```
İsim: İstanbul Hatırası
Yazar: Ahmet Ümit
Sayfa Sayısı: 561
Tür: Polisiye
```

len metodu

len metodu normalde özel olarak biz tanımlamazsak tanımlanan bir metod değil. Onun için bu metodu kendimiz tanımlamamız gereklidir.

In [77]:
```
len(kitap1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-77-ab6b326ab715> in <module>()
----> 1 len(kitap1)

TypeError: object of type 'Kitap' has no len()
```

In [81]:
```
class Kitap():
    def __init__(self,isim,yazar,sayfa_sayısı,tür):
        print("Kitap Objesi oluşuyor....")
        self.isim = isim
        self.yazar = yazar
        self.sayfa_sayısı = sayfa_sayısı
        self.tür = tür
    def __str__(self):
        # Return kullanmamız gerekli
        return "İsim: {}\nYazar: {}\nSayfa Sayısı: {}\nTür: {}".format(self.isim,self.yazar,self.sayf
    def __len__(self):
        return self.sayfa_sayısı
```

In [82]:
```
kitap1 = Kitap("İstanbul Hatırası","Ahmet Ümit",561,"Polisiye")
```

```
Kitap Objesi oluşuyor....
```

In [84]:
```
len(kitap1) # KEndi __len__ metodumuz çağrıldı.
```

Out[84]:
```
561
```

del metodu

del metodu Pythonda bir objeyi **del** anahtar kelimesiyle sildiğimiz zaman çalıştırılan metoddur. Bu metodu kendimiz tanımlayarak ekstra özellikler ekleyebiliriz.

In [85]:
```
kitap1 = Kitap("İstanbul Hatırası","Ahmet Ümit",561,"Polisiye")
```

```
Kitap Objesi oluşuyor....
```

In [86]:
```
del kitap1
```

In [87]:
```
kitap1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-87-361ccf2f9d6d> in <module>()
----> 1 kitap1

NameError: name 'kitap1' is not defined
```

In [88]:
```
class Kitap():
    def __init__(self,isim,yazar,sayfa_sayısı,tür):
        print("Kitap Objesi oluşuyor....")
        self.isim = isim
        self.yazar = yazar
        self.sayfa_sayısı = sayfa_sayısı
        self.tür = tür
    def __str__(self):
        # Return kullanmamız gerekli
        return "İsim: {}\nYazar: {}\nSayfa Sayısı: {}\nTür: {}".format(self.isim,self.yazar,self.sayf
    def __len__(self):
        return self.sayfa_sayısı
    def __del__(self):
        print("Kitap objesi siliniyor.......")
```

In [94]:
```
kitap1 = Kitap("İstanbul Hatırası","Ahmet Ümit",561,"Polisiye")
```

```
Kitap Objesi oluşuyor....
```

In [95]:
```
del kitap1  # Ekstra ekrana yazdırma özelliği ekledik.
```

```
Kitap objesi siliniyor.......
```

In [96]:
```
kitap1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-96-361ccf2f9d6d> in <module>()
----> 1 kitap1

NameError: name 'kitap1' is not defined
```

Siz de bunlar gibi çoğu özel metodu ihtiyacınız olduğu zaman kendiniz yazabilirsiniz. Özel metodlar için güzel bir ingilizce kaynak için şuradan faydalanabilirsiniz.

http://www.diveintopython3.net/special-method-names.html (http://www.diveintopython3.net/special-method-names.html)