More on Functions & an introduction to arrays

Clicker question

```
def f(x):
      x = x+1
      return x

x = 11

def  g():
      y = f(x)
      print x, y

x = 12
g()
```

What is printed?

A. 11 12
B. 12 13
C. 13 13
D. 12 11

Clicker question

```
def fun1 (x, y):
    x = 25
    print x+y

def fun2(a,b):
    a = a + 1
    return a+b

if __name__ == '__main__':
    s = 1
    x, y = 20, 100

    fun1(s, x)
    t = fun2(s, y)
    print s, t, x
```

What is printed?

A. 120
   1 102 20

B. 45
   1 102 20

C. 45
   2 102 20

D. 125
   1 102 20

3

Wednesday, February 4, 2009

Solutions

B
12 13

B
45
1 102 20

4

Wednesday, February 4, 2009

Example

```
def square(x):
        return x*x

def distance(p1, p2):
        d = math.sqrt(
                square(p2[0] - p1[0]) + square(p2[1] - p1[1]))
        return d
```

```
r = [3,5]
s = [-4.5,89]
t = [2*r[0], 3*s[1]]

val1 = distance(r, s)
val2 = distance(t, s)
```

```
>>>
84.3341567812
178.309422073
```

5

Wednesday, February 4, 2009

Getting results from functions

- Functions can return multiple results
- Get multiple results from call using multiple assignment

```
def sum_diff(x,y):
   sum = x+y
   diff = x-y
   return sum, diff

a, b = 5, 15
s,d = sum_diff(a,b)

print s, d
print sum_diff(a,b)
```

```
>>>
20 -10
(20, -10)
```

6

Wednesday, February 4, 2009

Getting results from functions

- Be careful not to forget the **return** for functions that have a result
- Functions without a **return** actually return a result **None**

Functions that modify parameters

- Assigning a value to a to a parameter (like x in function f) is a **local** change to the local parameter, not the argument
- This leaves x=10 and y=11

```
def f(x):
    x = x+1
    return x

x=10
y=f(x)
```

Functions that modify parameters

- Python passes arguments by assigning their values to the parameters
  - Applies to integers, floating point numbers, strings
- But, some values are actually references to "structures": lists, arrays, sequences
  - Assigning to the element of a structure does change the value of that element
  - The change remains after completion of the function

9

Wednesday, February 4, 2009

Functions that modify parameters

- Assigning a value to an **element of a parameter** (like x[0] of paramter x) changes that element.
- List x is changed in function f;
- After invoking function f, we have x = [2, 2, 3]

```
def f(x):
    x[0] = x[0]+1
    return

x = [1, 2, 3]
f(x)
```

10

Wednesday, February 4, 2009

Lists as arguments …

```
def alter(LA,LB):
    k = 15
    LA[0] = k
    LB[2] = LB[1] + LA[0]
    return

P = [12, 16, 88]
Q = [0, 0, 2, 2]

print "before:\t", P, Q
alter(P, Q)
print "after:\t", P, Q

R = Q
R[0] = -10
```


Functions and Style

- Functions should be used carefully so that program meaning is made clearer
- Overuse of functions can obfuscate, or obscure, the program meaning
- Try to use a consistent naming scheme
  - in_rect, readPoints, ShowLines
- Use direct verb names: drawRectangle

Functions and Style

- Parameters should be meaningful
- Parameters should capture logical varying inputs
- Avoid unrelated/unexpected/surprising side-effects
- Avoid under-generalizing
  - try to make a function broadly reusable
- Avoid over-generalizing
  - Swiss-Army Chainsaws are dangerous

Default arguments

- Python functions can have default arguments, evaluated in the defining scope

```
i = 5

def f(a=i, b=3):
    print a, b

i = 6
f()
```

Default arguments

- The default is evaluated just once when the function is defined

```
def f(a, L=[]):
    L.append(a)
    return L

print f(1)
print f(2)
print f(3)
```


Keyword arguments

- Functions can be called with arguments by keyword instead of position
- Any positional arguments must come before keyword arguments

```
def f(a, b):
    return a+b

print f(b=1,a=2)
print f(1,b=2)
```

Function arguments

- Functions can take functions as arguments
- "Higher-Order" functions

```
def f(g, x):
    return g(x)

def inc(x):
    return x+1

f(inc,2)
```


Arrays

- From the numpy library:
  - import numpy
  - Most common operations creating arrays: array, zeros, ones
- Efficient, math-oriented implementation of lists
- All elements in an array are of same data type, declared in advance
- Support multiple dimensions, reshaping dimensions, many matrix operations and more.
- http://www.scipy.org/Tentative_NumPy_Tutorial

```
from numpy import *

a = array([10, 20, 30, 40])  # create an array from a list

b = arange(4)   #create and initialize an array using range

b
array([0, 1, 2, 3])

c = ones((3,4))  #create and initialize a 2-d array with 1's

print c
[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]
```


Array Features

- Can specify data type of elements
  - x = zeros(100, dtype = int16)
  - 16 bit integers
  - Used for sound samples in audio project
- Useful functions
  - min(x)
  - max(x)
  - append(x,y) - returns new array that is x concatenated with y (x + y does element-by-element addition which is more expensive)

Looping Efficiency

- **Standard looping technique…**

```
x=0
for i in range(10000000):
 x += 1
```

Creates a very large list, taking time and memory

- **Alternative…**

```
x=0
for i in xrange(10000000):
 x +=1
```

No list is created, for loop handles "iterator" efficiently

- Use xrange for large values