

WORKING WITH FUNCTIONS

Divide and Conquer

- ❑ Large programs are often difficult to manage, thus large programs are divided into smaller units known as **functions**.
- ❑ It is simply a group of statements under any name i.e. function name and can be invoked (call) from other part of program.
- ❑ Take an example of School Management Software, now this software will contain various tasks like Registering student, Fee collection, Library book issue, TC generation, Result Declaration etc. In this case we have to create different functions for each task to manage the software development.

Introduction

- Set of functions is stored in a file called MODULE. And this approach is known as MODULARIZATION, makes program easier to understand, test and maintain.
- Commonly used modules that contain source code for generic need are called LIBRARIES.
- Modules contains set of functions. Functions is of mainly two types:
 - ▣ Built-in Functions
 - ▣ User-Defined Functions

Advantages of Function

- ❑ PROGRAM HANDLING EASIER : only small part of the program is dealt with at a time.
- ❑ REDUCED LoC: as with function the common set of code is written only once and can be called from any part of program, so it reduces Line of Code
- ❑ EASY UPDATING : if function is not used then set of code is to be repeated everywhere it is required. Hence if we want to change in any formula/expression then we have to make changes to every place, if forgotten then output will be not the desired output. With function we have to make changes to only one location.

User Defined Functions

- A function is a set of statements that performs a specific task; a common structuring element that allows you to use a piece of code repeatedly in different parts of a program. Functions are also known as sub-routine, methods, procedure or subprogram.
- Syntax to create USER DEFINED FUNCTION

```
def function_name([comma separated list of parameters]):
```

```
    statements....
```

```
    statements....
```



KEYWORD



FUNCTION DEFINITION

Points to remember...

- Keyword **def** marks the start of function header
- Function name must be unique and follows naming rules same as for identifiers
- Function can take arguments. It is optional
- A colon(:) to mark the end of function header
- Function can contains one or more statement to perform specific task
- An optional **return** statement to return a value from the function.
- Function must be **called/invoked** to execute its code

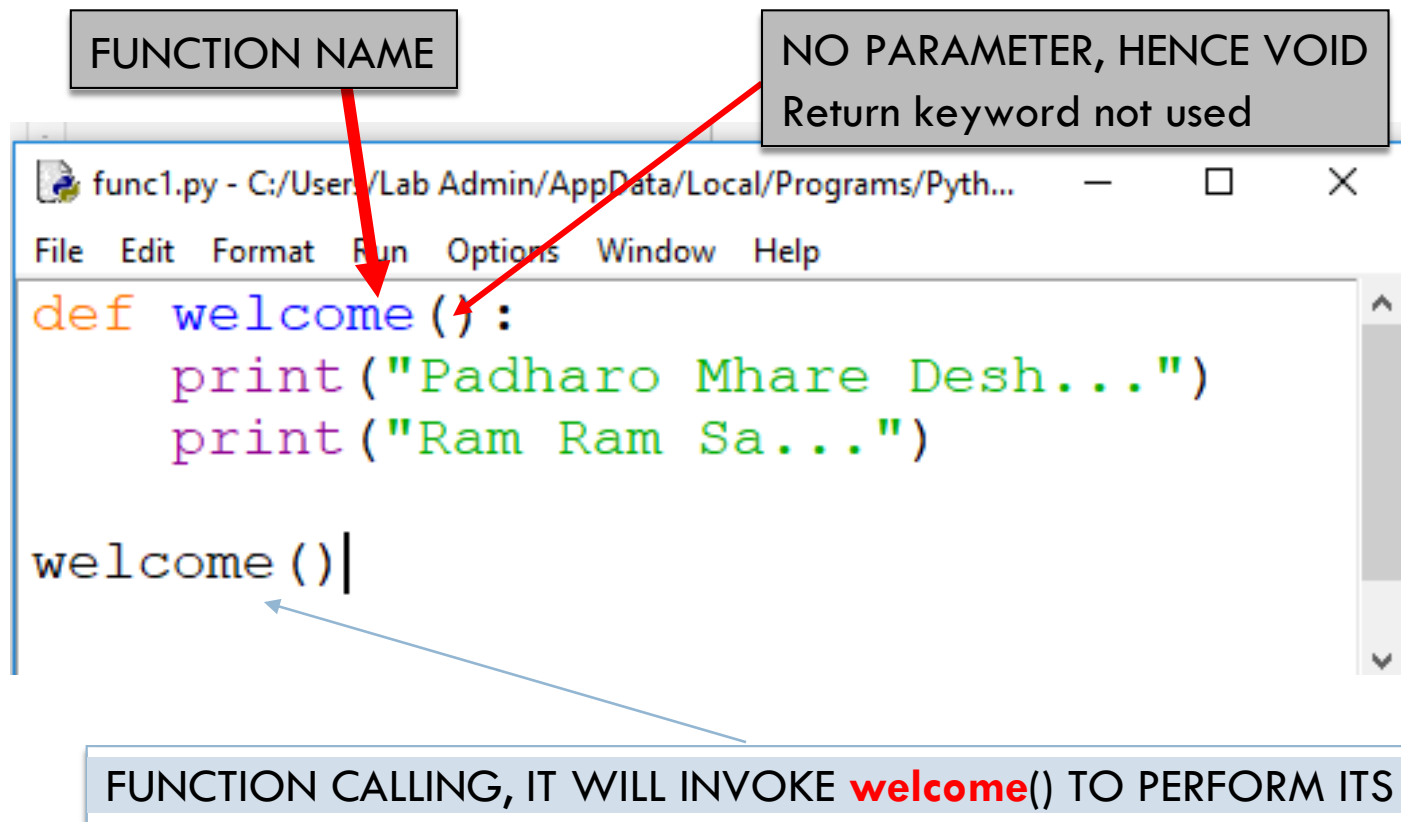
User Defined function can be....

1. Function with no arguments and no return
2. Function with arguments but no return value
3. Function with arguments and return value
4. Function with no argument but return value

Let us understand each of the function type with example....

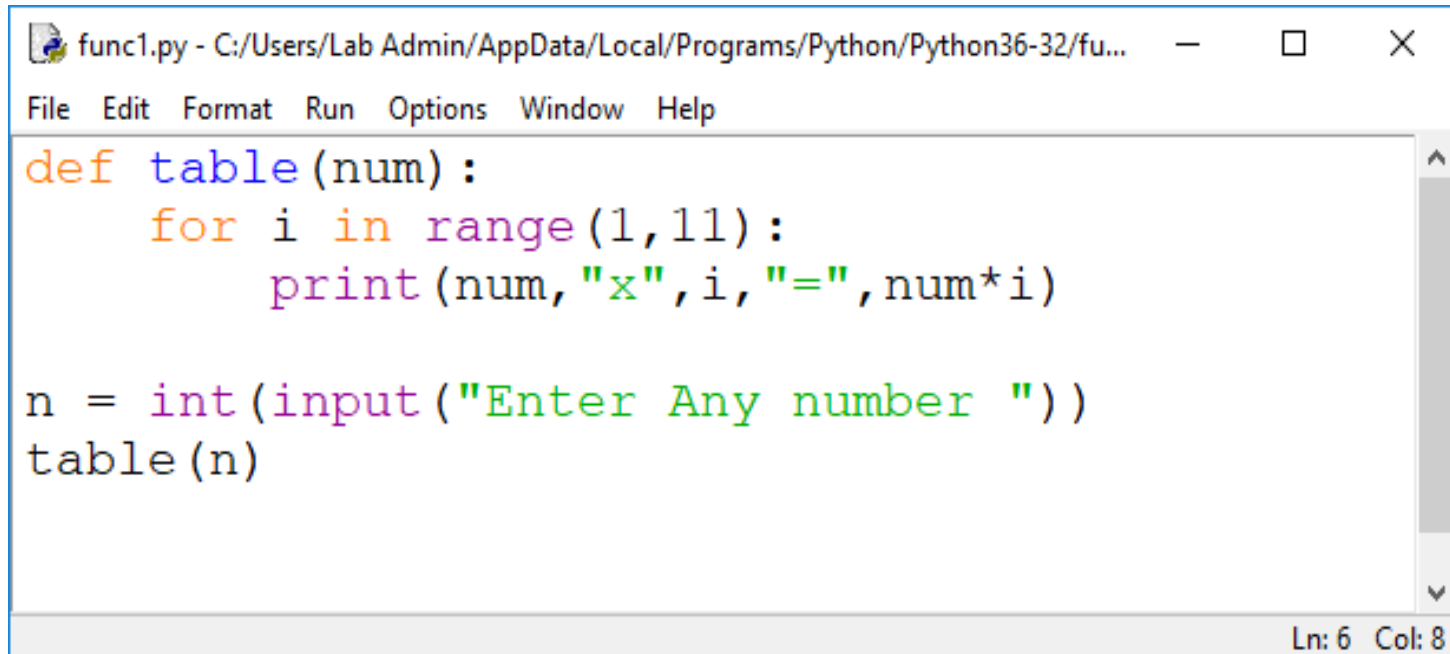
Function with no argument and no return

- This type of function is also known as **void** function



Function with parameters but no return value

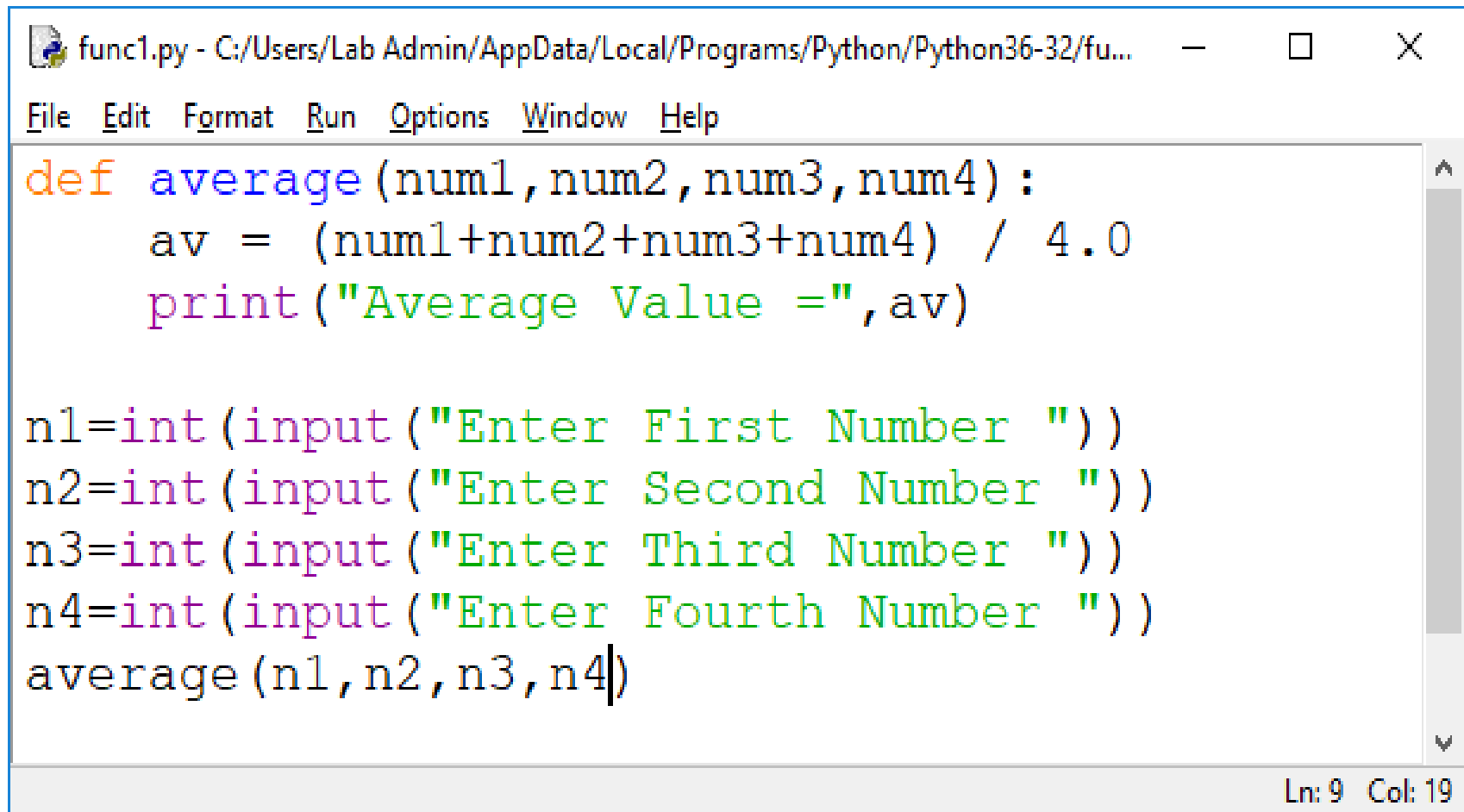
- ❑ Parameters are given in the parenthesis separated by comma.
- ❑ Values are passed for the parameter at the time of function calling.



```
func1.py - C:/Users/Lab Admin/AppData/Local/Programs/Python/Python36-32/fu...  
File Edit Format Run Options Window Help  
def table(num):  
    for i in range(1,11):  
        print(num, "x", i, "=", num*i)  
  
n = int(input("Enter Any number "))  
table(n)
```

Ln: 6 Col: 8

Function with parameters but no return value



The screenshot shows a Python IDE window titled 'func1.py - C:/Users/Lab Admin/AppData/Local/Programs/Python/Python36-32/fu...'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The main text area contains the following Python code:

```
def average(num1, num2, num3, num4) :  
    av = (num1+num2+num3+num4) / 4.0  
    print("Average Value =", av)  
  
n1=int(input("Enter First Number "))  
n2=int(input("Enter Second Number "))  
n3=int(input("Enter Third Number "))  
n4=int(input("Enter Fourth Number "))  
average(n1, n2, n3, n4|)
```

The status bar at the bottom right indicates 'Ln: 9 Col: 19'.

Function with parameter and return

- We can return values from function using return keyword.
- The return value must be used at the calling place by –
 - Either store it any variable
 - Use with print()
 - Use in any expression

Function with return

```
func1.py - C:/Users/Lab Admin/AppData/Local/Programs/Python/Python36-32/
File Edit Format Run Options Window Help

def cube(num):
    return num*num*num

n1=int(input("Enter Any Number "))
c = cube(n1)
print("Cube of number is ",c)|
```

```
func1.py - C:/Users/Lab Admin/AppData/Local/Programs/P...
File Edit Format Run Options Window Help

def cube(num):
    return num*num*num

n1=int(input("Enter Any Number "))
print("Cube of number is ",cube(n1))

Ln: 3 Col: 0
```

```
func1.py - C:/Users/Lab Admin/AppData/Local/Programs/Python/Python3...
File Edit Format Run Options Window Help

def cube(num):
    return num*num*num

n1=int(input("Enter Any Number "))
val = n1 + cube(n1)
print("Number + Cube =",val)
```

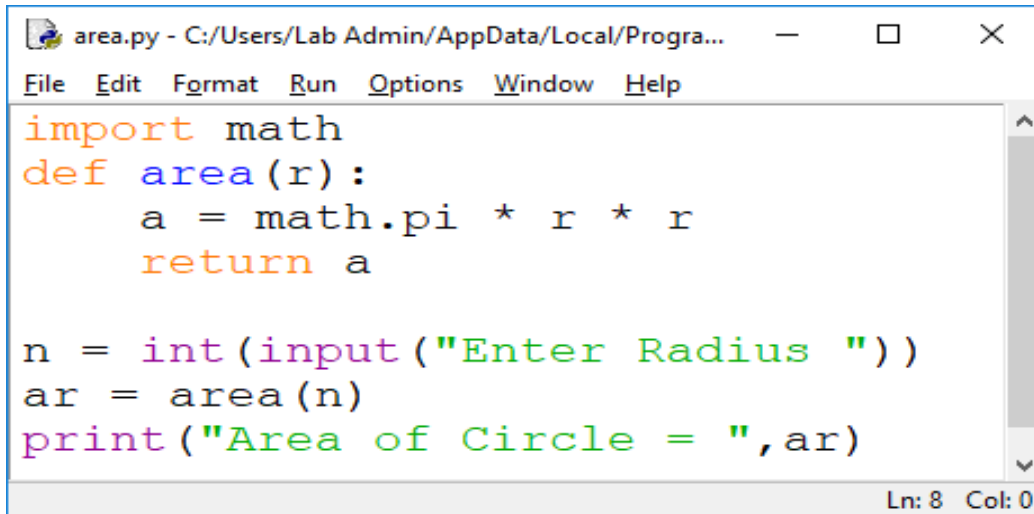
```
*func1.py - C:/Users/Lab Admin/AppData/Local/Programs/Python/Python...
File Edit Format Run Options Window Help

def cube(num):
    c = num * num * num
    return c

n1=int(input("Enter Any Number "))
val = cube(n1)
print("Cube =",val)

Ln: 4 Col: 0
```

Function with return



```
area.py - C:/Users/Lab Admin/AppData/Local/Progra...  
File Edit Format Run Options Window Help  
import math  
def area(r):  
    a = math.pi * r * r  
    return a  
  
n = int(input("Enter Radius "))  
ar = area(n)  
print("Area of Circle = ", ar)  
Ln: 8 Col: 0
```

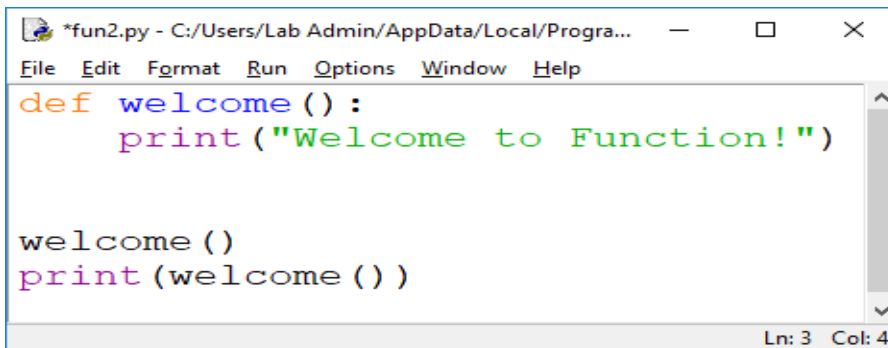
NOTE: *the RETURN statement ends a FUNCTION execution even if it is in the middle of FUNCTION. Anything written below RETURN statement will become UNREACHABLE code.*

```
def max(x,y):  
    if x>y:  
        return x  
    else:  
        return y  
    print("I am not reachable")
```

Function not returning value

- Function may or may not return a value. Non returning function is also known as VOID function. It may or may not contain return. If it contain return statement then it will be in the form of:

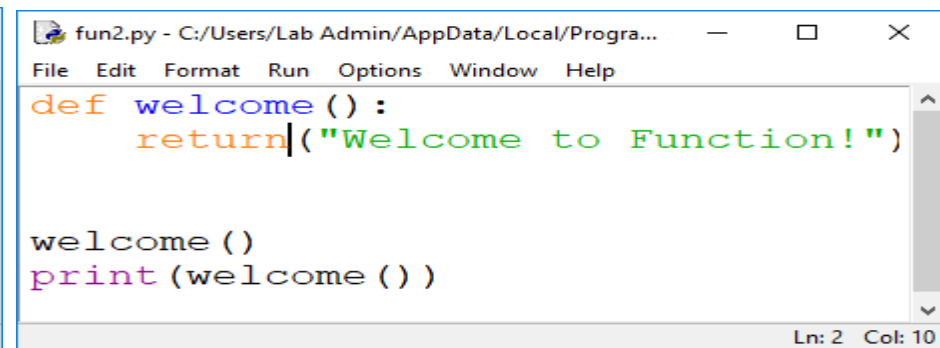
▣ **return** [no value after return]



```
*fun2.py - C:/Users/Lab Admin/AppData/Local/Progra...
File Edit Format Run Options Window Help
def welcome():
    print("Welcome to Function!")

welcome()
print(welcome())
Ln: 3 Col: 4
```

```
RESTART: C:/Users/Lab
Welcome to Function!
Welcome to Function!
None
```



```
fun2.py - C:/Users/Lab Admin/AppData/Local/Progra...
File Edit Format Run Options Window Help
def welcome():
    return("Welcome to Function!")

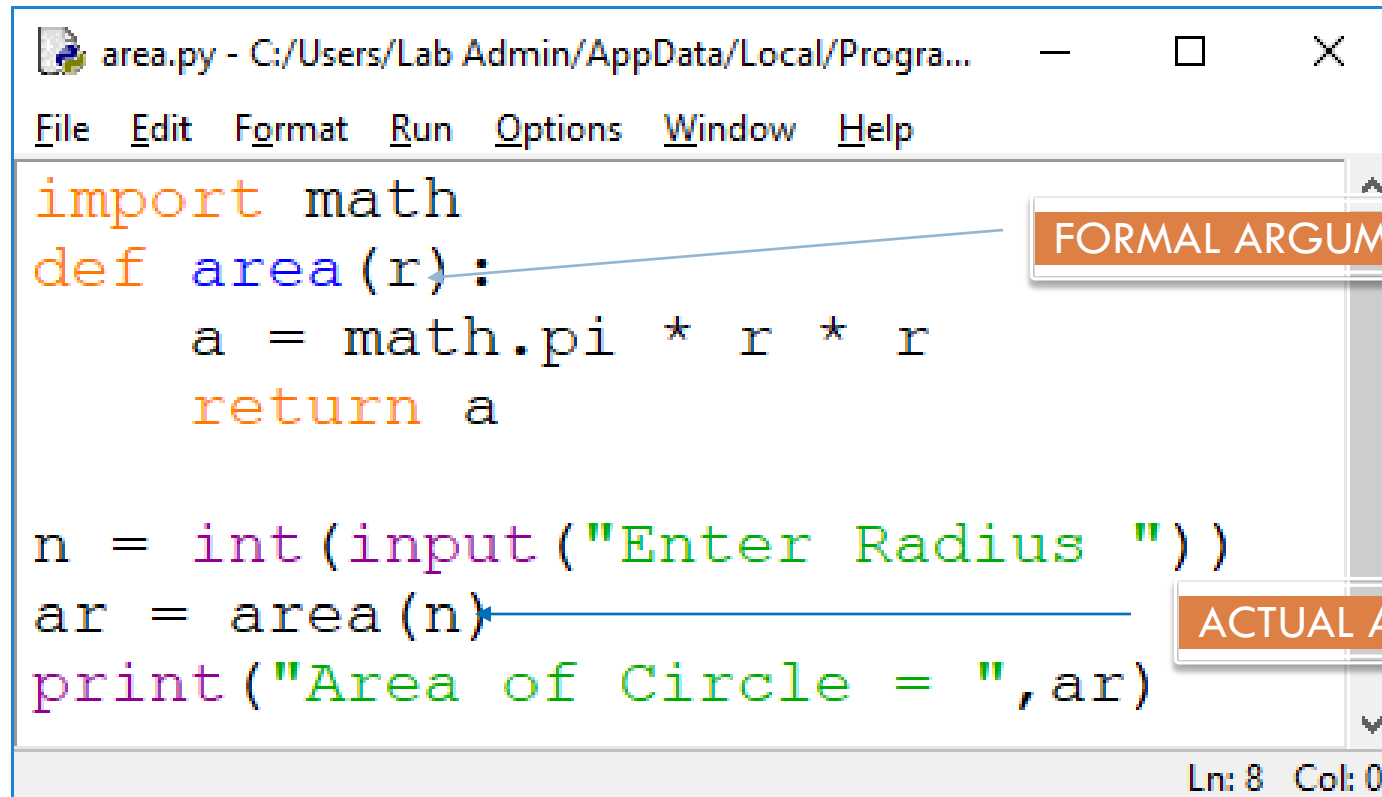
welcome()
print(welcome())
Ln: 2 Col: 10
```

```
RESTART: C:/Users/Lab
Welcome to Function!
>>>
```

Parameters and Arguments in Function

- Parameters are the value(s) provided in the parenthesis when we write function header. These are the values required by function to work
- If there are more than one parameter, it must be separated by comma(,)
- An Argument is a value that is passed to the function when it is called. In other words arguments are the value(s) provided in function call/invoke statement
- Parameter is also known as FORMAL ARGUMENTS/PARAMETERS
- Arguments is also known as ACTUAL ARGUMENTS/PARAMETER
- Note: Function can alter only MUTABLE TYPE values.

Example of Formal/Actual Arguments



```
area.py - C:/Users/Lab Admin/AppData/Local/Progra...  —  □  ×  
File Edit Format Run Options Window Help  
import math  
def area(r):  
    a = math.pi * r * r  
    return a  
  
n = int(input("Enter Radius "))  
ar = area(n)  
print("Area of Circle = ", ar)
```

Ln: 8 Col: 0

FORMAL ARGUMENT

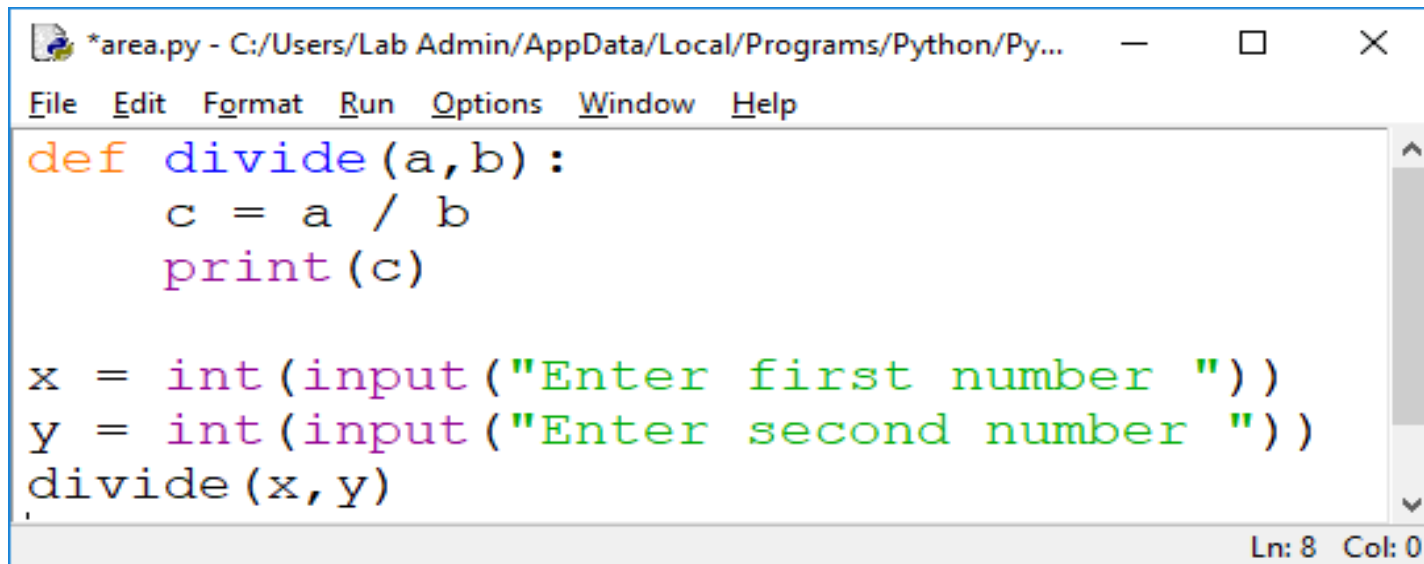
ACTUAL ARGUMENT

Types of Arguments

- There are 4 types of Actual Arguments allowed in Python:
 1. Positional arguments
 2. Default arguments
 3. Keyword arguments
 4. Variable length arguments

Positional arguments

- Are arguments passed to a function in correct positional order



```
*area.py - C:/Users/Lab Admin/AppData/Local/Programs/Python/Py...
File Edit Format Run Options Window Help

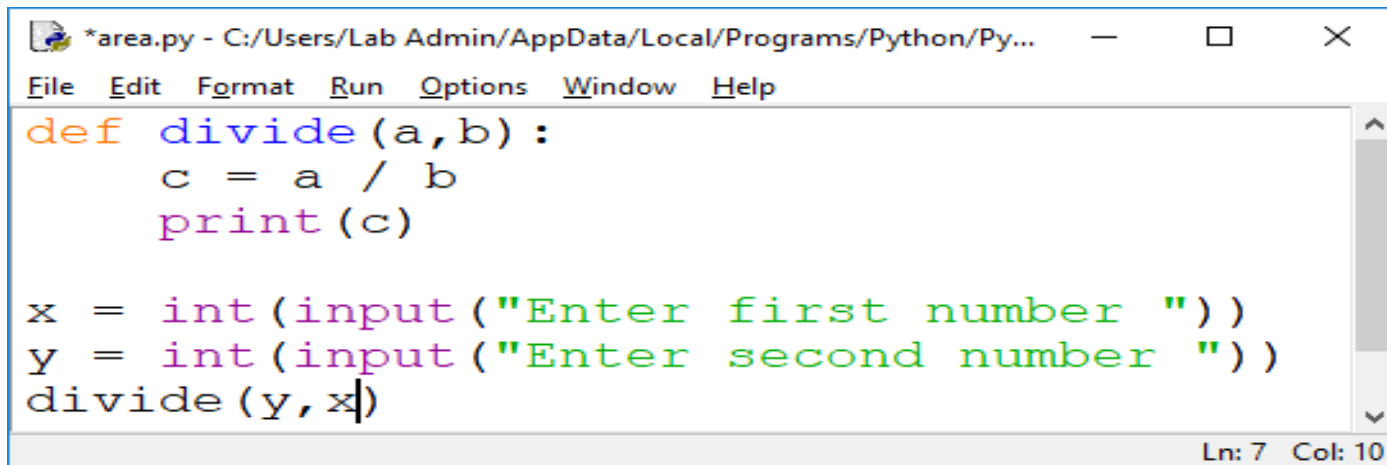
def divide(a,b):
    c = a / b
    print(c)

x = int(input("Enter first number "))
y = int(input("Enter second number "))
divide(x,y)
```

Ln: 8 Col: 0

- Here x is passed to a and y is passed to b i.e. in the order of their position

```
RESTART: C:/Users/Lab Ad
Enter first number 20
Enter second number 10
2.0
>>> |
```



```
*area.py - C:/Users/Lab Admin/AppData/Local/Programs/Python/Py...
File Edit Format Run Options Window Help
def divide(a,b):
    c = a / b
    print(c)

x = int(input("Enter first number "))
y = int(input("Enter second number "))
divide(y,x)
```

Ln: 7 Col: 10

```
RESTART: C:/Users/Lab Admin/AppData/Local/Progr
Enter first number 20
Enter second number 10
0.5
>>>
```

If the Number of formal argUMENT and ACTUAL differs then Python will raise an error

Default arguments

- Sometimes we can provide default values for our positional arguments. In this case if we are not passing any value then default values will be considered.
- Default argument must not followed by non-default arguments.

def interest(principal,rate,time=15):

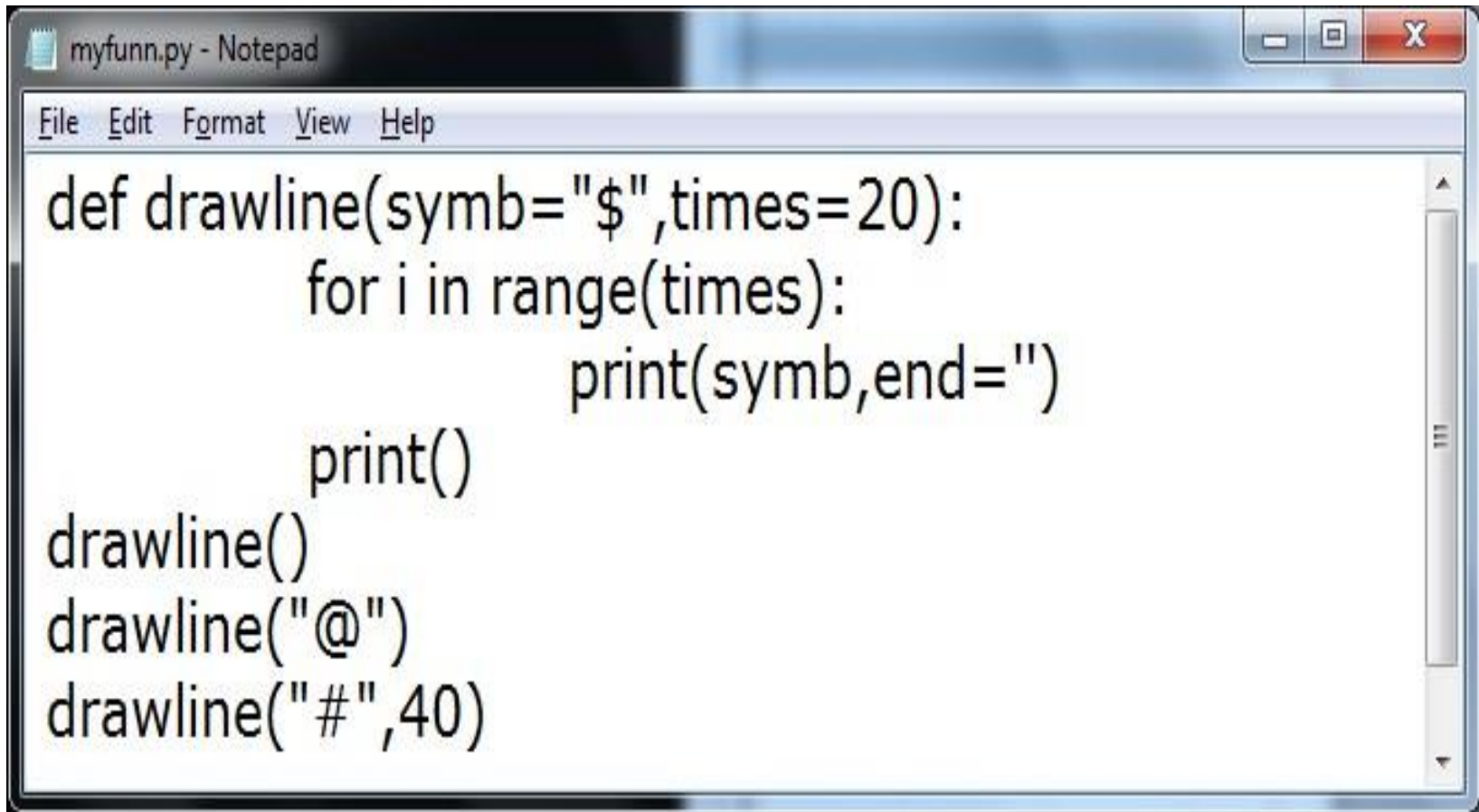
def interest(principal,rate=8.5,time=15):

def interest(principal,rate=8.5,time):

VALID

← INVALID

Default arguments

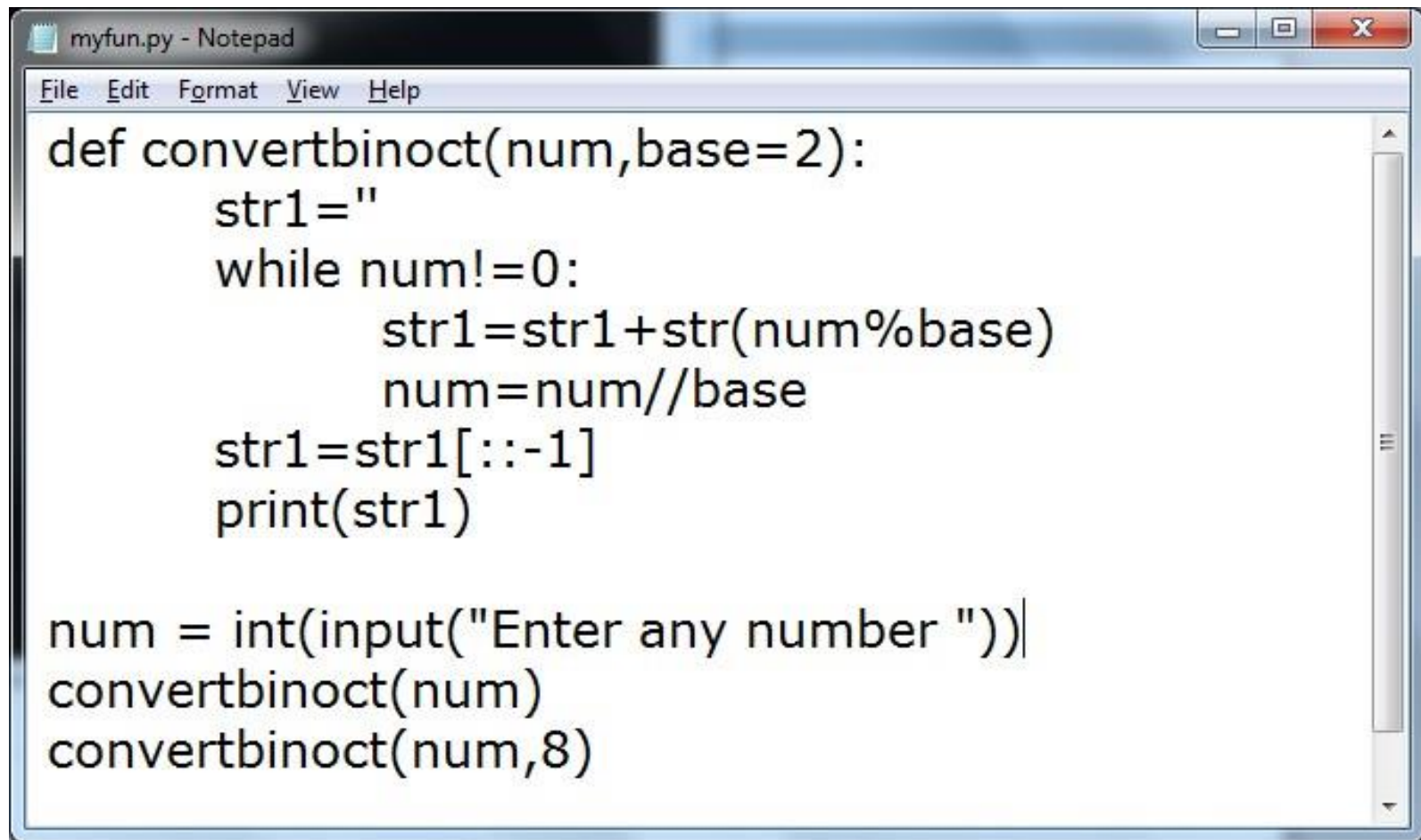


```
myfunn.py - Notepad
File Edit Format View Help

def drawline(symb="$",times=20):
    for i in range(times):
        print(symb,end="")
    print()

drawline()
drawline("@")
drawline("#",40)
```

Default arguments



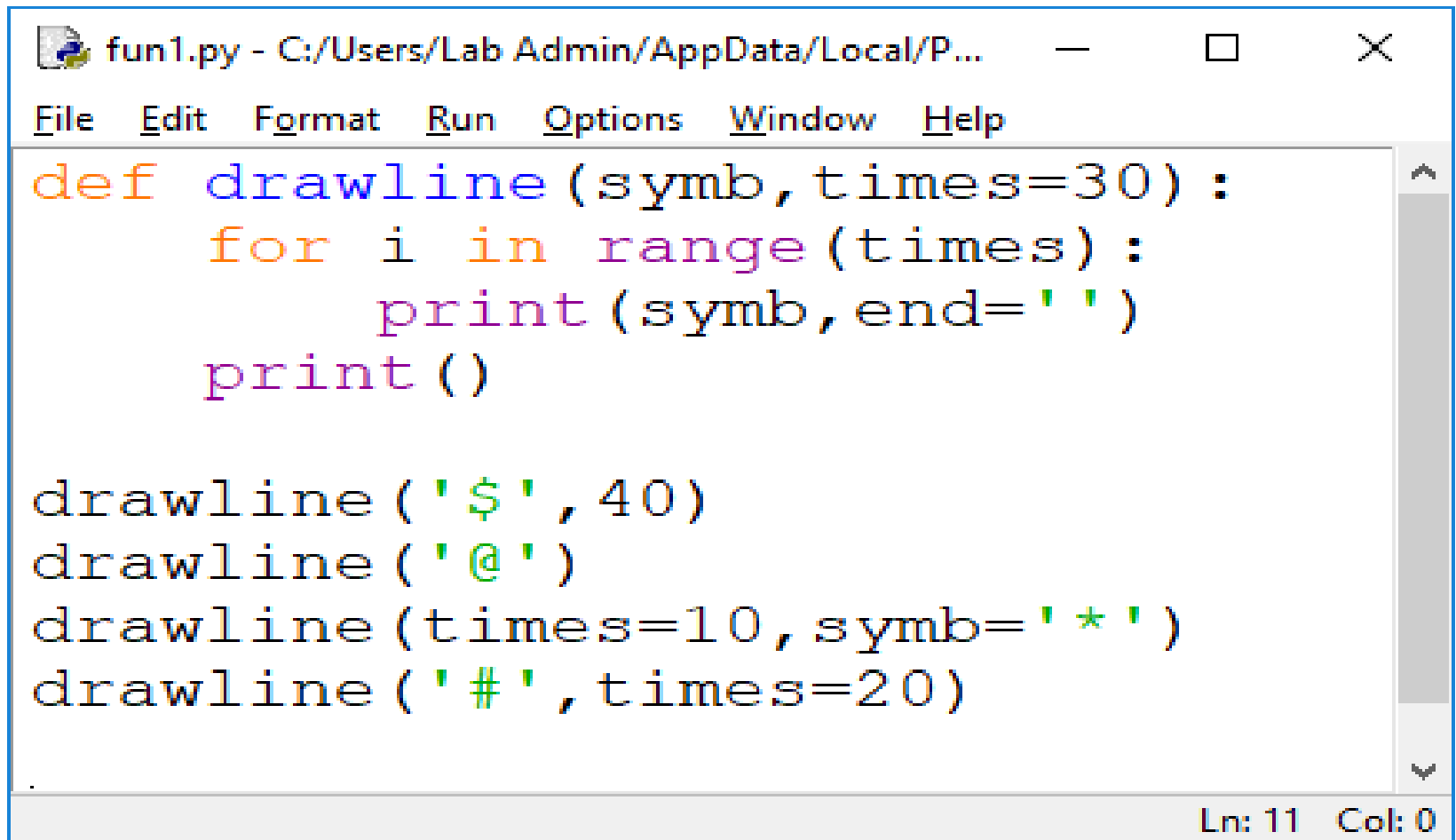
```
myfun.py - Notepad
File Edit Format View Help
def convertbinoct(num,base=2):
    str1=""
    while num!=0:
        str1=str1+str(num%base)
        num=num//base
    str1=str1[::-1]
    print(str1)

num = int(input("Enter any number "))
convertbinoct(num)
convertbinoct(num,8)
```

Keyword(Named) Arguments

- The default keyword gives flexibility to specify default value for a parameter so that it can be skipped in the function call, if needed. However, still we cannot change the order of arguments in function call i.e. you have to remember the order of the arguments and pass the value accordingly.
- To get control and flexibility over the values sent as arguments, python offers KEYWORD ARGUMENTS.
- This allows to call function with arguments in any order using name of the arguments.

Keyword(Named) Argument



The image shows a screenshot of a Python IDE window titled "fun1.py - C:/Users/Lab Admin/AppData/Local/P...". The window has a menu bar with "File", "Edit", "Format", "Run", "Options", "Window", and "Help". The main text area contains the following Python code:

```
def drawline(symb, times=30) :  
    for i in range(times):  
        print(symb, end=' ')  
    print()  
  
drawline('$', 40)  
drawline('@')  
drawline(times=10, symb='*')  
drawline('#', times=20)
```

The status bar at the bottom right indicates "Ln: 11 Col: 0".

Rules for combining all three type of arguments

- An argument list must first contain positional arguments followed by keyword arguments
- Keyword arguments should be taken from the required arguments
- You cannot specify a value for an argument more than once

Example of legal/illegal function call

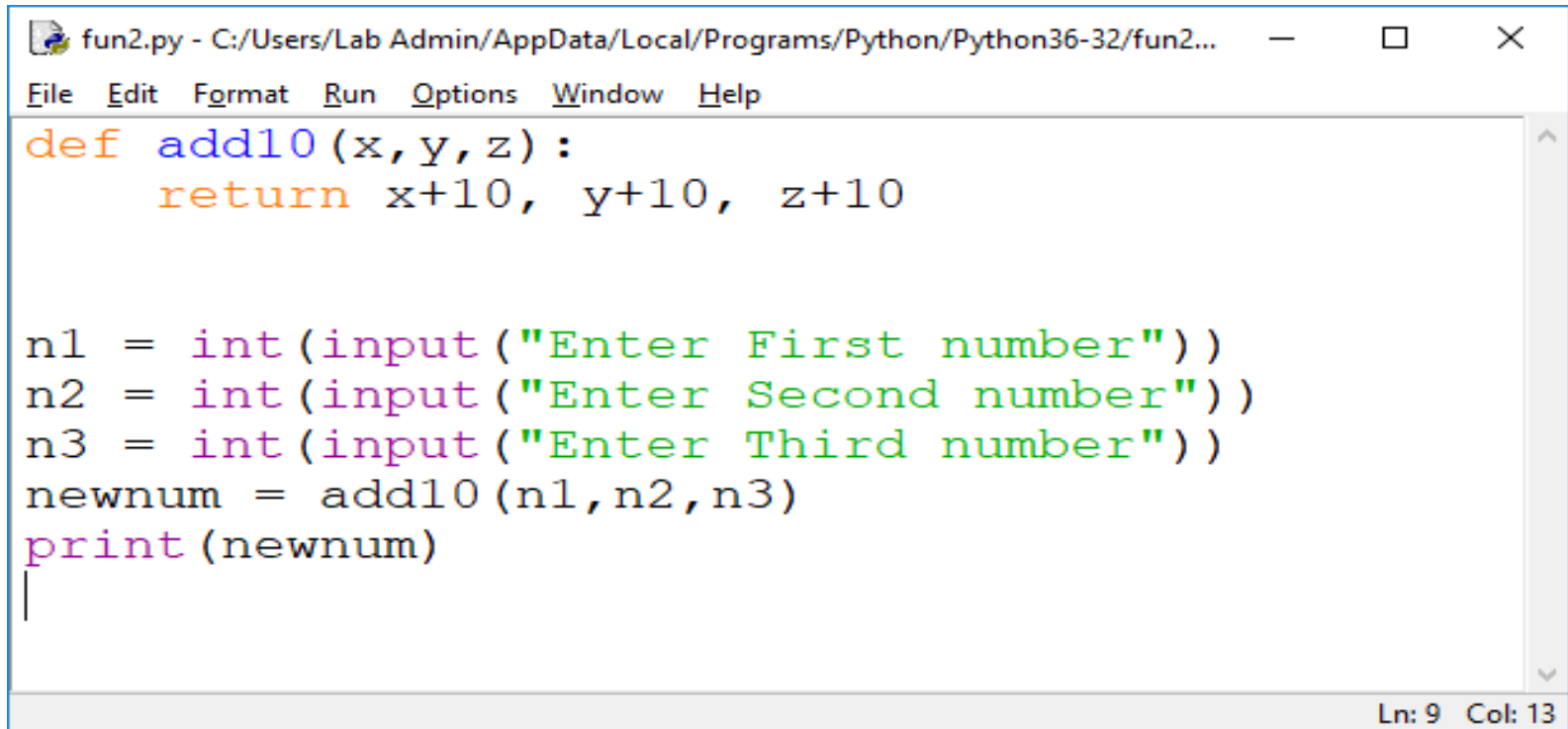
```
def Average(n1,n2,n3=100):  
    RETURN (n1+n2+n3)/3
```

FUNCTION CALL	LEGAL/ ILLEGAL	REASON
Average(n1=20, n2=40,n3=80)	LEGAL	Non default values provided as named arguments
Average(n3=10,n2=7,n1=100)	LEGAL	Keyword argument can be in any order
Average(100,n2=10,n3=15)	LEGAL	Positional argument before the keyword arguments
Average(n3=70,n1=90,100)	ILLEGAL	Keyword argument before the positional arguments
Average(100,n1=23,n2=1)	ILLEGAL	Multiple values provided for n1
Average(200,num2=90,n3=11)	ILLEGAL	Undefined argument NUM2
Average(21,num3=900)	ILLEGAL	A required argument n2 is missing

Returning Multiple values

- Unlike other programming languages, python lets you return more than one value from function.
- The multiple return value must be either stored in TUPLE or we can UNPACK the received value by specifying the same number of variables on the left of assignment of function call.
- Let us see an example of both :-

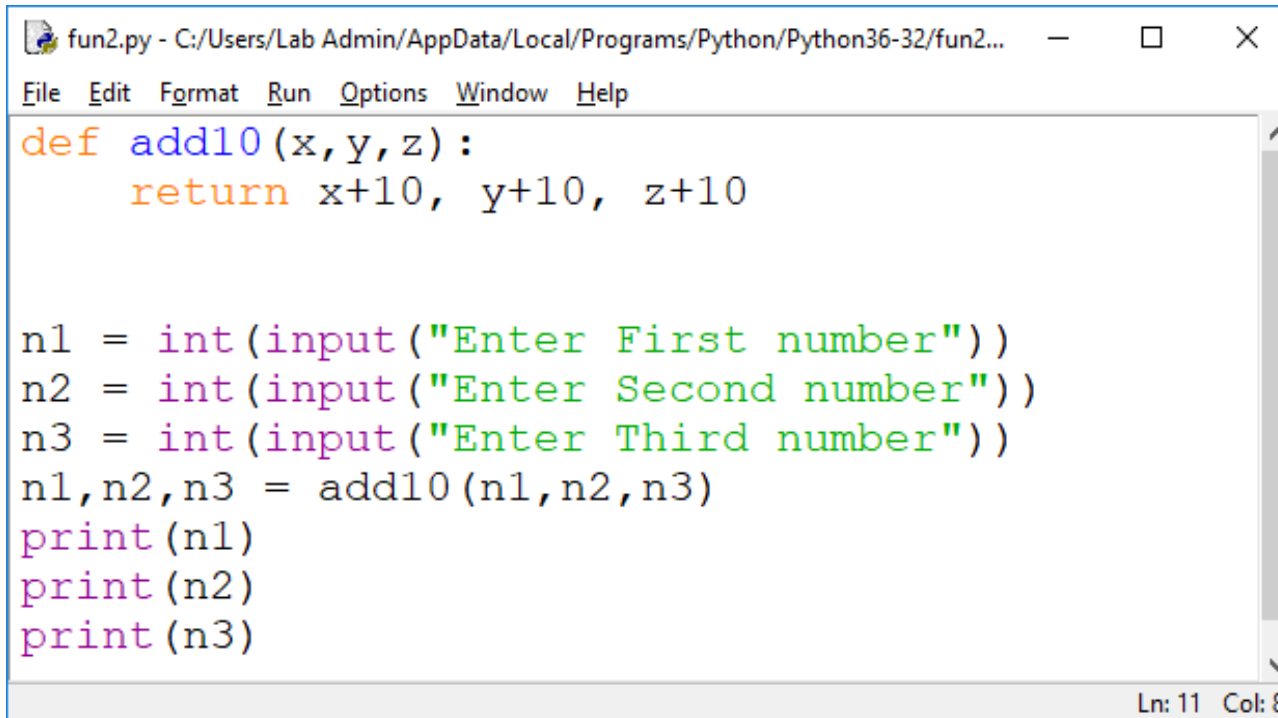
Multiple return value stored in TUPLE



```
fun2.py - C:/Users/Lab Admin/AppData/Local/Programs/Python/Python36-32/fun2...  
File Edit Format Run Options Window Help  
def add10(x, y, z):  
    return x+10, y+10, z+10  
  
n1 = int(input("Enter First number"))  
n2 = int(input("Enter Second number"))  
n3 = int(input("Enter Third number"))  
newnum = add10(n1, n2, n3)  
print(newnum)  
|  
Ln: 9 Col: 13
```

```
Enter First number10  
Enter Second number20  
Enter Third number30  
(20, 30, 40)  
>>>
```

Multiple return value stored by unpacking in multiple variables



```
fun2.py - C:/Users/Lab Admin/AppData/Local/Programs/Python/Python36-32/fun2...
File Edit Format Run Options Window Help

def add10(x,y,z):
    return x+10, y+10, z+10

n1 = int(input("Enter First number"))
n2 = int(input("Enter Second number"))
n3 = int(input("Enter Third number"))
n1,n2,n3 = add10(n1,n2,n3)
print(n1)
print(n2)
print(n3)
```

Ln: 11 Col: 8

```
Enter First number11
Enter Second number22
Enter Third number33
21
32
43
>>> |
```

Composition

- Refers to using an expression as a part of large expression, or a statement as a part of large statement.
- Examples
 - `Max((a+b),(c+a))` # Arithmetic
 - `Prize(Card or Cash)` # Logical
 - **`name="Vikram"`**
 - **`print(name.replace("m","nt").upper())`** #function

Scope of Variables

- SCOPE means in which part(s) of the program, a particular piece of code or data is accessible or known.
- In Python there are broadly 2 kinds of Scopes:
 - ▣ Global Scope
 - ▣ Local Scope

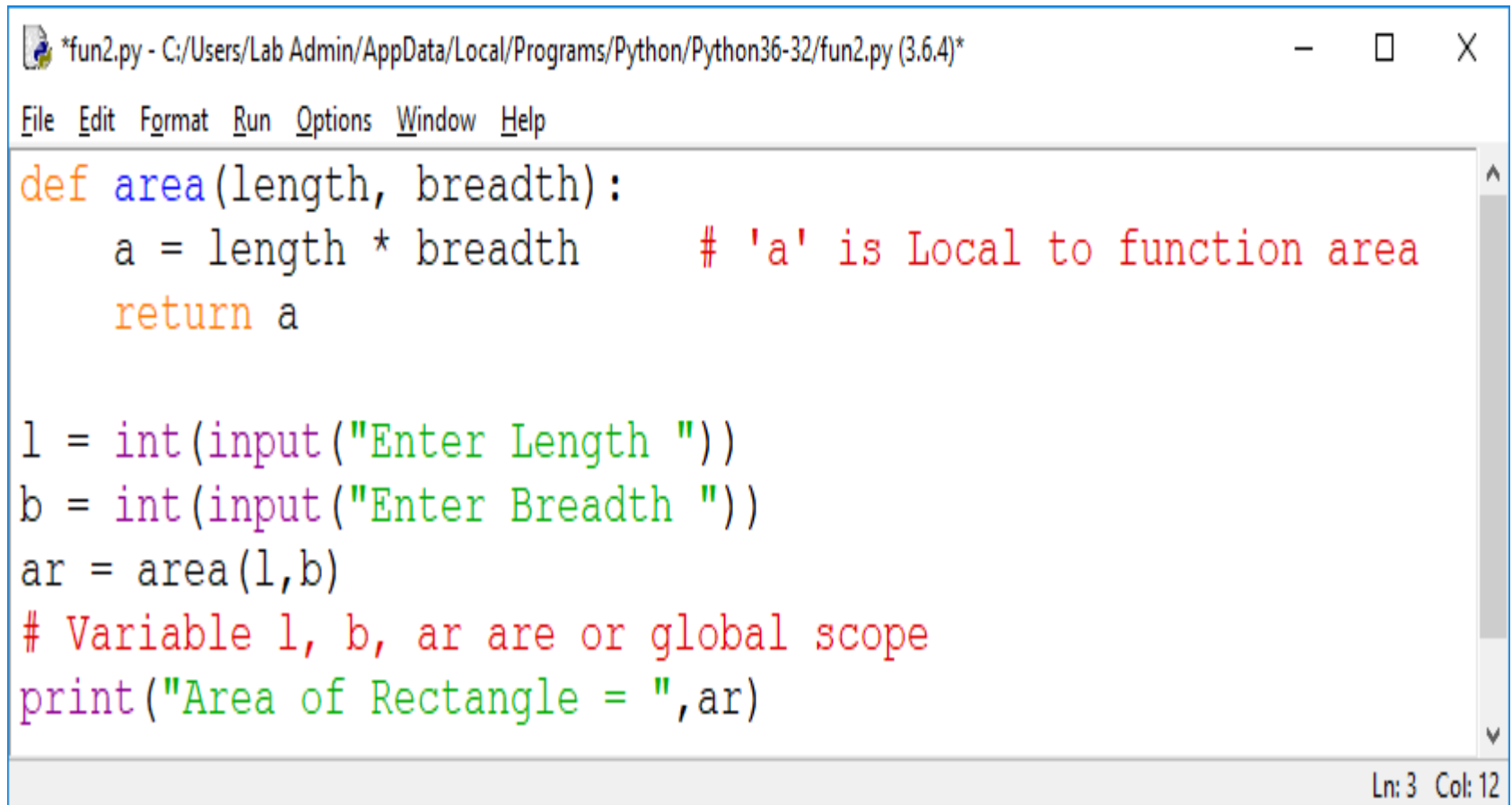
Global Scope

- A name declared in top level segment(_ main_) of a program is said to have global scope and can be used in entire program.
- Variable defined outside all functions are global variables.

Local Scope

- A name declare in a function body is said to have local scope i.e. it can be used only within this function and the other block inside the function.
- The formal parameters are also having local scope.
- Let us understand with example....

Example – Local and Global Scope



The screenshot shows a Python IDE window titled '*fun2.py - C:/Users/Lab Admin/AppData/Local/Programs/Python/Python36-32/fun2.py (3.6.4)*'. The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code defines a function 'area' that takes 'length' and 'breadth' as arguments, calculates the area, and returns it. Below the function, the script prompts the user for length and breadth, calls the 'area' function, and prints the result. A comment indicates that variables 'l', 'b', and 'ar' are in global scope. The status bar at the bottom right shows 'Ln: 3 Col: 12'.

```
*fun2.py - C:/Users/Lab Admin/AppData/Local/Programs/Python/Python36-32/fun2.py (3.6.4)*
File Edit Format Run Options Window Help
def area(length, breadth):
    a = length * breadth      # 'a' is Local to function area
    return a

l = int(input("Enter Length "))
b = int(input("Enter Breadth "))
ar = area(l,b)
# Variable l, b, ar are or global scope
print("Area of Rectangle = ",ar)
Ln: 3 Col: 12
```

Example – Local and Global Scope

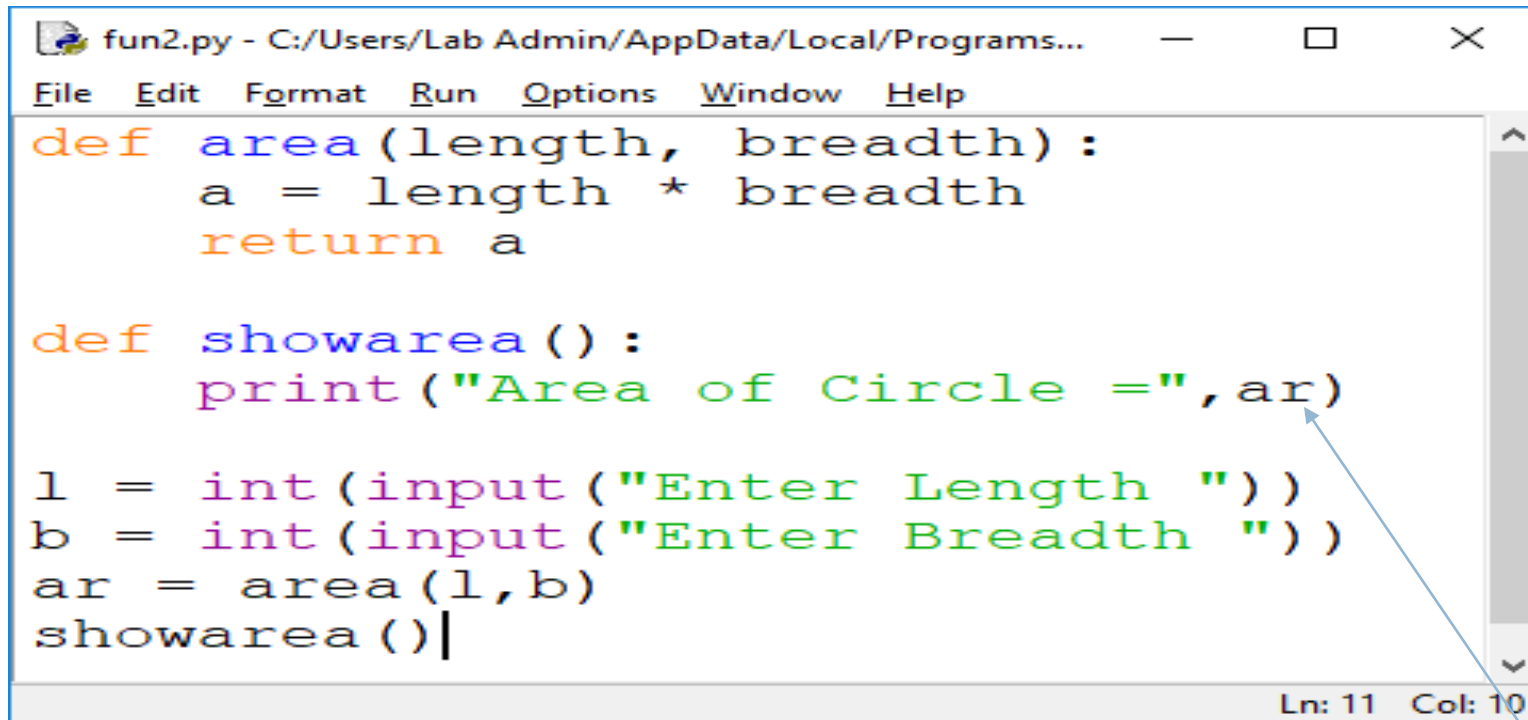
```
fun2.py - C:/Users/Lab Admin/AppData/Local/Programs/Python/Python36-32/fun2.py (3.6.4)
File Edit Format Run Options Window Help
def area(length, breadth):
    a = length * breadth      # 'a' is Local to function area
    return a

l = int(input("Enter Length "))
b = int(input("Enter Breadth "))
ar = area(l,b)
# Variable l, b, ar are or global scope
print("Area of Rectangle = ",a)
```

```
Enter Length 2
Enter Breadth 4
Traceback (most recent call last):
  File "C:/Users/Lab Admin/AppData/Local/Programs/Python/Python36-32/fun2.py", line 9, in <module>
    print("Area of Rectangle = ",a)
NameError: name 'a' is not defined
>>>
```

„a“ is not accessible here because it is declared in function area(), so scope is local to area()

Example – Local and Global Scope



```
fun2.py - C:/Users/Lab Admin/AppData/Local/Programs...  
File Edit Format Run Options Window Help  
def area(length, breadth):  
    a = length * breadth  
    return a  
  
def showarea():  
    print("Area of Circle =", ar)  
  
l = int(input("Enter Length "))  
b = int(input("Enter Breadth "))  
ar = area(l,b)  
showarea()
```

Ln: 11 Col: 10

Variable „ar' is accessible in function showarea() because it is having Global Scope

```
count=0
def add(x,y):
    global count
    count+=1
    return x+y
def sub(x,y):
    global count
    count+=1
    return x-y
def mul(x,y):
    global count
    count+=1
    return x*y
def div(x,y):
    global count
    count+=1
    return x//y
ans=1
while ans==1:
    n1=int(input("Enter First Number "))
    n2=int(input("Enter Second Number "))
    op=int(input("Enter operation desired(1-Add,2-Sub,3-Mul,4-Div) "))
    if op==1:
        print("Sum=",add(n1,n2))
    elif op==2:
        print("Sub=",sub(n1,n2))
    elif op==3:
        print("Mul=",mul(n1,n2))
    elif op==4:
        print("Div=",div(n1,n2))
    ans=int(input("Do More(1-Yes)| ?"))
print("Total Operation Done =",count)
```



This declaration “**global count**” is necessary for using global variables in function, other wise an error “**local variable 'count' referenced before assignment**” will appear because local scope will create variable “count” and it will be found unassigned

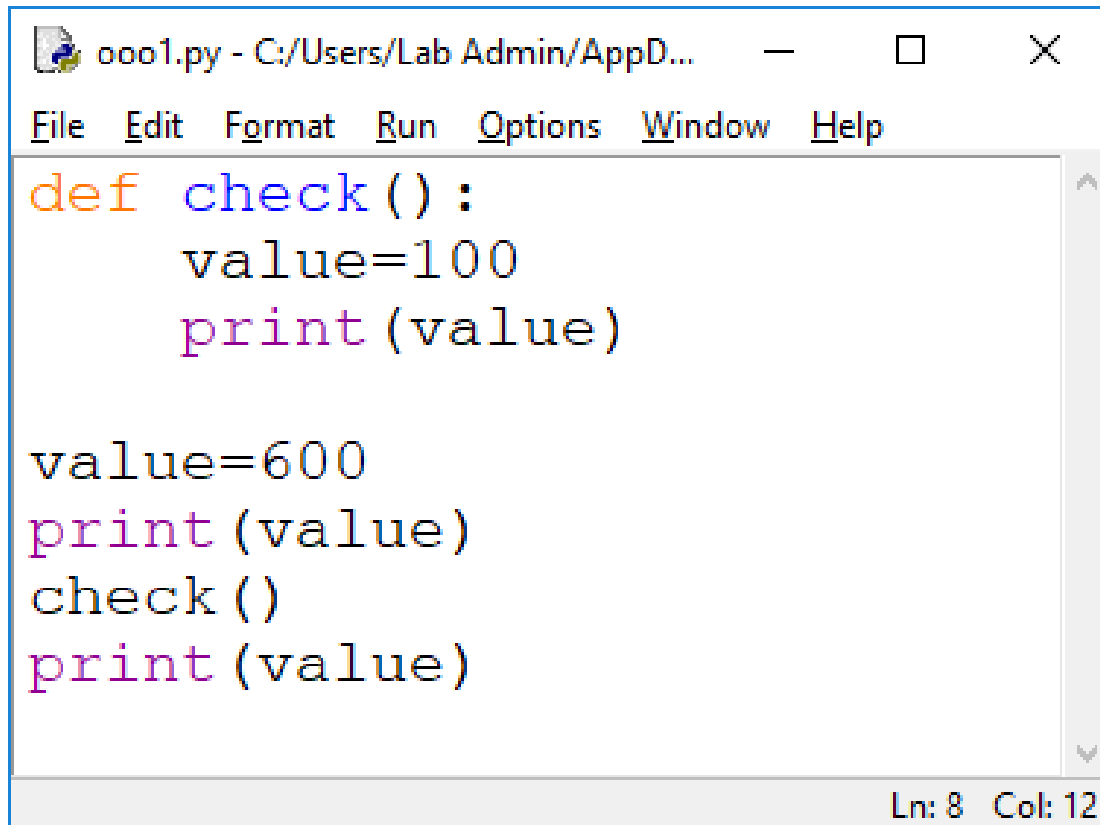
Lifetime of Variable

- Is the time for which a variable lives in memory. For Global variables the lifetime is entire program run i.e. as long as program is executing. For Local variables lifetime is their function's run i.e. as long as function is executing.

Name Resolution (Scope Resolution)

- For every name used within program python follows name resolution rules known as **LEGB** rule.
- (i) **LOCAL**: first check whether name is in local environment, if yes Python uses its value otherwise moves to (ii)
- (ii) **ENCLOSING ENVIRONMENT**: if not in local, Python checks whether name is in Enclosing Environment, if yes Python uses its value otherwise moves to (iii)
- **GLOBAL ENVIRONMENT**: if not in above scope Python checks it in Global environment, if yes Python uses it otherwise moves to (iv)
- **BUILT-IN ENVIRONMENT**: if not in above scope, Python checks it in built-in environment, if yes, Python uses its value otherwise Python would report the error:
- name <variable> not defined

Predict the output



```
ooo1.py - C:/Users/Lab Admin/AppD...
File Edit Format Run Options Window Help

def check():
    value=100
    print(value)

value=600
print(value)
check()
print(value)

Ln: 8 Col: 12
```

Program with
variable “value” in
both LOCAL and
GLOBAL SCOPE

Predict the output

```
ooo1.py - C:/Users/Lab Admin/AppD...  —  □  ×
File Edit Format Run Options Window Help
def check():
    value=100
    print(value)

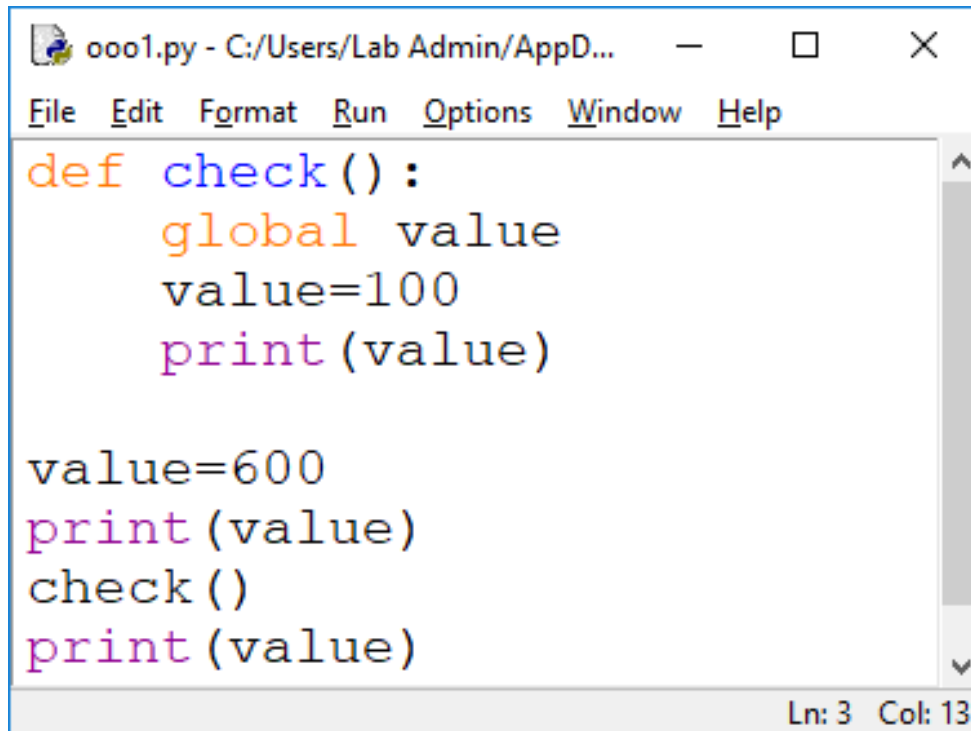
value=600
print(value)
check()
print(value)
```

Ln: 8 Col: 12

```
RESTART: C:/Users/Lab A
600
100
600
>>> |
```

Program with
variable "value" in
both LOCAL and
GLOBAL SCOPE

Predict the output



```
ooo1.py - C:/Users/Lab Admin/AppD...
File Edit Format Run Options Window Help

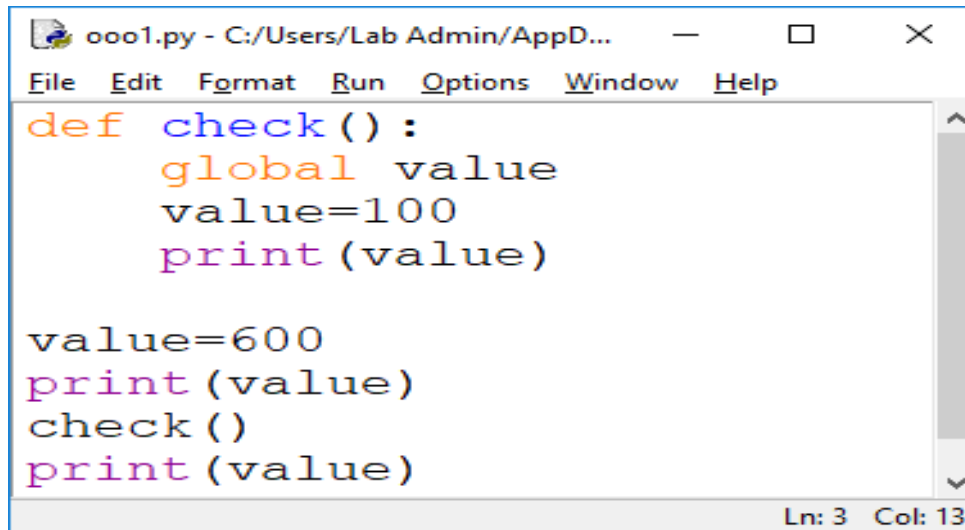
def check():
    global value
    value=100
    print(value)

value=600
print(value)
check()
print(value)
```

Ln: 3 Col: 13

Using GLOBAL
variable "value" in
local scope

Predict the output

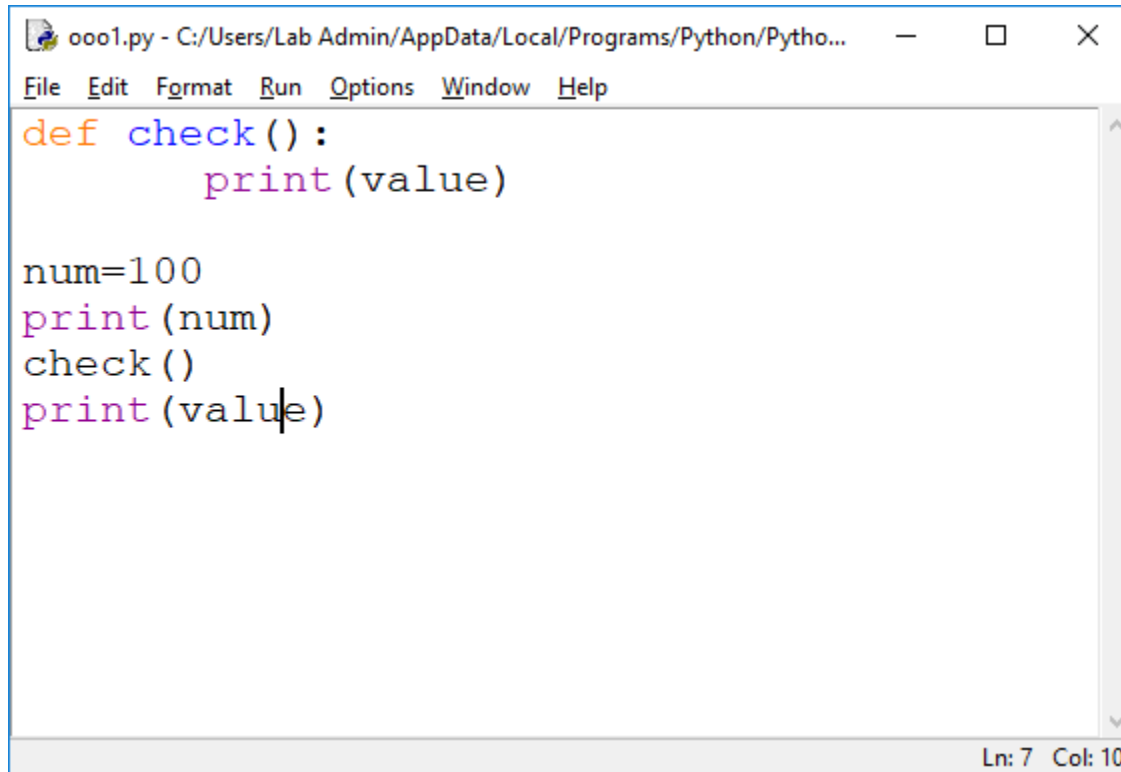


```
ooo1.py - C:/Users/Lab Admin/AppD...  
File Edit Format Run Options Window Help  
def check():  
    global value  
    value=100  
    print(value)  
  
value=600  
print(value)  
check()  
print(value)  
Ln: 3 Col: 13
```

Using GLOBAL
variable "value" in
local scope

```
RESTART: C:/Users/Lab Admin/AppD...  
600  
100  
100  
>>> |
```

Predict the output



The screenshot shows a Python IDE window titled 'ooo1.py - C:/Users/Lab Admin/AppData/Local/Programs/Python/Pytho...'. The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code in the editor is as follows:

```
def check():  
    print(value)  
  
num=100  
print(num)  
check()  
print(value)
```

The cursor is positioned at the end of the last line, 'print(value)'. The status bar at the bottom right indicates 'Ln: 7 Col: 10'.

Variable “value”
neither in local nor
global scope

Predict the output

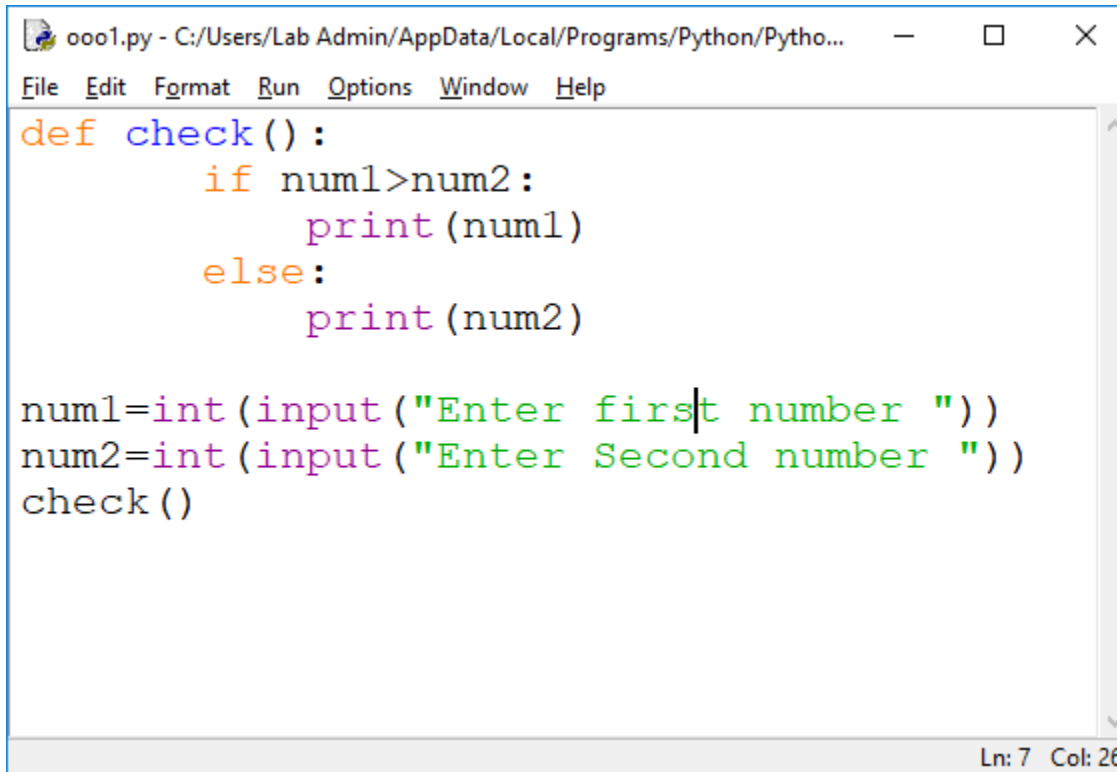
```
ooo1.py - C:/Users/Lab Admin/AppData/Local/Programs/Python/Pytho...
File Edit Format Run Options Window Help
def check():
    print(value)

num=100
print(num)
check()
print(value)
```

```
Traceback (most recent call last):
  File "C:/Users/Lab Admin/AppData/Local/Prog
ine 6, in <module>
    check()
  File "C:/Users/Lab Admin/AppData/Local/Prog
ine 2, in check
    print(value)
NameError: name 'value' is not defined
>>> |
```

Variable “value”
neither in local nor
global scope

Predict the output



```
ooo1.py - C:/Users/Lab Admin/AppData/Local/Programs/Python/Pytho...
File Edit Format Run Options Window Help

def check():
    if num1>num2:
        print(num1)
    else:
        print(num2)

num1=int(input("Enter first number "))
num2=int(input("Enter Second number "))
check()

Ln: 7 Col: 26
```

Variable in Global
not in Local
(input in variable at
global scope)

Predict the output

```
ooo1.py - C:/Users/Lab Admin/AppData/Local/Programs/Python/Pytho...
File Edit Format Run Options Window Help
def check():
    if num1>num2:
        print(num1)
    else:
        print(num2)

num1=int(input("Enter first number "))
num2=int(input("Enter Second number "))
check()
```

```
RESTART: C:/Users/Lab Ad
Enter first number 20
Enter Second number 10
20
>>>
```

Variable in Global
not in Local
(input in variable at
global scope)

Mutability/Immutability of Arguments/Parameters and function call

```
def check(num) :  
    print("*****")  
    print("value in function check() is ",num)  
    print("Id of num in function is ",id(num))  
    num+=10  
    print("value in function check() is ",num)  
    print("Id of num in function is ",id(num))  
    print("*****")  
myvalue=100  
print("value in main function is ",myvalue)  
print("Id of value in function is ",id(myvalue))  
check(myvalue)  
print("value in main function is ",myvalue)  
print("Id of value in function is ",id(myvalue))
```


Mutability/Immutability of Arguments/Parameters and function call

```
def check(num):  
    print("*****")  
    print("value in function check() is ",num)  
    print("Id of num in function is ",id(num))  
    num+=10  
    print("value in function check() is ",num)  
    print("Id of num in function is ",id(num))  
    print("*****")  
  
myvalue=100  
print("value in main function is ",myvalue)  
print("Id of value in function is ",id(myvalue))  
check(myvalue)  
print("value in main function is ",myvalue)  
print("Id of value in function is ",id(myvalue))
```

```
value in main function is 100  
Id of value in function is 1701457376  
*****  
value in function check() is 100  
Id of num in function is 1701457376  
value in function check() is 110  
Id of num in function is 1701457536  
*****  
value in main function is 100  
Id of value in function is 1701457376
```

Mutability/Immutability of Arguments/Parameters and function call

- From the previous example we can recall the concept learned in class XI that **Python variables are not storage containers**, rather Python variables are like memory references, they refer to memory address where the value is stored, thus **any change in immutable type data will also change the memory address**. So **any change to formal argument will not reflect back to its corresponding actual argument** and in case of mutable type, **any change in mutable type will not change the memory address of variable**.

Mutability/Immutability of Arguments/Parameters and function call

```
def updateData(myList):  
    print(myList)  
    for i in range(len(myList)):  
        myList[i] += 100  
    print(myList)  
  
List1 = [10, 20, 30, 40, 50]  
print(List1)  
updateData(List1)  
print(List1)
```

```
(base) C:\ProgramData\Minio  
[10, 20, 30, 40, 50]  
[10, 20, 30, 40, 50]  
[110, 120, 130, 140, 150]  
[110, 120, 130, 140, 150]
```

Because List is Mutable type, hence any change in formal argument **myList** will not change the memory address, So changes done to **myList** will be reflected back to **List1**.

However if we formal argument is assigned to some other variable or data type then link will break and changes will not reflect back to ACTUAL argument

For example (if inside function updateData()) we assign myList as:

myList = 20 OR myList = temp

Passing String to function

- ❑ Function can accept string as a parameter
- ❑ As per Python, string is immutable type, so function can access the value of string but cannot alter the string
- ❑ To modify string, the trick is to take another string and concatenate the modified value of parameter string in the newly created string.
- ❑ Let us see few examples of passing string to function...

Passing string to function and count how many vowels in it

```
#passing string to function
def countvowels(str1):
    count=0
    vowels=['a','e','i','o','u']
    for s in str1:
        if s.lower() in vowels:
            count+=1
    return count

msg = input("Enter Any String :")
c = countvowels(msg)
print("Total vowels in ", msg , " are ",c)
```

```
Enter Any String :the quick brown fox
Total vowels in  the quick brown fox  are  5
```

Program to count how many times any character is present in string

```
def countChar(str1,ch):  
    count=0  
    for s in str1:  
        if s == ch:  
            count+=1  
    return count  
  
msg = input("Enter Any String :")  
ch = input("Enter character to count in String :")  
c = countChar(msg,ch)  
if c==0:  
    print("Sorry! ",ch," is not in ",msg)  
else:  
    print(ch, " found ",c," times in ",msg)
```

```
Enter Any String :the quick brown fox  
Enter character to count in String :o  
o found 2 times in the quick brown fox
```

Program to Jumble the given string by passing it to function using temporary string

```
def Jumble(str1):  
    str2=""  
    count=0  
    for i in range(len(str1)):  
        if str1[i].islower():  
            str2=str2+str1[i].upper()  
        elif str1[i].isupper():  
            str2=str2+str1[i].lower()  
        elif str1[i].isdigit():  
            str2=str2+"*"  
        else:  
            str2=str2+"@"  
    return str2  
msg = input("Enter Any String :")  
msg = Jumble(msg)  
print("Jumbled String is :",msg)
```

```
Enter Any String :Hello India 123 Go Corona  
Jumbled String is : hELLO@iNDIA@***@gO@cORONA
```

Passing List to function

- We can also pass List to any function as parameter
- Due to the mutable nature of List, function can alter the list of values in place.
- It is mostly used in data structure like sorting, stack, queue etc.
- Let us see how to pass List to function by few examples:

Passing list to function, and just double each value

```
def DoubleIt(mylist):  
    for i in range(len(mylist)):  
        mylist[i]=mylist[i]*2
```

```
mylist = [10,20,30,40,50,60]  
DoubleIt(mylist)  
print("New List is :",mylist)
```

```
New List is : [20, 40, 60, 80, 100, 120]
```

Passing list to function to double the odd values and half the even values

```
def EvenOdd(mylist):  
    for i in range(len(mylist)):  
        if mylist[i]%2==0:  
            mylist[i]//=2  
        else:  
            mylist[i]*=2  
  
mylist = [11,20,30,40,17,5,3,2]  
EvenOdd(mylist)  
print("New List is :",mylist)
```

```
New List is : [22, 10, 15, 20, 34, 10, 6, 1]
```

Passing nested list to function and print all those values which are at diagonal position in the form of matrix

```
def PrintDiagonal(mylist):  
    for i in range(len(mylist)):  
        for j in range(len(mylist[i])):  
            if i==j:  
                print(mylist[i][j],end='\t')  
            else:  
                print('\t',end='')  
        print()  
  
mymatrix = [  
    [10,20,30],  
    [40,50,60],  
    [70,80,90]  
]  
PrintDiagonal(mymatrix)
```

10		
	50	
		90

Passing list to function to calculate sum and average of all numbers and return it in the form of tuple

```
def GetSumAverage(mylist):
    sum=0
    for i in mylist:
        sum+=i
    avg = sum/len(mylist)
    return sum,avg

mylist=[]
n = int(input("Enter how many numbers :"))
for i in range(n):
    num = int(input("Enter any number :"))
    mylist.append(num)

print(mylist)
sumavg = GetSumAverage(mylist)    # sumavg will be a tuple
print("Sum = ",sumavg[0]," Average =",sumavg[1])
```

Passing list to function to calculate sum and average of all numbers and return it in the form of tuple

```
def GetSumAverage(mylist):
    sum=0
    for i in mylist:
        sum+=i
    avg = sum/len(mylist)
    return sum,avg

mylist=[]
n = int(input("Enter how many numbers :"))
for i in range(n):
    num = int(input("Enter any number :"))
    mylist.append(num)

print(mylist)
sumavg = GetSumAverage(mylist) # sumavg will be a tuple
print("Sum = ",sumavg[0], " Average =",sumavg[1])
```

```
Enter how many numbers :6
Enter any number :10
Enter any number :20
Enter any number :30
Enter any number :40
Enter any number :50
Enter any number :60
[10, 20, 30, 40, 50, 60]
Sum = 210 Average = 35.0
```

Passing tuples to function

- We can also pass tuples to function as parameter
- Due to its immutability nature, function can only access the values of tuples but cannot modify it.
- Let us see how to pass tuples in function by few example...

Input n numbers in tuple and pass it function to count how many even and odd numbers are entered.

```
def CountEvenOdd(mytuple):
    counteven=0
    countodd=0
    for i in mytuple:
        if i%2==0:
            counteven+=1
        else:
            countodd+=1
    return counteven,countodd    #return multiple values

mytuple=()
n = int(input("Enter how many numbers :"))
for i in range(n):
    num = int(input("Enter any number :"))
    mytuple = mytuple +(num,)
eocount = CountEvenOdd(mytuple)    #eocount will also be a tuple
print("Even Counts are :",eocount[0]," Odd Counts are :",eocount[1])
```

Input n numbers in tuple and pass it function to count how many even and odd numbers are entered.

```
def CountEvenOdd(mytuple):  
    counteven=0  
    countodd=0  
    for i in mytuple:  
        if i%2==0:  
            counteven+=1  
        else:  
            countodd+=1  
    return counteven,countodd #return multiple values  
  
mytuple=()  
n = int(input("Enter how many numbers :"))  
for i in range(n):  
    num = int(input("Enter any number :"))  
    mytuple = mytuple +(num,)  
eocount = CountEvenOdd(mytuple) #eocount will also be a tuple  
print("Even Counts are :",eocount[0]," Odd Counts are :",eocount[1])
```

```
Enter how many numbers :6  
Enter any number :10  
Enter any number :21  
Enter any number :43  
Enter any number :20  
Enter any number :17  
Enter any number :19  
Even Counts are : 2  Odd Counts are : 4
```


Creating a login program with the help of passing tuple to function

```
def loginCheck(users,passwords,inputuser,inputpassword):  
    index=None  
    password=None  
    if inputuser in users:  
        index=users.index(inputuser)  
        password=passwords[index]  
        if password==inputpassword:  
            return "Login Successful!"  
        else:  
            return "Invalid Password Entered"  
    else:  
        return "Invaidd User Name"  
  
usernames=('Admin','System','Guest')  
passwords=('admin#123','manager','guest')  
print("..... Login ")  
print("*****")  
u = input("Enter User Name :")  
p = input("Enter Password :")  
response = loginCheck(usernames,passwords,u,p)  
print(response)
```

Creating a login program with the help of passing tuple to function

```
..::: Login
*****
Enter User Name :Admin
Enter Password :admin#123
Login Successful!
```

```
..::: Login
*****
Enter User Name :Admin
Enter Password :password
Invalid Password Entered
```

```
..::: Login
*****
Enter User Name :System
Enter Password :manager
Login Successful!
```

```
..::: Login
*****
Enter User Name :amit
Enter Password :admin#123
Invalid User Name
```

OUTPUT OF PREVIOUS PROGRAM

Passing Dictionary to function

- Python also allows us to pass dictionaries to function
- Due to its mutability nature, function can alter the keys or values of dictionary in place
- Let us see few examples of how to pass dictionary to functions.

Passing dictionary to function with list and stores the value of list as key and its frequency or no. of occurrence as value

```
def frequencyCount(mylist,myd):  
    for i in mylist:  
        if i not in myd:  
            myd[i]=1  
        else:  
            myd[i]+=1  
    return myd  
  
mylist=[10,20,10,20,30,10,40,10,40]  
d={}  
frequencyCount(mylist,d)  
print(d)
```

```
{10: 4, 20: 2, 30: 1, 40: 2}
```

Passing dictionary to function with key and value, and update value at that key in dictionary

```
def updateValue (myd, key, name) :  
    myd[key]=name  
  
student={'roll':1, 'name':'amit', 'per':89}  
  
print (student)  
print ("1. Change roll")  
print ("2. Change name")  
print ("3. Change percentage")  
choice=int(input("Enter your choice :"))  
key=None  
if choice==1:  
    r = int(input("Enter new rol number "))  
    key='roll'  
elif choice==2:  
    r=input("Enter new name ")  
    key='name'  
elif choice==3:  
    r=int(input("Enter new percentage :"))  
    key='per'  
else:  
    print("# Invalid Choice #")  
  
if key!=None:  
    updateValue (student, key, r)  
print (student)
```

Passing dictionary to function with key and value, and update value at that key in dictionary


```
{'roll': 1, 'name': 'amit', 'per': 89}
1. Change roll
2. Change name
3. Change percentage
Enter your choice :2
Enter new name Sumit
{'roll': 1, 'name': 'Sumit', 'per': 89}
>>>
===== RESTART: G:/python4csip.com/fv
{'roll': 1, 'name': 'amit', 'per': 89}
1. Change roll
2. Change name
3. Change percentage
Enter your choice :3
Enter new percentage :92
{'roll': 1, 'name': 'amit', 'per': 92}
```

Understanding of main() function in Python

- By default every program starts their execution from main() function. In Python including a main() function is not mandatory. It can structure our Python programs in a logical way that puts the most important components of the program in one function.
- We can get the name of current module executing by using built-in variable **__name__** (2 underscore before and after of name)

Understanding of main() function in Python


```
print("Welcome!!")  
print("My Name is ", __name__)
```



```
Welcome!!  
My Name is __main__
```

We can observe, by default the name of module will be `__main__`

```
def area(length,breadth):  
    return length * breadth  
  
def main():  
    l = int(input("Enter Length"))  
    b = int(input("Enter Breadth"))  
    a = area(l,b)  
    print("Area of Rectangle =",a)  
  
if __name__ == '__main__':  
    main()
```



Most non-python programmers are having the habit of writing `main()` function where the important and starter code of programs are written. **In Python we can also create `main()` and call it by checking `__name__` to `__main__` and then call any function, in this case `main()`**

Recursion

- It is one of the most powerful tool in programming language. It is a process where function calls itself again and again.
- Recursion basically divides the big problem into small problems up to the point where it can be solved easily, for example if we have to calculate factorial of a 5, we will divide factorial of 5 as $5 * \text{factorial}(4)$, then $4 * \text{factorial}(3)$, then $3 * \text{factorial}(2)$, then $2 * \text{factorial}(1)$ and now factorial of 1 can be easily solved without any calculation, now each pending function will be executed in reverse order.

Condition for Implementing Recursion

- It must contain **BASE CONDITION** i.e. at which point recursion will end otherwise it will become infinite.
- **BASE CONDITION** is specified using „if“ to specify the termination condition
- Execution in Recursion is in reverse order using STACK. It first divide the large problem into smaller units and then starts solving from bottom to top.
- It takes more memory as compare to LOOP statement because with every recursion call memory space is allocated for local variables.
- The computer may run out of memory if recursion becomes infinite or termination condition not specified.
- It is less efficient in terms of speed and execution time
- Suitable for complex data structure problems like TREE, GRAPH etc

Example - Recursion

```
def factorial(num):  
    if num == 1:  
        return 1  
    else:  
        return num * factorial(num-1)  
  
n = int(input("Enter any number "))  
f = factorial(n)  
print("Factorial of ", n, " is ", f)
```

```
Enter any number 5  
Factorial of 5 is 120
```

First time num=5, so goes to else:

return 5 * factorial(4) # pushed to Stack..(i)

Second time num=4, so goes to else:

return 4 * factorial(3) # pushed to Stack..(ii)

Third time num=3, so goes to else:

return 3 * factorial(2) # pushed to Stack..(iii)

Fourth time num=2, so goes to else:

return 2 * factorial(1) # pushed to Stack..(iv)

Fifth time num=1, so goes to if:

return 1 # goes to (iv)

then goes to (iii) then goes to (ii) then goes to (i) and finally back to calling function

Example - Recursion

```
def fibonacci(num):  
    if num <= 1:  
        return num  
    else:  
        return (fibonacci(num-1)+fibonacci(num-2))  
  
n = int(input("Enter how many terms "))  
for i in range(n):  
    print(fibonacci(i))
```

```
Enter how many terms 10  
0  
1  
1  
2  
3  
5  
8  
13  
21  
34
```

Questions based on functions

- WAP to create function Lsearch() which takes List and number to search and return the position of number in list using Linear Searching method
- WAP to create function Bsearch() which takes List and number to search and return the position of number in list using Binary Searching method
- What is the difference between Local and Global variables? Also give suitable Python code to illustrate both