Nesneye Yönelimli Programlama – BLM 205

Hafta 9: Decoratorlar

Fenerbahçe Üniversitesi

Öğretim Elemanları

Öğretim Üyesi: Dr. Vecdi Emre Levent

Ofis: 311

Email: emre.levent@fbu.edu.tr

Asistan: Arş. Gör. Uğur Özbalkan

Ofis: 307

Email: ugur.ozbalkan@fbu.edu.tr

Asistan: Arş. Gör. Ecenur Alioğulları

Ofis: 307

Email: ecenur.aliogullari@fbu.edu.tr

Ders Planı

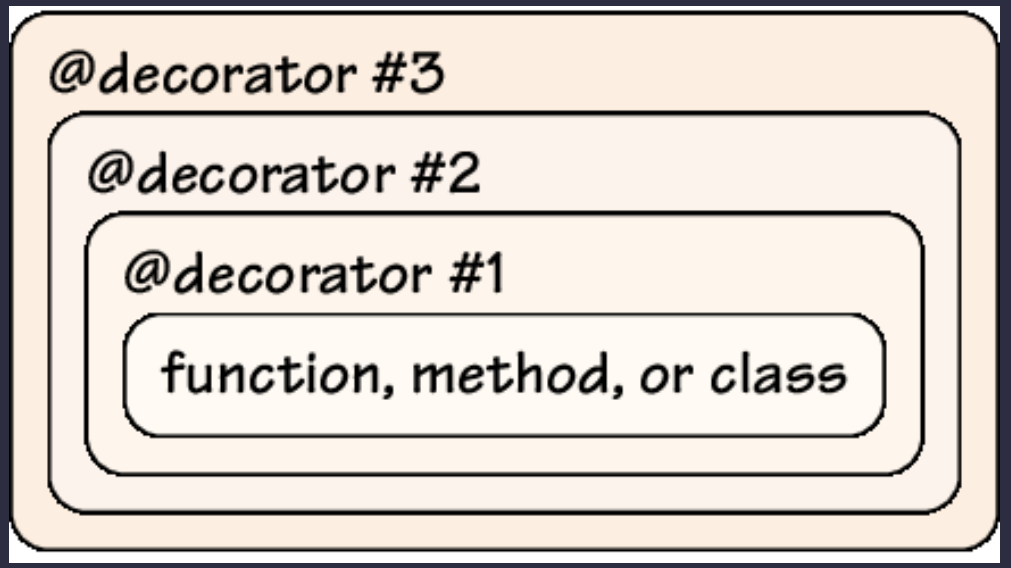
- Decoratorlar
  - Fonksiyon'u obje olarak kullanma
  - Fonksiyona yeni işlevler ekleme

Decoratorlar

Decorator

İki temel kullanım amacı vardır. Bunlar:

- Mevcut fonksiyonların veya sınıfların davranışlarının üzerine yeni bir işlev eklemek.
- Kod okunabilirliğini arttırmak



Decoratorlar

Decorator için önbilgiler

- Python dilinde fonksiyonlarda bir objedir.
- Yani, bir fonksiyon bir başka fonksiyona argüman olarak verilebilir.
- Bir değişken olarak atanabilir.

Decoratorlar

Decorator için önbilgiler

Örnek Kod Parçacığı

```
def testFnc1(msg):
    print(msg)

testFnc1("Merhaba")

denemeDegisken = testFnc1
denemeDegisken("Merhaba")
```

Çıktı

```
Merhaba
Merhaba
```

Fonksiyon Değişken olarak atandı

Decoratorlar

Decorator için önbilgiler

Örnek Kod Parçacığı

```
def arttir(x):
    return x + 1


def azalt(x):
    return x - 1


def calistir(func, x):
    result = func(x)
    return result

calistir(arttir,10)

calistir(azalt,10)
```

Çıktı

```
11
9
```

calistir fonksiyonuna, argüman olarak arttir ve azalt fonksiyonları verildi

Decoratorlar

Decorator için önbilgiler

Örnek Kod Parçacığı

```
def fonksiyonTest1():
    def fonksiyonTest2():
        print("Merhaba")

    return fonksiyonTest2


argumanTest = fonksiyonTest1()

argumanTest()
```

Çıktı

Merhaba

Geriye fonksiyon döndürülüyor

Geriye dönen fonksiyon alındı

Geriye dönen fonksiyon çalıştırıldı

Decoratorlar

Decorator

- Decarator’lar bir fonksiyondur.
- Dekore edilecek olan fonksiyonu argüman olarak alırlar.
- Geriye fonksiyon döndürürler

Decoratorlar

Decorator

Örnek Kod Parçacığı

```
def testFunc1(func):
    def testFunc2():
        print("Dekore edildi")
        func()

    return testFunc2


def testFunc3():
    print("siradan bir fonksiyon")


testFunc3()

donenFonksiyon = testFunc1(testFunc3)
donenFonksiyon()
```

Çıktı

```
siradan bir fonksiyon
Dekore edildi
siradan bir fonksiyon
```

testFunc1'de alınan func argümanı, testFunc2'de kullanılarak geriye testFunc2 döndürülüyor.

Dönen fonksiyonda, testFunc3 ve print işlemi yapılan yeni bir fonksiyon vardır.

Decoratorlar

Decorator

Örnek Kod Parçacığı

```
def testFunc1(func):
    def testFunc2():
        print("Dekore edildi")
        func()

    return testFunc2


def testFunc3():
    print("siradan bir fonksiyon")


testFunc3 = testFunc1(testFunc3)

testFunc3()
```

Çıktı

```
Dekore edildi
siradan bir fonksiyon
```

Çoğu zaman dekorator fonksiyondan dönen fonksiyon, dekorasyon yapılmak üzere verilen argüman fonksiyona atanır.

Yani, dekorasyon yapılan fonksiyon üzerine atanır.

Decoratorlar

Decorator

Örnek Kod Parçacığı

```
def testFunc1(func):
    def testFunc2():
        print("Dekore edildi")
        func()

    return testFunc2


def testFunc3():
    print("siradan bir fonksiyon")


testFunc3 = testFunc1(testFunc3)

testFunc3()
```

=

Örnek Kod Parçacığı

```
def testFunc1(func):
    def testFunc2():
        print("Dekore edildi")
        func()

    return testFunc2

@testFunc1
def testFunc3():
    print("siradan bir fonksiyon")


testFunc3()
```

Decorator kısayolu

Decoratorlar

Decorator

Syntax

```
@decoratorFonksiyon
def decoreEdilenFonksiyon(args)
        ...
```

Decoratorlar

Decorator

Aşağıdaki argümanları olan bir fonksiyon örneği verilmiştir.

Örnek Kod Parçacığı

```
def bolme(a, b):
    return a/b

bolme(3,0)
```

Hata

division by zero

Kod çalışma zamanında hata verecektir.

Decoratorlar

Decorator

Örnek Kod Parçacığı

```
def yeniBolme(func):
    def geriDonecekFonksiyon(a, b):
        print("Bolme yapilacak, sayilar: ", a,
" ve ", b)
        if b == 0:
            print("Bolme yapilamaz")
            return

        return func(a, b)
    return geriDonecekFonksiyon


@yeniBolme
def bolme(a, b):
    print(a/b)

bolme(3,0)
```

Çıktı

```
Bolme yapilacak, sayilar:  3  ve  0
Bolme yapilamaz
```

yeniBolme fonksiyonu ile dekorasyon yapıldı

Decoratorlar

Sınırsız sayıda argüman ile decorator

Örnek Kod Parçacığı

```
def genelDecorator(func):
    def geriDonecekFonksiyon(*args, **kwargs):
        print("Herhangi bir fonksiyon icin
calisabilir")
        return func(*args, **kwargs)

    return geriDonecekFonksiyon
```

Dekorator fonksiyonuna argüman olarak gelecek fonksiyonun herhangi bir tipte ve sayıda geldiğinde, dinamik olarak argümanlar geri döndürülebilir.

Decoratorlar

Çoklu(Zincir) Decorator

Örnek Kod Parçacığı

```
def yildiz(func):
    def inner(*args, **kwargs):
        print("*****")
        func(*args, **kwargs)
        print("*****")
    return inner


def yuzde(func):
    def inner(*args, **kwargs):
        print("%%%%%")
        func(*args, **kwargs)
        print("%%%%%")
    return inner


@yildiz
@yuzde
def printer(msg):
    print(msg)


printer("Merhaba")
```

Çıktı

```
*****
%%%%%
Merhaba
%%%%%
*****
```

Önce yuzde, sonra yildiz fonksiyonları dekorasyon yaptılar.

Decoratorlar

Çoklu(Zincir) Decorator

Örnek Kod Parçacığı

```
@yildiz
@yuzde
def printer(msg):
    print(msg)
```

=

Örnek Kod Parçacığı

```
def printer(msg):
    print(msg)
printer = yildiz(yuzde(printer))
```

Decoratorlar

Decorator

Örnek Kod Parçacığı

```
import time
import math

def zamanHesaplama(func):

    def inner1(*args, **kwargs):

        begin = time.time()
        func(*args, **kwargs)
        end = time.time()
        print("Toplam calisma zamani : ", end - begin)

    return inner1

@zamanHesaplama
def faktoriyel(num):
    time.sleep(2)
    factSonuc = math.factorial(num)
    print(factSonuc)

faktoriyel(5)
```

Çıktı

```
120
Toplam calisma zamani :  2.0009071826934814
```

Faktoriyel fonksiyonuna, zaman hesaplama kabiliyeti zamandırıldı