413-513 Notes- Week 4

**[ Functions – continued ]**

Topics:

Function overloading / argument passing args, kwargs

Multiple returns on functions

Lambda functions (one liners!)

Recursive functions

Creating modules

Math library

-Positional args vs. Keyword args

Use cases:

Function overloading / argument passing args, kwargs

How are keywords passed? Given in line after [positional args]

Ex. **def counter(start=0, step=1)**

**'''**

**There are two ways to assign argument values to function parameters, both are used.**

**By Position. Positional arguments do not have keywords and**

**are assigned first.**

**By Keyword. Keyword arguments have keywords and are assigned second, after positional arguments.**

**Note that you have the option to use positional arguments.**

**If you don't use positional arguments, then -- yes -- everything you wrote turns out to be a keyword argument.**

**When you call a function you make a decision to use position or keyword or a mixture. You can choose to do all keywords if you want. Some of us do not make this choice and use positional arguments.**

**'''**

**def foo(\*positional, \*\*keywords):**

**print ("Positional:", positional)**

**print ("Keywords:", keywords)**

**foo('one', 'two', 'three')**

**print()**

**foo(a='one', b='two', c='three')**

**print()**

**foo('one','two',c='three',d='four')**

-Multiple returns on functions

**def g(x):**

**y0 = x + 1**

**y1 = x \* 3**

**y2 = y0 \*\* y1**

**return y0, y1, y2**

**x,y,z = g(5)**

**print (x,y,z)**

>>>

**6 15 470184984576**

>>>

-Lambda functions (in line functions!)

rule set:

* Note: only **one** expression in the lambda body; Its value is always returned
* The lambda expression must fit on one line!

**def tryMe(x,y):**

**oneLineFunc = lambda x,y: x+y**

**print (oneLineFunc(x,y)) #Execute lambda**

**tryMe(2,3);**

>>>

**5**

>>>

-Recursive functions (work trace!)

**def gcd(x,y):**

**if x%y==0: #base case**

**return y**

**else:**

**return gcd(x,x%y)**

**print gcd(2,4)**

**print gcd(4,2)**

>>>

**2**

**2**

>>>

**#randDemo.py**

**import random**

**#hard code inventory items**

**a=1**

**b=2**

**c=3**

**#generate random inventory item**

**x= random.randint(1,5)**

**print (x)**

**'''check if x is repeating an inventory item (a b or c)'''**

**flg=True**

**while flg:**

**for val in [a,b,c]:**

**#regenerate x**

**if x in [a,b,c]: #x == val: --not 100% accurate**

**x = random.randint(1,7)**

**print("new x:",x)**

**else:**

**flg=False**

**print(a,b,c,x)**

-demo with function

**#itemChecker.py**

**import random**

**#incl. function to check repetitive inventory items**

**def itemRepeaterChecker(checker,\*args):**

**'''check if x is repeating an inventory item (a b or c)'''**

**flg=True**

**while flg:**

**for val in args:**

**#regenerate x**

**if checker in args:**

**checker = random.randint(1,7)**

**print("new x:",checker)**

**else:**

**flg=False**

**return checker #return x value**

-module driven

semantics:

**module\_name.method\_name**

**import func**

**func.foo('one')**

**from func import foo**

**foo('one')**

**#randDemo2.py**

**from itemChecker import itemRepeaterChecker**

**import random**

**#hard code inventory items**

**a=1**

**b=2**

**c=3**

**random.seed()**

**#generate random inventory item 1**

**x= random.randint(1,5)**

**print (x)**

**'''**

**pass x value to func**

**get result of x back locally**

**'''**

**x = itemRepeaterChecker(x,a,b,c)**

**#show non repeated inventory items**

**print(a,b,c,x)**

**#generate random inventory item 2**

**y= random.randint(1,10)**

**print (y)**

**'''**

**pass x value to func**

**get result of y back locally**

**'''**

**y = itemRepeaterChecker(y,a,b,c,x)**

**#show non repeated inventory items**

**print(a,b,c,x,y)**

**Next class**:

files and exceptions!