02-4 Fruitful Functions

CSI 500

Spring 2018

Course material derived from:
Downey, Allen B. 2012. "Think Python, 2nd Edition". O'Reilly Media Inc., Sebastopol CA.

"How to Think Like a Computer Scientist" by Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers. Oct 2012
http://openbookproject.net/thinkcs/python/english3e/index.html

Return values

- Most of our functions so far have not returned a useful value
  - Technically they return "None", which is an instance of Python NoneType
- You can write functions to return useful values
  - use the "return" operator, followed by the value you want to return

```
>>> import math
>>> def area( radius ):
        a = math.pi * radius ** 2
        return a

>>> area( 5 )
78.53981633974483
```

Incremental Development

- Often useful to develop complicated functions a little bit at a time
  - Start with something simple that works
  - make small incremental changes
  - if something doesn't work, you know where to look
- Use local variables to hold intermediate values - display and check them
  - After the program is working, you can remove the "scaffolding" if needed, but only if it doesn't break anything
- Example: Cartesian distance
  - $distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y)^2}$

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

>>> # now lets add some temp variables
>>> def distance( x1, y1, x2, y2):
            dx = x2 - x1
            dy = y2 - y1
            print( 'dx = ', dx )
            print( 'dy = ' , dy )
            return( 0. 0)

>>> # now do the computation
>>> def distance( x1, y1, x2, y2):
            dx = x2 - x1
            dy = y2 - y1
            dsqared = dx **2 + dy ** 2
            print( 'dx = ', dx )
            print( 'dy = ', dy )
            print( ' dsquared = ', dsquared)
            return( 0. 0)
```

Incremental Development

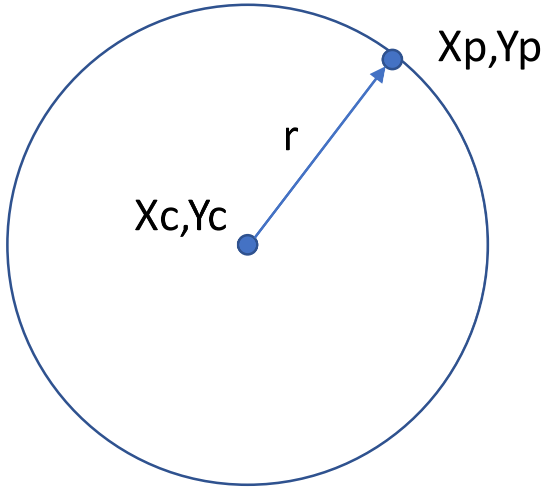
- Keep adding on
- When it's working, you can take out the debug print statements

```
>>> # now do the computation
>>> def distance( x1, y1, x2, y2):
            dx = x2 - x1
            dy = y2 - y1
            dsqared = dx **2 + dy ** 2
            result = math.sqrt( dsquared )
            print( 'dx = ', dx )
            print( 'dy = ', dy )
            print( ' dsquared = ', dsquared)
            print( ' result = ', result )
            return( result )

>>> # clean up final version
>>> def distance( x1, y1, x2, y2):
            dx = x2 - x1
            dy = y2 - y1
            dsqared = dx **2 + dy ** 2
            result = math.sqrt( dsquared )
            return( result )
```

Composition

- You can combine several functions together to make more complicated actions
- Example: given the center point of a circle and a point on the circle, compute the area of the circle
  - first we need the radius of the circle, so we'll use our distance function
  - then we need the area of the circle, so we'll use our area function

```
>>> def circle_area( xc, yc, xp, yp):
        radius = distance(xc, yc, xp, yp)
        result = area( radius )
        return result

>>> # this can be more concisely expressed
>>> def circle_area( xc, yc, xp, yp):
        return area( distance( xc, yc, xp, yp)
```



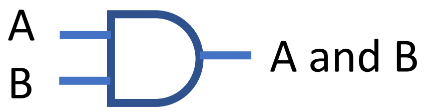
Boolean Functions

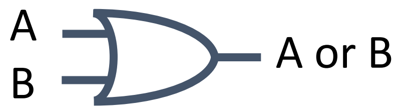
- A Boolean function returns only True or False
- Often used to hide complex logic used in other functions
- Example: test for divisibility

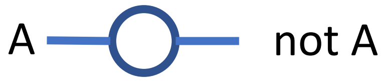
```
>>> def is_divisible( x, y ):
        if x % y == 0:
                return( True )
        else:
                return( False )

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

>>> # can be expressed more concisely
>>> def is_divisible( x, y ):
        return x % y == 0

>>> # can be used in conditional statements
>>> x = 6
>>> y = 3
>>> if is_divisible( x, y ):
        print ( '{} is divisible y {}'.format(str(x), str(y))
 6 is divisible by 3
```

More Boolean Functions

- Functions can be used to evaluate logical expressions
  - Python supports Boolean operators AND, OR, and NOT







A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

A	not A
0	1
1	0

```
A = 1
B = 0

print( A and B)
False

print( A or B )
True

print( not A )
False

def myAND( v1, v2 ):
        return v1 and v2
```

Summary

- Python functions can return useful values
  - dubbed "fruitful" functions
- When writing Python, incremental development is a good method
  - build a little, test a little
- Functions can be combined together
  - dubbed "composition"
- Boolean or logical functions work with True-False values