Python

Nesne Yönelimli Programlama

Dr. Yunus Santur

Nesne Yönelimli Programlama

Object Oriented Programming

OOP bir programlama yaklaşımıdır.

1. Sıralı Programlama (Kod ilk satırda doğar, çalışır, son satırda biter)
2. Metotlar (Tekrar kullanılabilirlik, kendini tekrar etmeme!)
3. OOP (Nesnelerin özellikleri ve davranışları vardır)
   - OOP ile gerçek dünyayı daha kolay modelleyebiliriz.

Avantajları

- Modülerlik
- Yazılım bakım kolaylığı
- Genişletilebilirlik
- Değiştirilebilirlik
- Yeniden kullanılabilirlik

Kavramlar

- OOP aşağıdaki 4 temel prensibi desteklemelidir

- Abstraction (Soyutlama)
- Encapsulation (Kapsülleme)
- Inheritance (Miras)
- Polymorphism (Çok biçimlilik)

Sınıf/Nesne Kavramı

- Sınıf soyut bir kavramdır.
  - OOP ile oluşturulan programlama yapısıdır.

- Nesne sınıfın somutlaşan bir cismidir.

Sınıf Oluşturma

```
class ilksinif:          # Sınıfları class anahtar sözcüğü ile oluşturuyoruz
   i=1                   # Sınıf değişkeni
   def yaz(self):        # Sınıf metotu, self anahtar sözcüğü sınıfa atıf yapar
      print "i=",self.i  # Sınıf değişkenlerine self.değişken şeklinde erişiyoruz


x=ilksinif()             # x sınıftan oluşturduğumuz nesne
print x.i                # Sınıf değişkenlerine erişim ve atama x.i=5
x.yaz()                  # Sınıf metodunu çağırma
```

Global Değişkenler

```
class ilksinif:
   global a
   a=2
    i=1
    def yaz(self):
            print "global a=",a

x=ilksinif()
x.yaz()
a=7
x.yaz()
```

- Sınıf değişkenlerine sınıf dışında doğrudan erişemeyiz

- Sınıf içinden
  - ***self.değişken***
- Sınıf dışından
  - ***nesne.değişken***

şeklinde erişebiliriz

Global Değişkenlere

- Her yerden erişebilir, değiştirebiliriz
- Self anahtar sözcüğüne gerek yok

Sınıfları Başlatma

```
class ilksinif:
   def __init__(self):
      print "merhaba"



x=ilksinif()
```

__init__(self):

- Özel bir metottur
- Metot çağrılmaksızın nesne oluşturulduğunda çalışır

- initialize : Başlatmak

Programı Başlatmak

Programı ilk satırdan itibaren biz başlatmış olduk

Programı doğrudan bağımsız olarak çalıştırmak için, bir main metoduna ihtiyacımız var (Java da ki main methodu gibi).

```
if __name__=="__main__":
    x=ilksinif()
```

Sınıflara Başka Programdan Erişmek

```
class ilksinif:
   i=1
   def yaz(self):
       print "i=",self.i
```

Bu programı **dene.py** olarak kaydettiğimizi varsayarsak

```
import dene
x=dene.ilksinif()
print x.goster()
```

test.py programı içinden erişmek için import ediyoruz.

Decorators (@)

Metotların üstünde yer alan tanımlama ifadeleridir.

=> Örnek metotları (decoratorsüz kullanım)

@classmethod => sınıf metotu

@staticmethod => statik metot

Sınıf Üyeleri

- Public => normal değişken
- Private => _ _ iki alt çizgi ile başlayan değişken
- Semi-private => _ tek alt çizgi ile başlayan değişken

@property

- Özel set ve get metotlar yazılmasına gerek kalmaz

Miras

- Miras türleri
  - Olduğu gibi miras alır
  - Yeniden tanımlanabilir
  - Değişikliğe uğratılabilir
  - super()