Biz, sınıfımızdan yaratmış olduğumuz örneklemlere nitelik atama işlemini **__init__** metodu ile gerçekleştiriyorduk. Bu nitelikler aynı zamanda örneklemin değişkenleridir. Örnekleme ait değişkenler ise isim, soyisim,maas ve departmandır. Bu bilgilere **Instance Veriables** yani **Örneklem Değişkenleri** denilmektedir. Sınıf içindeki diğer değişkenlerimize ise **Class Veriables** yani **Sınıf Değişkenleri** denilmektedir. Örneklem değişkenleri sadece örneklemi ilgilendirirken sınıf değişkenleri sınıfın bütününü ilgilendirmektedir.
Bu dersimizde **Class Veriables** (Sınıf Değişkenleri), **@classmethod** (Sınıf Metodu) ve **@staticmethod** (Statik Metot) kavramlarını anlamaya çalışacağız. Örneğimizi incelersek:

Calisanlar() adında bir sınıf oluşturduk ve isim, soyisim, maas ve departman olarak dört tane öznitelik atadık. Daha sonra **bilgileri_kaydet()** adında bir metot oluşturduk ve bu metodun işlevini örneklemlere ait bilgileri **calisanlar_listesi** adında bir listeye eklemesini sağladık. Daha sonra **bilgileri_goster()** adında bir metot oluşturduk ve bu metodun işlevini **calisanlar_listesi** adındaki listeyi ekrana bastırmasını sağladık. Son olarak **calisan_1** adında bir örneklem yarattık ve **calisan_1.bilgileri_goster()** diyerek bu örnekleme ait bilgileri ekrana yazdırdık.

```
class Calisanlar():

    calisanlar_listesi = []

    def __init__(self, isim, soyisim, maas, departman):

        self.isim      = isim

        self.soyisim    = soyisim

        self.maas      = maas

        self.departman  = departman

        self.bilgileri_kaydet()

    def bilgileri_kaydet(self):

        self.calisanlar_listesi.append(self.isim)

        self.calisanlar_listesi.append(self.soyisim)

        self.calisanlar_listesi.append(self.maas)

        self.calisanlar_listesi.append(self.departman)

    def bilgileri_goster(self): //örneklem

        print(self.calisanlar_listesi)

calisan_1 = Calisanlar("Mert", "Kaya", 6500, "Yazılım Geliştirme")

calisan_1.bilgileri_goster()

['Mert', 'Kaya', 6500, 'Bilgi İşlem']
```

Sınıfımızda tanımlamış olduğumuz calisanlar_listesi sadece bu sınıftan yaratılan örneklemleri ilgilendirmemektedir. Yazacağımız başka metotlarda da bu listeyi kullanarak bu listeden faydalanabiliriz. Peki, bu liste bütün sınıfı ilgilendirdiğine göre Calisanlar() sınıfı üzerinden bu listeye erişmek istersek nasıl bir sonuç alırız buna bakalım.

Aşağıdaki gibi bilgileri_goster() metoduna sınıfımız (Calisanlar) üzerinden ulaşmak istediğimizde bir hata almaktayız. Bu hatanın sebebi ise bilgileri_goster() metodunun bir sınıf metodu değil örneklem metodu olmasından dolayıdır. Bu sebepten dolayı calisan_1 örneklemi üzerinden bu metodu çalıştırabilirken Calisanlar() sınıfı üzerinden çalıştıramamaktayız.

```
Calisanlar.bilgileri_goster()

>> Calisanlar.bilgileri_goster()

  TypeError: bilgileri_goster() missing 1 required positional argument: 'self'
```

Bu hatayı almamak için @classmethod dekoratörünü kullanmamız gerekmektedir. @classmethod dekoratörü ile oluşturacağımız her bir metot sınıfın tümünü etkilemektedir. Sınıf metotlarına hem sınıfımızın adından hem de örneklemlerimizin adından erişebiliriz. Sınıf metodunda yapılacak bir değişiklik, sınıf içinde bu metodu kullanacak olan her örneklem için de geçerli olacaktır.

Bir metodu sınıf metodu yapabilmemiz için metodun başına @classmethod yazmamız yeterli olacaktır. Örneğimizi aşağıdaki gibi güncelleyebiliriz.

```
class Calisanlar():

   calisanlar_listesi = []

   def __init__(self, isim, soyisim, maas, departman):

      self.isim      = isim

      self.soyisim    = soyisim

      self.maas      = maas

      self.departman  = departman

      self.bilgileri_kaydet()

   def bilgileri_kaydet(self):

      self.calisanlar_listesi.append(self.isim)

      self.calisanlar_listesi.append(self.soyisim)

      self.calisanlar_listesi.append(self.maas)

      self.calisanlar_listesi.append(self.departman)

   @classmethod

   def bilgileri_goster(cls):

      print(cls.calisanlar_listesi)

calisan_1 = Calisanlar("Mert", "Kaya", 6500, "Bilgi İşlem")

calisan_1.bilgileri_goster()

Calisanlar.bilgileri_goster()

>> ['Mert', 'Kaya', 6500, 'Bilgi İşlem']

  ['Mert', 'Kaya', 6500, 'Bilgi İşlem']
```

bilgileri_goster() metodunu incelersek hemen üstünde @classmethod dekoratörünü görebiliriz. Ayrıca metodumuzun parantez içine baktığmızda self yerine cls anahtar sözcüğünü görmekteyiz. Bunun nedeni; nasıl ki örneklem metotlarında niteliklerimize veya değişkenlerimize ulaşmak için self kullanıyorsak, sınıf metotlarında tanımlanan her değer için cls (class) anahtar sözcüğünü kullanmaktayız.Biz, hem sınıf adı üzerinden hem de örneklem adı üzerinden ulaşabileceğimiz bir metot yazsak ve bu metodun sınıf içinde herhangi bir değişikliğe neden olmasını istemesek ama aynı zamanda sınıfa ait olmasını istersek bu işlemi nasıl yapabiliriz?Bu işlemi yapabilmek için @staticmethod dekoratörünü kullanabiliriz. Bu dekoratöre de tıpkı @classmethod gibi hem sınıf adıyla hem de örneklem adıyla ulaşabiliriz. Statik metodlarda parantez içinde herhangi bir anahtar sözcük (self ya da cls) kullanılmamaktadır.Aşağıdaki metodu incelediğimizde calisanlara prim vermek için kullanılabilecek sabit bir değer belirleyip bu metodun döndürmüş olduğu 0.12 değerini istediğimiz şekilde kullanabiliriz. Örnek olarak her ay çalışanlarımıza maaşlarına ek olarak prim ekleyebiliriz.

```
@staticmethod

def prim_katsayisi():

   return 0.12
```