Chapter 3 Working with Functions

Ex 3.1 Solutions

1. If return statement is not used inside the function, the function will return:

(a) 0
(b) None object
(c) an arbitrary integer
(d) Error! Functions in Python must have a return statement

b) None object

2. Which of the following keywords marks the beginning of the function block?

(a) func
(b) define
(c) def
(d) function

c) def keyword marks the beginning of the function block

3. What is the area of memory called, which stores the parameters and local variables of a function call?

(a) a heap
(b) storage area
(c) a stack
(d) an array

c) a stack

4. Find the errors in following function definitions:

```
(a) def main( )
        print ("hello")
(b) def func2() :
        print (2 + 3)
(c) def compute( ) :
        print (x • x)
(d) square (a)
        return (a)
```

```
a)def main ( ) # ':' missing

        print( " hello" )
```

b) def func2():
        print (2+3)
no errors
c) def compute ():
        print(x*x)
# x is not mentioned in local scope, it must be in the global scope
d) square (a)
        return a*a

'def' keyword is missing, ':' is missing

Back Exercise Part A

1. A program having multiple functions is considered better designed than a program without any functions. Why?

Functions are reusable piece of code which can be used inside a program by calling it. Programmers should adhere to the DRY (Don't repeat yourself) principle which is utilised using functions. It is better to use functions rather than writing the same code multiple times in a program. It helps in better understanding of the code. Thus, program having multiple functions is considered better designed than a program without any functions.

2. What all information does a function header give you about the function?

The function header is the first line of a function definition. It begins with the def' keyword and ends with a colon specifies the name of the function and its parameters which are variables that are listed within the parentheses of a function header. e.g.

def add(x, y): # function header

        return x+y

3. What do you understand by flow of execution?

The flow of execution refers to the order in which statements are executed during a program run. Statements are executed one at a time, in order from top to bottom. Function definitions do not alter the flow of execution of the program, but remember that statements inside the function are not executed until the function is called. e.g.

def func():

        return # the control is sent back to wherefrom the function was called

func ( ) # function is called

print(...)

4. What are arguments? What are parameters? How are these two terms different yet related? Give example.

The values being passed through a function-call statement are called arguments (or actual parameters or actual arguments). The values received in the function definition/header are called parameters (or formal parameters or formal arguments). e.g.

def add(x, y): # x & y are parameters

```
        return X+Y

add(2, 3) # 2 & 3 are arguments
```

5. What is the utility of:

i) default arguments

Python allows us to assign default value(s) to a function's parameter(s) which is useful in case a matching argument is not passed to the function call statement. The default values are specified in the function header of function definition. They are useful if one parameter is usually fixed. e.g.

```
def simple_interest(p, r, t=l) : # here 't' is the default argument

        return (p*t*r/100)
```

ii) keyword arguments

Keyword arguments are the named arguments with assigned values being passed in the function call statement. It lets us change the order of the arguments which are being passed. e.g.

```
def simple_interest(p, r, t=l) # here 't' is the default argument

        return print (simple_interest( t=5, r=10, p=10000) # function call with keyword arguments
```

7. Describe the different styles of functions in Python using appropriate examples.

Different styles of python functions are:
i) Built- in function: These are pre-defined functions and are always available for use. e.g. len(), type (), int () etc.
ii) Functions defined in modules: These functions are pre-defined in particular modules and can only be used when the corresponding module is imported. e.g. sin() function of math module
iii) User defined functions

```
def add (x, y):
        return x+y
```

8. Differentiate between fruitful functions and non-fruitful functions.

Fruitful functions are those which return a value. e.g.

```
def add (x, y):
        return x+y
```

Non-fruitful functions do not return a value. Instead the None object is returned.
e .g.

```
def add (x, y): # No value is being returned
        print ( x+y)
```

9. Can a function return multiple values? How?

Yes, python allows you to return multiple values from a function. The values can be received in a tuple variable or equal number of individual variables. e.g.

```
def square_cube(x):

        return x*x, x*x*x

        s, c = square_cube (5)

print ('square of 5 is', s)

print ('cube of 5 is', c)
```

11. What is the difference between local and global variables?

Local variables are those which are declared inside a function body. They can only be used within this function. Variables declared in the top level
segment (_ main_) of a program is said to have a global scope. They can be used anywhere and even inside functions, if the function doesn't have a local variable with the same name. e.g.

```
def func (x):
        y = 100 # Loca1 variable
        print (num) # Global variable being used inside a function
        return x/y
# __main__

num = 10000 # Global variable
print (func(num))
```

12. When is global statement used? Why is its use not recommended?

'global' statement is used when we want to modify a global variable inside a local scope. e .g.

```
def func (x, y):
        global num1
        num1 = num1/100 # the global variable' numl' is being modified
        print (x+y - num2) # the global variable 'num2' is not being modified
# we have not called 'num2' with global because it is not being modified
num1 = l00
num2 = l00
func( 500 , 600 )
```

Once a variable is defined global in a function, it cannot be undone, i.e. after the global statement, the function will always refer to the global variable and local variable cannot be created of the same name. Thus, the 'global' statement is not recommended.

13. Write the term suitable for following descriptions:

(a) A name inside the parentheses of a function header that can receive a value.
(b) An argument passed to a specific parameter using the parameter name.
(c) A value passed to a function parameter.
(d) A value assigned to a parameter name in the function header.
(e) A value assigned to a parameter name in the function call.
(f) A name defined outside all function definitions.
(g) A variable created inside a function body.

(a) A name inside the parentheses of a function header that can receive a value.

parameter

(b) An argument passed to a specific parameter using the parameter name.

keyword argument

(c) A value passed to a function parameter.

argument

(d) A value assigned to a parameter name in the function header.

default parameter

(e)A value assigned to a parameter name in the function call.

keyword argument

(f) A name defined outside all function definitions.

global variable

(g) A variable created inside a function body.

local variable

10. What is scope? What is the scope resolving rule of Python?

The program part(s) in which a particular piece of code or a data value (e.g., variable) can be accessed is known as variable scope. The scope resolving rule of python is called LEGB rule. It checks environments in the order: Local, Enclosing, Global and Built-in.
The two main scopes are local and global. In simple terms, variables declared inside The function have local scope while variables declared in top level segment(main) of a program is said to have a global scope.

Back Exercise Part B

1. What are the errors in following codes? Correct the code and predict output:

(a) total=0;
def sum(argl, arg2) :
total = arg1+ arg2;
print("Total: " , total)
return total
sum(10, 20)
print("Total: " , total)
	(b)
	def Tot (Number) #Method to find Total
	Sum = 0
	for C in Range (1, Number + 1):
	Sun += C
	RETURN sum
	print (Tot[3]) #Function Calls
	print (Tot[6])

a)

```
total = 0;                  # semi-colons are not required in python, although won't raise an error
    def sum( arg1, arg2 ):  # indentation was not needed
    total = arg1 + arg2;    # this 'total' is in local scope, won't change the value of global tuple
    print("Total :", total)
return total;               # should've been inside the function block
sum( 10., 20 );
print("Total :", total)     # this will print 0, because the value of global total is 0
```

```
# Solution

def sum(arg1, arg2):
    total = arg1+arg2
    print("Total :", total)
    return total

total = sum(10, 20)
print("Total :", total)
```

```
b)
def Tot(Number)      #Method to find Total,  # colon ':' is missing
    Sum = 0
    for C in Range (1, Number + 1) :        # Range should be 'range'
        Sum += C
    RETURN Sum                              # RETURN should be return
print (Tot[3])       #Function Calls,    # function call should be Tot(3)
print (Tot[6])           # function call should be Tot(6)
```

```
# Solution
def Tot (Number):
sum = 0
for C in range (1, Number+1):
        Sum += C
        return Sum
print (Tot (3))
print (Tot (6))
```

2. Consider the following code and write the flow of execution for this. Line numbers have been given for your reference.

```
1 def power (b, p):
2         return y=b**p
3 def calcSquare(x) :
4          a = power(x, 2)
5          return a
6 n=5
7 result = calcSquare(n)
8 print (result)
```

The control starts at line 6. Then on line 7 control goes to the function on line 3. line 3 to line 5 are executed and control goes back to line 7 after the return statement and then line 8 is executed. So
line 6
line 7
goes to line 3
line 4
line 1
line 2
line 5
back to line 7

3. What will the following function return?

```
def addEm(x, y, z):

            print(x + y + z)
```

The above function will return the "None" object.

4. What will the following function print when called?

```
def addEm(x, y, z):
```

```
    return x + y + z

    print(x + y + z)
```

The following function will print nothing because the print statement is after the return statement. The program will return back the control to the function call after the return statement.

5. What will be the output of following programs?

(i) num = 1 def myfunc() : return num print (nun) print (myfunc()) print(num) # Output 1 1 # the function is using the global 'num' here 1	(ii) num = 1 def myfunc() : nun = 10 return num print (num) print (myfunc()) print (num) # Output 1 10 1 # the local 'num' is being used # the global 'num' is being used # The local 'num' and global 'num' are two different variables
(iii) num = 1 def myfunc(): global num num =10 return num print(num) print (myfunc()) print(num) # Output 1 # the value of 'num' has not been changed 10 # the value of global ' num' modified 10 # Here, there is just the global 'num' whose value is being modified by the function	(iv) def display() print("Hello", end="") display() print("there ! ") # Output Hello there! # because end='` was used in the last print statement

6. Predict the output of the following code:

```
a = 10
y = 5
def myfunc() :
  y=a
  a=2
  print(“y = “, y, “a= “, a)
  print(“a+y = “, a+y)
  return a + y
print(“y = “, y, “a= “, a)
print(myfunc())
print(“y = “, y, “a= “, a)
# Output
```

UnboundLocalError: local variable ‘a' referenced before assignment

This is because we are assigning y the value of a before it has been declared in the local scope. If we were to not at all declare the local 'a', no error would've been raised and the global a's value would 've been assigned to the local y.

7. What is wrong with the following function definition?

```
def addEm(x, y, z):
    return x+y+z
    print( "the answer is ‟,x+y+z)
```

The print statement is after the return statement and so it would never get executed. The correct way is

```
def addEm(x, y, z):
      print( "the answer is ‟, x+y+z)
      return x+y+z
```

8. Write a function namely fun that takes no parameters and always returns None.

 # Code

```
def fun(): # No parameters
     pass # Nothing to execute, no return statement so None will be returned
print(fun()) # None
```

9. Consider the code below and answer the questions that follow:

```
def multiply(number1, number2) :
        answer = number1*number2
        print (number1, 'times', number2, answer)
```

```
        return (answer)
output = multiply (5, 5)
```
(i) When the code above is executed, what prints out?

        5 times 5 = 25

(ii) What is variable output equal to after the code is executed?

        25

10. Consider the code below and answer the questions that follow:

```
def multiply(number1, number2) :
        answer = number1*number2
        return (answer)
        print (number1, 'times', number2, answer)
output = multiply (5, 5)
```
(i) When the code above is executed, what gets printed?

Nothing gets printed as the print statement is after the return statement

(ii) What is variable output equal to after the code is executed?

        25

11. Find the errors in code given below:

(a)
```
def minus (total, decrement) # ' : ' Missing
output = total - decrement
print (output)
return (output)
```

(b)
```
define check() # ' def' should be used instead of define , ':' is missing
N = input('Enter N:')
i=3
answer = 1 + i**4/N
Return answer # ' return' should be used instead of ' Return '
```
(c)
```
def alpha (n, string ='xyz', k=10):
        return beta (string)
        return n          # this return statement will never be executed
def beta (string)        #':' is missing
        return string str(n) # n is not defined in either local or global scope

print(beta(string 'true'))
print(alpha("Valentine's Day")) # ':' outstide quotes
print(beta(string = 'true'))
print(alpha(n = 5, "Good-bye")) # ':' outstide quotes,
```
#positional argument follows keyword arguments

12. Draw the entire environment, including all user-defined variables at the time line 10 is being executed

1. def sum(a, b, c, d) :

```
2.      result =0
3.      result = result + a + b + c d
4.      return result
5.
6.  def length():
7.      return 4
8.
9.  def mean(a, b, c, d):
10.     return float(sum (a, b, c, d))/ length()
11.
12. print(sum(a, b, c, d ), length(), mean(a, b, c, d))
```

If line 10 is being executed, then it must have been called from line 12.
So, when line 12 is called sum (), length () & mean () are called respectively.
First sum () is called
A local environment is created for 'sum' within the global environment, formal arguments a, b, c, d is created in the local environment, variable result is added to the 'sum' environment and then result is returned to the function call. Environment sum' is removed from the global environment.
Control comes back to line 12.
length () function is called, a local environment is created for 'length' within the global environment, on line 7 control is returned back to line 12 and ' length' environment is removed.
mean() function is called, a local environment is created within the global environment. function sum is called. a local environment is created for 'sum' nested within the 'mean' environment, formal arguments a, b, c, d are created in the local environment, variable result is added to the 'sum' environment and then result is returned to the function call. Environment ' sum' is removed from the local environment 'mean' .
length() function is called, a local environment is created for 'length' within the 'mean' environment, on line 7 control is returned back to tine 10 and ' length' environment is removed.

13. Draw flow of execution for above program.

Line 1 :'def' is encountered, lines 2 , 3 , 4 are ignored
Line 6 :'de f' is encountered, line 7 i s ignored
Line 9: 'def' is encountered, line 18 i s ignored
line 12 : control goes to line 1 and then 1s returned back to line 12
control goes to line 6 and is returned back to line 12 and then
control goes to line 9. here, function sum () i s called so
control goes to line 1 and is returned back to line 18 , then function length() is called, so
control goes to line 6 and is returned back to Line 10 , then control is returned to line 12
The print statement is executed.

14. In the following code, which variables are in the same scope?

```
def func1():
        a = 1
        b = 2
def func2():
        c = 3
        d = 4
global : e
```

local (func1) : a, b
local (func2) :c , d

15. Write a program with a function that takes an integer and prints the number that follows after it. Call the function with these arguments:

4 , 6 , 8 , 2 + 1 , 4 – 3 * 2 , -3 -2

```
# Code
def func(x):
        print (x+1)
func(4)
func(6)
func(8)
func(2+1)
func(4-3*2)
func(-3-2)
```

16. Write a program with non-void version of above function and then write flow of execution for both the programs.

```
# Code
 def func(x):      # Line 1 here
        print(x+1)
# non-void version
 def func2(x):      # Line 5
        print (x+1)
        return x+1
 func(5)        # Line 9
 func2(5)       # Line 10
```

```
# flow of execution
Line 1 : 'def' keyword is encountered, line 2 ignored
Line 5 : 'def' keyword is encountered, lines 6 , 7 are ignored
Line 9 : func is called, control goes to Line 1 , Line 2 is executed, control back to Line 9
Line 10 : func2 is called, control goes to Line 5 , Line 6 , 7 are executed, control back to
Line 10, program end
```

17. What is the output of following code fragments?

```
( i )
def increment (n):
        n.append([ 4])      # an object of list type is being appended
        return n
L =[ 1 , 2 , 3 ]
M = increment(L)
print (L, H)
# Output

[1, 2 , 3 , [4 ]] [1 , 2 , 3 , [ 4 ] ]
# because list is a mutable type and changes made to it are reflected outside the function
```

as well

```
( ii)
def increment (n) :
        n.append([ 49 ])
        return n[ 0 ], n[ 1], n[2], n[ 3]
L = [23 , 35 , 47 ]
m1, m2, m3, m4 =increment(L)
print (L)
print (m1, m2, m3, m4)
print (L[ 3 ] == m4)
# Out put
[ 23 , 35 , 47 , [49 ]] , # list is a mutable type and changes made to it reflected outside the function
13 35 47 [49 ]
True , # because l[3] is same as m4 ([ 49])
```

Back Exercise Part C

1. Write a function that takes amount-in-dollars and dollar-to-rupee conversion price; it then returns the amount converted to rupees. Create the function in both void and non-void forms.

```
# Code
# void version
def usd_to_inr (amt, exc):
        print ('Rs ' , amt*exc)
# non-void version
def usd_to_inr2 (amt, exc):
        print ( 'Rs' ,amt*exc)
        return amt*exc
usd_to_inr( 100 , 70 )
print ( 'Rs' , usd_to_inr2( 500 , 70))        # value is being returned
# Output
Rs 7000
Rs 35000
Rs 35000
```

2. Write a function to calculate volume of a box with appropriate default values for its parameters. Your function should have the following input parameters:

```
# Code
def vol (l=1 , b=1 , h= 1 ):
        return l *b*h
vl = vol()
v2 = vol( 10 , 5)
v3 = vol(b=6)
```

```
print (v1)
print (v2)
print (v3)
```

3. Write a program to have following functions:

(i) a function that takes a number as argument and calculates Cube for it. The function does not return a value. If there is no value passed to the function in function call, the function should calculate cube of 2.
(ii) a function that takes two char arguments and returns True if both the arguments are equal otherwise False.
Test both functions by giving appropriate function call statements

```
# (i)
def cube (x=2):
        print ('cube of', x, 'is', x*x*x)
# (ii)
def equal (x, y):
        return x == y
cube ()
cube(5)
print (equal( 'a' , 'b' ))
print (equal( 'z' , 'z' ))
print (equal(4 , ' 4' ))
# Output
cube of 2 is 8
cube of 5 is 125
False
True
False # One is integer type, the other is string type
```

4. Write a function that receives two numbers and generates a random number from that range. Using this function, the main program should be able to print three numbers randomly.

```
# Code
from random import randint # to generate random number
# randint (x, y) function generates a random integer between x & y
def ran (x, y):
        return randint(x, y)
print ('Three random numbers are: ', ran(3 , 36 ), ran( 1 , 100 ), ran( 500 , 600 ))
# Output
Three random numbers are 11 32 573
```

5. Write a function that receives two string arguments and checks whether they are same-length strings (returns True in this case otherwise false).

```
# Code
def equ (x, y):
        return len (x) == len (y) # returns a Boolean value, True or False
x = input (‘Enter first string ‘)
y = input (‘Enter second string; ‘)
print (‘length of’, x, 'is equal to length of’, y, ' is’, equ(x, y))
```

```
# Output
Enter first string: abed
Enter second string: dflfdfd
length of abed is equal to length of dflfdfd is False
```

6. Write a function namely zzfhRoot( ) that receives two parameters x and n and returns nth root of x i.e., $x^{1/n}$, the default value of n is 2

```
# Code
def nthRoot (x, n=2):
        return round (x**( l /n), 2 )
# round () function to round the value till 2 decimal places
print ('nth root of', 19, 'for n=3 is', nthRoot(19 , 3))
print ('square root of', 30, 'is', nthRoot(30))
# Output
nth root of 10 for n=3 is 2. 15
square root of 30 is 5. 48
```

7. Write a function that takes a number n and then returns a randomly generated number having exactly n digits (not starting with zero) e.g., if n is 2 then function can randomly return a number 10-99 but 07, 02 etc. are not valid two-digit numbers.

```
# Code
from random import randint

def ran(n):
    x = int('9'*(n-1))+1 # first n-digit number, string is converted back to int
    y = int('9'*n) # last n-digit number
    return randint(x, y)
try:
    x=int(input("Number of digits in the number to be generated (>1): "))
    print('Random number of digit', x, 'is', ran(x))

except ValueError as e:
    print("Error: ",e)
    print("Please enter a number greater than 1")

#Output
Random number of digit 3 is 883
Random number of digit 5 is 78720
```

8. Write a function that takes two numbers and returns the number that has minimum one's digit. [For example, if numbers passed are 491 and 278, then the function will return 491 because it has got minimum one's digit out of two given numbers (491's 1 is < 278's 8)].

```
def minOne (x, y):
# remainder after dividing by 10 gives the one's digit
        if x%10 < y%10:
                return x
        elif x%10 == y%10:
                return 'both are equal'
```

```
        else:
                return y
print (minOne(73 , 88 ))
print (minOne(89 , 79 ))
print (minOne (67, 55))
# Output
73
both are equal
55
```

9. Write a program that generates a series using a function which takes first and last values of the series and then generates four terms that are equidistant e.g., if two numbers passed are 1 and 7 then function returns 1 3 5 7.

```
# Code

def func(x, y):
    d = (y-x)/3 # divides the difference in 3 equal parts
    return [x, x+d, x+2*d, y] # returns the series as a list

print(func(1, 9))
print(func(4, 8))
print(func(4, 10))
print(func(1, 7))

# Output
[1, 3.6666666666666665, 6.333333333333333, 9]
[4, 5.333333333333333, 6.666666666666666, 8]
[4, 6.0, 8.0, 10]
[1, 3.0, 5.0, 7]
```