screenshot-localhost-8888-2017-10-19-16-15-45-167
http://localhost:8888/notebooks/Desktop/Python%20Modules/Lab%204/Lab-4B-Functions-.ipynb
19.10.2017

jupyter Lab-4B-Functions- Last Checkpoint: Yesterday at 11:28 AM (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Code

Functions

```
Formally, a function is a useful device that groups together a set of statements so they can be run more than once. They can
also let us specify parameters that can serve as inputs to the functions
```

```
Functions are a convenient way to divide your code into useful blocks, allowing us to order our code, make it more readable,
reuse it and save some time. Also functions are a key way to define interfaces so programmers can share their code
```

Functions will be one of most basic levels of reusing code in Python, and it will also allow us to start thinking of program design

On a more fundamental level, functions allow us to not have to repeatedly write the same code again and again. If you remember back to the labs on strings and lists, remember that we used a function len() to get the length of a string. Since checking the length of a sequence is a common task you would want to write a function that can do this repeatedly at command.

In [4]:
```
def name_of_function(arg1,arg2):
    '''
    This is where the function's Document String (doc-string) goes
    '''
    # Do stuff here
    #return/print desired result
```

Arguments are basically the inputs that you give to your function

Now here is the important step, you must indent to begin the code inside your function correctly. Python makes use of whitespace to organize code. Lots of other programing languages do not do this, so keep that in mind.

Next you'll see the doc-string, this is where you write a basic description of the function. Using iPython and iPython Notebooks, you'll be able to read these doc-strings by pressing Shift+Tab after a function name.

Doc strings are not necessary for simple functions, but its good practice to put them in so you or other people can easily understand the code you write.

In [13]:
```
def say_hello(a):
    print('Hello ' + a +'!')
```

In [14]:
```
say_hello('Professor')
```

```
Hello Professor!!
```

In [42]:
```
def happybday(name):
    "This function prints out the happy birthday message"
    print ("Happy Birhtday to you!")
    print ("Happy birthday to you!!")
    print ("Happy Birthday to you, %s!!!" %name)
    print ("Happy birthday to you!")
    return
```

In [45]:
```
happybday('abc')
```

```
Happy Birhtday to you!
Happy birthday to you!!
Happy Birthday to you, abc!!!
Happy birthday to you!
```

In [3]:
```
# How to build a function in python ! What's the syntax
```

In [6]:
```
say_hello('Professor')
```

```
Hello Professor
```

Using return

Let's see some example that use a return statement. return allows a function to return a result that can then be stored as a variable, or used in whatever manner a user wants.

In [18]:
```
# lets multiply two numbers and return the result
```

In [18]:
```
def prod_two_num(a,b):
    return(a*b)
```

In [22]:
```
prod_two_num(3,5)
```

Out[22]: 15

In [60]:
```
def prod_two_num(a,b):
    c = a * b
    return (c)

def div_two_num(a,b):
    c = (a/b)
    return(c)

def pro_div(a,b):
    print(prod_two_num(a,b))
    print(div_two_num(a,b))
```

In [61]:
```
pro_div(6,3)
```

```
18
2.0
```

In [37]:
```
# Lets write a function to check if a number is prime or not
# Lets use all the iterative arguments that we used before
```

In [9]:
```
def is_prime(num):
    for factor in range(2,num):
        if num%factor==0:
            return False
            print (factor)
            break
    else:
            return True
```

In [13]:
```
print(is_prime(18))
```

```
False
```

In [39]:
```
# How do you take this sentence and print only the words that start with s
# 'Hi I am Srivatsan. I Love Data Science.
```

In [76]:
```
def print_words_with_s(a):
    for word in a.split():
        if word[0] =='s' or word[0]== 'S':
            print(word)
        else:
            pass
```

In [77]:
```
print_words_with_s('Hi I am Srivatsan I love Data Science')
```

```
Srivatsan
Science
```

In [86]:
```
#How to find factorial using function

def factorial(a):
    res = 1
    if a == 0:
        return(res)
    else:
        for i in range(1,a+1):
            res = res*i
        print(res)
```

In [52]:
```
# Function definition is here
def changeme( mylist ):
    #This changes a passed list into this function
    mylist.append([1,2,3,4])
    print("Values inside the function: ", mylist)
    return

# Now you can call changeme function
mylist = [10,20,30];
changeme( mylist );
print ("Values outside the function: ", mylist)
```

```
Values inside the function:  [10, 20, 30, [1, 2, 3, 4]]
Values outside the function:  [10, 20, 30, [1, 2, 3, 4]]
```

In [53]:
```
# Function definition is here
def changeme( mylist ):
    "This changes a passed list into this function"
    mylist = [1,2,3,4]; # This would assig new reference in mylist
    print ("Values inside the function: ", mylist)
    return

# Now you can call changeme function
mylist = [10,20,30];
changeme( mylist );
print ("Values outside the function: ", mylist)
```

```
Values inside the function:  [1, 2, 3, 4]
Values outside the function:  [10, 20, 30]
```

Lambda functions

Lamda functions are anonymous funcitons. They are defined by lambda Lambda forms can take any number of arguments but return just one value in the form of an expression. They cannot contain commands or multiple expression.Only Single expression.

In [ ]:
```
# How will I use lambda function to add 2 numbers
```

In [66]:
```
add = lambda a1, a2: a1 + a2
```

In [71]:
```
def square(a):
    return(a **2)
```

In [72]:
```
square1 = lambda num : num **2
```

In [73]:
```
square(10)
```

Out[73]:
```
100
```

In [74]:
```
square1(10)
```

Out[74]:
```
100
```

In [75]:
```
# how do you create a sequence of numbers in python3 without using numpy
# using map and filter
```

In [89]:
```
a = list(range(1,11))
```

In [90]:
```
a
```

Out[90]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [101]:
```
map_ex = list(map(lambda x: x**2,a))
```

In [102]:
```
map_ex
```

Out[102]:
```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

In [105]:
```
filter_ex = list(filter(lambda x: x%2==0,a))
```

In [106]:
```
filter_ex
```

Out[106]:
```
[2, 4, 6, 8, 10]
```

Recursion

In [113]:
```
def fibbo(n):
    if n == 1:
        return 1
    else:
        return(n + fibbo(n-1))
```

In [115]:
```
fibbo(8)
```

Out[115]:
```
36
```

In [ ]:
```

```