W18 Reading Unit 5

June 24, 2016

1 Functions

1. Define the function
2. Call the function

```
In [1]: # when we define the fuction, the body of the function is not executed
        def print_stuff():
            print("some stuff")
            print("more stuff")
```

```
In [2]: print("Things on my todo list:")
        print_stuff()     # call the function!
        print_stuff()
```

```
Things on my todo list:
some stuff
more stuff
some stuff
more stuff
```

```
In [4]: def sqrt(x,epsilon):
            """Newton's Method for Finding Square Roots
            to within a precision epsilon"""     # this is a documentation
            ans = 1
            num_guesses = 0
            while abs(x/ans-ans) > epsilon:
                ans = (x/ans +ans)/2
                num_guesses +=1

            return ans
```

```
In [4]: help(sqrt)
```

```
Help on function sqrt in module __main__:

sqrt(x, epsilon)
    Newton's Method for Finding Square Roots
    to within a precision epsilon
```

```
In [5]: root = sqrt(5,0.00001)
        print(root,"is close to the square root of 5")
        fourth_root = sqrt(root,0.00001)
        print(fourth_root,"is close to the fourth root of 5")
```

```
2.2360688956433634 is close to the square root of 5
1.495349088238384 is close to the fourth root of 5
```

1.1 1. Calling Functions

```
In [ ]: # Parameters and arguments
        # whrn we are inside a function, we call the values passed in paramieters
        # when we call a function from the outside, we called the bojects we pass
        # in arguments
```

1.2 2. Nesting Functions

functions can be nested insied another function

```
In [1]: def sqrt(x,epsilon):
            """Newton's Method for Finding Square Roots
            to within a precision epsilon"""      # this is a documentation
            ans = 1
            num_guesses = 0
            while abs(x/ans-ans) > epsilon:
                ans = (x/ans +ans)/2
                num_guesses +=1

            return ans
        def distance_to_origin(x,y):
            """Find the  disance from a point at (x,y) to the origin"""
            ans = sqrt(x**2 + y**2, 0.00001)
            return ans

        def geometric_mean(x,y):
            """compute the square root of x * y """
            return sqrt(x*y, 0.00001)

        x = float(input("Enter an x-coordnate: "))
        y = float(input("Enter an y-coordnate: "))
        magnitude = distance_to_origin(x,y)
        print("The magnitude of your vetor is", magnitude)
        geo_mean = geometric_mean(x,y)
        print("The geopmetric mean of x and y is ", geo_mean)
```

```
Enter an x-coordnate: 3
Enter an y-coordnate: 4
The magnitude of your vetor is 5.000000000053722
The geopmetric mean of x and y is  3.4641016533502986
```

1.3 3. Functiosn and the Call Stack

How does Python keep track of where conrol needs to go when you call functions.

It does this with the help of a special data structrue called an execution stack, or call stack

1.4 4. The stack trace

help us understand where in our function is not correct!

```
In [7]: def sqrt(x,epsilon):
            """Newton's Method for Finding Square Roots
            to within a precision epsilon"""      # this is a documentation
            ans = 1
            num_guesses = 0/0       # what if we mand an error here!
```

```
            while abs(x/ans-ans) > epsilon:
                ans = (x/ans +ans)/2
                num_guesses +=1

            return ans
        def distance_to_origin(x,y):
            """Find the  disance from a point at (x,y) to the origin"""
            ans = sqrt(x**2 + y**2, 0.00001)
            return ans

        def geometric_mean(x,y):
            """compute the square root of x * y """
            return sqrt(x*y, 0.00001)

        x = float(input("Enter an x-coordnate: "))
        y = float(input("Enter an y-coordnate: "))
        magnitude = distance_to_origin(x,y)
        print("The magnitude of your vetor is", magnitude)
        geo_mean = geometric_mean(x,y)
        print("The geopmetric mean of x and y is ", geo_mean)
```

```
Enter an x-coordnate: 3
Enter an y-coordnate: 4
```

```
        ---------------------------------------------------------------------------

        ZeroDivisionError                         Traceback (most recent call last)

        <ipython-input-7-a6d741cfb74e> in <module>()
         20 x = float(input("Enter an x-coordnate: "))
         21 y = float(input("Enter an y-coordnate: "))
    ---> 22 magnitude = distance_to_origin(x,y)
         23 print("The magnitude of your vetor is", magnitude)
         24 geo_mean = geometric_mean(x,y)


        <ipython-input-7-a6d741cfb74e> in distance_to_origin(x, y)
         11 def distance_to_origin(x,y):
         12     """Find the  disance from a point at (x,y) to the origin"""
    ---> 13     ans = sqrt(x**2 + y**2, 0.00001)
         14     return ans
         15


        <ipython-input-7-a6d741cfb74e> in sqrt(x, epsilon)
          3     to within a precision epsilon"""     # this is a documentation
          4     ans = 1
    ----> 5     num_guesses = 0/0      # what if we mand an error here!
          6     while abs(x/ans-ans) > epsilon:
          7         ans = (x/ans +ans)/2


        ZeroDivisionError: division by zero
```

1.5 5. Advantages of functions

1. decomposition

   break up programming tasks into self-contained pirces

2. modularity

   write flecible colde that can be reused
   make code more readable

3. abstraction

   hide implementation details
   create layers for low level and high level tasks
   define interation etween programm components

2 Namespaces

How does python treat variable names

Why does each function get a name space 1. it prevents a function from accidentally altering variables in other parts of the program 2. different functions may be written by different programmers. it would be a huge pain to make sure that thet use different names 3. in large programs, we won'trun out of nice short vairbale names since they can be reused 4. the effects of a function are limited to the parameters it takes, making it eeasier to understand.

```
In [12]: def add_one(x):
             x = x + 1
             print("in the function, x = ", x) # this is a local variable because
             return x                          # because it is insied a function
         x = 3
         result = add_one(x)
         print("the function returned", result)
         print("Outside the function x = ",x) # this is a global variable
```

```
in the function, x =  4
the function returned 4
Outside the function x =  3
```

```
In [15]: def add_one(x):
             y = x + 1
             print("in the function, y = ", y) # this is a local variable because
             return y                          # because it is insied a function
         x = 3
         y = 3
         result = add_one(x)
         print("the function returned", result)
         print("Outside the function y = ",y) # this is a global variable
```

```
in the function, y =  4
the function returned 4
Outside the function y =  3
```

2.1 1. Accessing global variables

```
In [17]: def add_y(x):
             ans = x+y #note: there is no local y
                     # what happens here is that python will go up the stack and
                     # using that y
             print("In the function, y =",y)
             return ans

         y = 5
         result = add_y(3)
         print("the function returned", result)
         print("outside the function, y = ", y)
```

```
In the function, y = 5
the function returned 8
outside the function, y =  5
```

```
In [19]: # the previous is not too bad, but what about this one that acctualy
         # changes the global value
         def change_y():
             y = y+1
             print("In this function, y =", y)
         y =5
         change_y()
         print("Outside the function, y=", y)
```

```
---------------------------------------------------------------------------

UnboundLocalError                         Traceback (most recent call last)

<ipython-input-19-8b921de46501> in <module>()
  5     print("In this function, y =", y)
  6 y =5
----> 7 change_y()
  8 print("Outside the function, y=", y)


<ipython-input-19-8b921de46501> in change_y()
  2 # changes the global value
  3 def change_y():
----> 4     y = y+1
  5     print("In this function, y =", y)
  6 y =5


UnboundLocalError: local variable 'y' referenced before assignment
```

The previous example actually didnot change the value but gives an error, the reason of the error is the following:

Steps in runing a function

1. create a new local namespace
2. for each parameter, create a new local variable and bind it to argument

3. for each assignment statement, if name is not in local namespace, create a new local variable
4. execute the statement

```
In [22]: # If you really really want to do this, you can using the global key world
         # however, it may not be a good thing to do!
         def change_y():
             global y
             y = y+1
             print("In this function, y =", y)
         y =5
         change_y()
         print("Outside the function, y=", y)
```

```
In this function, y = 6
Outside the function, y= 6
```

3 Using parameters

```
In [26]: def feedback(grade,comment = ""):
             if grade >= 90 and grade <= 100:
                 return("A " + comment)
             elif grade >= 80 and grade < 90:
                 return("B " + comment)
             elif grade >= 70 and grade < 80:
                 return("C " + comment)
             elif grade >= 60 and grade < 70:
                 return("D " + comment)
             else:
                 return("F " + comment)
```

```
In [27]: print(feedback(90,"Great work!"))
```

```
A Great work!
```

```
In [28]: print(feedback(80))
```

```
B
```

```
In [29]: print(feedback(75,"Please study more"))
```

```
C Please study more
```

```
In [31]: type(None)
```

```
Out[31]: NoneType
```

```
In [34]: def feedback(grade = None, comment = None):
             text = "" if comment == None else "-" + comment

             if grade == None:
                 return("Grade is Missing. " + text)
             elif grade >= 90 and grade <= 100:
                 return("A " + text)
             elif grade >= 80 and grade < 90:
                 return("B " + text)
             elif grade >= 70 and grade < 80:
```

```
                return("C " + text)
            elif grade >= 60 and grade < 70:
                return("D " + text)
            else:
                return("F " + text)
```

```
In [39]: print(feedback(90,"Great work!\n"))
         print(feedback(80),"\n")
         print(feedback(75,"Please study more\n"))
         print(feedback())
```

```
A -Great work!

B

C -Please study more

Grade is Missing.
```

```
In [42]: def add_total(order_list = []):
             total = sum([quantity for name, quantity in order_list])
             order_list.append(("Total", total))
             print(order_list)
         order_list = [("Bob",8),("Teri",3),("Pat",4)]
         print(order_list)
         add_total()
         add_total()
```

```
[('Bob', 8), ('Teri', 3), ('Pat', 4)]
[('Total', 0)]
[('Total', 0), ('Total', 0)]
```

3.1 1. Keyword Arguments

```
In [45]: print(feedback(90,comment = "Keep it up!"))
         print(feedback(comment = "Not bad.", grade = 88))
         print("Default")
         print("print")
         print("behavior")
         print("Modifider", end = " ")
         print("print", end = " ")
         print("behavior", end= ".")
```

```
A -Keep it up!
B -Not bad.
Default
print
behavior
Modifider print behavior.
```

3.2 2. Functions are objects

```
In [2]: def round10(x):
            return (x+5) // 10 * 10
```

```
In [4]: type(round10)
```

```
Out[4]: function

In [6]: a = round10
        a(12)

Out[6]: 10

In [8]: def apply_to_grades(operation, rade_list):
            return[(name,operation(grade)) for name, grade in grade_list]

In [10]: grade_list = [("Betty", 88),("Steve",75),("Bob",73),("Teri",94),("Sandy",97)]
         print(apply_to_grades(round10,grade_list))

[('Betty', 90), ('Steve', 80), ('Bob', 70), ('Teri', 90), ('Sandy', 100)]

In [13]: list(map(round10,(23,45,42,66)))
         # this map apply round10 to each of the variables

Out[13]: [20, 50, 40, 70]

In [15]: # define a lamda function which is an anominous function, this is eazy to
         # define a short function
         lambda x : 100 - (100-x)/2

Out[15]: <function __main__.<lambda>>

In [18]: apply_to_grades(lambda x : 100-(100-x)/2, grade_list)

Out[18]: [('Betty', 94.0),
          ('Steve', 87.5),
          ('Bob', 86.5),
          ('Teri', 97.0),
          ('Sandy', 98.5)]

In [21]: list(map(lambda x:x**2,(23,45,42,66)))
         # define an annominous function that squares the numbers

Out[21]: [529, 2025, 1764, 4356]
```

3.3 Exceptions

```
In [24]: x = int(input("Enter a number: "))
         print("The reciprocal of your number is", 1/x)

Enter a number: K


        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-24-d99fbe94254a> in <module>()
    ----> 1 x = int(input("Enter a number: "))
          2 print("The reciprocal of your number is", 1/x)


        ValueError: invalid literal for int() with base 10: 'K'
```

```
In [26]: try:
             x = float(input("Enter a number: "))
             print("The reciprocal of your number is", 1/x)
         except:
             print("You did not enter a valid number ")

Enter a number: P
You did not enter a valid number

In [28]: try:
             x = float(input("Enter a number: "))
             print("The reciprocal of your number is", 1/x)
         except ValueError:    # this ValueError is the error you can see when you
                               # do not use try and enter a cha for the input
             print("You did not enter a valid number ")
         except ZeroDivisionError:    # this ZeroDivisionError is the error you can
                                    # see when you do not use try and enter a
                                    # 0 for the input
             print("Zero does not have a reciprocal ")
         except:
             print("Smething else went wrong")

Enter a number: 0
Zero does not have a reciprocal

In [31]: def sell(item, quantity, inventory):
             if item not in inventory:
                 raise Exception(str(itm)+" does not appear in inventory.")
             q = inventory[item]
             if q < quantity:
                 raise Exception("Inventory does not contain enough of item to sell.")
             inventory[item] = q-quantity
         inventory = {"oranges" : 10, "apples" : 5, "plums" : 3}

In [33]: sell("oranges", 15, inventory)


         ---------------------------------------------------------------------------

         Exception                                 Traceback (most recent call last)

         <ipython-input-33-005dd0be1eb9> in <module>()
     ----> 1 sell("oranges", 15, inventory)


         <ipython-input-31-13c5ddfc718a> in sell(item, quantity, inventory)
           4     q = inventory[item]
           5     if q < quantity:
     ----> 6         raise Exception("Inventory does not contain enough of item to sell.")
           7     inventory[item] = q-quantity
           8 inventory = {"oranges" : 10, "apples" : 5, "plums" : 3}


         Exception: Inventory does not contain enough of item to sell.
```

```
In [34]: try:
             sell("oranges",15,inventory)
         except Exception as e:
             print("Could not complete sale: " + str(e))
```

```
Could not complete sale: Inventory does not contain enough of item to sell.
```