Nesne Tabanlı Programlama (Object-Oriented Programming -OOP)

Nesne yönelimli programlama, özelliklerin ve davranışların ayrı ayrı nesnelerde toplanacağı şekilde programları yapılandırmanın bir yolunu sağlayan bir programlama paradigmasıdır.Başka bir deyişle, nesne yönelimli programlama, arabalar gibi somut, gerçek dünyadaki şeylerin yanı sıra şirketler ve çalışanlar veya öğrenciler ve öğretmenler gibi şeyler arasındaki ilişkileri modellemeye yönelik bir yaklaşımdır. OOP, gerçek dünya varlıklarını, kendileriyle ilişkili bazı verilere sahip olan ve belirli işlemleri gerçekleştirebilen yazılım nesneleri olarak modeller.

In [1]:
```
kirk = ["James Kirk", 34, "Captain", 2265]
spock = ["Spock", 35, "Science Officer", 2254]
mccoy = ["Leonard McCoy", "Chief Medical Officer", 2266]
```

Örneğin, bir kuruluştaki çalışanları izlemek isteyebilirsiniz. Her çalışan hakkında adı, yaşı, pozisyonu ve işe başladığı yıl gibi bazı temel bilgileri saklamanız gerekir.

Bunu yapmanın bir yolu, her çalışanın bilgilerini bir liste halinde temsil etmektir:

Bu yaklaşımla ilgili bir takım sorunlar var.

Birincisi, daha büyük kod dosyalarının yönetilmesini zorlaştırabilir. Kirk listesini bildirdiğiniz yerden birkaç satır uzakta kirk[0]'a başvurursanız, 0 indeksli öğenin çalışanın adı olduğunu hatırlayacak mısınız?

İkincisi, çalışanların kendi listelerinde aynı sayıda öğeye sahip olmaması hatalara neden olabilir. Yukarıdaki mccoy listesinde yaş eksik olduğundan mccoy[1], Dr. McCoy'un yaşı yerine "Baş Sağlık Memuru" ifadesini döndürecektir.

Bu tür kodları daha yönetilebilir ve bakımı daha kolay hale getirmenin harika bir yolu sınıfları kullanmaktır.

Python'da Bir Sınıfı (class) Nasıl Tanımlarsınız?

Sınıflar(classes), kullanıcı tanımlı veri yapıları oluşturmanıza olanak tanır. Sınıflar, sınıftan oluşturulan bir nesnenin kendi verileriyle gerçekleştirebileceği davranışları ve eylemleri tanımlayan, yöntemler (methods) adı verilen işlevleri tanımlar.

Python'da, class anahtar sözcüğünü ve ardından bir adı ve iki nokta üst üste işaretini kullanarak bir sınıfı tanımlarsınız. Daha sonra sınıfın her örneğinin hangi niteliklere sahip olması gerektiğini bildirmek için .__init__() işlevini kullanırsınız:

In [2]:
```
class çalışan:
    def __init__(self,isim,soy_isim,yaş,bölüm):
        self.isim=isim
        self.soy_isim=soy_isim
        self.yaş=yaş
        self.bölüm=bölüm
```

The init() Function

Tüm çalışan nesnelerinin sahip olması gereken özellikleri .__init__() adı verilen bir yöntemde tanımlarız. Her yeni Dog nesnesi oluşturduğunuzda, .__init__() nesnenin özelliklerinin değerlerini atayarak nesnenin başlangıç ••durumunu ayarlar. Yani, .__init__() sınıfın her yeni örneğini başlatır.

.__init__()'de oluşturulan niteliklere örnek nitelikler (instance attributes) denir. Bir örnek niteliğinin değeri, sınıfın belirli bir örneğine özeldir. Tüm çalışan nesnelerinin bir adı ve yaşı vardır, ancak ad ve yaş niteliklerinin değerleri çalışan örneğine bağlı olarak değişecektir.

The self Parameter

Self parametresi sınıfın geçerli örneğine bir referanstır ve sınıfa ait değişkenlere erişmek için kullanılır.

```
3 Self olarak adlandırılmasına gerek yoktur, onu istediğiniz gibi adlandırabilirsiniz, ancak sınıftaki herhangi
  bir işlevin ilk parametresi olmalıdır:
```

In [3]:
```
1 işçi_1 = çalışan('Ali','çokçalışkan',25,'muhasebe')
```

In [4]:
```
1 işçi_2 = çalışan('Veli','Tembel',25,'güvenlik')
```

In [5]:
```
1 işçi_1.isim
```

Out[5]: 'Ali'

In [6]:
```
1 işçi_1.soy_isim
```

Out[6]: 'çokçalışkan'

In [7]:
```
1 işçi_2.isim
```

Out[7]: 'Veli'

In [8]:
```
1 işçi_2.bölüm
```

Out[8]: 'güvenlik'

In [9]:
```
1 class MyClass:
2     def __init__(self):
3
```

```
  File "C:\Users\mbenturk\AppData\Local\Temp\ipykernel_2160\863528696.py", line 3

    ^
IndentationError: expected an indented block
```

PASS

In [ ]:
```
class MyClass:
    def __init__(self,ders_ismi,gün,saat,sınav_tarihi):
        self.ders_ismi=ders_ismi
        self.gün=gün
        self.saat=saat
        def sınav_tarihi(self,sınav_türü):

```

In [ ]:
```
class MyClass:
    def __init__(self,ders_ismi,gün,saat,sınav_tarihi):
        self.ders_ismi=ders_ismi
        self.gün=gün
        self.saat=saat
        def sınav_tarihi(self,sınav_türü):
            pass
```

In [ ]:
```
x_1 =MyClass('matematik','perşembe','13:00','vize')
```

Myclass sınıfının sınav tarihi alt elementi bir fonksiyon olarak tanımlanmıştır ama fonksiyon eksik olarak yazılmıştır. Kodun doğru çalışması için sınav tarihi fonsiyonun tamamlanmasını gerektirmektedir. Sınan tarihi fonksiyonu tamamlanmadan önce çalıştırmanın tek yolu pass ifadesini kullanmaktadır.

In [ ]:
```
x_1.ders_ismi
```

In [ ]:
```
x_1.sınav_tarihi
```

In [ ]:
```

```

In [ ]:
```python
# dog.py

class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

In [ ]:
```python
# dog.py

class Dog:
    species = "Canis familiaris"

    def __init__(self, name, age):
        self.name = name
        self.age = age
```

In [ ]:
```python
miles = Dog("Miles", 4)
buddy = Dog("Buddy", 9)
```

In [ ]:
```python
print(miles.name,miles.age)
print(buddy.name,buddy.age)
```

In [ ]:
```python
miles.species
```

In [ ]:
```python
buddy.species
```

In [ ]:
```python
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
```

In [ ]:
```
p1 = Person("John", 36)
print(p1.name)
print(p1.age)
```

Değişkenleri Göstermek

In [ ]:
```
vars(miles)
```

In [ ]:
```
vars(buddy)
```

In [ ]:
```
vars(p1)
```

In [ ]:
```

```

Niteliklerin (attributes) var olduğu garanti edilse de değerleri dinamik olarak değişebilir:

In [ ]:
```
print(buddy.age)
```

In [ ]:
```
buddy.age=10
```

In [ ]:
```
print(buddy.age)
```

In [ ]:
```
print(miles.species)
```

In [ ]:
```
miles.species = "Felis silvestris"
```

In [ ]:
```
print(miles.species)
```

The str() Function

```
__str__() işlevi, sınıf nesnesi bir dize olarak temsil edildiğinde neyin çıktı olarak verilmesi gerektiğini
kontrol eder.


```

In [ ]:
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def __str__(self):
    return ('Kişi ismi {} ve  yaşı {} dir.'.format(self.name,self.age))
```

In [ ]:
```
p1 = Person("John", 36)

print(p1)
```

Örnek yöntemleri(Instance Methods)

```
Örnek yöntemleri(instance methods), bir sınıf içinde tanımladığınız ve yalnızca o sınıfın bir örneğini
çağırabileceğiniz işlevlerdir. Tıpkı .__init__() gibi, bir örnek yöntem de her zaman self'i ilk parametresi
olarak alır.
```

In [ ]:
```python
# dog.py

class Dog:
    species = "Canis familiaris"

    def __init__(self, name, age):
        self.name = name
        self.age = age

    # Instance method
    def description(self):
        return ('Köpek ismi {} ve  yaşı {} dir.'.format(self.name,self.age))


    # Another instance method
    def speak(self, sound):
        return ('Köpek ismi {} ve  sesi {} dir.'.format(self.name,sound))

```

```
1.description(), köpeğin adını ve yaşını görüntüleyen bir dize döndürür.
2.speak()'in ses adı verilen bir parametresi vardır ve köpeğin adını ve köpeğin çıkardığı sesi içeren bir dize döndürür.
```

In [ ]:
```python
miles = Dog("Miles", 4)
print(miles.description())

```

In [ ]:
```python
miles.speak("Bow Wow")
print(miles.speak("Woof Woof"))
```

In [ ]:
```python
miles.speak("Bow Wow")
print(miles.speak("Woof Woof"))
```

In [ ]:
```
1
```

In [ ]:
```
class Person():
    def __init__(self, first, last, age):
        self.first = first
        self.last = last
        self.age = age

```

In [ ]:
```
bobby = Person('bobby', 'hadz', 30)
```

In [ ]:
```
print(bobby.__dict__)
```

In [ ]:
```
attributes = list(bobby.__dict__.keys())
```

In [ ]:
```
print(attributes)
```

In [ ]:
```
print(dir(bobby))
```

In [ ]:
```
# 👇 dict_items([('first', 'bobby'), ('last', 'hadz'), ('age', 30)])
print(bobby.__dict__.items())

result = ', '.join(f'{key}={str(value)}' for key,
                   value in bobby.__dict__.items())
```

In [ ]:
```
1
```

Delete Object Properties

In [ ]:
```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)
```

In [ ]:
```
p1 = Person("John", 36)
p1.myfunc()
```

In [ ]:
```
del p1
```

In [ ]:
```
p1.myfunc()
```

In [ ]:
```

```

In [1]:
```
class araba:
    def __init__(self,marka,model,yıl):
        self.marka = marka
        self.model = model
        self.yıl = yıl

    def göster (self):
        print('Arabanın markası :{} , modeli:{} ve yılı:{} '.format(self.marka,self.model,self.yıl))
    def araba_bilgisi_güncelle (self,marka,model,yıl):
        self.marka = marka
        self.model = model
        self.yıl = yıl

```

In [2]:
```
# ilk araba sınıfı
araba_1 = araba("Toyota", "Camry", 2010)
```

In [4]:
```
araba_1.göster()
```

```
Arabanın markası :Toyota , modeli:Camry ve yılı:2010
```

In [5]:
```
araba_1.araba_bilgisi_güncelle("Honda", "Civic", 2012)
```

In [6]:
```
araba_1.göster()
```

```
Arabanın markası :Honda , modeli:Civic ve yılı:2012
```

In [ ]:
```

```

Tanımlanmış Bir Sınıfın Tüm Methodlarını Görmek

```
Bir sınıfın tüm methods almak için inspect.getmembers() yöntemini kullanın.
```

In [10]:
```
import inspect


class Employee():
    def __init__(self, name, salary):
        self.salary = salary
        self.name = name

    def get_name(self):
        return self.name

    def get_salary(self):
        return self.salary
```

In [11]:
```
list_of_methods = inspect.getmembers(Employee, predicate=inspect.isfunction)
```

In [12]:
```
print(list_of_methods)
```

```
[('__init__', <function Employee.__init__ at 0x000001DF92E8ECA0>), ('get_name', <function Employee.get_name at 0x000
001DF92E8E790>), ('get_salary', <function Employee.get_salary at 0x000001DF92E8E040>)]
```

In [13]:
```
bob = Employee('Bobbyhadz', 100)
```

In [14]:
```
list_of_methods = inspect.getmembers(bob, predicate=inspect.ismethod)
```

In [15]:
```
print(list_of_methods)
```

```
[('__init__', <bound method Employee.__init__ of <__main__.Employee object at 0x000001DF92E3BAF0>>), ('get_name', <b
ound method Employee.get_name of <__main__.Employee object at 0x000001DF92E3BAF0>>), ('get_salary', <bound method Em
ployee.get_salary of <__main__.Employee object at 0x000001DF92E3BAF0>>)]
```

Python'da Başka Bir Sınıftan Nasıl Miras Alırsınız?

In [ ]:
```
```

```
Miras, bir sınıfın diğerinin niteliklerini ve yöntemlerini devraldığı süreçtir. Yeni oluşturulan sınıflara alt sınıflar (child classes), alt sınıflardan türettiğiniz sınıflara ise üst sınıflar (parent classes) denir.
```

In [16]:
```python
# inheritance.py

class Parent:
    hair_color = "brown"

class Child(Parent):
    pass
```

In [17]:
```python
Child.hair_color
```

Out[17]:
```
'brown'
```

```
Bu minimal örnekte, Child alt sınıfı Parent sınıfından miras alır. Çocuk sınıfları ebeveyn sınıflarının niteliklerini ve yöntemlerini aldığından, Child.hair_color siz açıkça tanımlamasanız da "kahverengidir".
```

Saç renginizi ailenizden almış olabilirsiniz. Bu, doğuştan sahip olduğunuz bir özelliktir. Ama belki saçınızı mora boyamaya karar verirsiniz. Ebeveynlerinizin mor saçları olmadığını varsayarsak, ebeveynlerinizden miras aldığınız saç rengi özelliğini geçersiz kılmış olursunuz:

In [18]:
```python
# inheritance.py

class Parent:
    hair_color = "brown"

class Child(Parent):
    hair_color = "purple"
```

In [19]:
```python
Parent.hair_color
```

Out[19]:
```
'brown'
```

In [20]:
```
class kişi:
    def __init__ (self,isim,soy_isim):
        self.isim = isim
        self.soy_isim = soy_isim
    def print_isim_soy_isim(self):
        print(self.isim,self.soy_isim)
```

In [21]:
```
x = kişi("Ali", "Veli")
x.print_isim_soy_isim()
```

```
Ali Veli
```

In [22]:
```
class öğrenci (kişi):
    def __init__ (self,isim,soy_isim):
        super().__init__(isim,soy_isim)
        self.soy_isim = self.soy_isim.upper()
```

In [23]:
```
y = öğrenci("Ali", "Veli")
y.print_isim_soy_isim()
```

```
Ali VELI
```

Yeni Özellik Eklemek

In [24]:
```
class öğrenci (kişi):
    def __init__ (self,isim,soy_isim,numara):
        super().__init__(isim,soy_isim)
        self.soy_isim = self.soy_isim.upper()
        self.okul_no=numara
```

In [25]:
```
y = öğrenci("Ali", "Veli",15)
y.print_isim_soy_isim()
y.okul_no
```

```
Ali VELI
```

Out[25]:
```
15
```

In [ ]:
```

```

```
Sample Employee Data:
"ALİ", "M2001", 50000, "MUHASEBE"
"VELİ", "M2010", 45000, "ARAŞTIRMA"
"MARTIN", "S2020", 50000, "SATIŞ"
"AYŞE", "S2019", 55000, "OPERASYON"
```

```
Bir çalışanın departmanını değiştirmek için 'departman_atama' yöntemini kullanın.
Bir çalışanın ayrıntılarını yazdırmak için 'personel_bilgilerinin_yazdırılması' yöntemini kullanın.
'maaşı_hesapla' : (iki bağımsız değişken alır): maaş ve çalışanın çalıştığı saat sayısı. Çalışılan saat sayısı 50'den fazla ise yöntem fazla mesaiyi hesaplayarak maaşa ekler. Fazla mesai aşağıdaki formülle hesaplanır:
```

```
fazla_mesai=çalışma_saati-50
fazla_mesai_parası=(fazla_mesai*(maaş/50))
```

In [26]:
```python
class Personel:
    def __init__(self,isim,personel_no,maaş,departman):
        self.isim = isim
        self.personel_no = personel_no
        self.maaş = maaş
        self.departman = departman

    def maaşı_hesapla(self, maaş, çalışma_saati):
        fazla_mesai = 0
        if çalışma_saati > 50:
            fazla_mesai = çalışma_saati - 50
        self.maaş = self.maaş + (fazla_mesai * (self.maaş / 50))

    def departman_atama(self, departman):
        self.departman = departman

    def personel_bilgilerinin_yazdırılması(self):
        print("\nİsim: ", self.isim)
        print("Personel No: ", self.personel_no)
        print("Maaş: ", self.maaş)
        print("Departman: ", self.departman)
        print("----------------------")

```

In [27]:
```python
Personel1 = Personel("ALİ", "M2001", 50000, "MUHASEBE")
Personel2 = Personel("VELİ", "M2010", 45000, "ARAŞTIRMA")
Personel3 = Personel("MARTIN", "S2020", 50000, "SATIŞ")
Personel4 = Personel("AYŞE", "S2019", 55000, "OPERASYON")
```

In [28]:
```
print("Personel Bilgileri:")
Personel1.personel_bilgilerinin_yazdırılması()
Personel2.personel_bilgilerinin_yazdırılması()
Personel3.personel_bilgilerinin_yazdırılması()
Personel4.personel_bilgilerinin_yazdırılması()
```

```
Personel Bilgileri:

İsim:  ALİ
Personel No:  M2001
Maaş:  50000
Departman:  MUHASEBE
----------------------

İsim:  VELİ
Personel No:  M2010
Maaş:  45000
Departman:  ARAŞTIRMA
----------------------

İsim:  MARTIN
Personel No:  S2020
Maaş:  50000
Departman:  SATIŞ
----------------------

İsim:  AYŞE
Personel No:  S2019
Maaş:  55000
Departman:  OPERASYON
----------------------
```

In [29]:
```
# Çalışan1 ve çalışan4 departmanlarını değiştirin
Personel1.departman_atama("OPERASYON")
Personel4.departman_atama("SATIŞ")
```

In [30]:
```
# FAZLA MESAİ:
Personel2.maaşı_hesapla(45000, 52)
Personel4.maaşı_hesapla(45000, 60)
```

In [31]:
```
print("Güncelleşmiş Personel Bilgisi:")
Personel1.personel_bilgilerinin_yazdırılması()
Personel2.personel_bilgilerinin_yazdırılması()
Personel3.personel_bilgilerinin_yazdırılması()
Personel4.personel_bilgilerinin_yazdırılması()
```

```
Güncelleşmiş Personel Bilgisi:

İsim:  ALİ
Personel No:  M2001
Maaş:  50000
Departman:  OPERASYON
----------------------

İsim:  VELİ
Personel No:  M2010
Maaş:  46800.0
Departman:  ARAŞTIRMA
----------------------

İsim:  MARTIN
Personel No:  S2020
Maaş:  50000
Departman:  SATIŞ
----------------------

İsim:  AYŞE
Personel No:  S2019
Maaş:  66000.0
Departman:  SATIŞ
----------------------
```

In [ ]:
```
1
```

In [ ]:
```
1
```