CHAPTER 3 : WORKING WITH FUNCTIONS

Understanding Functions

Defining

Flow of execution

Passing Parameters

Returning values from functions

Composition

Scope of variables

FUNCTIONS

- A function is a subprogram that acts on data and often returns a value.
- Functions are named sequence that performs a computation.
- It contains lines of code(s) that are executed sequentially from top to bottom by Python interpreter.

Defining Functions in Python

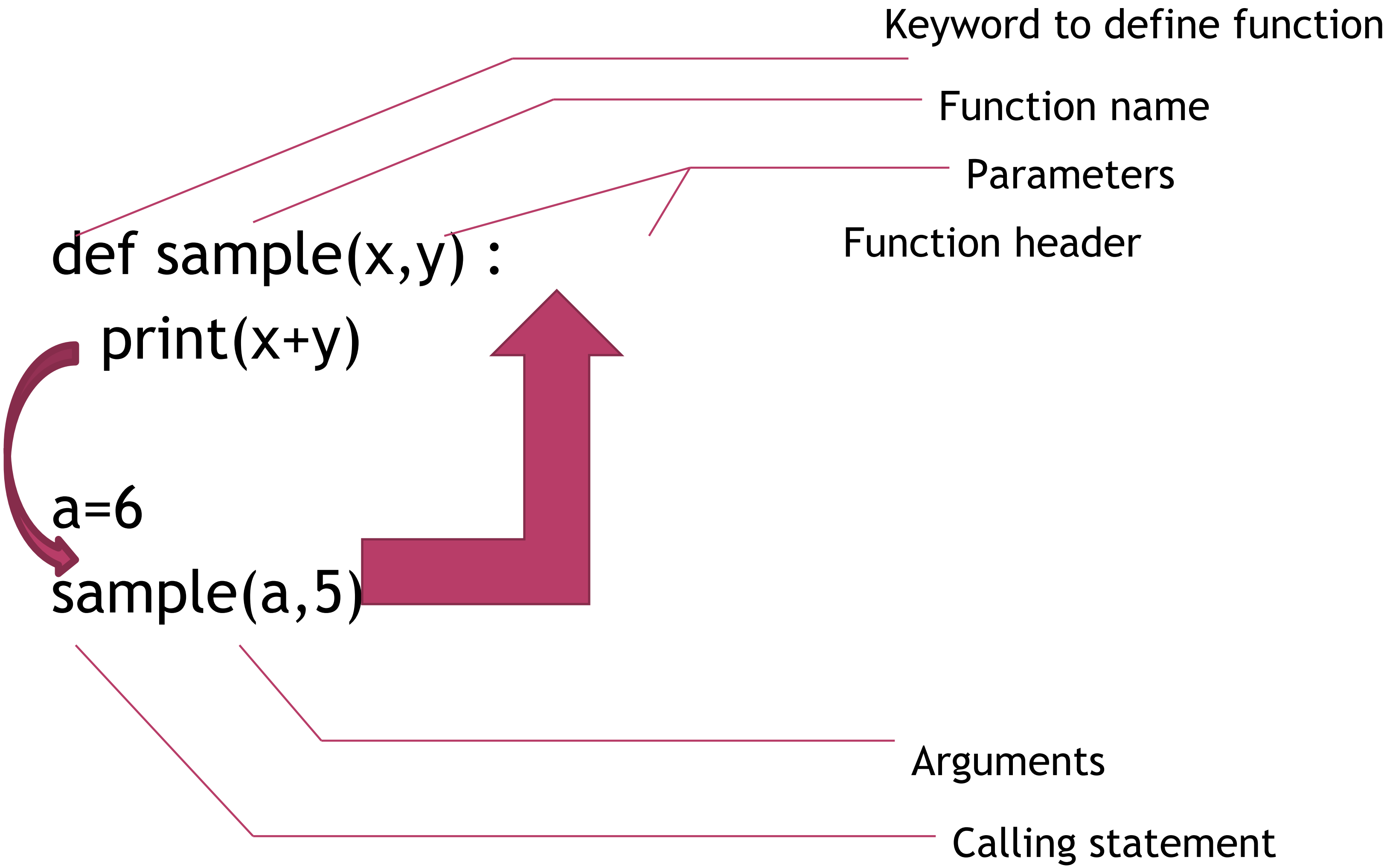
```
def <function name> ([parameters]): #function header
        <statement>
        :
        :
        [<statement>]

#top level segment
__main__

# by default, Python names the segment with top-level
statements(main program) as __main__)
```

TYPES OF FUNCTIONS

- USER DEFINED FUNCTIONS
- BUILT-IN FUNCTIONS
- FUNCTIONS DEFINED IN MODULE

USER DEFINED FUNCTIONS

EX (VOID TYPE /NON-FRUITFUL FUNCTIONS)



```
def sample(x,y) :
  print(x+y)


a=6
sample(a,5)
```

Keyword to define function

Function name

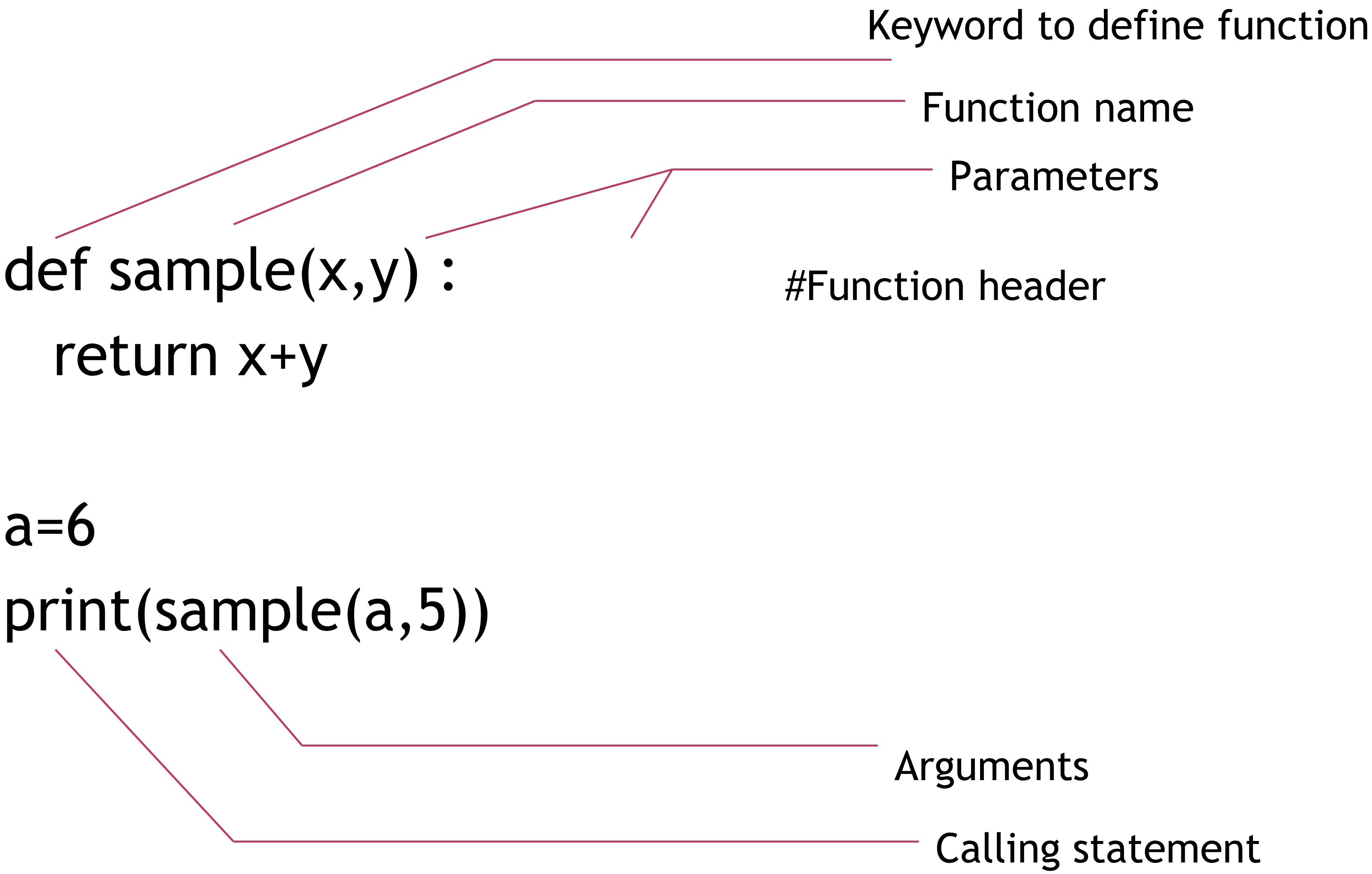
Parameters

Function header

Arguments

Calling statement

EXAMPLE (NON - VOID/FRUITFUL TYPE)



Keyword to define function

Function name

Parameters

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

#Function header

```
a=6
print(sample(a,5))
```

Arguments

Calling statement

Arguments – literals/variables/expression passed from calling(caller) to called(callee)

Parameters – variables used in called function

PASSING PARAMETERS

- POSITIONAL (REQUIRED or MANDATORY)
- DEFAULT
- KEYWORD or NAMED
- VARIABLE LENGTH

POSITIONAL ARGUMENTS

<table>
<tr><th>Example</th><th>Output</th></tr>
<tr><td>

```
#example 1 for positional arguments

def check (a,b):
    if a > b:
        return a
    else:
        return b

#top level program area or __main__ part
a,b = 5,6
print(check(a,b))
```

</td><td>6</td></tr>
</table>

POSITIONAL ARGUMENTS

Example	Output

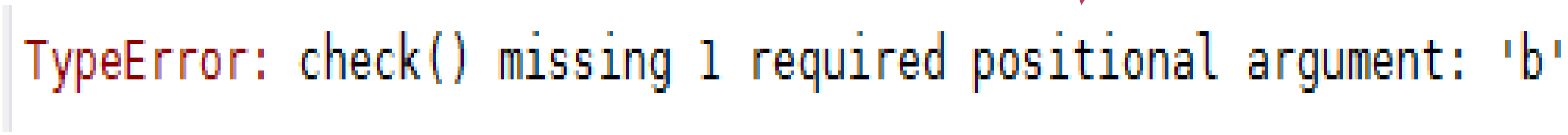
#example 2 for positional arguments
 	
def check (a,b):
    if a > b:
        return a
    else:
        return b
 	
#top level program area or __main__ part
a = 16
print(check(a,7))
	16

POSITIONAL ARGUMENTS

<table>
<tr><th>Example</th><th>Output</th></tr>
<tr><td><pre>
#example 3 for positional arguments

def check (a,b):
    if a > b:
        return a
    else:
        return b

#top level program area or __main__ part
a = 16
print(check(a))
</pre></td><td>Error</td></tr>
</table>



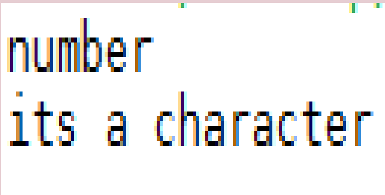
```
TypeError: check() missing 1 required positional argument: 'b'
```

POSITIONAL ARGUMENTS

<table>
<tr><th>Example</th><th>Output</th></tr>
<tr><td>

```python
#example 4 for positional arguments

def check (a,b):
    if a==0:
        print('number ')
    if b=='a':
        print('its a character')

#top level program area or __main__ part

check('a',0)
check(0,'a')
```

</td><td>



```
number
its a character
```

</td></tr>
</table>

POSITIONAL PARAMETERS

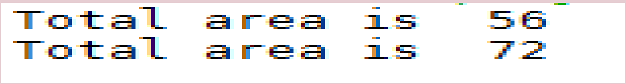
- Need to match the number of parameters
- For all arguments values must be provided
- Values of arguments are to be matched with parameter position(order) wise (positional)

DEFAULT ARGUMENTS

Example 1

```
#example 1 for default
arguments

def area_calc(l,b=9):
    print('Total area is ',l*b)

#top level part

area_calc(8,7)

area_calc(8)
```

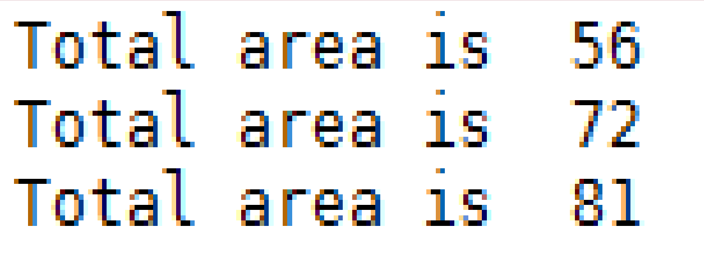
Output



```
Total area is  56
Total area is  72
```

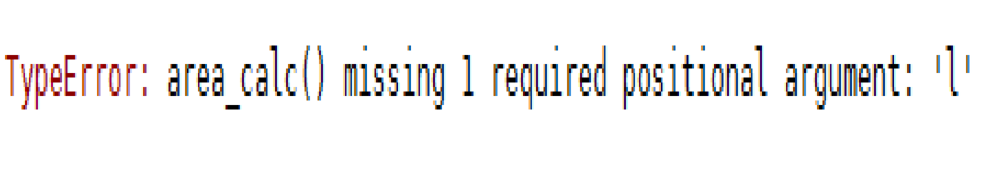
DEFAULT ARGUMENTS

Example 2	Output

#example 2 for default arguments
 	
def area_calc(l=9,b=9):
    print('Total area is ',l*b)
 	
#top level part
 	
area_calc(8,7)
 	
area_calc(8)
 	
area_calc()
	
	Total area is 56
	Total area is 72
	Total area is 81

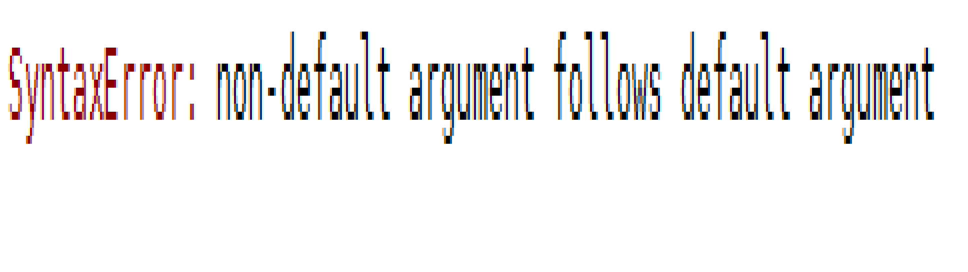
DEFAULT ARGUMENTS

Example 3	Output

#example 3 for default arguments
 	
def area_calc(l,b=9):
    print('Total area is ',l*b)
 	
#top level part
 	
area_calc()
	
	TypeError: area_calc() missing 1 required positional argument: 'l'

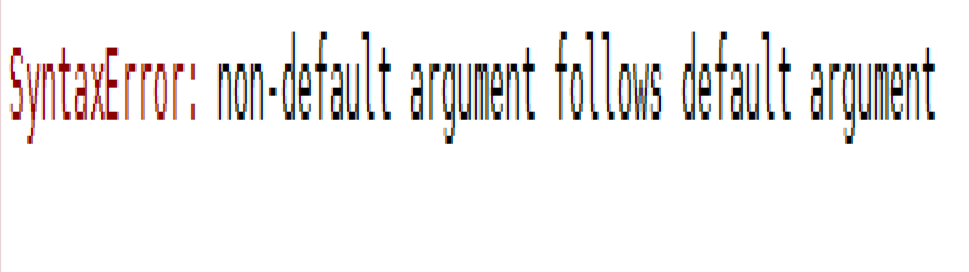
DEFAULT ARGUMENTS

Example 4	Output

#example 4 for default arguments
 	
def area_calc(l=3,b):
    print('Total area is ',l*b)
 	
#top level part
 	
area_calc()
	
	SyntaxError: non-default argument follows default argument

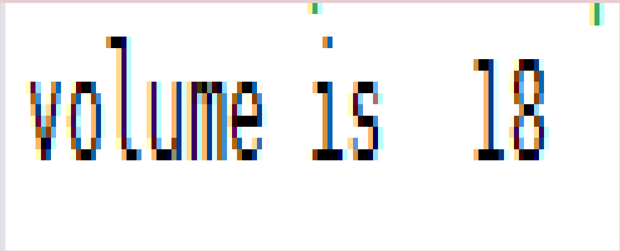
DEFAULT ARGUMENTS

Example 5	Output

#example 5 for default arguments
 	
def vol_calc(l=2,b,h=3):
    print('volume is ',l*b*h)
 	
#top level part
 	
vol_calc()
	
	SyntaxError: non-default argument follows default argument

DEFAULT ARGUMENTS

Example 6	Output

#example 6 for default arguments
 	
def vol_calc(b,l=2,h=3):
    print('volume is ',l*b*h)
 	
#top level part
 	
vol_calc(3)
	
	volume is 18

DEFAULT PARAMETER

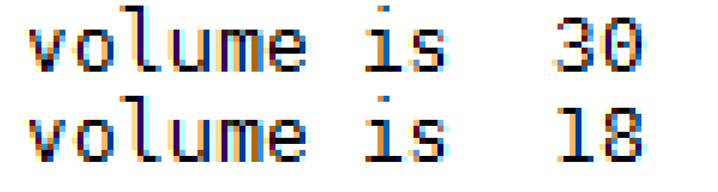
- Parameter having default value in function header is known as default argument
- Facilitates partial, full or no argument list or call
- From right to left (non-default argument cannot follow default)
- In a function header, a parameter cannot have a default value unless all parameters appearing on its right have their default values.

KEYWORD ARGUMENTS

Example 1

```
#example 1 for keyword arguments

def vol_calc(b,l=2,h=3):
    print('volume is ',l*b*h)

#top level part

vol_calc(h=5,b=3,l=2)

vol_calc(b=3)
```

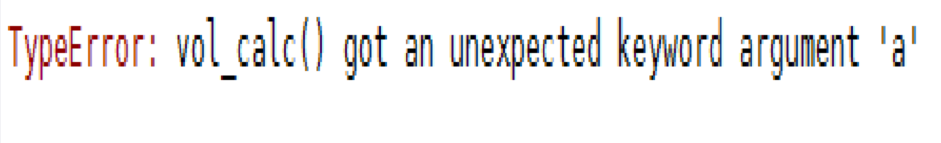
Output



```
volume is  30
volume is  18
```

KEYWORD ARGUMENTS

Example 2	Output

#ex 2 for keyword arguments
 	
def vol_calc(b,l=2,h=3):
    print('volume is ',l*b*h)
 	
#top level part
 	
vol_calc(a=5,b=3,l=2)
	
	TypeError: vol_calc() got an unexpected keyword argument 'a'

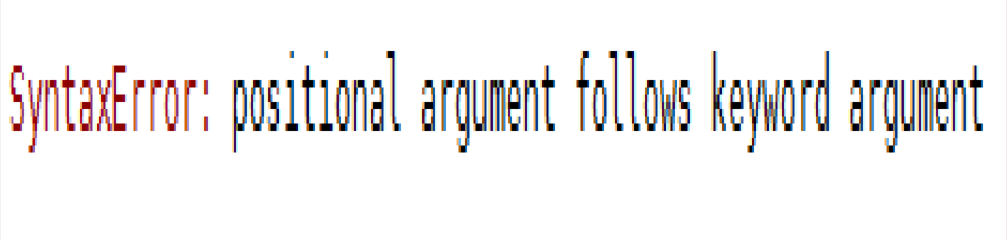
KEYWORD OR NAMED

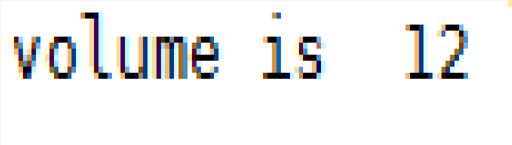
- Named arguments with assigned values being passed in the function call
- Argument can be named and sent in any order, but named keyword argument should be matched with parameter name used

MULTIPLE ARGUMENTS - IN FUNCTION CALL

- Keyword argument before positional is an error (or) Argument list must first contain positional followed by keyword

Example 1	Output

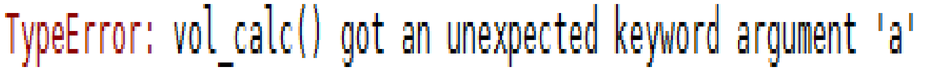
#ex 1 for multiple arguments
 	
def vol_calc(b,l=2,h=3):
    print('volume is ',l*b*h)
 	
#top level part
 vol_calc(b=3,3,l=2)
	
	SyntaxError: positional argument follows keyword argument

#ex 2 for multiple arguments
 	
def vol_calc(b,l=2,h=3):
    print('volume is ',l*b*h)
 	
#top level part
vol_calc(3,2,h=2)
	
	volume is 12

MULTIPLE ARGUMENTS – IN FUNCTION CALL

- Keyword arguments should be taken from the required arguments

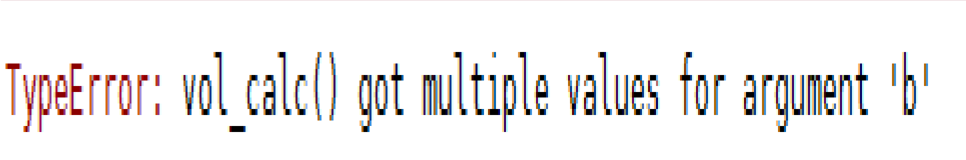
Example 2	Output

#ex 2 for keyword arguments
 	
def vol_calc(b,l=2,h=3):
    print('volume is ',l*b*h)
 	
#top level part
 	
vol_calc(a=5,b=3,l=2)
	
	TypeError: vol_calc() got an unexpected keyword argument 'a'

MULTIPLE ARGUMENTS – IN FUNCTION CALL

- Value for an argument cannot be specified more than once

Example	Output

#ex 3 for multiple arguments
 	
def vol_calc(b,l=2,h=3):
    print('volume is ',l*b*h)
 	
#htop level part
 	
vol_calc(3,2,b=2)
	
	TypeError: vol_calc() got multiple values for argument 'b'

RETURNING VALUES FROM FUNCTIONS

Non-void (fruitful)	Void (non-fruitful)
If the return value is not used in function, Python will not throw error	Can also have return statement, but it will return None to the caller
return statement marks the end of function, any statement after that will not be executed	

```
1)  def sample() :
          print(‘***’)
          return
    sample()
```

Output

```
***
```

```
2) def sample():
         return(‘$$$’)

   sample()
```

Output

No output

```
3) def sample():
         print(‘***’)

   print(sample())
```

Output

```
***
None
```

RETURNING MULTIPLE VALUES

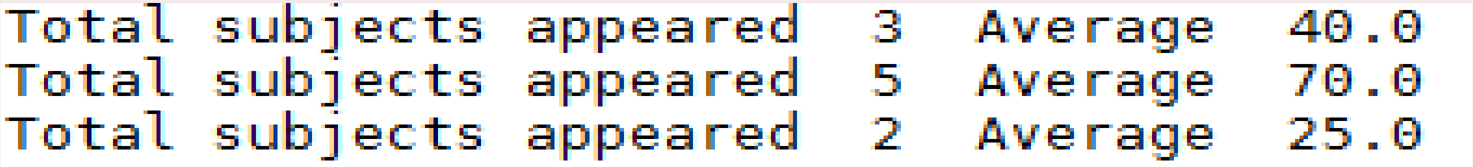
```
def sample():
   a,b,c,=9,8,7
   return a,b,c

t=sample()  #tuple will be created
print(t)

(or)
a,b,c = sample()
print(a,b,c)
```

VARIABLE LENGTH ARGUMENTS

Example

```
# VARIABLE LENGTH ARGUMENTS IN FUNCTION
def subject_avg(*sub):
    tot,count = 0,0
    for i in sub:

        tot = tot + i
        count+=1
    print('Total subjects appeared ',len(sub),' Average ',tot/count)

subject_avg(30,40,50)

subject_avg(70,50,60,80,90)

subject_avg(20,30)
```

Output :



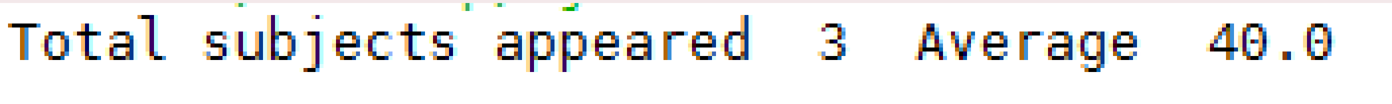
```
Total subjects appeared  3  Average  40.0
Total subjects appeared  5  Average  70.0
Total subjects appeared  2  Average  25.0
```

PASSING ARRAY/LISTS TO FUNCTIONS

Example

```
# passing list as ARGUMENTS IN FUNCTION
def subject_avg(sub):
    tot = 0
    for i in sub:


        tot = tot + i


    print('Total subjects appeared ',len(sub),' Average ',tot/len(sub))



sub=[30,40,50]
subject_avg(sub)
```

Output :



```
Total subjects appeared  3  Average  40.0
```

COMPOSITION

- Is an art of combining simple functions to build more complicated ones (ie) result of one function is used as input to other.

```
int(str(‘2’))
```

```
import math
print(math.sqrt(math.pi*8))
```

- (or)

```
from math import sqrt,pi
print(sqrt(pi*8))
```

SCOPE OF VARIABLES

- Part(s) of a program within which a name(identifier) is legal and accessible, is called scope of the name.

- Lifetime of the name(identifier) – duration for which the variable exists is called its lifetime.

TYPES OF SCOPE

LOCAL SCOPE

- A name declared in the function body.(Parameters or **Formal** arguments)
- It can be used within the function and their blocks contained within it.

- Lifetime – is the time for which a variable or name remains in memory

Example	Output

def sample():
    a=6 # Local scope
    print(a)
sample()
	6

ENCLOSED SCOPE

- Names in local scope of any or all enclosing functions from inner to outer functions.

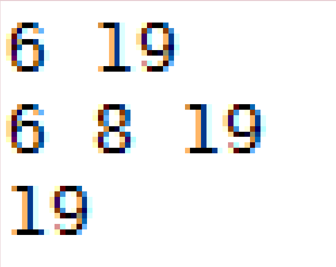
Example	Output

def try1(a):
        b=9
        print(a,b)
        a=10
        print(a)
 	
def fun():
    a=6
    print(a)
    try1(a)
    print(a)
 fun()
	6
	6 9
	10
	6

GLOBAL SCOPE

- A name declared in top level segment __main__ of a program is said to have global scope (usable inside the whole program and all blocks)

Example	Output

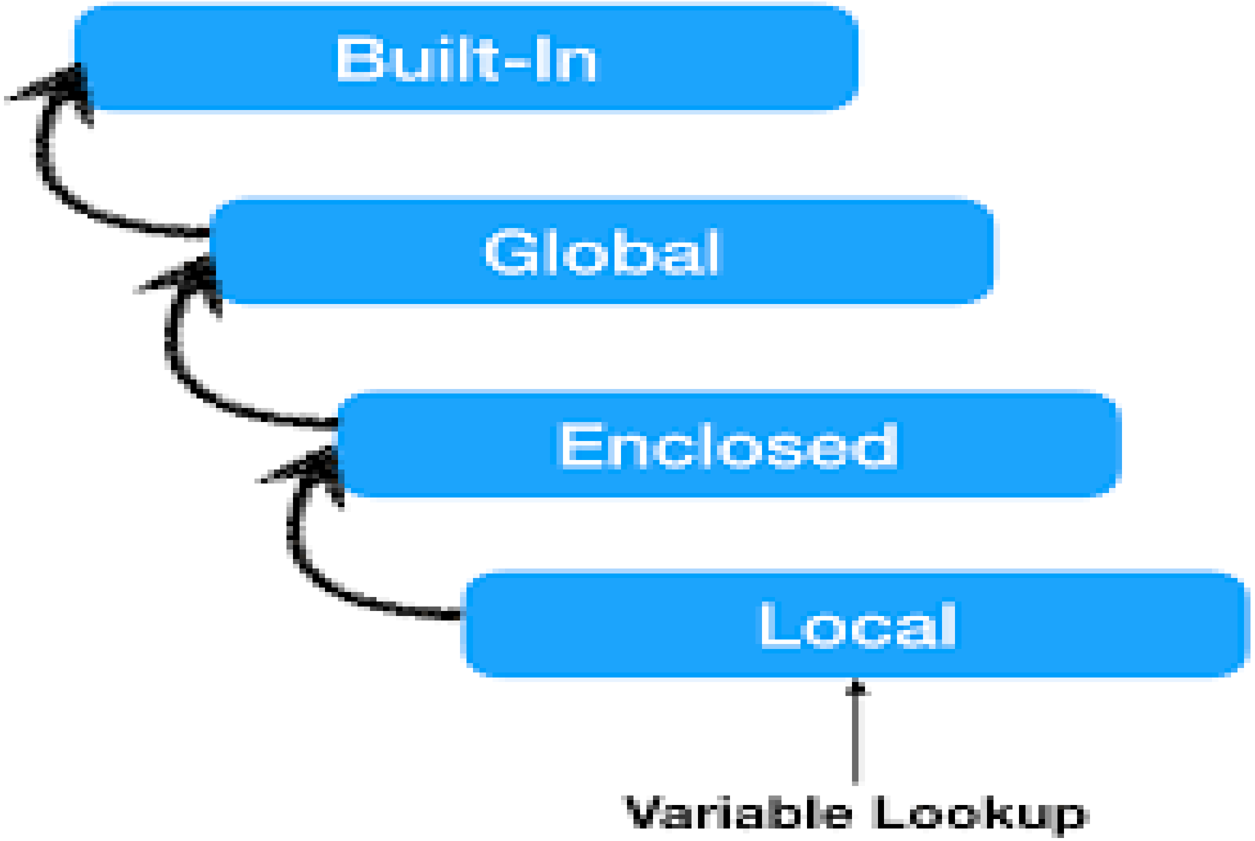
def trial(a):
    b=8   # local scope
    print(a,b,c)
def fun():
    a=6   #enclosing scope
    print(a,c)
    trial(a)
 	
c=19   #global
scope
fun()
print(c)
	
	6 19
	6 8 19
	19

BUILT-IN SCOPE

- Contains all built-in variables and functions of Python.
- If there is a variable with the same name; if yes, Python uses its value.

- Reserved words – keywords
- Functions – len(), id(), type() ....

HOW DOES PYTHON DECIDE ON SCOPE OF VARIABLES?



LOCAL VARIABLE VS GLOBAL VARIABLE

Local Variable	Global Variable
It is a variable which is declared within a function or within a block	It is a variable which is declared outside all the functions
It is accessible only within a function/block in which it is declared	It is accessible throughout the program

HOW DOES PYTHON DECIDE ON SCOPE OF VARIABLES?

- Variable in global scope but not in local scope – will be accessed (It will first check for the variable in local, then goes for global)

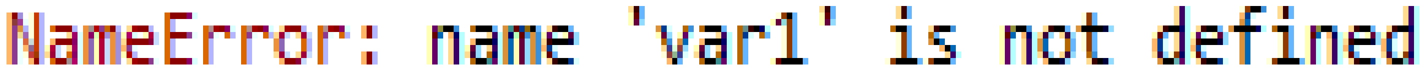
Example	Output

var = 6
def func1():
    print(var)
func1()
	6

HOW DOES PYTHON DECIDE ON SCOPE OF VARIABLES?

- Variable neither in local nor in global scope – NameError occurs

```
var = 6
def func1():
    print(var1)
func1()
```



```
NameError: name 'var1' is not defined
```

HOW DOES PYTHON DECIDE ON SCOPE OF VARIABLES?

- Same variable name in local as well as in global scope
  - Refers to only local scope
  - To refer to global variable use global statement (any change of value in global variable will be reflected in all its places)

Example 1:

```
var = 6
def func1():
    var = 5
    print(var)
func1()
```

Output:

```
5
```

Example 2:

```
def func1():
    global var
    print(var)
    var=8

var = 6
func1()
print(var)
```

Output:

```
6
8
```

MUTABLE/IMMUTABLE VARIABLES - THEIR EFFECTS IN FUNCTION CALL

- Python variables are not storage containers, rather Python variables are like memory references, they refer to memory address where values are stored.
- Depending on mutability/immutability the variable behave. If mutable the called function changes any value they are either reflected or not reflected.

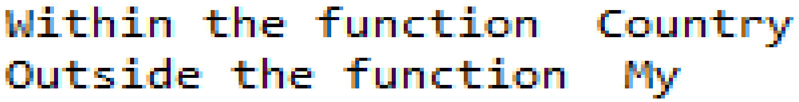
MUTABLE/IMMUTABLE VARIABLES - THEIR EFFECTS IN FUNCTION CALL

- Changes in Immutable types are not reflected in the caller function at all.

Example :

```
def sample(x):
    x='Country'
    print('Within the function ',x)      # immutable
x='My'
sample(x)
print('Outside the function ',x)
```

Output :



```
Within the function  Country
Outside the function  My
```

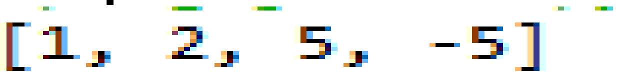
MUTABLE/IMMUTABLE VARIABLES - THEIR EFFECTS IN FUNCTION CALL

Changes, if any, in mutable types:

Option1 : are reflected in caller function if its name is not assigned a different variable or datatype.

```
def sample(x):
    x.append(-5)
z=[1,2,5]
sample(z)
print(z)
```

Output:



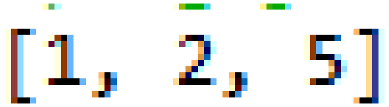
```
[1, 2, 5, -5]
```

MUTABLE/IMMUTABLE VARIABLES - THEIR EFFECTS IN FUNCTION CALL

- Changes, if any, in mutable types:
- Option 2: are not reflected in the called function if it is assigned a different variable or data type.

```
def sample(x):
    x=(1,6) #tuple
z=[1,2,5]   #list
sample(z)
print(z)
```

Output :



```
[1, 2, 5]
```

TO TRACE THE FLOW OF EXECUTION OF FUNCTION

EXAMPLE 1	FLOW OF EXECUTION
1. def add(x): 2. x = x + 1 3. 4. #top level segment 5. x=3 6. print(x) 7. add(x) 8. print(x)	1→ 5→6→7→1→2→8

TO TRACE THE FLOW OF EXECUTION OF FUNCTION

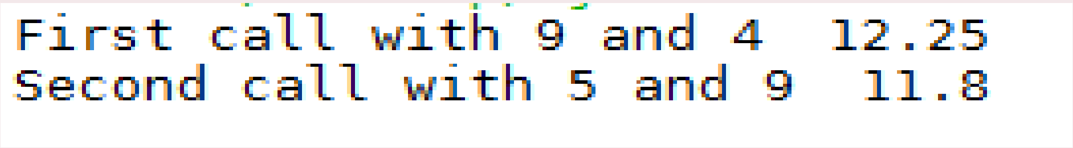
EXAMPLE 2	FLOW OF EXECUTION
1. def expo(x,y): 2. res = x ** y 3. return res 4. 5. def square(x): 6. res = expo(x,2) 7. return res 8. 9. x = 6 10. answer = square(x) 11. print(answer)	1→5→9→10→5→6→1→2→3→6 →7→10→11

ADDITIONAL PROGRAM

EXAMPLE

```
def division(a,b):
    d=10                                  #local scope
    if a<b:
        c = b/a + d      #d is enclosed scope, c,a,b are local
    else:
        c = a/b + d
    return c

if __name__ == '__main__':
    print('First call with 9 and 4 ',division(9,4))
    print('Second call with 5 and 9 ',division(5,9))
```

OUTPUT



```
First call with 9 and 4  12.25
Second call with 5 and 9  11.8
```

THANK U