Content Delivered in class_9_17-August-2016

- Chapter 5: Functions
  - User-defined functions
    - Functions without input arguments
    - Functions with inputs arguments
    - Scope: Local and Global Variables
    - Default and Keyword arguments
    - return statement
    - DocStrings
    - Variable Arguments (args and *kwargs)
    - lambda
    - map()
    - filter()
    - zip()
    - reduce()
    - recursive functions

Interview Questions Discussed:

Interview Question 1: Explain the difference between global and local values, with an example?

Interview Question 2: Write a function to highlist the vowels in the input string

Interview Question 3: What is the difference between map() and zip()?

Interview Question 4: Perform string concatenation using reduce()

Interview question 5: what is the maximum recursion depth of any python object? Does it differ from object to object? Is it dependent on the machine being used?

Assignments Given:

Assignment 1: Using the above written logic, write evenOddTest function within the map() and get the even values between 23 and 97

Assignment 2: Do modifications to this logic, to remove the spaces between letters; not words

Assignment 3: Write a script two functions, celsiusToFahrenheit() and fahrenheitToCelsius(). In runtime, the user should be given choice to enter either Fahrenheit or Celcius; then use the functions to convert correspondingly. Then, display the result from main function, using string formatting?

Assignment 4: Write three functions absolute(). Prompt the user to feed an integer (). The result should mimick the functionality of built-in function abs()

```
ex:
    In [1]: abs(9)
    Out[1]: 9

    In [2]: abs(-9)
    Out[2]: 9

    In [3]: abs(0)
    Out[3]: 0

    In [4]: abs(2+3j)
    Out[4]: 3.6055512754639896
```

Assignment 5: Write a function that validates and prints whether the run-time input given is palindrome or not.

```
Ex: 'otto'
    'Evil is a deed as I live' Hint: ignore white-space here.
```

Assignment 6: write a function that generates palindromes, for the given run-time input. There should be two outputs, both for even and odd case. Ref : http://www.jimsabo.com/palindrome.html (http://www.jimsabo.com/palindrome.html).

Assignment 7: Write a function to implement the caesar cipher, for the given input.

```
ex1: input  --> 'Python'
    output --> 'Qzuipo'
ex2: input -->  'One day, I will achieve my goal!'
     output ->  'Pof!ebz-!J!xjmm!bdijfwf!nz!hpbm"'
```

Assignment 8: Define a procedure histogram() that takes a list of integers and prinst a histogram to the screen. For example, histogram([2, 5, 7]) should print:

```
**
*****
*******
```

Assignment 9: Write a function to get the fibinocci series of numbers till 100; and another function to print the corresponding number of astericks. Hint: 0, 1, 1, 2, 3, 5, 8, ...

```
Output expected :
    *
    *
    **
    ***
    *****
    ********
    ...so on
```

Assignment 10: Write a function to calculate the factorial of a number.

```
Hint: factorial(4) = 4*3*2*1 = 24
```

Assignment 11: Using recursive functions, generate the fibonacci series and display all the series still a given number

Assignment 12: using timeit module, conclude which of the above two factorial() functions, is faster?

Assignment 13: Modify this stringReverse() recursive function to result as below:

'Python Programming' -> 'nohtyP gnimmargorP'

Assignmnet 14: Write a recursive function to compute the sum of first n whole number

Hint : 10 -> 10+9+8+ .....1+0
if n is 0, return 0

Assignment 15: Write a recursive function which displays the Pascal's triangle:

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5   10  10  5   1
```

Assignments 16 (Interview Question): Implement a recursive function to multiply 2 numbers recursively using + and - operators only.

Assignment 17: Write a recursive function to display first 50 prime numbers

Assignment 18: WAP to display the sum of digits of a number Hint: n//10, n%10

Assignment 19: Execute the below Script in a new .py file, and observe the result

```
#!/usr/bin/python
# listDirsNfilesUsingRecursions.py
# Purpose: To list the directories and files, in the given location
import os

def get_dirlist(path):
    """
      Return a sorted list of all entries in path.
      This returns just the names, not the full path to the names.
    """
    dirlist = os.listdir(path)
    dirlist.sort()
    return dirlist

def print_files(path, prefix = ""):
    """ Print recursive listing of contents of path """
    if prefix == "":  # Detect outermost call, print a heading
        print("Folder listing for", path)
        prefix = "| "

    dirlist = get_dirlist(path)
    for f in dirlist:
        print(prefix+f)                     # Print the line
        fullname = os.path.join(path, f)    # Turn name into full pathname
        if os.path.isdir(fullname):         # If a directory, recurse.
            print_files(fullname, prefix + "| ")

print_files(r'C:\Python27\Tools')
```

Functions

- To reduce the code duplication for repetive logic
- Modularizing the complex problem, into simpler pieces.
- For better code reusage
- Information Hiding
- Functions in Python are called **First-class citizens.**
  - Function object have equal tsatus as other objects
  - Functions too, can be assigned to variables, stored in collections (list, tuple, dictionary, set) or can be passed as arguments.
- Functions are of two types:
  1. Built-functions
  2. user-defined functions

Syntax:

```
def functionName(argument1, arguments2, ... ):
    statement1
    statement2
    ....
    return expression/Value  # return is optional, None object goes when return is
 not present.
```

Functions without input arguments

In [4]:
```
def helloWorld():                # function definition
    print "Hello World!!!"
```

In [5]:
```
helloWorld()                    # function call
```

```
Hello World!!!
```

In [6]:
```
hw = helloWorld()              # assigning the result of function call
```

```
Hello World!!!
```

In [7]:
```
print hw, type(hw)
```

```
None <type 'NoneType'>
```

Problem 1: write a function to get all the team members for a new project in run-time, and display them along with their count.

In [11]:
```
def teamMembers():
    team = raw_input('Enter the names of team members: ')
    print team, type(team)
    print len(team)

teamMembers()                # function call
```

```
Enter the names of team members: Yash, ropit, Eash, benerji
Yash, ropit, Eash, benerji <type 'str'>
26
```

In [13]:
```
def teamMembers():
    team = raw_input('Enter the names of team members: ')
    #print team, type(team)
    #print len(team)
    teamNames = team.split(',')
    print teamNames, len(teamNames)

teamMembers()                # function call
```

```
Enter the names of team members: Yash, ropit, Eash Parikaar, benerji
['Yash', ' ropit', ' Eash Parikaar', ' benerji '] 4
```

Functions with inputs

Problem 2: Write a script to print the greater number, among two numbers

In [14]:
```
def greaterNumber(a,b):
    if a>b:
        print "%d is greater than %d"%(a,b)
    elif a<b:
        print "%d is lesser than %d"%(a,b)
    else:
        print "%d is equal to %d"%(a,b)

greaterNumber(-32, -12)
```

```
-32 is lesser than -12
```

Problem 3: Displaying a given name

In [16]:
```
def displayName(name):
    print "The name of the candidate is %r"%(name)

displayName('Yash')
```

```
The name of the candidate is 'Yash'
```

Scope: Local and Global Variables

In [18]:
```
a = 10
def localEx(a):
    print "In localEx() function, a = %d"%(a)
    a= 5
    print "In localEx() function, a = %d"%(a)

localEx(a)
print "Outside: a = %d"%(a)
```

```
In localEx() function, a = 10
In localEx() function, a = 5
Outside: a = 10
```

Interefernce: Changes made to the variable within function, does affect outside it

global : It is a keyword to reflect the local changes (within the function) to the global level

In [24]:
```
global a             # global declaration
a = 10
def globalEx():             # Observe that 'a' is not passed as input
    global a         # global declaration
    print "In globalEx() function, a = %d"%(a)
    a= 5
    print "In globalEx() function, a = %d"%(a)

localEx()
print "Outside: a = %d"%(a)
```

```
In localEx() function, a = 10
In localEx() function, a = 5
Outside: a = 5
```

Interview Question 1: Explain the difference between global and local values, with an example?

Default and Keyword arguments

In [25]:
```
def hello(name = "World!!!"):
    print "Hello ", name
```

In [26]:
```
hello()
```

```
Hello  World!!!
```

In [27]:
```
hello('Yash')     # there is only one input
```

```
Hello  Yash
```

In [28]:
```
hello(name = 'Yash')
```

```
Hello  Yash
```

In [29]:
```
def cricket(balls = 3, bats = 2):
    print "There are %d balls and %d bats"%(balls, bats)
```

In [30]:
```
cricket()
```

```
There are 3 balls and 2 bats
```

In [31]:
```
cricket(4,5)    # taken based on position
```

```
There are 4 balls and 5 bats
```

In [32]:
```
cricket(balls = 5, bats= 4)   # taken based on assignment
```

```
There are 5 balls and 4 bats
```

In [35]:
```
def nTimesDisplay(message, noOfTimes = 3):
    print message*noOfTimes
```

Here, as only one argument is default, the other should be given mandatorily.

The default arguments must be placed in the last in the function definition.

In [36]:
```
nTimesDisplay('Python ')
```

```
Python Python Python
```

In [37]:
```
nTimesDisplay('Python ', 5)
```

```
Python Python Python Python Python
```

In [38]:
```
nTimesDisplay(5,'Python ')
```

```
Python Python Python Python Python
```

In [40]:
```
def funcExpressions(a,b = 12, c = 123):
    print "The value of a is %d"%(a)
    print "The value of b is %d"%(b)
    print "The value of c is %d"%(c)
```

In [42]:
```
funcExpressions(12)
```

```
The value of a is 12
The value of b is 12
The value of c is 123
```

In [43]:
```
funcExpressions(12, c= 999)
```

```
The value of a is 12
The value of b is 12
The value of c is 999
```

In [44]:
```
funcExpressions(c = 123, a = 234)
```

```
The value of a is 234
The value of b is 12
The value of c is 123
```

return statement

- By default, user-defined functions will return 'None'

In [45]:
```
def evenOddTest(a):
    result = a%2
    if result == 0:
        return '%d is an even number'%(a)
    else:
        return '%d is an odd number'%(a)
```

In [46]:
```
evenOddTest(223)
```

Out[46]:
```
'223 is an odd number'
```

In [47]:
```
print evenOddTest(223)
```

```
223 is an odd number
```

In [48]:
```
eot = evenOddTest(223)
```

In [49]:
```
print eot, type(eot)
```

```
223 is an odd number <type 'str'>
```

DocStrings

- Essential for specifying something about the function
- Enclosed within ''' ''' or """ """
- Docstrings are different from comments

In [51]:
```
def evenOddTest(a):
    '''
    Purpose: To validate the even-ness or odd-ness of a given integer.
    Input: Variable a
    Input Type: Integer
    Output: result statement
    Output Type: String
    '''
    result = a%2
    if result == 0:
        return '%d is an even number'%(a)
    else:
        return '%d is an odd number'%(a)
```

In [52]:
```
print evenOddTest.func_doc
```

```
    Purpose: To validate the even-ness or odd-ness of a given integer.
    Input: Variable a
    Input Type: Integer
    Output: result statement
    Output Type: String
```

Variable Arguments (*args and **kwargs)

In [59]:
```
#!/usr/bin/python

'''
    Purpose: Demonstration of the usage of *args and **kwargs

    *args   stores variables in tuple
    ** kwargs stores variables in dictionaries

'''
def foo(firstArg, *args, **kwargs):
    print 'Necessary argument is ', firstArg
    print 'args = ', args
    print 'type(args) is', type(args)
    if args:
        for arg in args:
            print arg

    print 'kwargs = ', kwargs
    print 'type(kwargs) is', type(kwargs)
    if kwargs:
        print "The keyword arguments are\n",
        # for kwarg in kwargs:
        #    print kwarg
        for k,v in kwargs.items():
            print '%15r  --> %10r'%(k,v)

if __name__ == "__main__":
    print '\n','_'*70
    foo(321)
    print '\n','_'*70
    foo(1,2,3,4)
    print '\n','_'*70
    foo(99,22.0, True, [12,34, 56])
    print '\n','_'*70
    foo(32, 56, 32, (2, [34, (2.3, 99, 0)]))
    print '\n','_'*70
    foo(2.0, a = 1, b=2, c=3)
    print '\n','_'*70
    foo('a', 1, None, a = 1, b = '2', c = 3)
    print '\n','_'*70
    foo(123, courseName = 'Python', currentStatus = '40% completed', studentLi
st = ['Yash', 'Raj', 'Johson'])
```

```
Necessary argument is  321
args =  ()
type(args) is <type 'tuple'>
kwargs =  {}
type(kwargs) is <type 'dict'>
```

```
Necessary argument is  1
args =  (2, 3, 4)
type(args) is <type 'tuple'>
2
3
4
kwargs =  {}
type(kwargs) is <type 'dict'>
```

```
Necessary argument is  99
args =  (22.0, True, [12, 34, 56])
type(args) is <type 'tuple'>
22.0
True
[12, 34, 56]
kwargs =  {}
type(kwargs) is <type 'dict'>
```

```
Necessary argument is  32
args =  (56, 32, (2, [34, (2.3, 99, 0)]))
type(args) is <type 'tuple'>
56
32
(2, [34, (2.3, 99, 0)])
kwargs =  {}
type(kwargs) is <type 'dict'>
```

```
Necessary argument is  2.0
args =  ()
type(args) is <type 'tuple'>
kwargs =  {'a': 1, 'c': 3, 'b': 2}
type(kwargs) is <type 'dict'>
The keyword arguments are
            'a'  -->          1
            'c'  -->          3
            'b'  -->          2
```

```
Necessary argument is  a
args =  (1, None)
type(args) is <type 'tuple'>
1
None
kwargs =  {'a': 1, 'c': 3, 'b': '2'}
type(kwargs) is <type 'dict'>
The keyword arguments are
```

```
            'a'  -->          1
            'c'  -->          3
            'b'  -->        '2'
```

```
Necessary argument is  123
args =  ()
type(args) is <type 'tuple'>
kwargs =  {'courseName': 'Python', 'currentStatus': '40% completed', 'student
List': ['Yash', 'Raj', 'Johson']}
type(kwargs) is <type 'dict'>
The keyword arguments are
   'courseName'  -->   'Python'
'currentStatus'  --> '40% completed'
  'studentList'  --> ['Yash', 'Raj', 'Johson']
```

Lambda

In [61]:
```
def printSquares(x):
    for i in range(x):
        print i,":",i**2

printSquares(7)
```

```
0 : 0
1 : 1
2 : 4
3 : 9
4 : 16
5 : 25
6 : 36
```

In [62]:
```
[lambda x:x**2  for x in range(7)]
```

Out[62]:
```
[<function __main__.<lambda>>,
 <function __main__.<lambda>>,
 <function __main__.<lambda>>,
 <function __main__.<lambda>>,
 <function __main__.<lambda>>,
 <function __main__.<lambda>>,
 <function __main__.<lambda>>]
```

In [63]:
```
map(lambda x:x**2, range(7))   # map() builtin function, that takes function a
nd list as input
```

Out[63]:
```
[0, 1, 4, 9, 16, 25, 36]
```

In [64]:
```
filter(lambda x:x%2 == 0, xrange(12))
```

Out[64]:
```
[0, 2, 4, 6, 8, 10]
```

Assignment 1: Using the above written logic, write evenOddTest function within the map() and get the even values between 23 and 97

map()

In [65]:
```
def double(number):
    return number*2  # experession isn't returned, but the result is
```

In [66]:
```
map(double, range(45, 56))
```

Out[66]:
```
[90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110]
```

filter()

In [67]:
```
def OddNessTest(number):
    return number%2 != 0
```

In [68]:
```
listOfValues = [12, 34, 56, 78, 923, 23]
```

In [69]:
```
filter(OddNessTest, listOfValues)
```

Out[69]:
```
[923, 23]
```

In [70]:
```
map(OddNessTest, listOfValues)
```

Out[70]:
```
[False, False, False, False, True, True]
```

Interview Question 2: Write a function to highlist the vowels in the input string

In [71]:
```
def highlightVowels(string):
    vowels = 'ae1ouAEIOU'
    for i in string:
        if i in vowels:
            print i.upper(),
        else:
            print i.lower(),
```

In [72]:
```
highlightVowels('I will give my best to achieve my goal!')
```

```
I   w i l l   g i v E   m y   b E s t   t O   A c h i E v E   m y   g O A l !
```

Assignment 2: Do modifications to this logic, to remove the spaces between letters; not words

In [73]:
```
def highlightVowels(string):
    vowels = 'ae1ouAEIOU'
    newString = ""
    for i in string:
        if i in vowels:
            #print i.upper(),
            newString+=i.upper()
        else:
            #print i.lower(),
            newString+=str(i.lower())
    return newString
```

In [74]:
```
highlightVowels('I will give my best to achieve my goal!')
```

Out[74]:
```
'I will givE my bEst tO AchiEvE my gOAl!'
```

zip()

- To pair the list given
- Results in list of tuples

In [82]:
```
zip('Python', 'Python')
```

Out[82]:
```
[('P', 'P'), ('y', 'y'), ('t', 't'), ('h', 'h'), ('o', 'o'), ('n', 'n')]
```

In [83]:
```
zip('python', xrange(len('python')))
```

Out[83]:
```
[('p', 0), ('y', 1), ('t', 2), ('h', 3), ('o', 4), ('n', 5)]
```

In [84]:
```
zip(xrange(len('python')), 'Python')
```

Out[84]:
```
[(0, 'P'), (1, 'y'), (2, 't'), (3, 'h'), (4, 'o'), (5, 'n')]
```

In [85]:
```
zip('Python')
```

Out[85]:
```
[('P',), ('y',), ('t',), ('h',), ('o',), ('n',)]
```

Interview Question 3: What is the difference between map() and zip()?

In [86]:
```
print map(None, range(10), range(12))    # None type function taken to pair th
e values in list
```

```
[(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7), (8, 8), (9,
 9), (None, 10), (None, 11)]
```

In [87]:
```
print zip(range(10), range(12))
```

```
[(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7), (8, 8), (9,
 9)]
```

Observe that map() can handle lists of assymmetric lengths, whereas zip() can't

reduce()

- It accepts an iterator to process, but it's not an iterator itself. It returns a single result

In [88]:
```
reduce( (lambda a, b: a * b), [1, 2, 3, 4] )   # results in 1*2*3*4
```

Out[88]: 24

In [89]:
```
reduce( (lambda a, b: a ** b), [1, 2, 3, 4] )   # results in 1**2**3**4; anyth
ing to the power of 1 results in 1
```

Out[89]: 1

In [90]:
```
reduce( (lambda a, b: a ** b), [2, 3, 4] )    # results in 2**3**4
```

Out[90]: 4096

Interview Question 4: Perform string concatenation using reduce()

In [92]:
```
string = 'I am confident about my Success'
list = string.split(' ')
print list
```

```
['I', 'am', 'confident', 'about', 'my', 'Success']
```

In [93]:
```
reduce(lambda a,b: a+b, list)
```

Out[93]: 'IamconfidentaboutmySuccess'

In [95]:
```
reduce(lambda a,b: a+' '+b, list)   # with spaces between words
```

Out[95]: 'I am confident about my Success'

Assignment 3: Write a script two functions, celsiusToFahrenheit() and fahrenheitToCelsius(). In runtime, the user should be given choice to enter either Fahrenheit or Celcius; then use the functions to convert correspondingly. Then, display the result from main function, using string formatting?

Assignment 4: Write three functions absolute(). Prompt the user to feed an integer (). The result should mimick the functionality of built-in function abs()

```
ex:
    In [1]: abs(9)
    Out[1]: 9

    In [2]: abs(-9)
    Out[2]: 9

    In [3]: abs(0)
    Out[3]: 0

    In [4]: abs(2+3j)
    Out[4]: 3.6055512754639896
```

Assignment 5: Write a function that validates and prints whether the run-time input given is palindrome or not.

```
Ex: 'otto'
    'Evil is a deed as I live' Hint: ignore white-space here.
```

Assignment 6: write a function that generates palindromes, for the given run-time input. There should be two outputs, both for even and odd case. Ref : http://www.jimsabo.com/palindrome.html (http://www.jimsabo.com/palindrome.html).

Assignment 7: Write a function to implement the caesar cipher, for the given input.

```
    ex1: input  --> 'Python'
        output --> 'Qzuipo'
    ex2: input -->  'One day, I will achieve my goal!'
         output ->  'Pof!ebz-!J!xjmm!bdijfwf!nz!hpbm"'
```

Assignment 8: Define a procedure histogram() that takes a list of integers and prinst a histogram to the screen. For example, histogram([2, 5, 7]) should print:

```
**
*****
*******
```

Assignment 9: Write a function to get the fibinocci series of numbers till 100; and another function to print the corresponding number of astericks. Hint: 0, 1, 1, 2, 3, 5, 8, ...

```
Output expected :
    *
    *
    **
    ***
    *****
    ********
    ...so on
```

Assignment 10: Write a function to calculate the factorial of a number.

Recursive functions

```
def funcName(<input paramaters>):
    <some logic>
    return funcName(<input parameters>)
```

Recursion is a programming technique in which a call to a function results in another call to that same function. Iteration is calling an object, and moving over it.

Problem 1: Return teh fibonacci series (0, 1, 1, 2, 3, 5, 8) using recursions

In [1]:
```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1)+fib(n-2)
```

In [3]:
```
fib(5)    # 5th element    # fib(4)+fib(3)
                           # fib(4) -> fib(3)+fib(2)
                           #             fib ...
```

Out[3]: 5

In [4]:
```
fib(6)   # 6th element
```

Out[4]: 8

Assignment 11: Using recursive functions, generate the fibonacci series and display all the series still a given number

Problem 2: Write a recursive function to get the factorial of a given number

In [5]:
```
def factorial(n):
    if n==0:
        return 1
    else:
        return n*factorial(n-1)
```

In [6]:
```
factorial(5) # 5*3*2*1*1
```

Out[6]: 120

In [7]:
```
def factorial(n):                                # same logic, but consolidated
    return 1 if n==0 else n*factorial(n-1)
```

In [8]:
```
factorial(5)
```

Out[8]: 120

Assignment 12: using timeit module, conclude which of the above two factorial() functions, is faster?

Problem 3: write a recursive function to display the reverse of a given string, using recursive function.

ex:'Python' -> 'nohtyP'

In [9]:
```
def stringReverse(string):
    if string == '':
        return ''
    else:
        return stringReverse(string[1:]) + string[0]
```

In [10]:
```
stringReverse('Python Programming')
```

Out[10]: 'gnimmargorP nohtyP'

In [11]:
```
stringReverse('Programming')
```

Out[11]: 'gnimmargorP'

Assignment 13: Modify this stringReverse() recursive function to result as below:

'Python Programming' -> 'nohtyP gnimmargorP'

NOTE: We also know that string reversal is possible using slicing (str[::-1]) and using built-in function reversed()

Problem 3: Write a recursive function to display the sum of squares of whole numbers, till given number, n.

```
1**2 +2**2 + 3**2 + ...
```

In [12]:
```
def squareSum(n):
    if n == 0:
        return 0
    else:
        return pow(n,2) + squareSum(n-1)
```

In [13]:
```
squareSum(4)       #   0**2 + 1**2 + 2**2 + 3**2 + 4**2
```

Out[13]: 30

Assignmnet 14: Write a recursive function to compute the sum of first n whole number

Hint : 10 -> 10+9+8+ .....1+0
if n is 0, return 0

Assignment 15: Write a recursive function which displays the Pascal's triangle:

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
    1   5   10  10  5   1
```

Assignments 16 (Interview Question): Implement a recursive function to multiply 2 numbers recursively using + and - operators only.

Assignment 17: Write a recursive function to display first 50 prime numbers

Assignment 18: WAP to display the sum of digits of a number Hint: n//10, n%10

Problem 4: Create an infinite loop using recursive functions

In [14]:
```
noOfRecursions = 0
def loop(noOfRecursions):                          # Infinite loop
    print 'Hi! I am in Loop '
    noOfRecursions+=1              # to get the count of number of recursions
 occurred
    print 'This is Loop %d'%noOfRecursions
    return loop(noOfRecursions)
```

In [15]:
```
loop(noOfRecursions)
```

```
Hi! I am in Loop
This is Loop 1
Hi! I am in Loop
This is Loop 2
Hi! I am in Loop
This is Loop 3
Hi! I am in Loop
This is Loop 4
Hi! I am in Loop
This is Loop 5
Hi! I am in Loop
This is Loop 6
Hi! I am in Loop
This is Loop 7
Hi! I am in Loop
This is Loop 8
Hi! I am in Loop
This is Loop 9
Hi! I am in Loop
This is Loop 10
Hi! I am in Loop
This is Loop 11
Hi! I am in Loop
This is Loop 12
Hi! I am in Loop
This is Loop 13
Hi! I am in Loop
This is Loop 14
Hi! I am in Loop
This is Loop 15
Hi! I am in Loop
This is Loop 16
Hi! I am in Loop
This is Loop 17
Hi! I am in Loop
This is Loop 18
Hi! I am in Loop
This is Loop 19
Hi! I am in Loop
This is Loop 20
Hi! I am in Loop
This is Loop 21
Hi! I am in Loop
This is Loop 22
Hi! I am in Loop
This is Loop 23
Hi! I am in Loop
This is Loop 24
Hi! I am in Loop
This is Loop 25
Hi! I am in Loop
This is Loop 26
Hi! I am in Loop
This is Loop 27
Hi! I am in Loop
This is Loop 28
Hi! I am in Loop
```

```
This is Loop 941
Hi! I am in Loop
This is Loop 942
Hi! I am in Loop
This is Loop 943
Hi! I am in Loop
This is Loop 944
Hi! I am in Loop
This is Loop 945
Hi! I am in Loop
This is Loop 946
Hi! I am in Loop
This is Loop 947
Hi! I am in Loop
This is Loop 948
Hi! I am in Loop
This is Loop 949
Hi! I am in Loop
This is Loop 950
Hi! I am in Loop
This is Loop 951
Hi! I am in Loop
This is Loop 952
Hi! I am in Loop
This is Loop 953
Hi! I am in Loop
This is Loop 954
Hi! I am in Loop
This is Loop 955
Hi! I am in Loop
This is Loop 956
Hi! I am in Loop
This is Loop 957
Hi! I am in Loop
This is Loop 958
Hi! I am in Loop
This is Loop 959
Hi! I am in Loop
This is Loop 960
Hi! I am in Loop
This is Loop 961
Hi! I am in Loop
This is Loop 962
Hi! I am in Loop
This is Loop 963
Hi! I am in Loop
This is Loop 964
Hi! I am in Loop
This is Loop 965
Hi! I am in Loop
This is Loop 966
Hi! I am in Loop
This is Loop 967
Hi! I am in Loop
This is Loop 968
Hi! I am in Loop
This is Loop 969
```

```
Hi! I am in Loop
This is Loop 970
Hi! I am in Loop
This is Loop 971

---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-15-479a6f6d0cf4> in <module>()
----> 1 loop(noOfRecursions)

<ipython-input-14-5aebfb03032c> in loop(noOfRecursions)
      4     noOfRecursions+=1             # to get the count of number of re
cursions occurred
      5     print 'This is Loop %d'%noOfRecursions
----> 6     return loop(noOfRecursions)

... last 1 frames repeated, from the frame below ...

<ipython-input-14-5aebfb03032c> in loop(noOfRecursions)
      4     noOfRecursions+=1             # to get the count of number of re
cursions occurred
      5     print 'This is Loop %d'%noOfRecursions
----> 6     return loop(noOfRecursions)

RuntimeError: maximum recursion depth exceeded while calling a Python object
```

In [16]:
```
print noOfRecursions   # It results the default value given before entering th
e function, as returns from subframes was halted.
```

```
0
```

Notice that in python, infinite loop will not run for infinite time. But, they gets halted after reaching the maximum recursion depth.

Interview question 5: what is the maximum recursion depth of any python object? Does it differ from object to object? Is it dependent on the machine being used?

Problem 5: Write a function to demonstrate the MUTUAL recursion between two functions

In [17]:
```
# Infinite loop between these functions  --- Mutual recursion
def func1():
    print 'I am in function 1 .'
    return func2()

def func2():
    print 'I am in function 2 .'
    return func1()
```

In [18]:
```
func1()
```

```
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
I am in function 2 .
I am in function 1 .
```

```
I am in function 2 .
I am in function 1 .

---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-18-043607b7b3c9> in <module>()
----> 1 func1()

<ipython-input-17-a0185b57d968> in func1()
      2 def func1():
      3     print 'I am in function 1 .'
----> 4     return func2()
      5
      6 def func2():

<ipython-input-17-a0185b57d968> in func2()
      6 def func2():
      7     print 'I am in function 2 .'
----> 8     return func1()

... last 2 frames repeated, from the frame below ...

<ipython-input-17-a0185b57d968> in func1()
      2 def func1():
      3     print 'I am in function 1 .'
----> 4     return func2()
      5
      6 def func2():

RuntimeError: maximum recursion depth exceeded while calling a Python object
```

NOTE :

- Python doesnot have Tail Call optimization(TCO), to handle the recursive functions.
- It is very difficult to add TCO to python, as it is a dynamic language.

Assignment 19: Execute the below Script in a new .py file, and observe the result

```
#!/usr/bin/python
# listDirsNfilesUsingRecursions.py
# Purpose: To list the directories and files, in the given location
import os

def get_dirlist(path):
    """
      Return a sorted list of all entries in path.
      This returns just the names, not the full path to the names.
    """
    dirlist = os.listdir(path)
    dirlist.sort()
    return dirlist

def print_files(path, prefix = ""):
    """ Print recursive listing of contents of path """
    if prefix == "":  # Detect outermost call, print a heading
        print("Folder listing for", path)
        prefix = "| "

    dirlist = get_dirlist(path)
    for f in dirlist:
        print(prefix+f)                     # Print the line
        fullname = os.path.join(path, f)    # Turn name into full pathname
        if os.path.isdir(fullname):         # If a directory, recurse.
            print_files(fullname, prefix + "| ")

print_files(r'C:\Python27\Tools')
```

In **conclusion**, note that recursive calls are expensive (inefficient) as they take up a lot of memory and time. Recursive functions are hard to debug.

Recommended References:

1. Animated explanation of recursions (http://www.composingprograms.com/pages/17-recursive-functions.html)