İşlev nedir?

TODO

işlev

fonksiyon (işlev): belli bir işlemi gerçekleştirmek üzere isimlendirilmiş cümle (komut) serisidir.

→ işlevi bir kereliğine **tanımlarsınız**: ne yapacağına dair cümleleri yazarsınız

→ işlevi çok defa **çağırırsınız**: aynı işi (farklı **parametrelerle**) tekrar tekrar yaptırırsınız

işlev: tanımlama

→ işlev tanımlanırken,

```
1  def ISIM( PARAMETRE LISTESI ):
2      CUMLELER
```

→ işlevin ne yaptığını söyleyen: ISIM

→ bu işte kullanacağı dış veriler: PARAMETRELER

→ ne yapacağını söyleyen: CUMLELER

bileşik cümle

- → def anahtar kelime (*definition*)
- → satır sonunda iki nokta üst üste -:
- → CUMLELER, def satırına göre içerden başlamalıdır
- → işlevler aslında def ile başlayıp (**başlık** satırı)
- → CUMLELER ile devam eden (**gövde** bölümü)
- → bileşik bir cümledir
- → işlev isminden sonra parametreler olmasa da parantez vardır

PEP isimlendirme kuralları

todo

işlev

- → diğer dillerdeki (C++, Pascal vs) gibi arayüz ve gerçekleme ayrı değildir
- → işleve ihtiyaç duyduğunuzda sadece onu deklare edersiniz (tanımlarsınız)

```
def merhaba_de():
```

- → def bu deklerasyonu başlatır
- → varsa argümanları parantez içerisinde yer alır, birden fazla iseler virgülle ayrılır
- → işlevin döndüreceği değerin türünün belirtilmesi zorunluluğu yoktur
- → boş return dönütleri None değerlidir

merhaba dünya

→ ilk işlevimiz

```
1  def merhaba_de():
2      print "merhaba, dunya"
```

→ parametre almayan

→ merhaba_de isminde

→ tek cümle (print “merhaba, dunya”) gövdeye sahip

→ bir işlev tanımıdır

pass ifadesi

→ yeni kodla çalışıyorken işlev gövdesini boş bırakmamak için

→ farklı kullanımları da olan (while ile belli bir süre bekleme gibi)

→ pass ifadesi

```
1  def initlog():
2      pass   # Remember to implement this!
```

işlev çağrısı

- → işlevi tanımlamak tek başına hiçbir anlama gelmez
- → onu çağırmalısınız ki bir işi icra etsin
- → işlevi çağırısı ise
- → işlev ismini, takiben parantez içerisinde
- → varsa parametrelerle gerçekleştirilir

```
1  # gercekle
2  def merhaba_de():
3      print "merhaba, dunya"
4
5  # main
6  print "BASLA"
7  merhaba_de()     # cagir
8  merhaba_de()     # tekrar cagir
9  merhaba_de()     # tekrar cagir
10 print "BITIR"
```

işlev çağrısı

→ çıtı

```
1  BASLA
2  merhaba, dunya
3  merhaba, dunya
4  merhaba, dunya
5  BITIR
```

işlevi başka bir işlevde kullanma

→ üç çağrıyı tek bir işlevde birleştirebiliriz

```
1  def merhaba_de3():
2      print "merhaba, dunya"
3      print "merhaba, dunya"
4      print "merhaba, dunya"
```

→ ya da daha güzeli

```
1  def merhaba_de3():
2      merhaba_de()
3      merhaba_de()
4      merhaba_de()
```

→ daha da güzeli var geleceğiz

→ yordam (işlev) içerisinde birden fazla kez çağırıyoruz

neden işlevler

- → Yeni bir fonksiyon yaratma, size cümle gruplarına bir isim verme şansı verir.
- → Yeni fonksiyonlar yaratmak programınızı küçültebilir.
  - → *basit olarak ekranda 6 kez merhaba dedirtmek nasıl olurdu?*

yürütme akışı

→ merhaba_de ve merhaba_de3, main üzerinden çizim yaparak açıkla

uzun metin

→ üç tırnakla uzun - çok satırlı çıktı

```
1  def davet_et():
2      print """\
3      Sayin Mehmet Bey,
4
5      19.05.2010 tarihinde bolumumuzde yapacagimiz
6      OYAK etkinliginde yer almanizi gonulden arzu ederiz.
7
8      Saygılarımızla,
9      OYAK Kulubu"""
10
11 davet_et()
```

→ Mehmet bey dışındakileri nasıl davet edeceğiz?

→ tarihi değiştirmenin başka bir yolu olabilir mi?

argümanlar

→ davet_et işlevine davet edilecek kişiyi nasıl söyleriz

→ şimdiye kadar aslında gördünüz

```
>>> x = input("Bir sayi giriniz:")
Bir sayi giriniz:-4
>>> y = abs(x)
>>> print "y =", y
y = 4
```

→ input işlevindeki “Bir sayi giriniz:”

→ abs işlevindeki x

→ print işlevindeki “y=”, y

argümanlar

argüman (parametre): fonksiyonun görevini yaparken kullandığı ve bir bakıma bu görevi nasıl yapacağını belirleyen değerler.

- → bazıları tek parametre alır: abs(x)
- → bazıları iki: pow(a, b)
- → bazılarında ucu açıktır: max(1, 4, 2, 0, 9, 5)

argüman

→ selamlama işlevi

```
1  def selamla():
2      print "merhaba"
```

→ kimi selamlayacak

```
1  def selamla(kisi):
2      print "merhaba", kisi
```

→ kisi argümanıdır

davet_et - v2

→ davet_et işlevini dışarıdan argüman alacak biçimde düzenleyelim

```
1  def davet_et(kimi, nezaman, kim):
2      print """\
3      Sayin %s,
4
5      %s tarihinde bolumumuzde yapacagimiz
6      OYAK etkinliginde yer almanizi gonulden arzu ederiz.
7
8      Saygılarımızla,
9      %s""" % (kimi, nezaman, kim)
10
11 davet_et("Mehmet Atar", "21.02.2010", "OYAK Kulubu")
12 davet_et("Robotik Kulubu Uyeleri", "21.02.2010", "Bilgisayar M
```

import cümlesi

→ selamla işlevini d03_selamla.py ismiyle kaydedin

→ herhangi bir betikte/kabukta bunu çağırmak için

```
1  from d03_selamla import *
2
3  kim = raw_input("Kimi selamlayayim? ")
4  selamla(kim)
```

argüman kısmında deyim kullanımı

→ argüman değer veya değişken olabileceği gibi

→ deyim de olabilir

```
1  selamla('Python' * 5)
```

yerel değişken

→ argümanlar yerel değişkenlerdir

```
1  def swap(a,
2       a, b = b, a
3
4  x, y = 5, 4
5  swap(x, y)
6  print x, y
```

→ ekran çıktısı nedir?

→ 5 4? 4 5?

docstring - doctest

- → docstring, Python işlevlerinizi dokumente etmede kullanırız
- → tasarladığınız her işlev için bunu yazmak zorunda değilsiniz
- → FAKAT dokumente etseniz iyi olur
- → böylelikle her bir işlevin tasarımıyla dokumentasyonu aynı dosyada yer alır
- → Python bu dokumentasyona erişmek için iki yol sunar: __doc__, help

docstring: __doc__ ve help()

→ örnek

```
1  # d03_docstring.py
2
3  def selamla(kisi):
4      """
5      selamla(kisi), islevi.
6      kisi yi selamlar
7      """
8      print "merhaba", kisi
```

docstring: __doc__ ve help()

→ örnek

```
1  # d03_docstring.py
2
3  def selamla(kisi):
4      """
5      selamla(kisi), islevi.
6      kisi yi selamlar
7      """
8      print "merhaba", kisi
```

→ __doc__ yöntemi ve help()

```
>>> from d03_docstring import *
>>> selamla.__doc__
'\n\tselamla(kisi), islevi. \n\
>>> help(selamla)
Help on function selamla in mod

selamla(kisi)
    selamla(kisi), islevi.
    kisi yi selamlar
```

docstring örnekleri

→ PEP-0257

```
def complex(real=0.0, imag=0.0):
    """Form a complex number.

    Keyword arguments:
    real -- the real part (default 0.0)
    imag -- the imaginary part (default 0.0)

    """
    if imag == 0.0 and real == 0.0: return complex_zero
    ...
```

doctest

- → doctest, yazılım geliştirmede kaynak kodun otomatik birim sınamasını yapmak yaygın bir pratiktir.
- → Birim sınama, fonksiyonlar gibi bağımsız kod parçalarının otomatik olarak doğru çalıştığını onaylamak için bir yol sağlar.
- → Bu daha sonra fonksiyonun gerçekleştirimini değiştirmeyi ve yine de bekleneni yapmasını olanaklı kılar.

Python'da doctest modülü ve kullanımı

- → Doctestler fonksiyon gövdesinin veya betiğin ilk satırında üç tırnaklı karakter dizileri (docstring) içerisinde yazılabilir
- → Bunlar bir Python bilgi istemine girdileri ve beklenen çıktıyı örnekleyen yorumlayıcı oturumları şeklindedir.
- → doctest modülü >>> ile başlayan herhangi bir cümleyi otomatik olarak çalıştırarak,
- → takip eden cümleyi yorumlayıcının çıktısıyla karşılaştırır.

doctest

→ örnek

```
1  # d03_doctest.py
2
3  def is_divisible_by_2_or_5(n):
4      """
5        >>> is_divisible_by_2_or_5(8)
6        True
7      """
8
9  if __name__ == '__main__':
10     import doctest
11     doctest.testmod()
```

test edelim

→ test

```
$ python d03_doctest.py -v
Trying:
    is_divisible_by_2_or_5(8)
Expecting:
    True
**********************************************************************
File "d03_doctest.py", line 5, in __main__.is_divisible_by_2_or_5
Failed example:
    is_divisible_by_2_or_5(8)
Expected:
    True
Got nothing
1 items had no tests:
    __main__
**********************************************************************
1 items had failures:
   1 of   1 in __main__.is_divisible_by_2_or_5
1 tests in 2 items.
0 passed and 1 failed.
***Test Failed*** 1 failures.
```

daha gerçeksi

→ argüman değişken olması durumu

```
1  def is_divisible_by_2_or_5(n):
2      """
3        >>> is_divisible_by_2_or_5(8)
4        True
5        >>> is_divisible_by_2_or_5(7)
6        False
7        >>> is_divisible_by_2_or_5(5)
8        True
9        >>> is_divisible_by_2_or_5(9)
10       False
11     """
12     print n % 2 == 0 or n % 5 == 0
13
14 if __name__ == '__main__':
15     import doctest
16     doctest.testmod()
```

sıra sizde

→ aşağıdaki problemi çözün

```
1  def cat_n_times(s, n):
2      <kodu buraya yazin>
3
4  >>> cat_n_times('Spam', 7)
5  SpamSpamSpamSpamSpamSpamSpam
```

→ veya şöyle diyeceğiz

```
1  def cat_n_times(s, n):
2      """
3          >>> cat_n_times('Spam', 7)
4          SpamSpamSpamSpamSpamSpamSpam
5      """
6      <kodu buraya yazin>
```

→ diyeceğiz