Lecture_7_Code

April 19, 2017

0.0.1 Review: Void and Returning function differences

```
In [4]: #A None-returning or Void function
        def printer (n):
            print ("A Printer Prints:",n)

        a = printer(5)
        print (type(a),a)

        #A function that returns
        def returner (n):
            return (n)

        b = returner(5)
        print (type(b),b)

        def summarizer (a,b,c):
            """
            Sums 3 numbers
            Returns sum
            """
            result = sum([a,b,c])
            return result

        c = summarizer(1,2,3)
        print (type(c),c)
```

```
A Printer Prints: 5
<class 'NoneType'> None
<class 'int'> 5
<class 'int'> 6
```

0.0.2 Functions can call other functions

Exploring the stack

```
In [ ]: def hooli():
            print ("http://www.hooli.xyz/")
```

```
        def piedpiper():
            hooli()
            print ("http://www.piedpiper.com/")
        def coderag(n):
            piedpiper()
            print ("http://www.coderag.com/")
            print (n*n)

        coderag(5)
        print ("Done")
        print ("Go back to Lecture Slides")
```

0.0.3 Matryoshka Height: Recursion

0.0.4 Matryoshka 1 = 3cm tall

0.0.5 Each next one is 2 cm taller

The height of the smallest one is 1 cm

```
In [ ]: def height(n):
            if n == 1:
                return 1
            else:
                return 2 + height(n-1)

        height(5)
```

```
In [ ]:
```

```
In [ ]: def height(n):
            count = 0
            small = 1
            while (n > 1):
                n -= 1
                count +=1

            return small + count*2

        height(5)
```

0.0.6 Iterative Factorial Functions

```
In [ ]: def fact1(n):
            """
            Calculatesw Factorial Iteratively
            while loop
            """
            prod = n
            while (n > 1):
```

```
            n -= 1
            prod *= n
        return prod

    def fact2(n):
        """
        Calculates Factorial Iteratively
        for loop
        """
        prod = n
        for i in reversed(range(1,n)):
            prod *=i
        return prod

    print (fact1(5))
    print (fact2(5))
```

0.0.7 Recursive Factorial Function

```
In [5]: def fact3(n):
            """
            Recursively calculates n!
            """
            #Base Case
            if n == 1:
                return 1
            #Recursive Case
            else:
                return n * fact3(n-1)

        print (fact3(5))
```

```
120
```

0.0.8 Recursive List Sum Function

```
In [ ]: def sumIt(myList):
            """
            Recursively returns a sum given a list of numbers
            """
            if (myList==[]):
                return 0
            else:
                return myList[0] + sumIt(myList[1:])

        print (sumIt([1,2,3]))
```

```
        #QUESTION: WHY DO WE NOT GET AN ERROR HERE?
        #WE ARE TRYING TO SLICE AT OFFSET THAT DOESN'T EXIST
        l = [1,2,3]
        print (l[4:])
```

```
In [ ]: import my_bio
```

```
In [ ]: import this
```

0.0.9 dir returns the attributes of a module or object

```
In [ ]: dir(my_bio)
```

0.0.10 This is how we access the docstring

this is how you can learn what a function does
remember """ is not for commenting things out!
I prefer help, but you can use doc as well

```
In [ ]: print (my_bio.nmers.__doc__)
```

```
In [ ]: help(my_bio.nmers)
```

```
In [ ]: import sys
        #default is 1000 depth
        #max is platform dependent
        help(sys.setrecursionlimit)
```

```
In [ ]: sys.getrecursionlimit()
```

```
        def stack():
            stack()

        stack()
```

```
In [ ]: help(sys.getrecursionlimit)
```

0.1 Pickles!

0.1.1 Store your progress, store your data structure, store your data types.

```
In [ ]: d = {"name":"FUGU"}
        my_list = ["Spam",d]
        print (my_list)

        import pickle
        with open('first.pickle', 'wb') as f:
            pickle.dump(my_list, f)

        #Let's make my_list refer to some empty list now
```

```
my_list = []
print ("Proof that it's empty: ", my_list) #yup it's empty at this point

#now let's load the pickle file

with open('first.pickle', 'rb') as f:
    my_list = pickle.load(f)
print ("We loaded it from the first.pickle file:",my_list)
```

0.1.2 Fileless pickling

It exists on the stack while your program is running

In [ ]:
```
a = pickle.dumps(my_list)

#then you do some kind of code here

unpickled_list = pickle.loads(a)
print (unpickled_list)
```

In [ ]:

In [ ]: