Geri dönüş değerleri

→ abs, pow, max sonuç üretiyorlardı

```
>>> mx = max(3, 7, 2, 5)
>>> x = abs(3 - 11) + 10
>>> mx, x
(7, 18)
```

→ her işlev çağrısı bir değer üretmedir

→ üretilen değer ya bir değişkene atanır ya da deyimin bir parçasıdır

→ şimdiye kadar tasarladığımız işlevler değer üretip, döndürmedi

ürün veren işlevler

→ işlev değer hesaplayıp, döndürüyorsa ürünlüdür

→ böyle yazardık

```
1  def area(radius):
2      tmp = 3.14159 * radius ** 2
3      print "alan =", tmp
4
5  # main
6  area(5)
7  area(11)
```

ürün veren işlevler

→ işlev değer hesaplayıp, döndürüyorsa ürünlüdür

→ böyle yazardık

```
1  def area(radius):
2      tmp = 3.14159 * radius ** 2
3      print "alan =", tmp
4
5  # main
6  area(5)
7  area(11)
```

→ şimdi değeri geri döndürelim

```
1  def area(radius):
2      tmp = 3.14159 * radius ** 2
3      return tmp
4
5  # main
6  print "Kucuk dairenin alani =", area(5)
7  print "Buyuk dairenin alani =", area(11)
```

return

- → return <deger>(ler): değer(ler) döndürür
- → çağrıldığın yere geri dön, devamındaki ifadeyi geri dönüş değeri olarak kullan
- → döndürülecek değer değişkende veya doğrudan deyim olabilir

```
1  def area(radius):
2      return 3.14159 * radius ** 2
```

- → tmp benzeri geçici değişkenler hata ayıklamada kullanışlıdır

birden fazla return cümlesi

→ birden fazla return cümlesi

```
1  def myabs(x):
2      if x < 0:
3          return -x
4      else:
5          return x
```

birden fazla return cümlesi

→ birden fazla return cümlesi

```
1  def myabs(x):
2      if x < 0:
3          return -x
4      else:
5          return x
```

→ nasıl olsa return'e rastlanılan yerden hemen çıkıyor

→ else'yi kaldırabiliriz

```
1  def myabs(x):
2      if x < 0:
3          return -x
4      return x
```

her durum idare edilmeli yoksa ...

→ işlevler her durumu idare etmelidir

```
1  def myabs(x):
2      if x < 0:
3          return -x
4      elif x > 0:
5          return x
```

→ burada hata nerededir?

her durum idare edilmeli yoksa ...

→ işlevler her durumu idare etmelidir

```
1  def myabs(x):
2      if x < 0:
3          return -x
4      elif x > 0:
5          return x
```

→ burada hata nerededir?

→ x=0 ise ne olacak?

doctest'ten kaçmaz

→ doctest'ler bunun için var

```
1   def myabs(x):
2       """\
3           myabs(x), mutlak deger alma islevidir.
4           >>> myabs(-5)
5           5
6           >>> myabs(0)
7           0
8           >>> myabs(5)
9           5
10      """
11      if x < 0:
12          return -x
13      elif x > 0:
14          return x
15
16  if __name__ == '__main__':
17      import doctest
18      doctest.testmod()
```

doctest sonuçları

→ sonuçlar

```
$ python -i d05_myabs.py
**********************************************************************
File "d05_myabs.py", line 5, in __main__.myabs
Failed example:
    myabs(0)
Expected:
    0
Got nothing
**********************************************************************
1 items had failures:
   1 of   3 in __main__.myabs
***Test Failed*** 1 failures.
```

→ Got nothing veya None

→ işlev hakkında birşey söylemiyorsa, döndürmüyorsa None'dır

program geliştirme

→ programlar büyüdükçe çalışma zamanı ve anlambilimsel hatalar artmaya başlar

→ karmaşık programlarla başa çıkmak için

→ arttırımsal geliştirme

arttırımsal geliştirme

hata ayıklama süreçlerini kısaltmak

1. bir anda küçük bir kod ekle
2. ekleneni sına
3. tekrar 1. adım

Ör. distance hesaplayıcı

→ iki nokta arasındaki mesafenin hesaplanması

→ pisagor teoremi: distance=sqrt((x2-x1)^2+(y2-y1)^2)

1. bunu Python'da nasıl ifade ederim?
2. girdiler (parametreler) ve çıktı (dönüş değeri) nedir?

Ör. distance hesaplayıcı

→ iki nokta arasındaki mesafenin hesaplanması

→ pisagor teoremi: distance=sqrt((x2-x1)^2+(y2-y1)^2)

1. bunu Python'da nasıl ifade ederim?
2. girdiler (parametreler) ve çıktı (dönüş değeri) nedir?

→ girdiler: noktalar

→ çıktı: hesaplanan uzaklık değeri (kayan noktalı olarak)

tasarım

→ tasarım: işlevin anahatları

```
1  def distance(x1, y1, x2, y2):
2      return 0.0
```

→ önce bunu bir sına

```
1  >>> distance(1, 2, 4, 6)
2  0.0
```

→ hesap yok, doğru değer döndürmüyor

→ sözdizimsel olarak doğru, çalışabilir kod

girdi set seçimi

→ neden (1, 2, 4, 6) seçtik

→ biliyoruz ki 4-1=3, 6-2=4, 3-4-5 pisagorundan uzaklık 5.0 olacak

→ girdi: (1, 2, 4, 6)

→ çıktı: 5.0

hata ayıkla

→ her bir yöndeki mesafeleri hesaplamakla devam et

```
1  def distance(x1, y1, x2, y2):
2      dx = x2 - x1
3      dy = y2 - y1
4      print "dx = %s, dy = %s" % (dx, dy)
5      return 0.0
```

hata ayıkla

→ her bir yöndeki mesafeleri hesaplamakla devam et

```
1  def distance(x1, y1, x2, y2):
2      dx = x2 - x1
3      dy = y2 - y1
4      print "dx = %s, dy = %s" % (dx, dy)
5      return 0.0
```

→ neden print satırı var

→ hata ayıklama

→ böyle yaparsak idaresi daha kolay

```
1  def distance(x1, y1, x2, y2):
2      dbg = True  #debug on/off
3      dx = x2 - x1
4      dy = y2 - y1
5      if dbg: print "dx = %s, dy = %s" %
6      return 0.0
```

geliştirme

→ kareyi hesapla

```
1  def distance(x1, y1, x2, y2):
2      dx = x2 - x1
3      dy = y2 - y1
4      dsqr = dx**2 + dy**2
5      if dbg: print "dsqr = %s" % (dsqr)
6      return 0.0
```

→ önceki print cümlesini kaldırdık

→ inşaat kurarken iskele kullan,

→ söz konusu aşamayı geçince iskeleyi kaldır

geliştirme

→ karekök eklentisi, ve geridönüş

```
1  def distance(x1, y1, x2, y2):
2      dx = x2 - x1
3      dy = y2 - y1
4      dsqr = dx**2 + dy**2
5      result = dsqr**0.5
6      return result
```

→ bunu test etme sırasıdır ...

anahatlar

1. Çalışan bir programla başla

   → *küçük artımlı değişikler yap*

   → *herhangi bir noktada bir hata varsa hatanın nerede olduğunu bulmak kolay*

2. Ara değerleri tutmak için geçici değişkenler kullan

   → *böylece bu değerleri ekran gösterip kontrol edebilirsin*

3. Program çalışır hale gelince, iskelet kodlarının bazılarını kaldır

   → *birden fazla cümleyi bileşik deyimler haline getir*

   → *birleştirme x okunurluk?*

kompozisyon

- bir işlevi başka işlevden çağırabiliriz
- dairenin alanını hesaplama
- girdi: dairenin merkezi, daire üzerindeki bir nokta
- çıktı: dairenin alanı
- önce yarıçapı hesapla
- sonra alanı hesapla

yarıçap hesaplama

→ dairenin merkezi: (xc, yc)

→ daire üzerinde yer alan bir nokta: (xp, yp)

→ yarıçap

```
1  radius = distance(xc, yc, xp, yp)
```

alan hesaplama

→ dairenin yarıçapından alan hesaplayabiliriz (daha önce verilmişti)

```
1  result = area(radius)
```

parçaları bir araya getir

→ önce yarıçap hesabı sonra alan

```
1  def area2(xc, yc, xp, yp):
2      radius = distance(xc, yc, xp, yp)
3      result = area(radius)
4      return result
```

parçaları bir araya getir

→ önce yarıçap hesabı sonra alan

```
1  def area2(xc, yc, xp, yp):
2      radius = distance(xc, yc, xp, yp)
3      result = area(radius)
4      return result
```

→ geçici radius ve result değişkenleri hata ayıklama içindi

→ kısaltabiliriz

```
1  def area2(xc, yc, xp, yp):
2      return area(distance(xc, yc, xp, yp))
```

dikdörtgen çizme

→ çizgi çizdirmeyi kullanarak dikdörtgen çizdirmek, nasıl?

→ dikdöretgenin sol-üst ve sağ-alt köşeleri elimizde

→ köşeler: (x1, y1) ve (x2, y2)

```
1  def dikdortgen(x1, y1, x2, y2):
2      Line((x1, y1), (x2, y1))
3      Line((x2, y1), (x2, y2))
4      Line((x2, y2), (x1, y2))
5      Line((x1, y2), (x1, y1))
```

→ şöyle de test edebiliriz

```
1  dikdortgen(50, 50, 150, 200)
```

dikdörtgen çizme-v2

→ köşe noktasından ziyade genişlik ve yüksekliğe göre olsa

→ değişkenler: w, h

```
1  def dikdortgen2(x, y, w, h):
2      x1, y1 = x, y
3      x2, y2 = x + w, y + h
4
5      Line((x1, y1), (x2, y1))
6      Line((x2, y1), (x2, y2))
7      Line((x2, y2), (x1, y2))
8      Line((x1, y2), (x1, y1))
```

dikdörtgen çizme-v2

→ köşe noktasından ziyade genişlik ve yüksekliğe göre olsa

→ değişkenler: w, h

```
1  def dikdortgen2(x, y, w, h):
2      x1, y1 = x, y
3      x2, y2 = x + w, y + h
4
5      Line((x1, y1), (x2, y1))
6      Line((x2, y1), (x2, y2))
7      Line((x2, y2), (x1, y2))
8      Line((x1, y2), (x1, y1))
```

→ neden tekrar edelim ki

→ son dört satıra zaten bir isim vermiştik: dikdörtgen

```
1  def dikdortgen2(x, y, w, h):
2      x2, y2 = x + w, y + h
3      dikdortgen(x, y, x2, y2)
```

→ ara değişkenler kaldırmakta mümkün

dikdörtgen çizme-v3

→ kare çizdirmek istersek

→ kenar: a

```
1  def kare(x, y, a):
2      dikdortgen2(x, y, a, a)
```

öntanımlı değerler

→ işlevlerde değişkenlere parametre aktarımında

```
1  def Circle(center, radius, filled=False, \
2             color=(0,0,0), thickness=1):
```

boolean işlevler

→ işlev boolean değer üretebilir

```
1  def is_divisible(x, y):
2      if x%y == 0:
3          return True
4      else:
5          return False
```

→ işlevin ismi "bölünebilir mi?"
⟹ is_divisible

→ el-cevap: "Evet/Hayır"
⟹ True/False

boolean işlevler

→ işlev boolean değer üretebilir

```
1  def is_divisible(x, y):
2      if x%y == 0:
3          return True
4      else:
5          return False
```

→ işlevin ismi "bölünebilir mi?"
⟹ is_divisible

→ el-cevap: "Evet/Hayır"
⟹ True/False

→ boolean deyimi return cümlesine alabiliriz

```
1  def is_divisible(x, y):
2      return x%y == 0
```

test

→ işlevin testi kolay

```
>>> is_divisible(6, 4)
False
>>> is_divisible(6, 3)
True
```

→ bu yapıyı aynen doctest te kullanırız

→ boolean işlevler genelde koşul cümlelerinde kullanılır

```
if is_divisible(x, y):
    print "x, y tam bolunebilir
else:
    print "x, y tam bolunemez"
```

function türü

→ işlevler type ile çağrıldığında ne döndürür

```
>>> type(distance)
<type 'function'>
```

işlevleri argüman olarak geçmek mümkündür

→ işlevleri argüman olarak geçmek mümkündür

```
1  def f(n):
2      return 3*n - 6
3
4  def g(n):
5      return 5*n + 2
6
7  def h(n):
8      return -2*n + 17
9
10 def doto(value, func):
11     return func(value)
12
13 print doto(7, f)
14 print doto(7, g)
15 print doto(7, h)
```

→ sonuç

fonksiyon pointerları

→ fonksiyon pointerları

```
1   def topla(x, y):
2       return x + y
3
4   def cikar(x, y):
5       return x - y
6
7   islem = {'+':topla, '-':cikar}
8   x, y = 3, 4
9   islec = '+'
10  fn = islem[islec]
11  print fn(x, y)
```

program yazım kuralları

→ PEP

docstring - doctest

1. kodun otomatik birim sınaması
2. hakkında yardım sağlama

gerçekleme

→ tek/çift?

```
1       def isodd(n):
2           """\
3               isodd(n), tek mi?
4
5               >>> isodd(3)
6               True
7               >>> isodd(4)
8               False
9           """
10          return n%2 == 1
11
12      if __name__ == '__main__':
13          import doctest
14          doctest.testmod()
```

örnek: birim sınama

→ test

```
$ python d05_parity.py -v
Trying:
    isodd(3)
Expecting:
    True
ok
Trying:
    isodd(4)
Expecting:
    False
ok
1 items had no tests:
    __main__
1 items passed all tests:
   2 tests in __main__.isodd
2 tests in 2 items.
2 passed and 0 failed.
Test passed.
```

→ -v seçeneğine dikkat

Örnek: hakkında yardım

→ test

```
$ python
Python 2.6.4 (r264:75706, Dec  7 2009, 18:45:15)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from d05_parity import *
>>> help(isodd)
Help on function isodd in module d05_parity:

isodd(n)
    isodd(n), tek mi ?

    >>> isodd(3)
    True
    >>> isodd(4)
    False
```

→ in module d05_parity cümlesine dikkat

default argument values

→ öntanımlı değerler

```
1  def ask_ok(prompt, retries=4, complaint='Yes or no, please!'):
2      while True:
3          ok = raw_input(prompt)
4          if ok in ('y', 'ye', 'yes'):
5              return True
6          if ok in ('n', 'no', 'nop', 'nope'):
7              return False
8          retries = retries - 1
9          if retries < 0:
10             raise IOError('refusenik user')
11         print complaint
```

default argument values

→ öntanımlı değerler

```
def ask_ok(prompt, retries=4, complaint='Yes or no, please!'):
    while True:
        ok = raw_input(prompt)
        if ok in ('y', 'ye', 'yes'):
            return True
        if ok in ('n', 'no', 'nop', 'nope'):
            return False
        retries = retries - 1
        if retries < 0:
            raise IOError('refusenik user')
        print complaint
```

→ kullanırken/çağırırken

```
ask_ok('Do you really want to quit?')
ask_ok('OK to overwrite the file?', 2)
ask_ok('OK to overwrite the file?', 2, 'Come on, only yes or n
```

öntanımlı değerler: bir kereliğine

→ öntanımlı değerler bir kereliğine atanır

```
1  def f(a, L=[]):
2      L.append(a)
3      return L
4
5  print f(1)
6  print f(2)
7  print f(3)
```

→ ekran çıktısı

```
1  [1]
2  [1, 2]
3  [1, 2, 3]
```

keyword argümanlar

→ argümanları keyword = value olarakta aktarabilirsiniz

```
1  def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):
2      print "-- This parrot wouldn't", action,
3      print "if you put", voltage, "volts through it."
4      print "-- Lovely plumage, the", type
5      print "-- It's", state, "!"
```

keyword argümanlar

→ gerçerli: çağrılar

```
1  parrot(1000)
2  parrot(action = 'VOOOOOM', voltage = 1000000)
3  parrot('a thousand', state = 'pushing up the daisies')
4  parrot('a million', 'bereft of life', 'jump')
```

→ gerçersiz: çağrılar

```
1  parrot()                     # required argument missing
2  parrot(voltage=5.0, 'dead')  # non-keyword argument following
3  parrot(110, voltage=220)     # duplicate value for argument
4  parrot(actor='John Cleese')  # unknown keyword
```

belirsiz sayıda argüman

→ argüman sayısını tam bilmiyorsak: bakınız print cümleleri

```
1  def cheeseshop(kind, * arguments, ** keywords):
2      print "-- Do you have any", kind, "?"
3      print "-- I'm sorry, we're all out of", kind
4      for arg in arguments: print arg
5      print "-" * 40
6      keys = keywords.keys()
7      keys.sort()
8      for kw in keys: print kw, ":", keywords[kw]
```

belirsiz sayıda argüman

→ çağırırken

```
cheeseshop("Limburger", "It's very runny, sir.",
           "It's really very, VERY runny, sir.",
           shopkeeper='Michael Palin',
           client="John Cleese",
           sketch="Cheese Shop Sketch")
```

→ ekran çıktısı

```
-- Do you have any Limburger ?
-- I'm sorry, we're all out of Limburger
It's very runny, sir.
It's really very, VERY runny, sir.
----------------------------------------
client : John Cleese
shopkeeper : Michael Palin
sketch : Cheese Shop Sketch
```

lambda

- → fonksiyonel programlama
- → LISP
- → işlerin kestirme yolları vardır

```
>>> def f (x): return x**2
...
>>> print f(8)
64
>>>
>>> g = lambda x: x**2
>>>
>>> print g(8)
64
```

lambda

→ bazı standart işlevleri işleri kolaylaştırır

```
>>> foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]
>>>
>>> print filter(lambda x: x % 3 == 0, foo)
[18, 9, 24, 12, 27]
>>>
>>> print map(lambda x: x * 2 + 10, foo)
[14, 46, 28, 54, 44, 58, 26, 34, 64]
>>>
>>> print reduce(lambda x, y: x + y, foo)
139
```

dizgiler + lambda

→ lambda güçlüdür

```
1  >>> sentence = 'It is raining cats and dogs'
2  >>> words = sentence.split()
3  >>> print words
4  ['It', 'is', 'raining', 'cats', 'and', 'dogs']
5  >>>
6  >>> lengths = map(lambda word: len(word), words)
7  >>> print lengths
8  [2, 2, 7, 4, 3, 4]
```

sistem programcısı

→ sistem programcısı böyle kullanır

```
>>> import commands
>>>
>>> mount = commands.getoutput('mount -v')
>>> lines = mount.split('\n')
>>> points = map(lambda line: line.split()[2], lines)
>>>
>>> print points
['/', '/var', '/usr', '/usr/local', '/tmp', '/proc']
```