Python Programming Language

Prepared by: Mohamed Ayman

Machine Learning Engineer

spring 2018

facebook.com/sw.eng.MohamedAyman

sw.eng.MohamedAyman@gmail.com

wuzzuf.net/me/engMohamedAyman

codeforces.com/profile/Mohamed_Ayman

Functions

Outline

Function Definition

- A function is a block of organized, reusable code that is used to perform a single, related action.
- Functions provide better modularity for your application and a high degree of code reusing. As you already know, Python given you many built-in function like print(), etc. but you can also create your own functions. These functions are called user defined functions.

Syntax

```
def functionname( parameters ):
   "function_docstring"
   function_suite
   return [expression]
```

Example

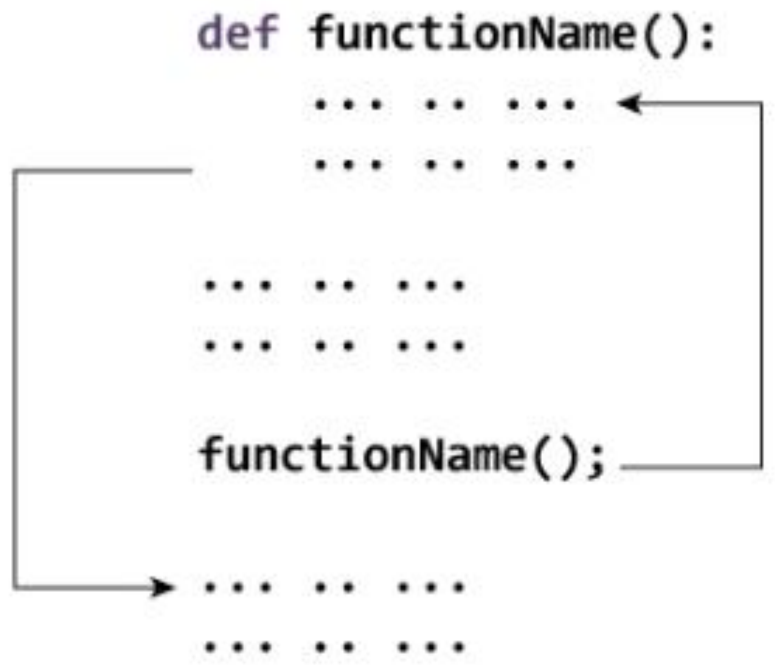
```
def printme( str ):
   "This prints a passed string into this function"
   print (str)
```

Function Definition

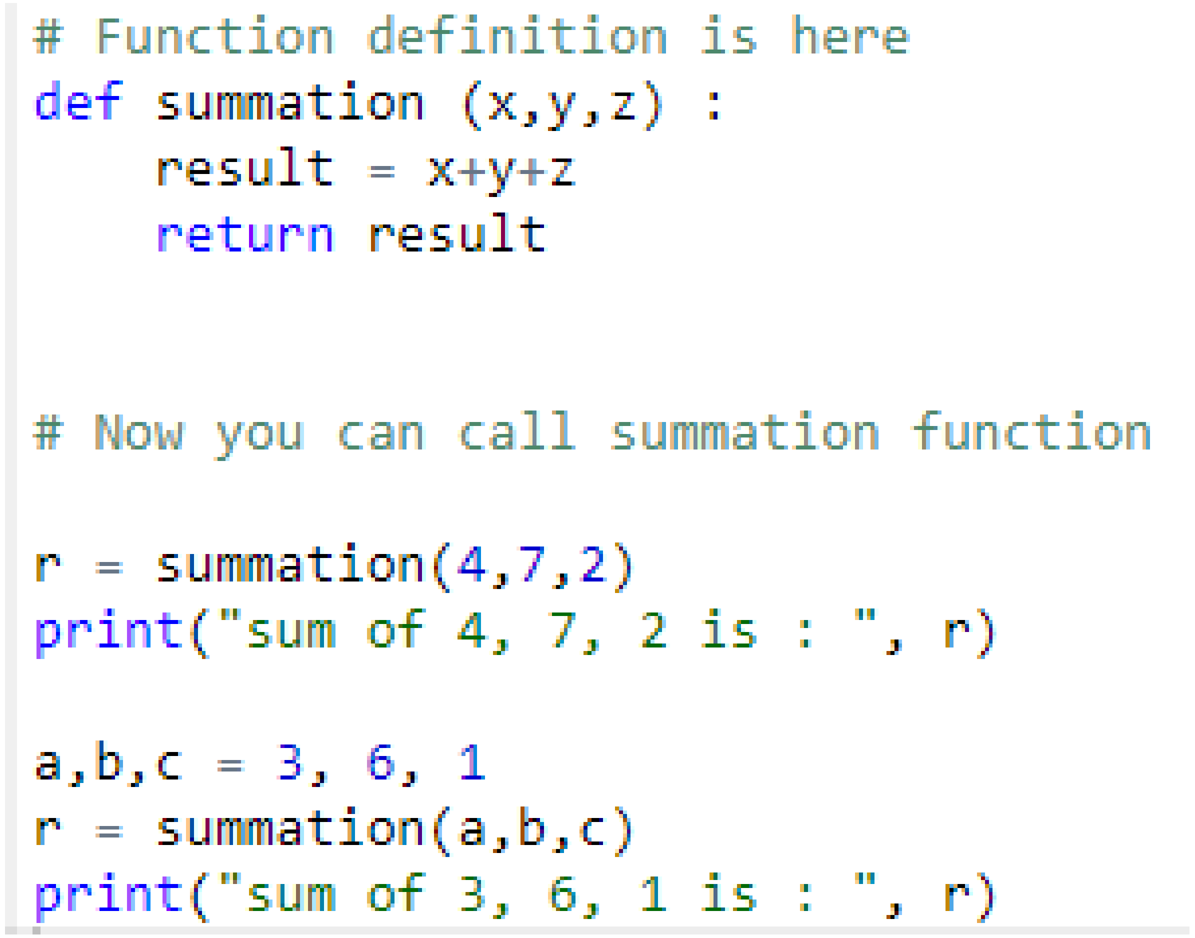
- You can define function to provide the required functionality. Here are simple rules to define a function in Python.
- Function blocks begin with the keyword def followed by the function name and parentheses (( )).
- Any input parameters or argument should be placed within these parentheses you can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement — the documentation string of the function or doctoring.
- The code blocks within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no argument is the same as return None.

Calling a Function

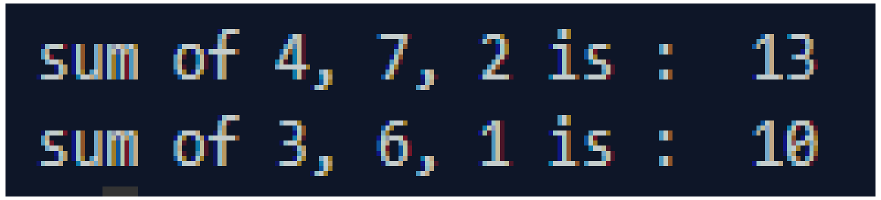
- Defining a function gives it a name, specifies the parameters that are to be include in the function and structure the blocks of code.
- Once the basics structure of a function is finalized, you can execute it by calling it form another function or directly from the Python prompt.
- Once we have defined a function, we can call it from another function, program or even the Python prompt. To call a function we simply type the function name with appropriate parameters.



Calling a Function



```
# Function definition is here
def summation (x,y,z) :
    result = x+y+z
    return result


# Now you can call summation function

r = summation(4,7,2)
print("sum of 4, 7, 2 is : ", r)

a,b,c = 3, 6, 1
r = summation(a,b,c)
print("sum of 3, 6, 1 is : ", r)
```

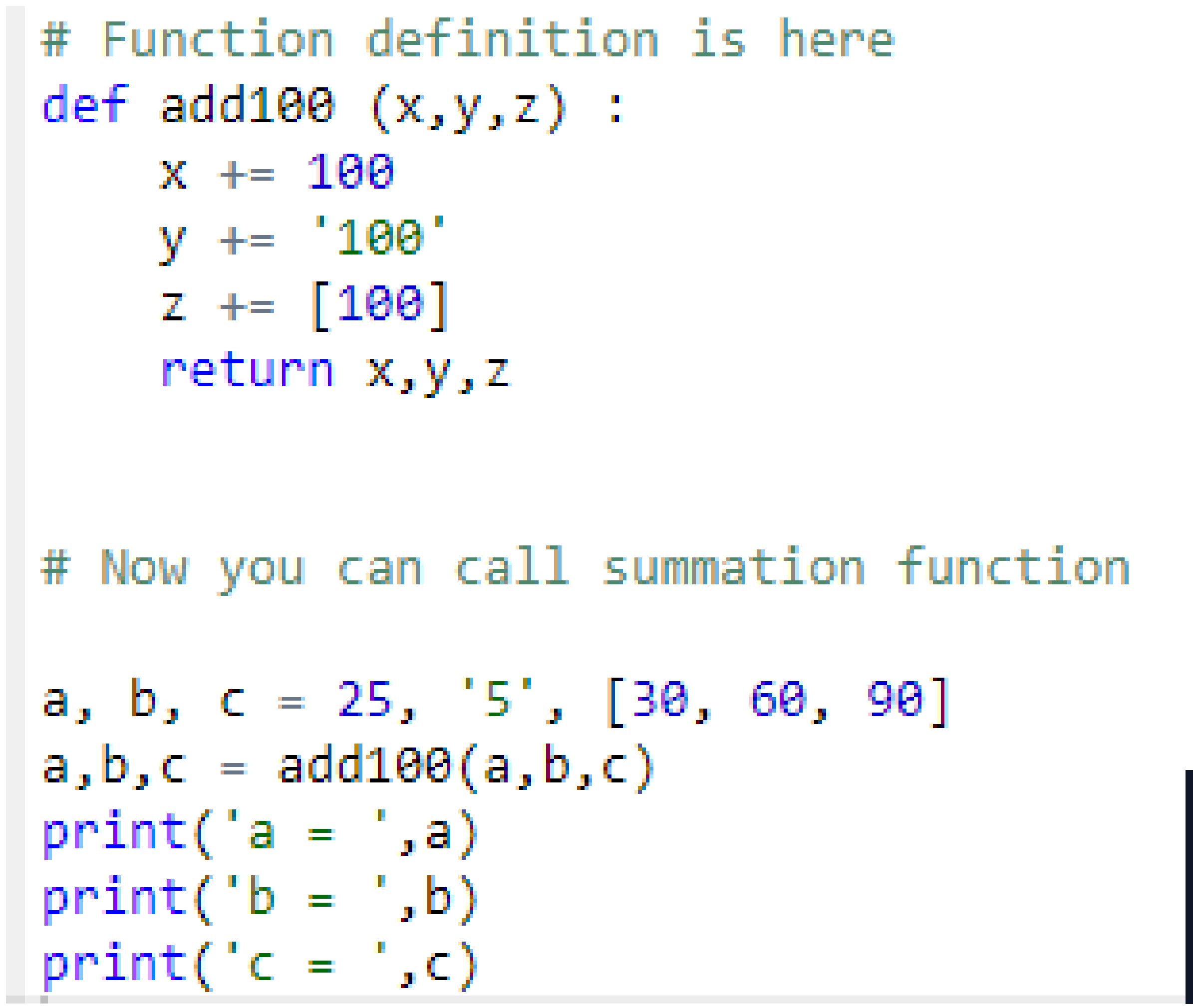


```
sum of 4, 7, 2 is :  13
sum of 3, 6, 1 is :  10
```

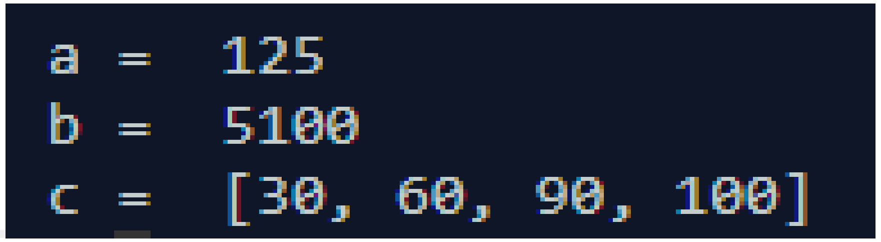
return Statement

- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.
- This statement can contain expression which gets evaluated and the value is returned. If there is no expression in the statement or the return statement itself is not present inside a function, then the function will return the None object.
- Python support multiple return objects with different data types.
- The return statement causes your function to exit and hand back a value to its caller. The point of functions in general is to take in inputs and return something. The return statement is used when a function is ready to return a value to its caller.

return Statement



```
# Function definition is here
def add100 (x,y,z) :
    x += 100
    y += '100'
    z += [100]
    return x,y,z


# Now you can call summation function

a, b, c = 25, '5', [30, 60, 90]
a,b,c = add100(a,b,c)
print('a = ',a)
print('b = ',b)
print('c = ',c)
```

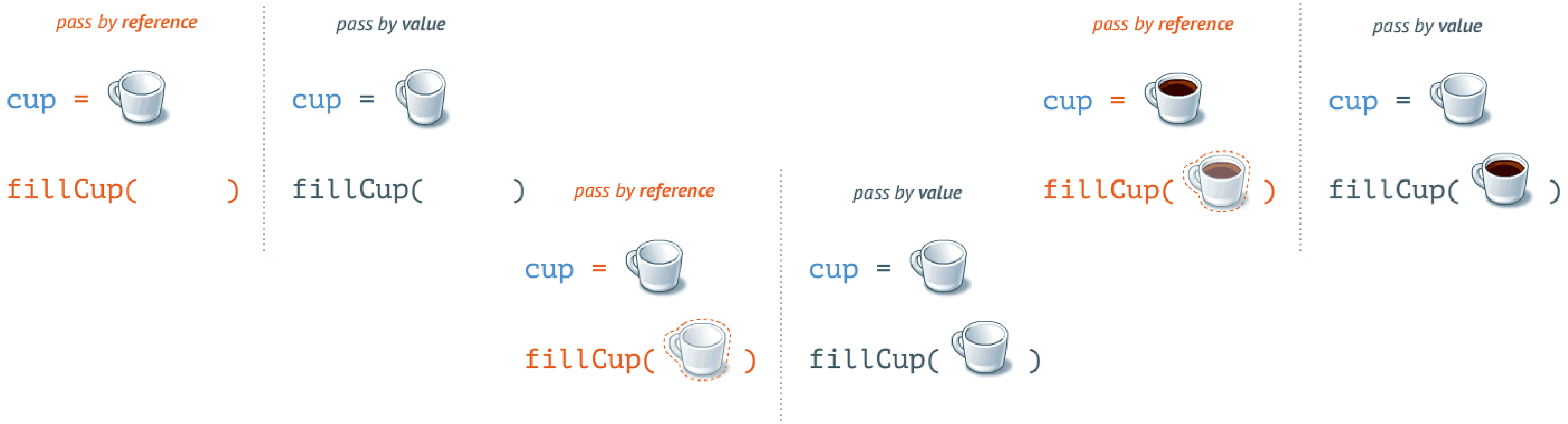


```
a =  125
b =  5100
c =  [30, 60, 90, 100]
```

© Prepared by: Mohamed Ayman

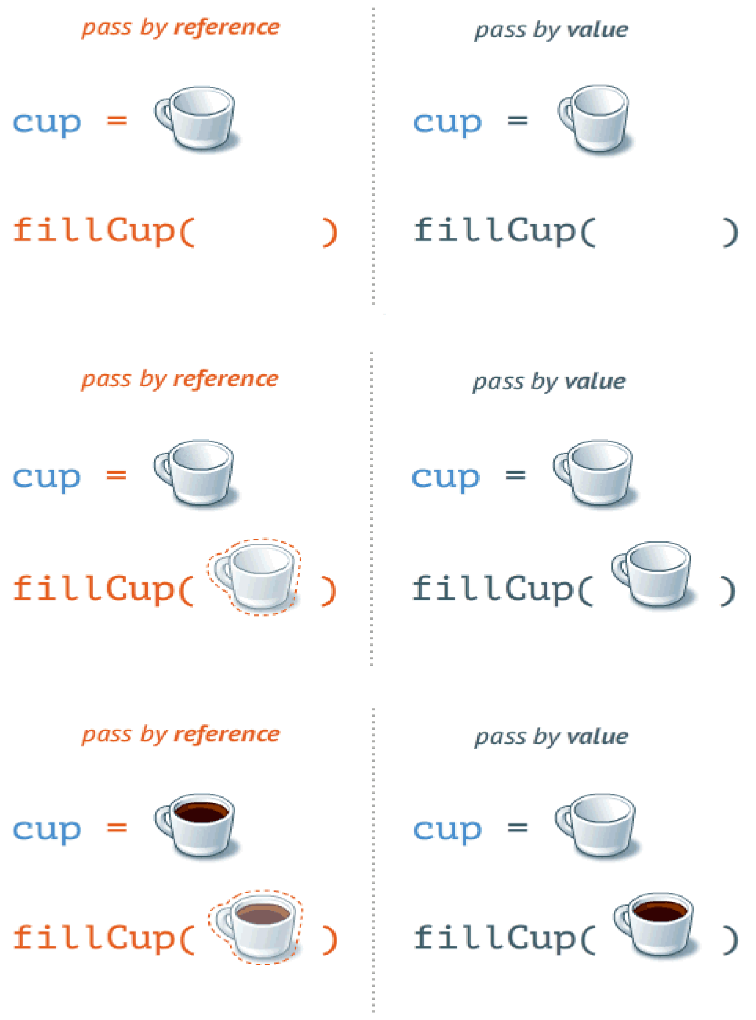
Passing by Reference & Value

- In call-by-value, the argument expression is evaluated, and the result of this evaluation is bound to the corresponding variable in the function. So, if the expression is a variable, a local copy of its value will be used, i.e. the variable in the caller's scope will be unchanged when the function returns.



Passing by Reference & Value

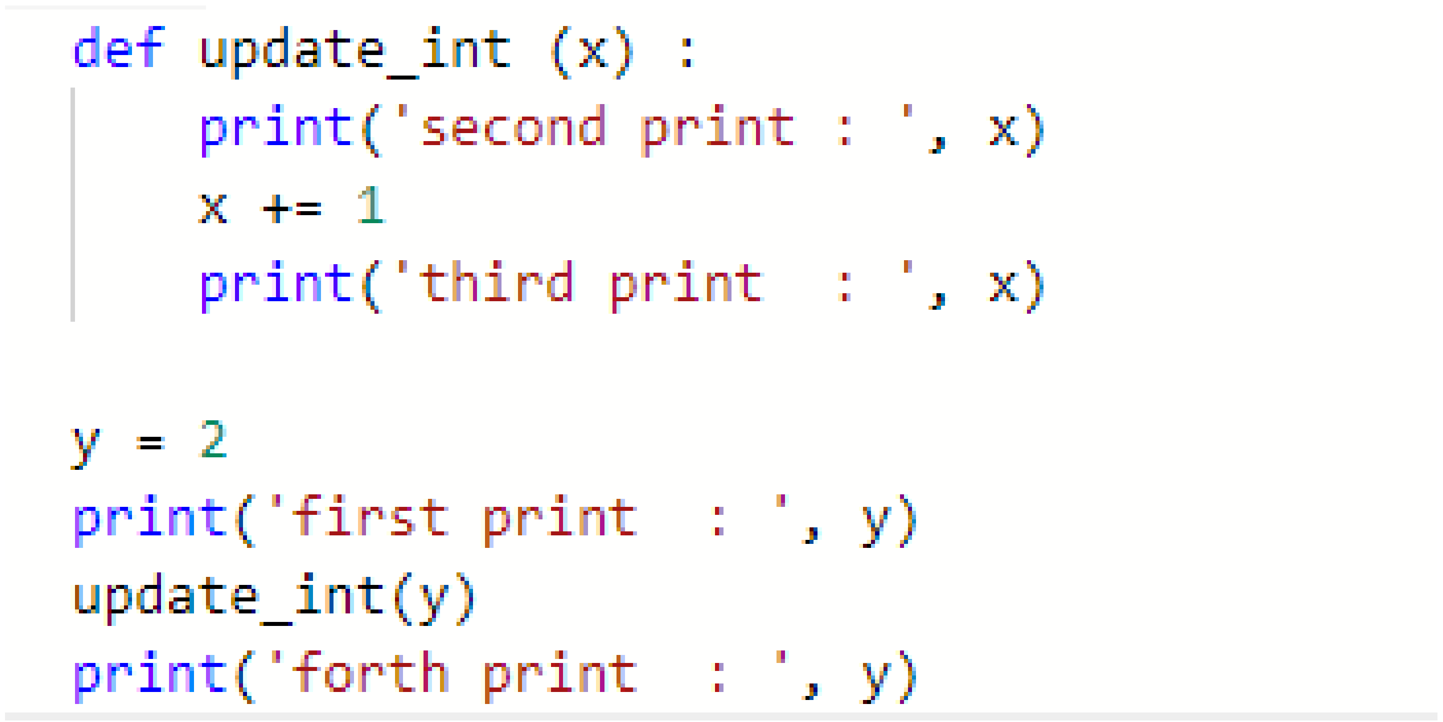
- In call-by-reference evaluation, which is also known as pass-by-reference, a function gets an implicit reference to the argument, rather than a copy of its value. As a consequence, the function can modify the argument, i.e. the value of the variable in the caller's scope can be changed.
- The advantage of call-by-reference consists in the advantage of greater time- and space-efficiency, because arguments do not need to be copied.
- On the other hand this harbours the disadvantage that variables can be "accidentally" changed in a function call. So special care has to be taken to "protect" the values, which shouldn't be changed.



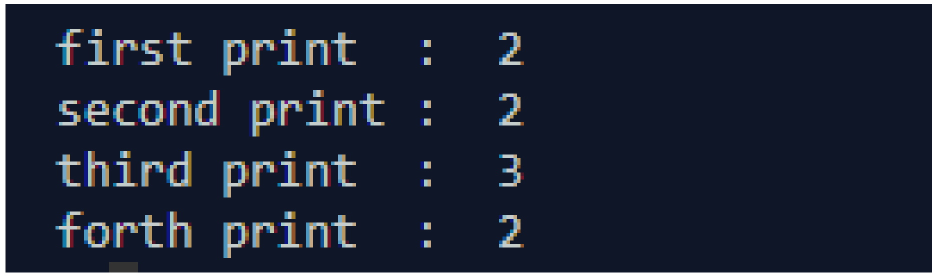
Passing by Reference & Value

- All parameters (arguments) in the Python language are passed by reference.
- Python uses a mechanism, which is known as "Call-by-Object", sometimes also called "Call by Object Reference" or "Call by Sharing".
- If you pass immutable arguments like integers, strings or tuples to a function, the passing acts like call-by-value. The object reference is passed to the function parameters.
- They can't be changed within the function, because they can't be changed at all, i.e. they are immutable. It's different, if we pass mutable arguments. They are also passed by object reference, but they can be changed in place in the function. If we pass a list to a function, we have to consider two cases: Elements of a list can be changed in place, i.e. the list will be changed even in the caller's scope. If a new list is assigned to the name, the old list will not be affected, i.e. the list in the caller's scope will remain untouched.

Passing by Reference & Value

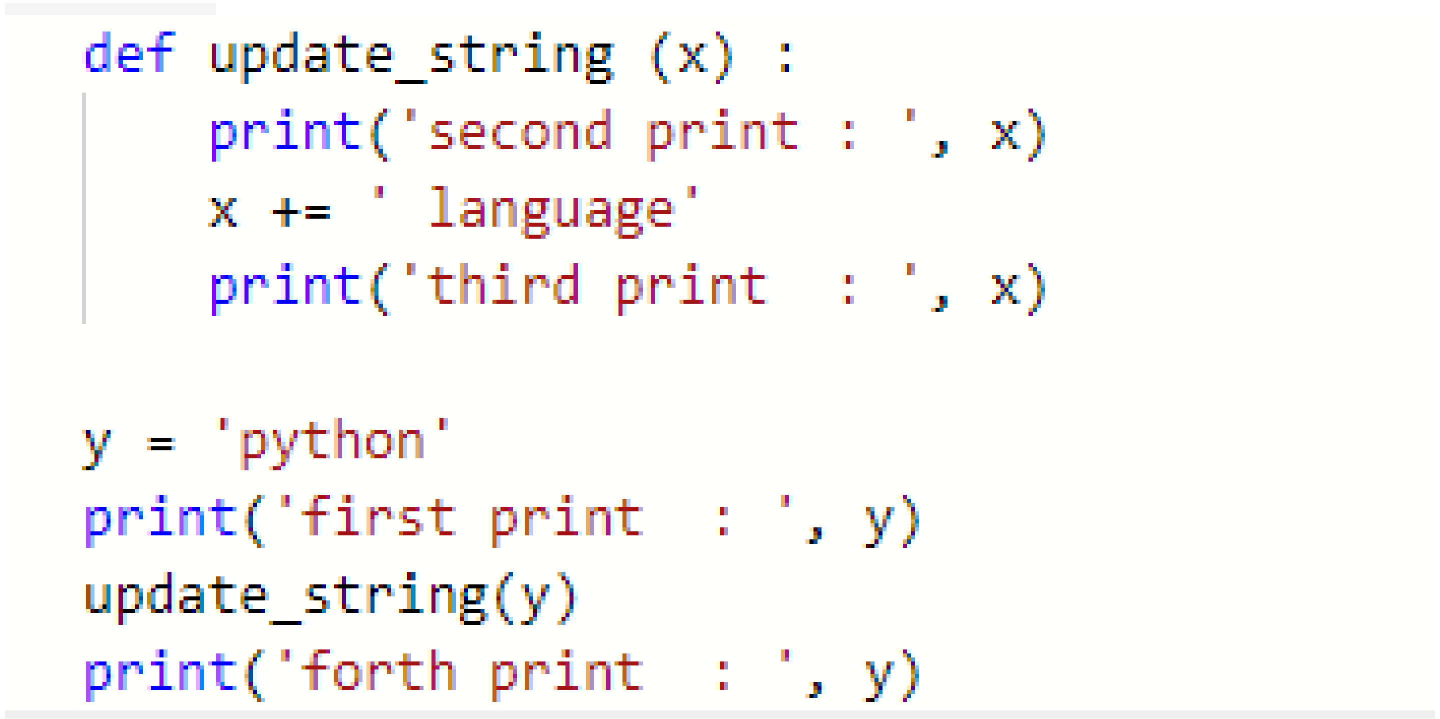


```python
def update_int (x) :
    print('second print : ', x)
    x += 1
    print('third print  : ', x)

y = 2
print('first print  : ', y)
update_int(y)
print('forth print  : ', y)
```

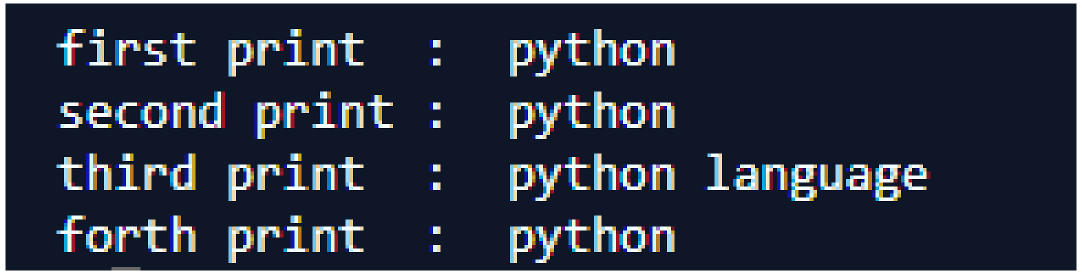


```
first print  :  2
second print :  2
third print  :  3
forth print  :  2
```

Passing by Reference & Value

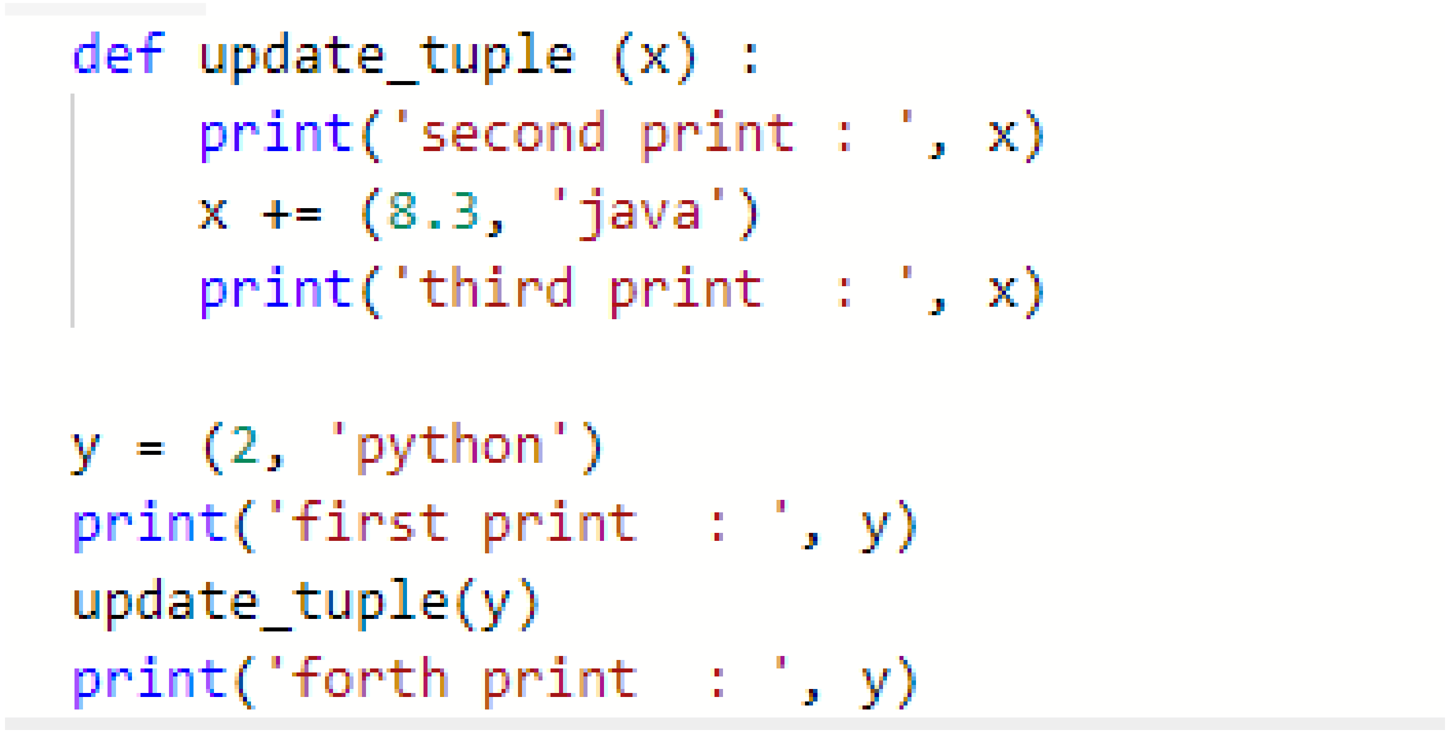


```python
def update_string (x) :
    print('second print : ', x)
    x += ' language'
    print('third print  : ', x)

y = 'python'
print('first print  : ', y)
update_string(y)
print('forth print  : ', y)
```

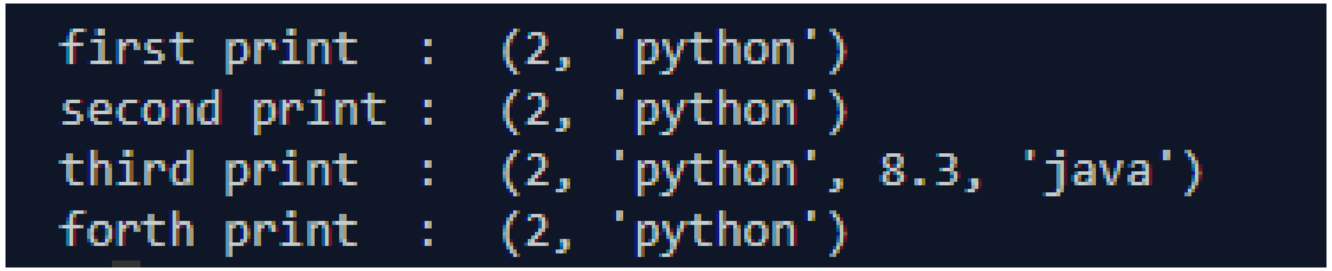


```
first print  :  python
second print :  python
third print  :  python language
forth print  :  python
```

Passing by Reference & Value

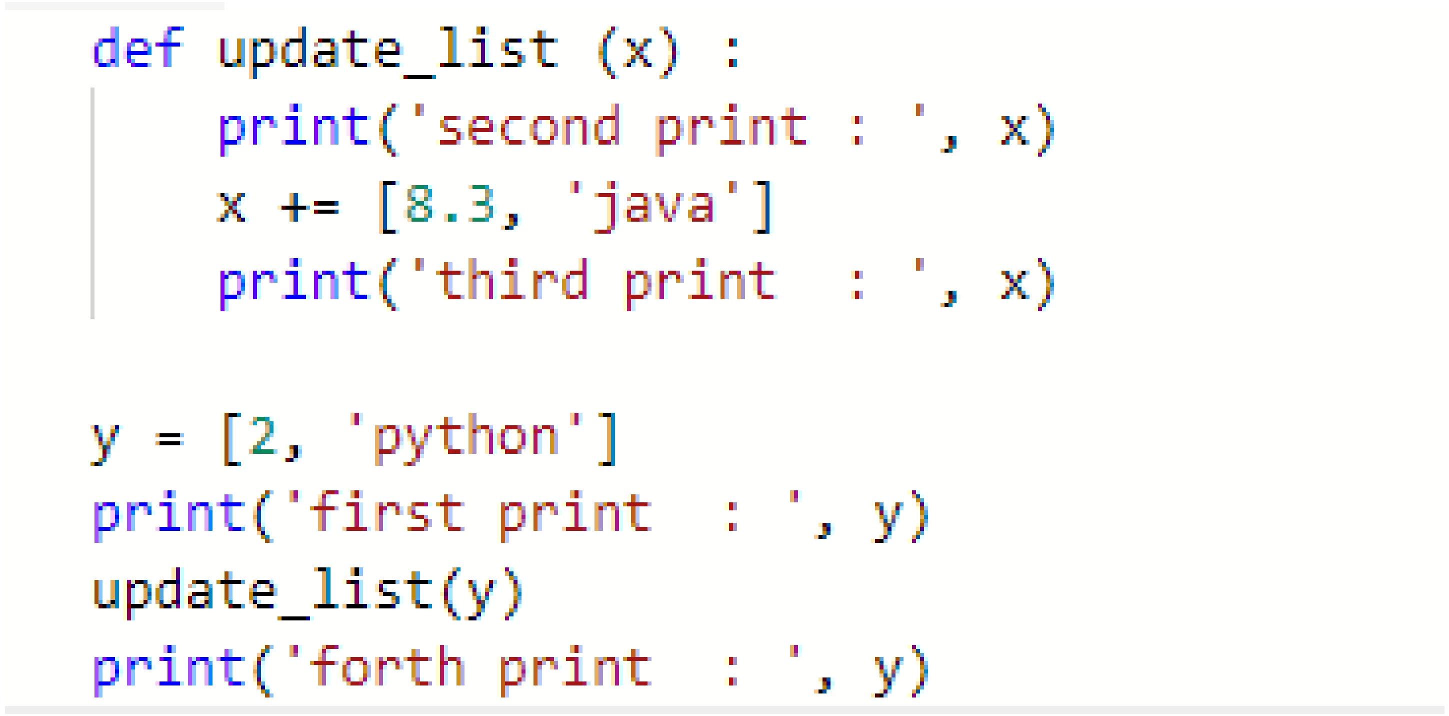


```python
def update_tuple (x) :
    print('second print : ', x)
    x += (8.3, 'java')
    print('third print  : ', x)

y = (2, 'python')
print('first print  : ', y)
update_tuple(y)
print('forth print  : ', y)
```

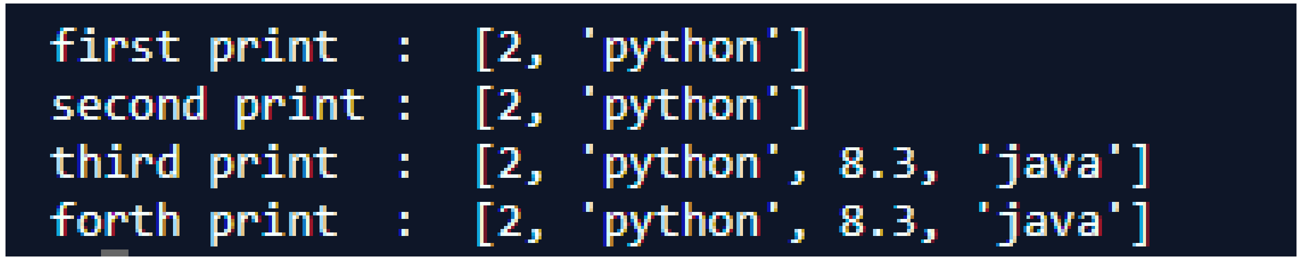


```
first print  :  (2, 'python')
second print :  (2, 'python')
third print  :  (2, 'python', 8.3, 'java')
forth print  :  (2, 'python')
```

Passing by Reference & Value

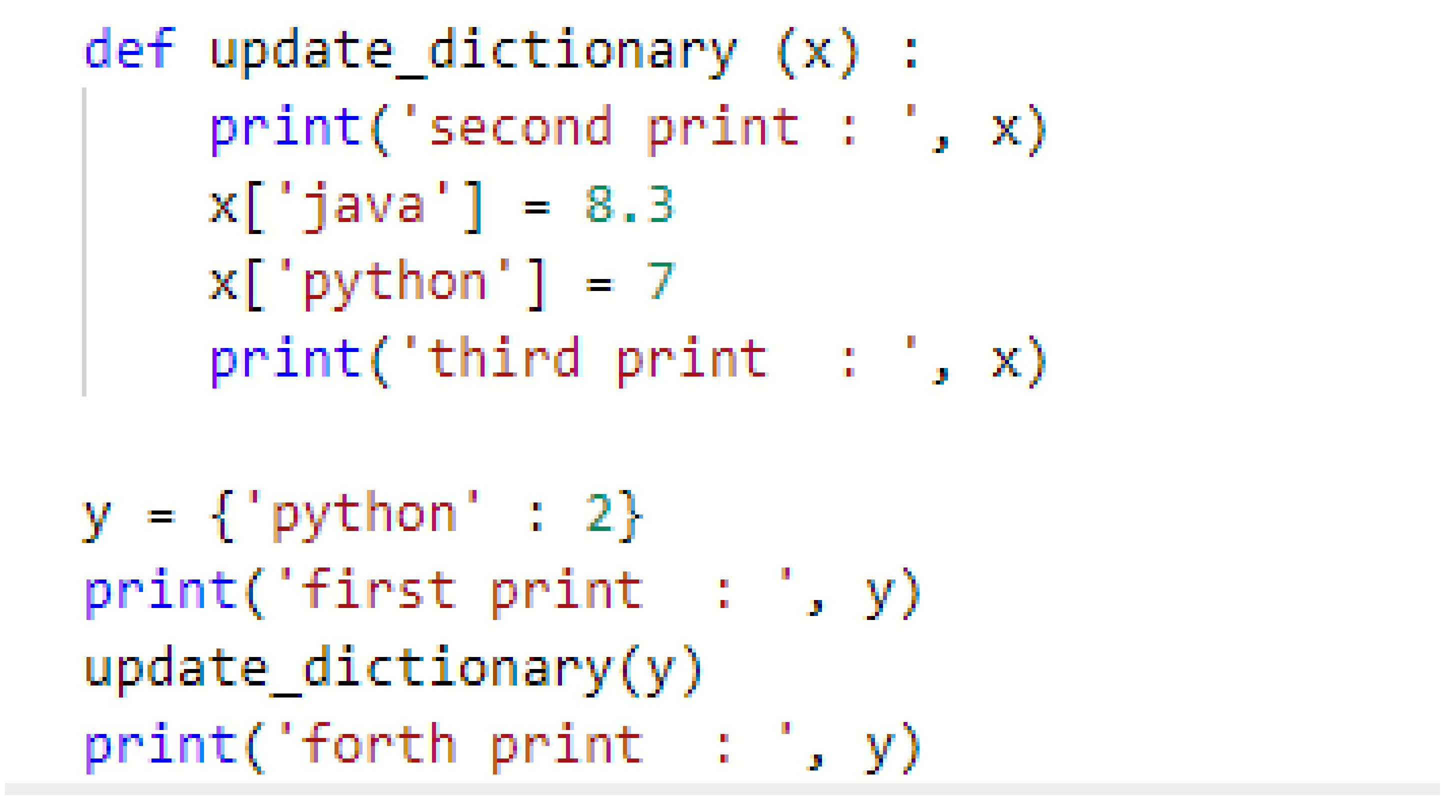


```python
def update_list (x) :
    print('second print : ', x)
    x += [8.3, 'java']
    print('third print  : ', x)

y = [2, 'python']
print('first print  : ', y)
update_list(y)
print('forth print  : ', y)
```

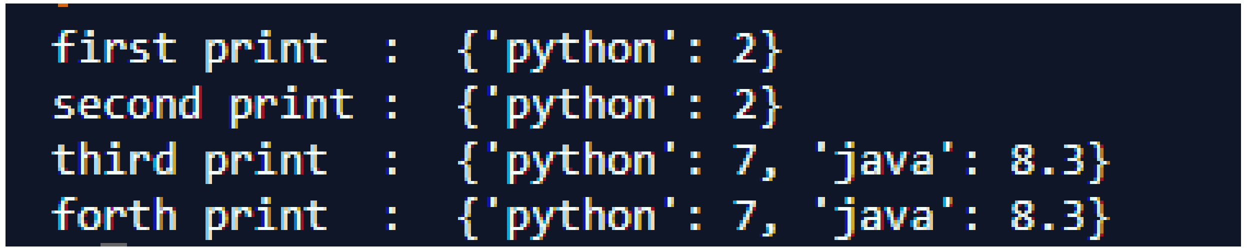


```
first print  :  [2, 'python']
second print :  [2, 'python']
third print  :  [2, 'python', 8.3, 'java']
forth print  :  [2, 'python', 8.3, 'java']
```

Passing by Reference & Value

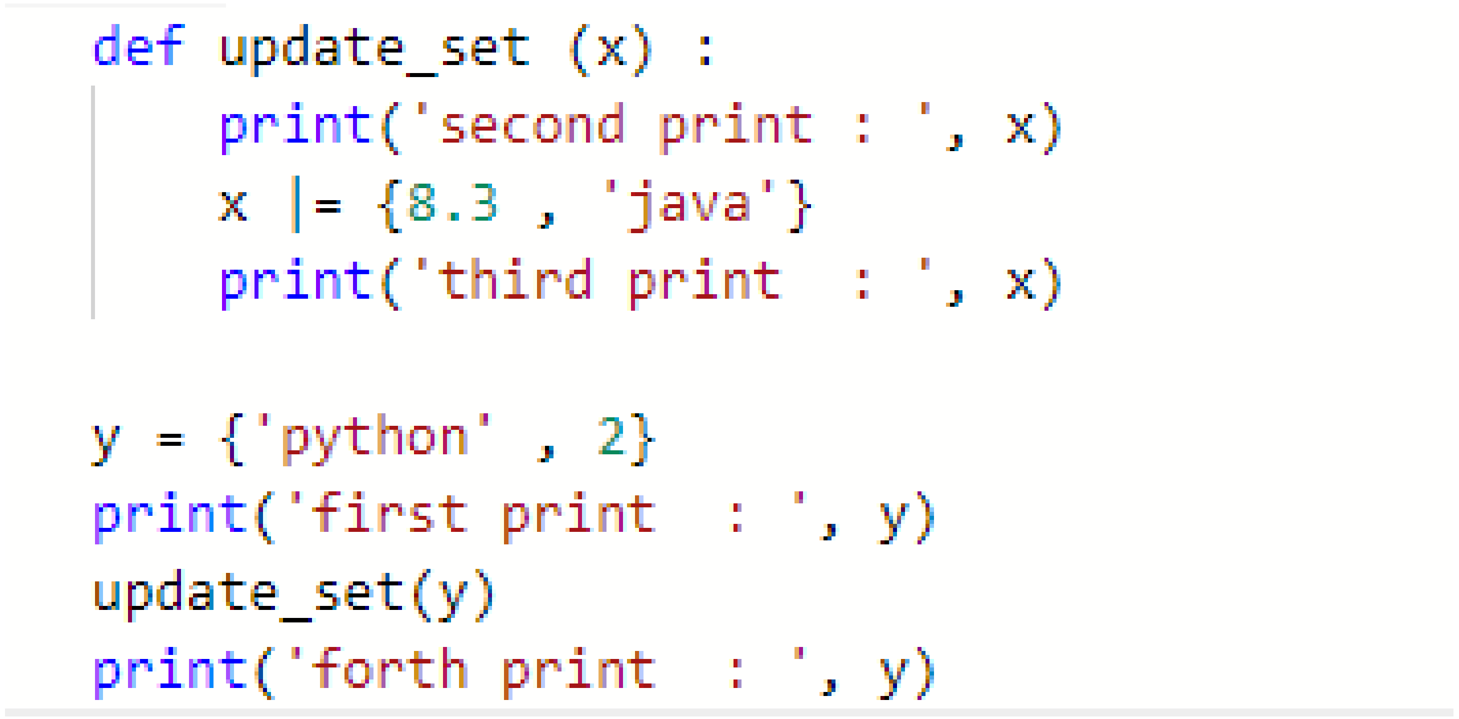


```python
def update_dictionary (x) :
    print('second print : ', x)
    x['java'] = 8.3
    x['python'] = 7
    print('third print  : ', x)

y = {'python' : 2}
print('first print  : ', y)
update_dictionary(y)
print('forth print  : ', y)
```

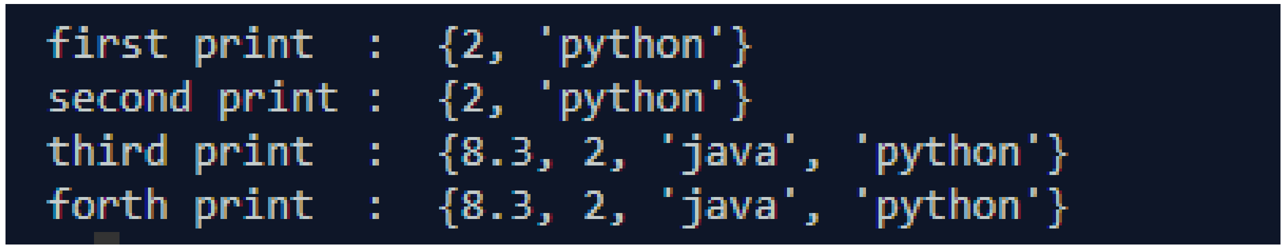


```
first print  :  {'python': 2}
second print :  {'python': 2}
third print  :  {'python': 7, 'java': 8.3}
forth print  :  {'python': 7, 'java': 8.3}
```

Passing by Reference & Value



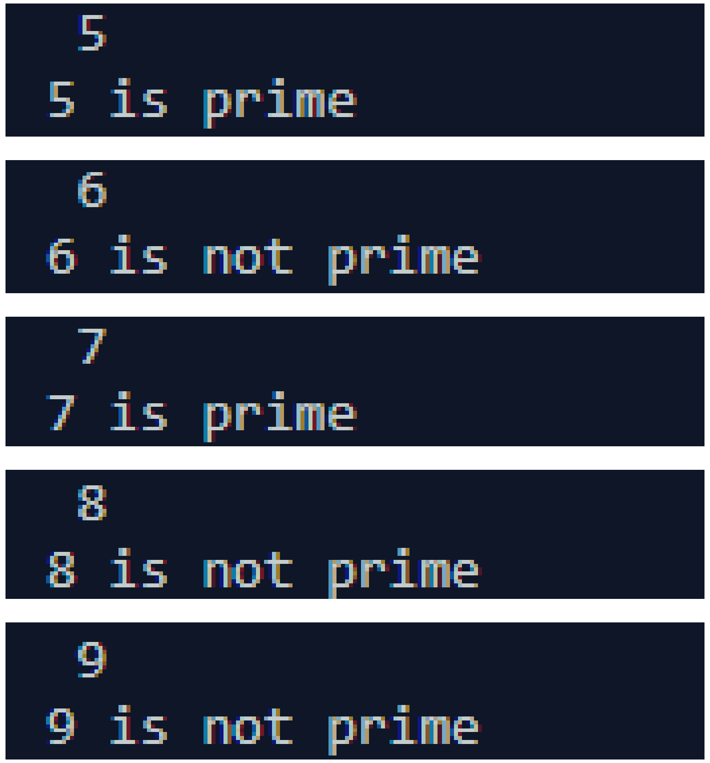
```
def update_set (x) :
    print('second print : ', x)
    x |= {8.3 , 'java'}
    print('third print  : ', x)

y = {'python' , 2}
print('first print  : ', y)
update_set(y)
print('forth print  : ', y)
```



```
first print  :  {2, 'python'}
second print :  {2, 'python'}
third print  :  {8.3, 2, 'java', 'python'}
forth print  :  {8.3, 2, 'java', 'python'}
```

Problem 1
Check Prime

- Take as an input n and implement function which determine this number prime or not such that n >= 1 and integer
- Function Name: isPrime
- Parameters: n
- Return: True if number prime otherwise False
- Test Cases:



```
 5
5 is prime
```

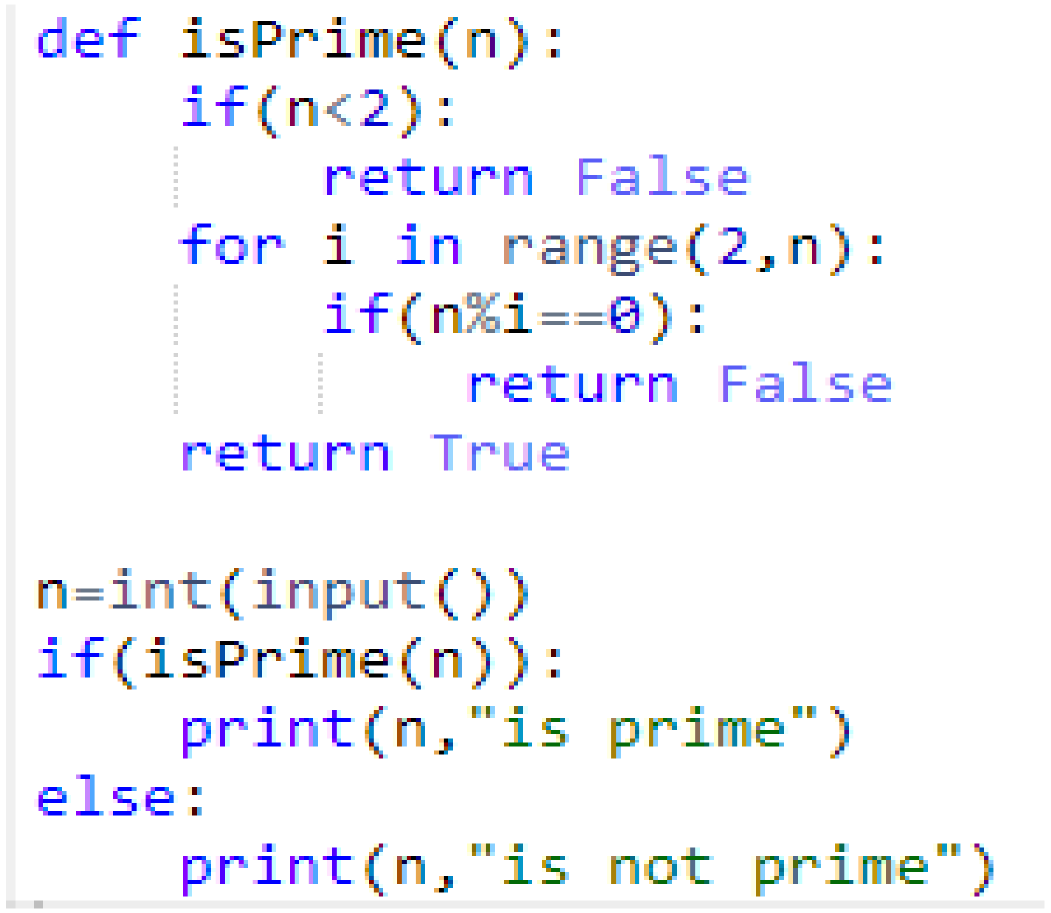
```
 6
6 is not prime
```

```
 7
7 is prime
```

```
 8
8 is not prime
```

```
 9
9 is not prime
```

Problem 1 Solution
Check Prime

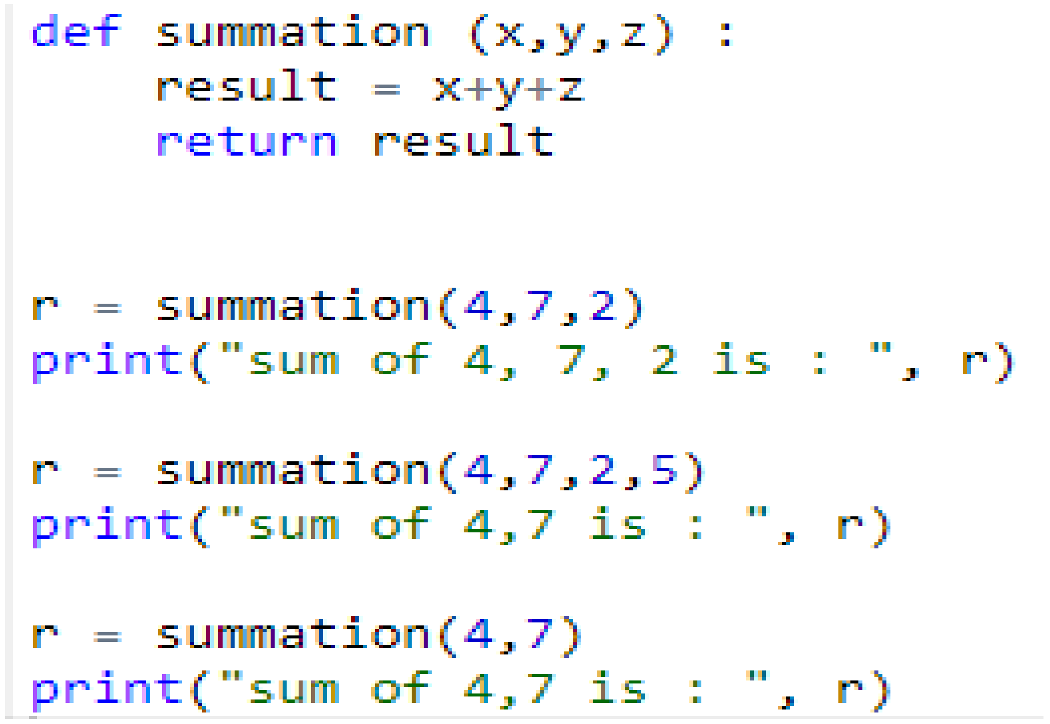


```python
def isPrime(n):
    if(n<2):
        return False
    for i in range(2,n):
        if(n%i==0):
            return False
    return True

n=int(input())
if(isPrime(n)):
    print(n,"is prime")
else:
    print(n,"is not prime")
```

Function Arguments

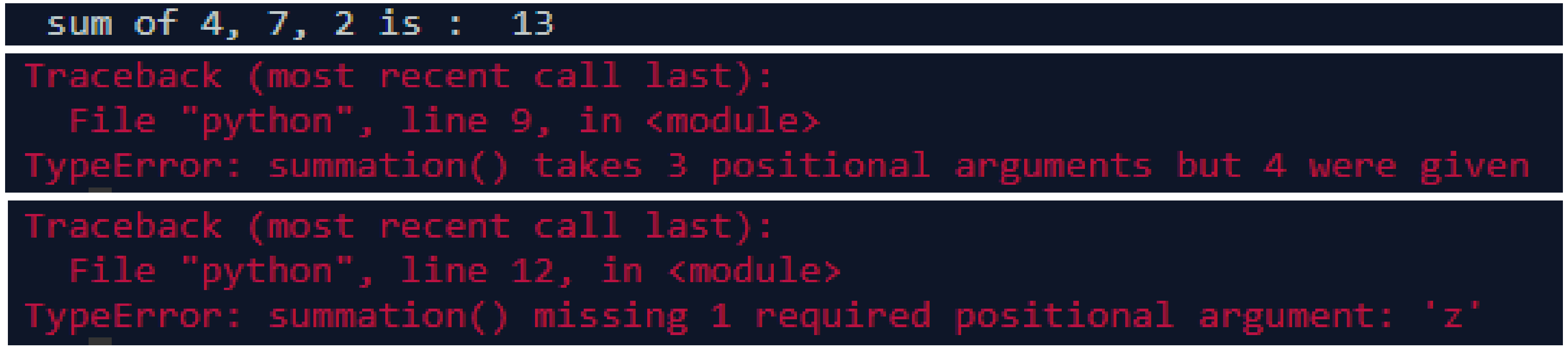
- You can call a function by using the following types of formal arguments
  - Required argument
  - Keyword argument
  - Default argument
  - Variable-length argument

Required Arguments



```
def summation (x,y,z) :
    result = x+y+z
    return result


r = summation(4,7,2)
print("sum of 4, 7, 2 is : ", r)

r = summation(4,7,2,5)
print("sum of 4,7 is : ", r)

r = summation(4,7)
print("sum of 4,7 is : ", r)
```

- Required argument are the argument passed to a function in correct positional order. Here the number of argument in the function call should match exactly with the function definition.



```
 sum of 4, 7, 2 is :  13
```

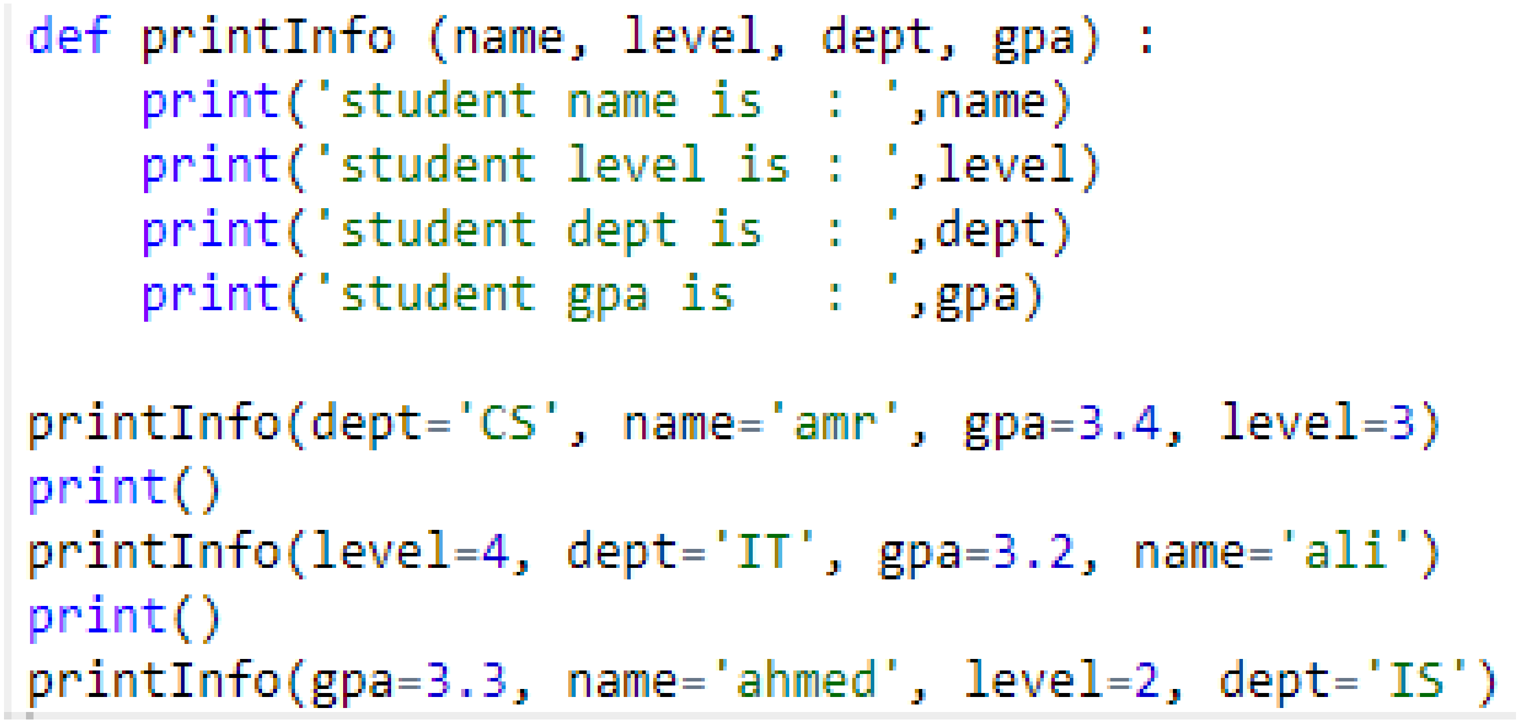
```
Traceback (most recent call last):
  File "python", line 9, in <module>
TypeError: summation() takes 3 positional arguments but 4 were given
```

```
Traceback (most recent call last):
  File "python", line 12, in <module>
TypeError: summation() missing 1 required positional argument: 'z'
```

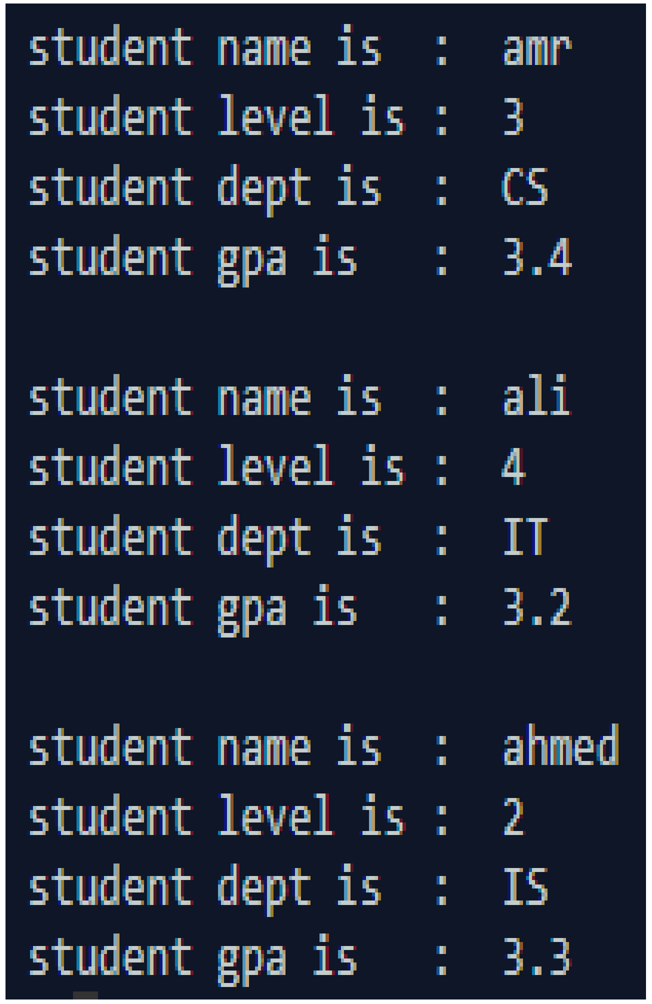
Keyword Arguments

- Keyword argument are related to the function calls. When you use keyword arguments in a function call, the caller identifier the arguments by the parameter name. This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keyword provided to match the value with parameters.
- When calling functions in Python, you will often have to choose between using keyword arguments or positional arguments. Keyword arguments can often be used to make function calls more explicit. You will likely see keyword arguments quite a bit in Python.
- Python has a number of functions that take an unlimited number of positional arguments. These functions sometimes have arguments that can be provided to customize their functionality. Those arguments must be provided as named arguments to distinguish them from the unlimited positional arguments.

Keyword Arguments

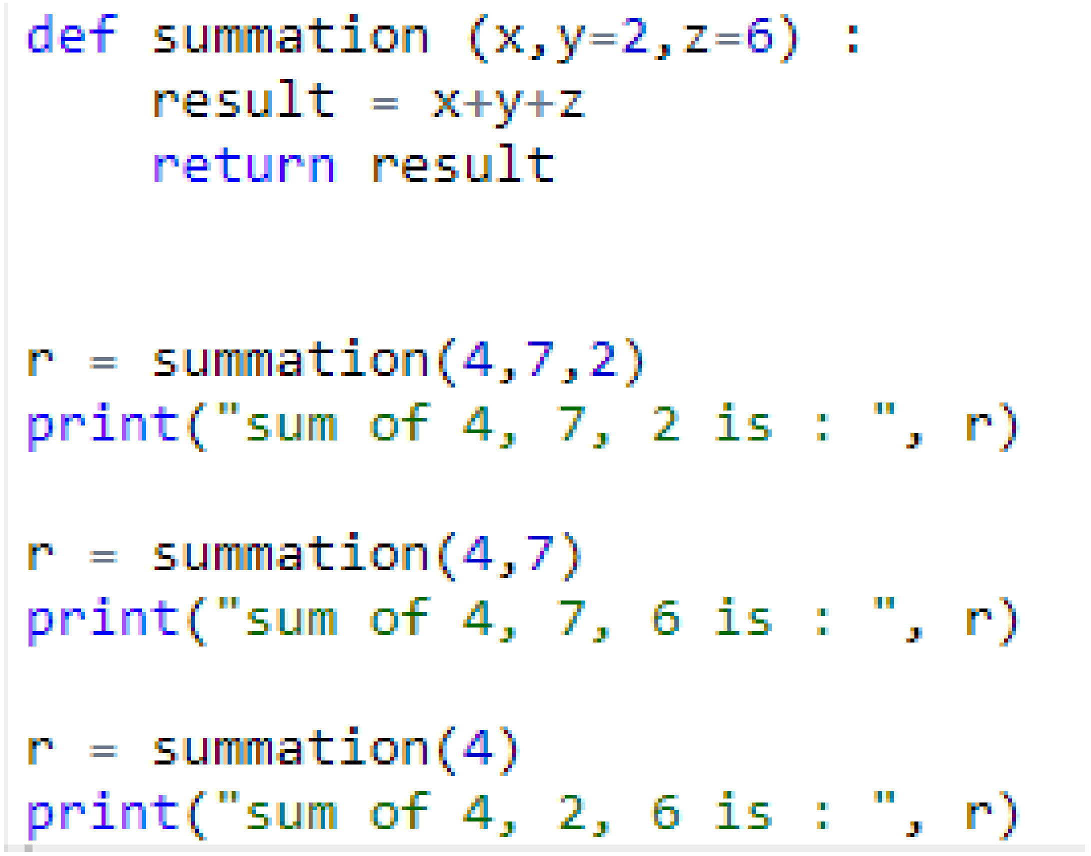


```python
def printInfo (name, level, dept, gpa) :
    print('student name is  : ',name)
    print('student level is : ',level)
    print('student dept is  : ',dept)
    print('student gpa is   : ',gpa)

printInfo(dept='CS', name='amr', gpa=3.4, level=3)
print()
printInfo(level=4, dept='IT', gpa=3.2, name='ali')
print()
printInfo(gpa=3.3, name='ahmed', level=2, dept='IS')
```

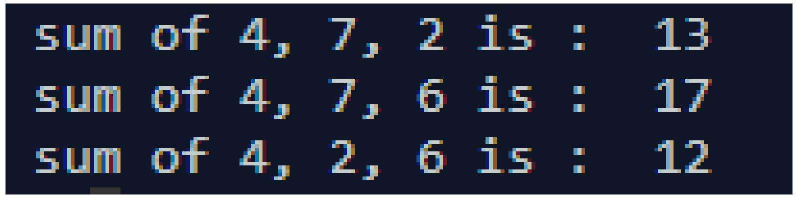


```
student name is  :  amr
student level is :  3
student dept is  :  CS
student gpa is   :  3.4

student name is  :  ali
student level is :  4
student dept is  :  IT
student gpa is   :  3.2

student name is  :  ahmed
student level is :  2
student dept is  :  IS
student gpa is   :  3.3
```

Default Arguments



```python
def summation (x,y=2,z=6) :
    result = x+y+z
    return result


r = summation(4,7,2)
print("sum of 4, 7, 2 is : ", r)

r = summation(4,7)
print("sum of 4, 7, 6 is : ", r)

r = summation(4)
print("sum of 4, 2, 6 is : ", r)
```



```
sum of 4, 7, 2 is :  13
sum of 4, 7, 6 is :  17
sum of 4, 2, 6 is :  12
```

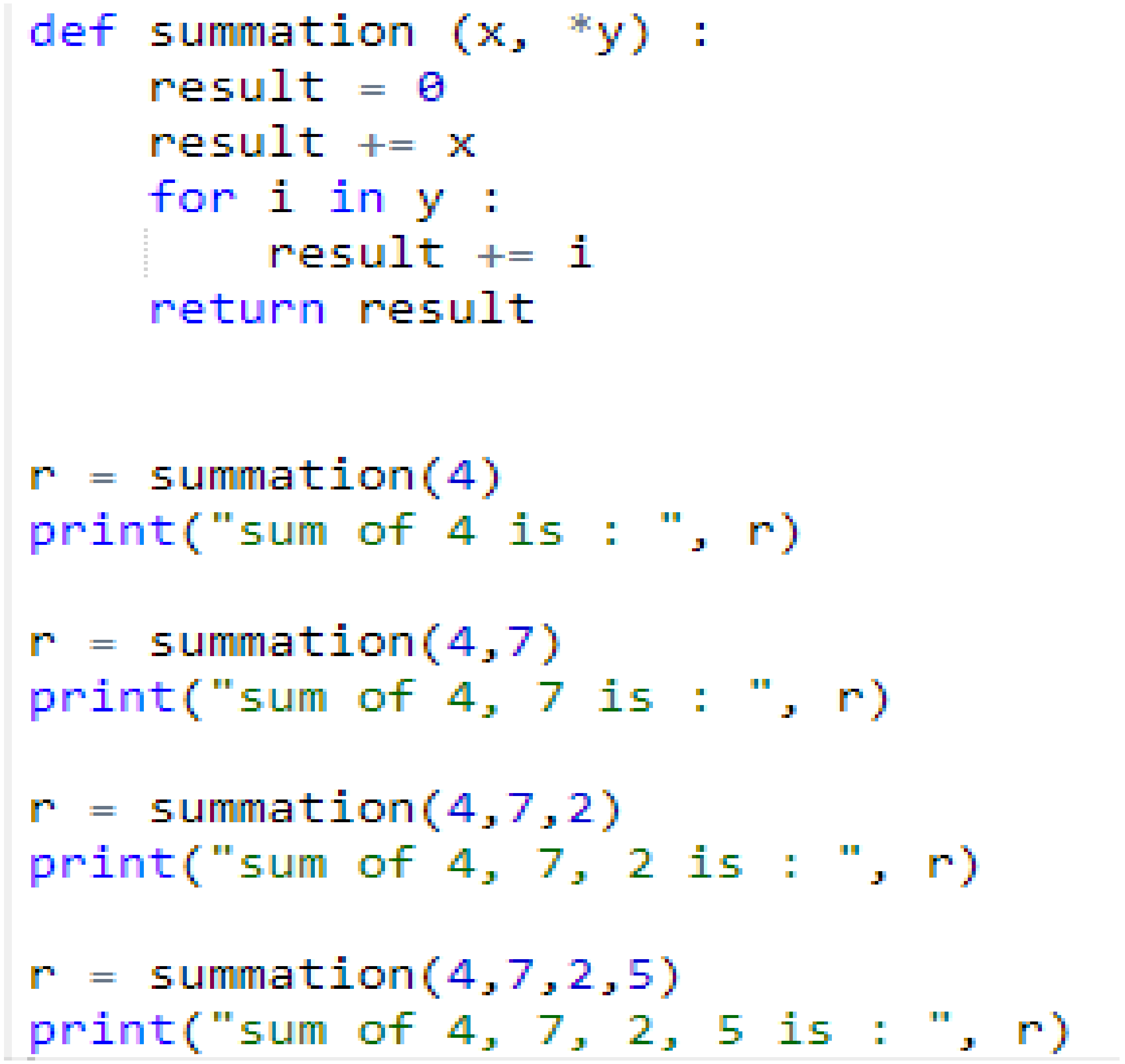
- A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument. The following example givens an idea on default arguments, it prints default age if it is not passed.

Variable-length Arguments

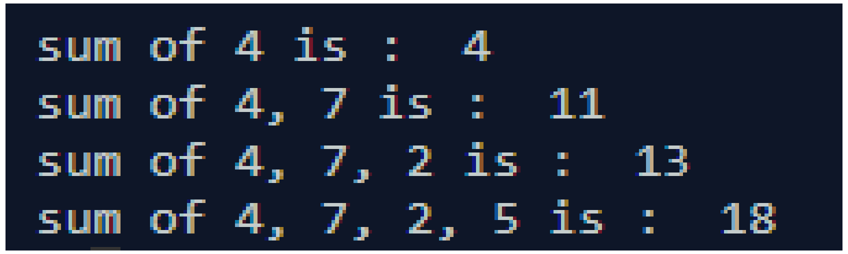
- You may need to process a function for more arguments than you specified while defining the function. These argument are called variable-length argument and are not named in the function definition, unlike required and default argument.
- An asterisk (*) is placed before the variable name that holds the values of all non keyword variable arguments. This tuple remains empty if no additional argument are specified during the function call

```
def functionname([formal_args,] *var_args_tuple ):
   "function_docstring"
   function_suite
   return [expression]
```

Variable-length Arguments



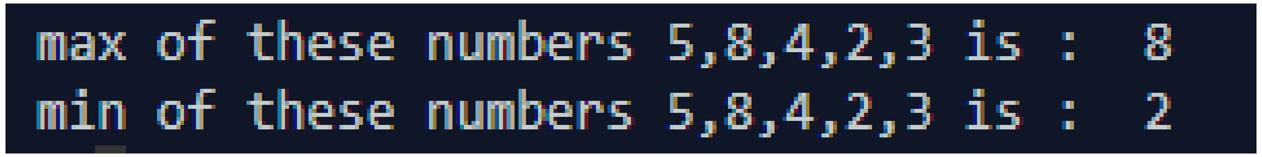
```python
def summation (x, *y) :
    result = 0
    result += x
    for i in y :
        result += i
    return result


r = summation(4)
print("sum of 4 is : ", r)

r = summation(4,7)
print("sum of 4, 7 is : ", r)

r = summation(4,7,2)
print("sum of 4, 7, 2 is : ", r)

r = summation(4,7,2,5)
print("sum of 4, 7, 2, 5 is : ", r)
```



```
sum of 4 is :  4
sum of 4, 7 is :  11
sum of 4, 7, 2 is :  13
sum of 4, 7, 2, 5 is :  18
```

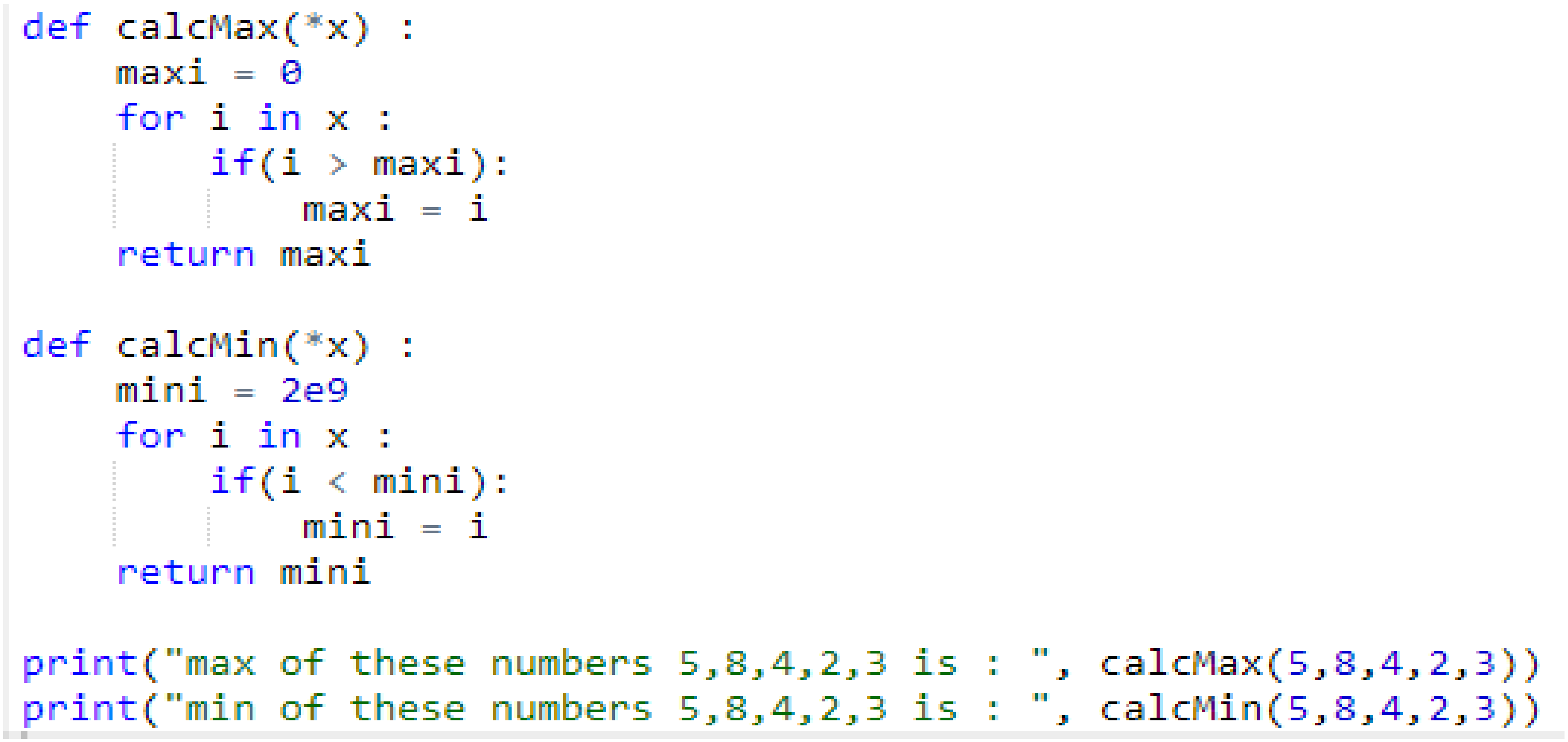
Problem 2
Get min & max

- Implement two functions which determine the maximum value and minimum value for set of number such that these numbers integer and non-negative
  - Function Name: calcMax
  - Parameters: all numbers which user enter them
  - Return: maximum value of them
  - Function Name: calcMin
  - Parameters: all numbers which user enter them
  - Return: maximum value of them
- Test Cases:



```
max of these numbers 5,8,4,2,3 is :  8
min of these numbers 5,8,4,2,3 is :  2
```

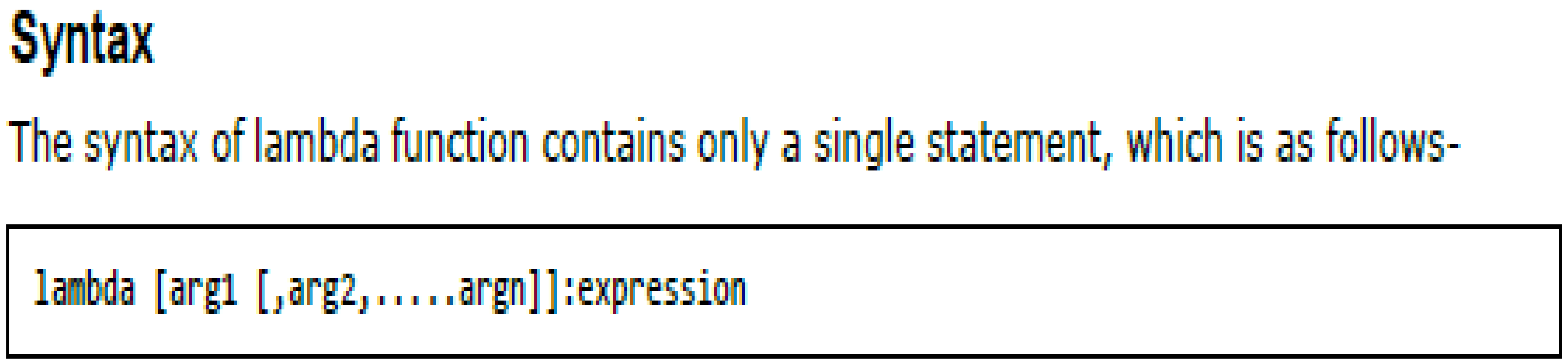
Problem 2 Solution
Get min & max



```python
def calcMax(*x) :
    maxi = 0
    for i in x :
        if(i > maxi):
            maxi = i
    return maxi

def calcMin(*x) :
    mini = 2e9
    for i in x :
        if(i < mini):
            mini = i
    return mini

print("max of these numbers 5,8,4,2,3 is : ", calcMax(5,8,4,2,3))
print("min of these numbers 5,8,4,2,3 is : ", calcMin(5,8,4,2,3))
```

The Anonymous Functions

- These function are called anonymous because they are not declared in the standard manner by using the def keyword. You can use the lambda keyword to create small anonymous functions.
- Lambda forms can take any number of arguments but return just one value in the form of an expression. They cannot contain commands or multiple expressions.
- An anonymous function cannot be a direct call to print because lambda requires an expression.
- Lambda functions have their own local namespace and cannot access variable other than those in their parameter list and those in the global namespace.

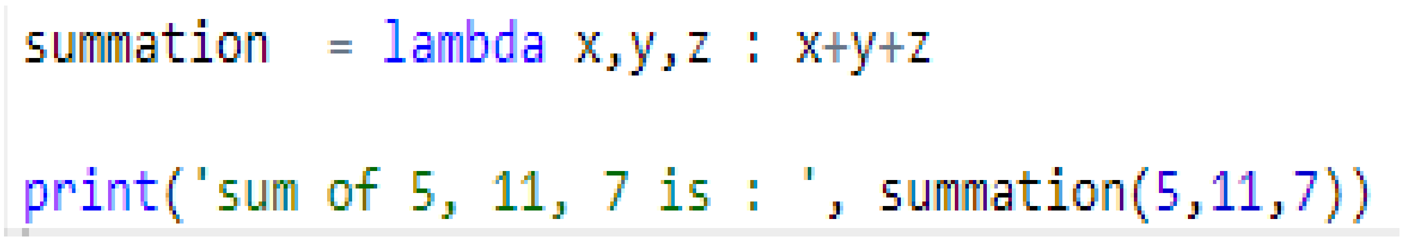


Syntax

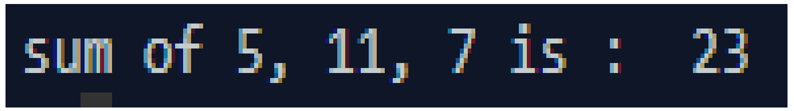
The syntax of lambda function contains only a single statement, which is as follows-

```
lambda [arg1 [,arg2,.....argn]]:expression
```

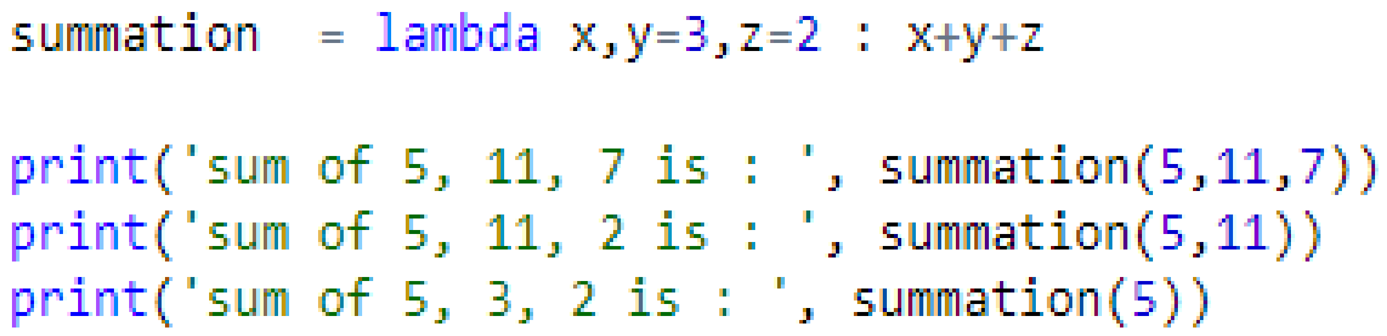
The Anonymous Functions



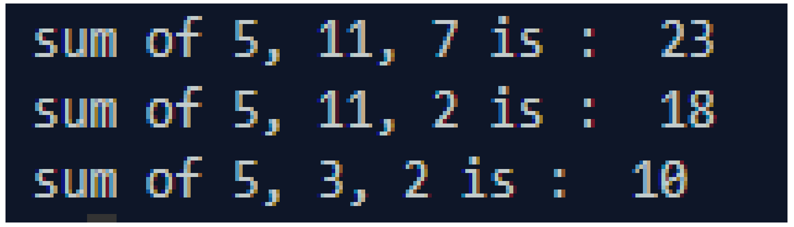
```python
summation  = lambda x,y,z : x+y+z

print('sum of 5, 11, 7 is : ', summation(5,11,7))
```



```
sum of 5, 11, 7 is :  23
```



```python
summation  = lambda x,y=3,z=2 : x+y+z

print('sum of 5, 11, 7 is : ', summation(5,11,7))
print('sum of 5, 11, 2 is : ', summation(5,11))
print('sum of 5, 3, 2 is : ', summation(5))
```

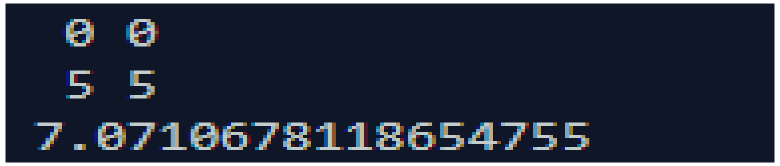


```
sum of 5, 11, 7 is :  23
sum of 5, 11, 2 is :  18
sum of 5, 3, 2 is :  10
```

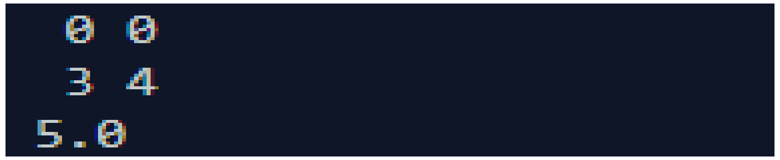
Problem 3
Distance

- Take as an input two points and Implement anonymous functions which determine the distance between two points such that coordination maybe float

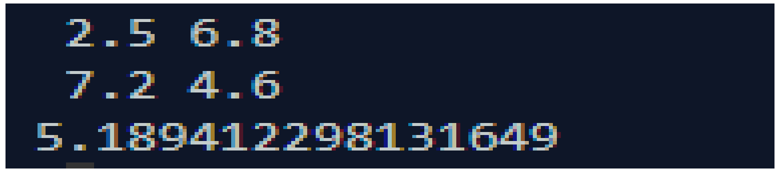
- Function Name: distance
- Parameters: x1, y1, x2, y2
- Return: float value represent distance
- Test Cases:



```
 0 0
 5 5
7.0710678118654755
```

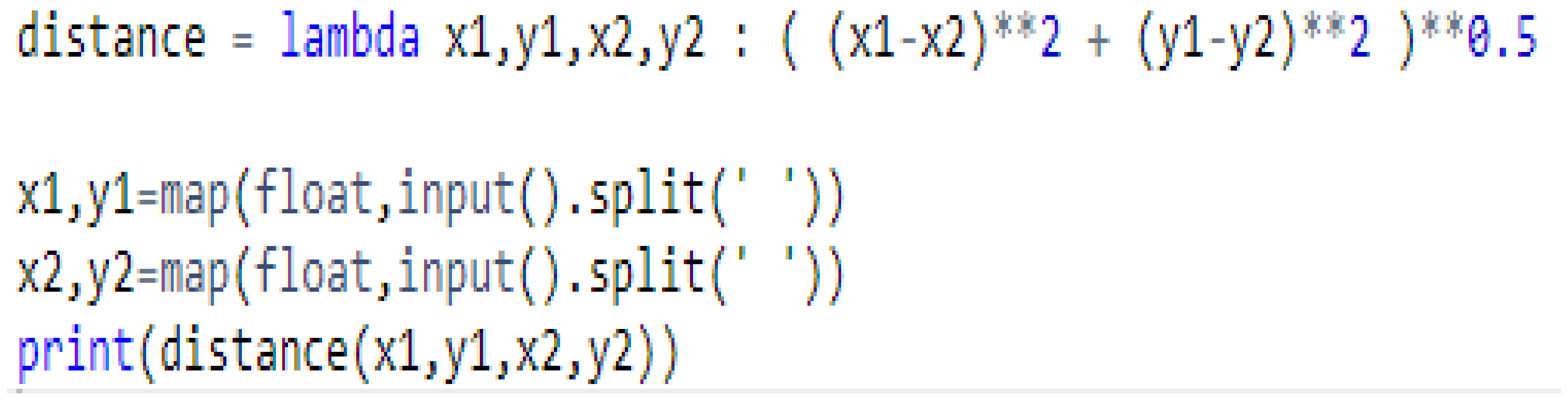


```
 0 0
 3 4
5.0
```



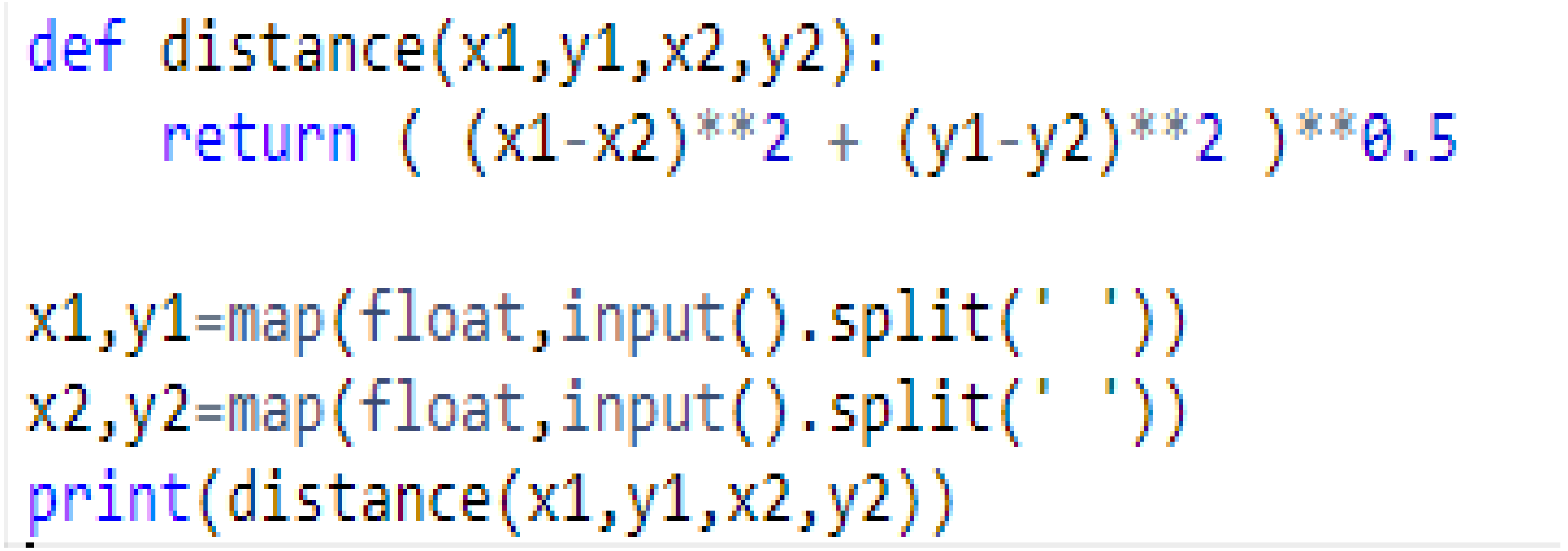
```
 2.5 6.8
 7.2 4.6
5.189412298131649
```

Problem 3 Solution
Distance



```
distance = lambda x1,y1,x2,y2 : ( (x1-x2)**2 + (y1-y2)**2 )**0.5

x1,y1=map(float,input().split(' '))
x2,y2=map(float,input().split(' '))
print(distance(x1,y1,x2,y2))
```

Problem 3 Solution
Distance

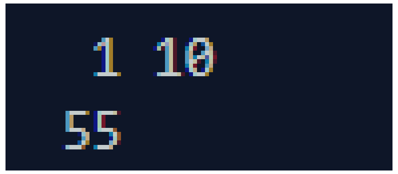


```python
def distance(x1,y1,x2,y2):
    return ( (x1-x2)**2 + (y1-y2)**2 )**0.5

x1,y1=map(float,input().split(' '))
x2,y2=map(float,input().split(' '))
print(distance(x1,y1,x2,y2))
```

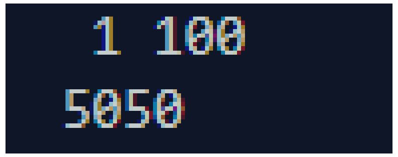
Problem 4
Summation

- Take as an input two number x , y and implement anonymous function which calculate summation form 1 to n to use it in our problem, such that x<=y and x, y integers and x, y >=1

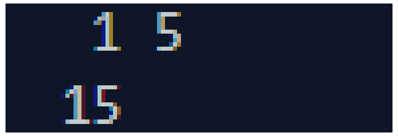
- Function Name: summation
- Parameters: n
- Return: int value represent summation from 1 to n
- Test Cases:



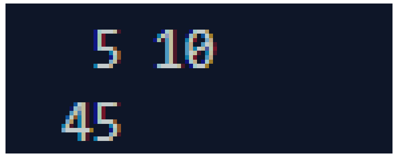
```
 1 10
55
```



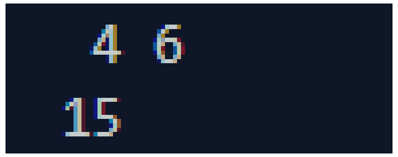
```
 1 100
5050
```



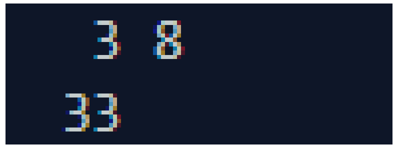
```
 1 5
15
```



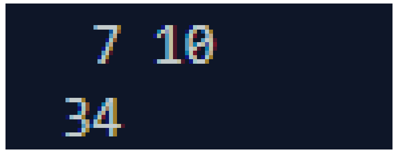
```
 5 10
45
```



```
 4 6
15
```

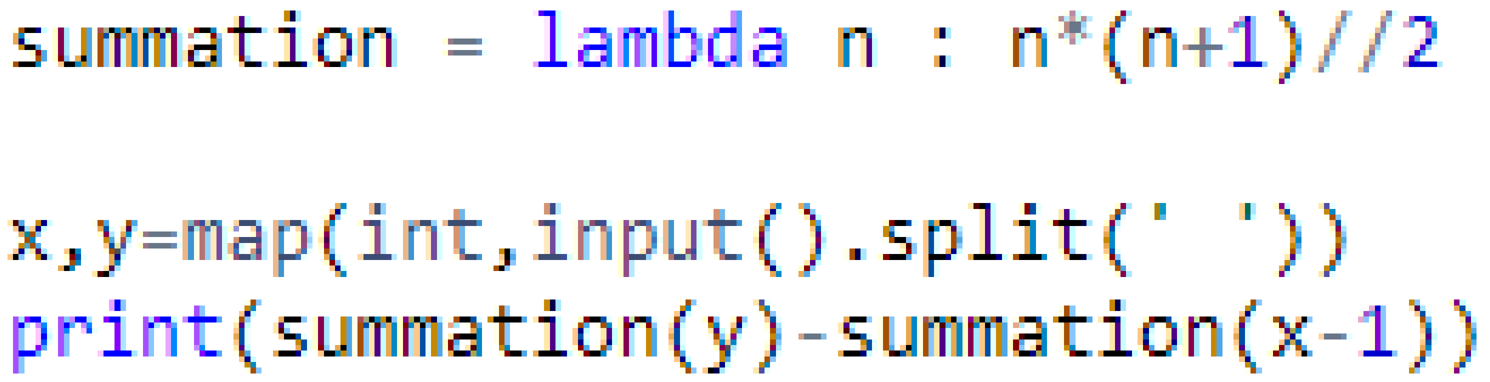


```
 3 8
33
```



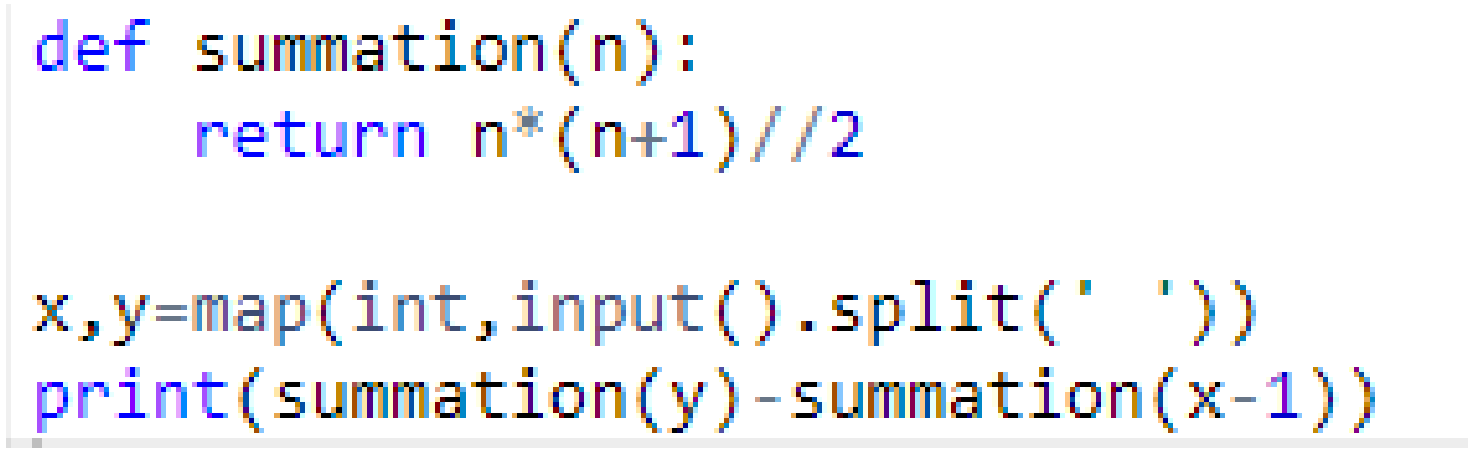
```
 7 10
34
```

Problem 4 Solution
Summation



```python
summation = lambda n : n*(n+1)//2

x,y=map(int,input().split(' '))
print(summation(y)-summation(x-1))
```

Problem 4 Solution
Summation

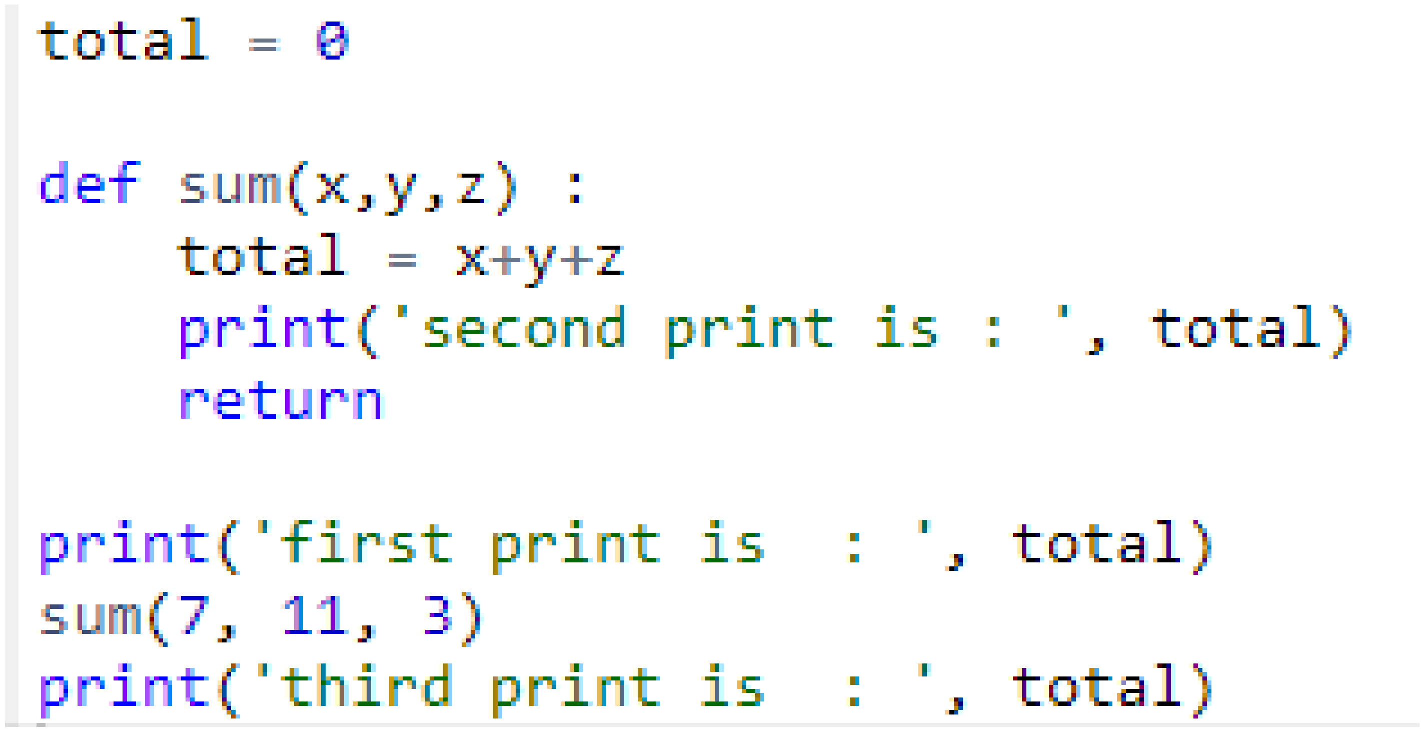


```
def summation(n):
    return n*(n+1)//2

x,y=map(int,input().split(' '))
print(summation(y)-summation(x-1))
```

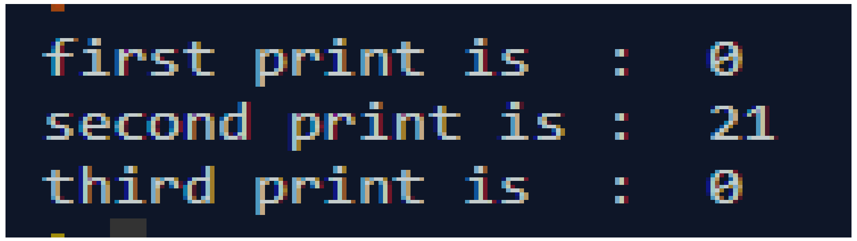
Global & Local variables

- All variables in a program may not be accessible at all locations in that program, this depend on where you have declared a variable. The scope of a variable determine the portion of the program where you can access a particular identifier.
- There are two basic scopes of variable in Python: Global variable - Local variable
- Variable that are defined inside a function body have a local scope, and those defined outside have a global scope.
- This means that local variables can be accessed only inside the function in which they are declared, where as global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope.

Global & Local variables

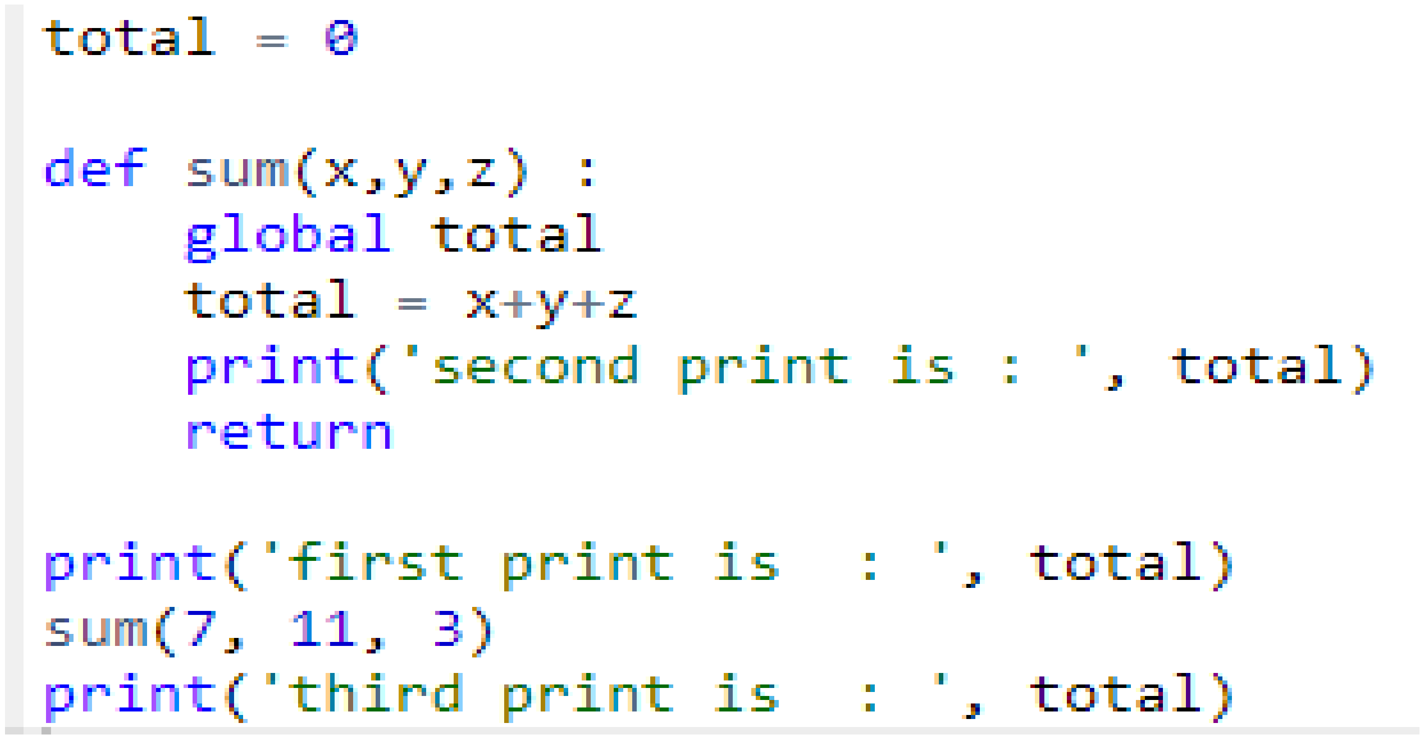


```python
total = 0

def sum(x,y,z) :
    total = x+y+z
    print('second print is : ', total)
    return

print('first print is  : ', total)
sum(7, 11, 3)
print('third print is  : ', total)
```

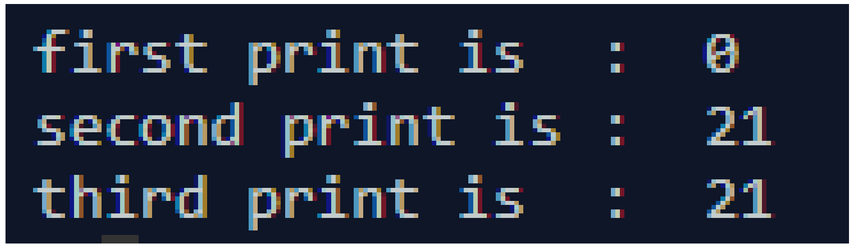


```
first print is  :  0
second print is :  21
third print is  :  0
```

Global & Local variables



```python
total = 0

def sum(x,y,z) :
    global total
    total = x+y+z
    print('second print is : ', total)
    return

print('first print is  : ', total)
sum(7, 11, 3)
print('third print is  : ', total)
```

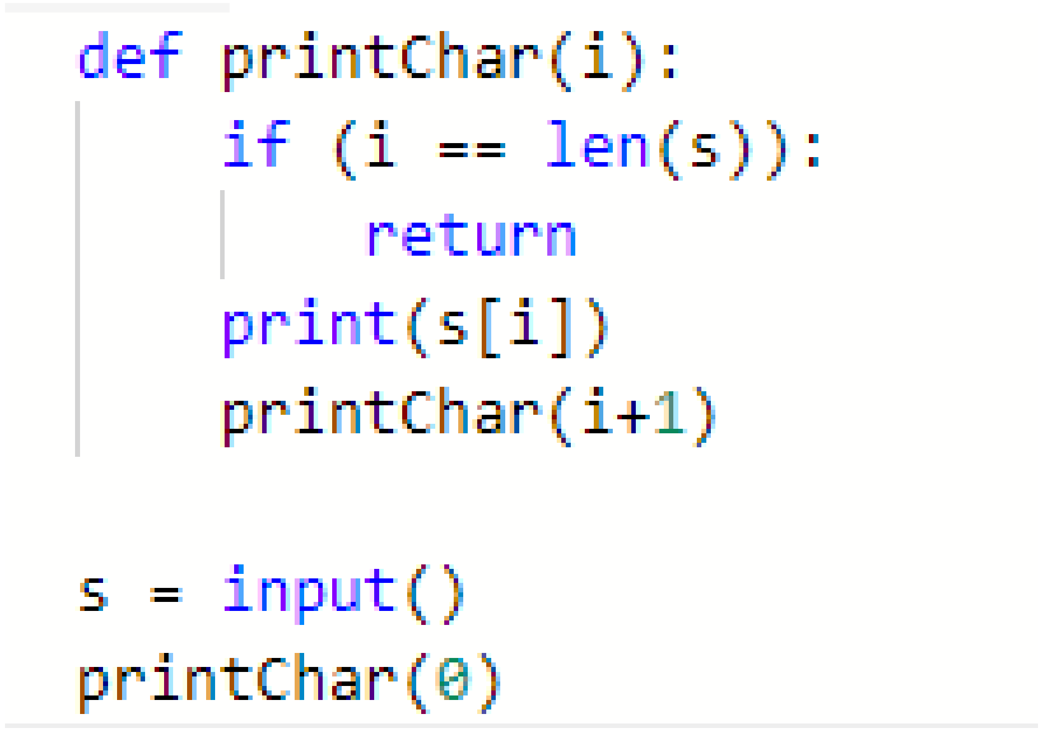


```
first print is  :  0
second print is :  21
third print is  :  21
```

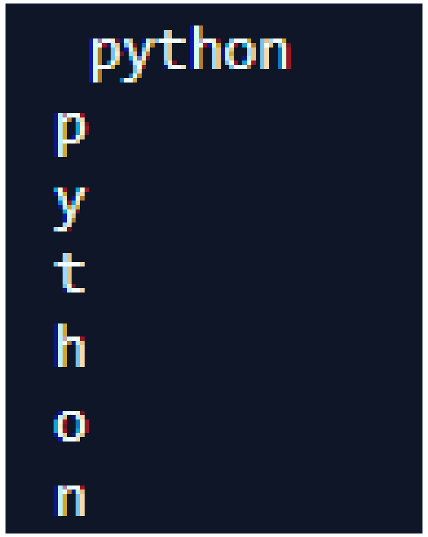
Recursion

- We know that in Python, a function can call other functions. It is even possible for the function to call itself. These type of construct are termed as recursive functions.
- Advantages of Recursion:
  - Recursive functions make the code look clean and elegant.
  - A complex task can be broken down into simpler sub-problems using recursion.
  - Sequence generation is easier with recursion than using some nested iteration.
- Disadvantages of Recursion:
  - Sometimes the logic behind recursion is hard to follow through.
  - Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
  - Recursive functions are hard to debug.

Recursion

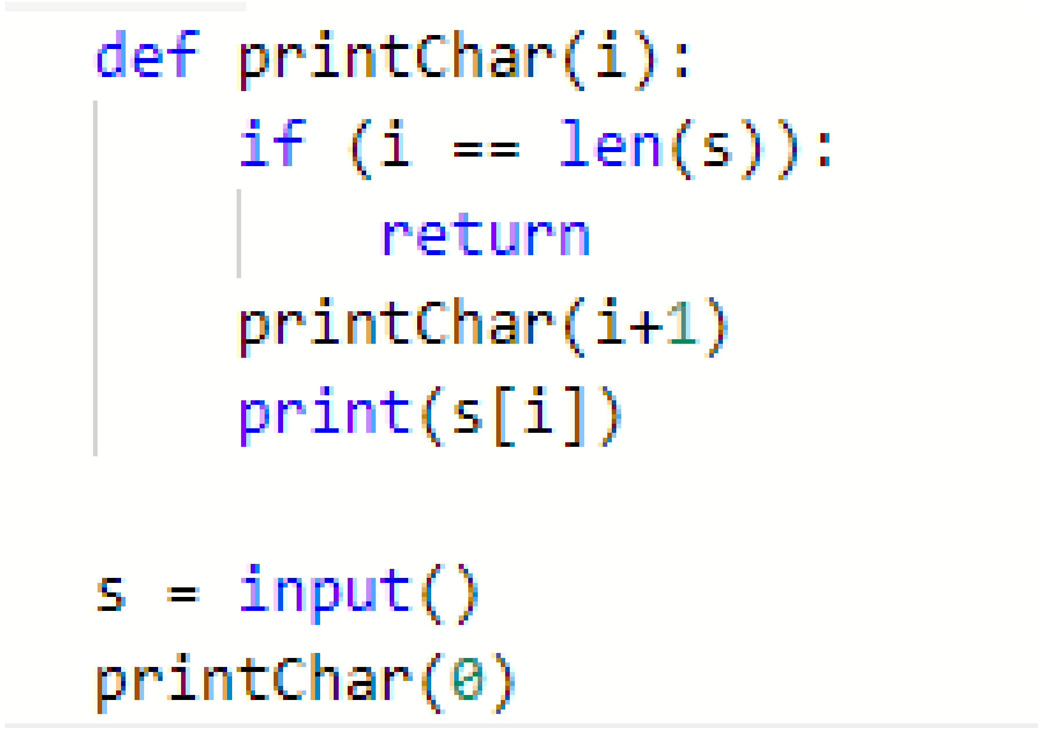


```
def printChar(i):
    if (i == len(s)):
        return
    print(s[i])
    printChar(i+1)

s = input()
printChar(0)
```

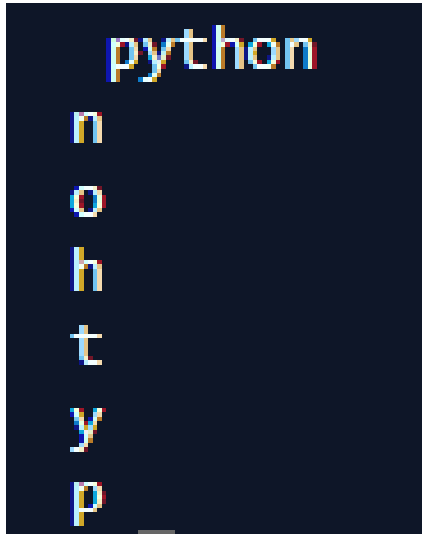


```
 python
p
y
t
h
o
n
```

Recursion



```
def printChar(i):
    if (i == len(s)):
        return
    printChar(i+1)
    print(s[i])

s = input()
printChar(0)
```

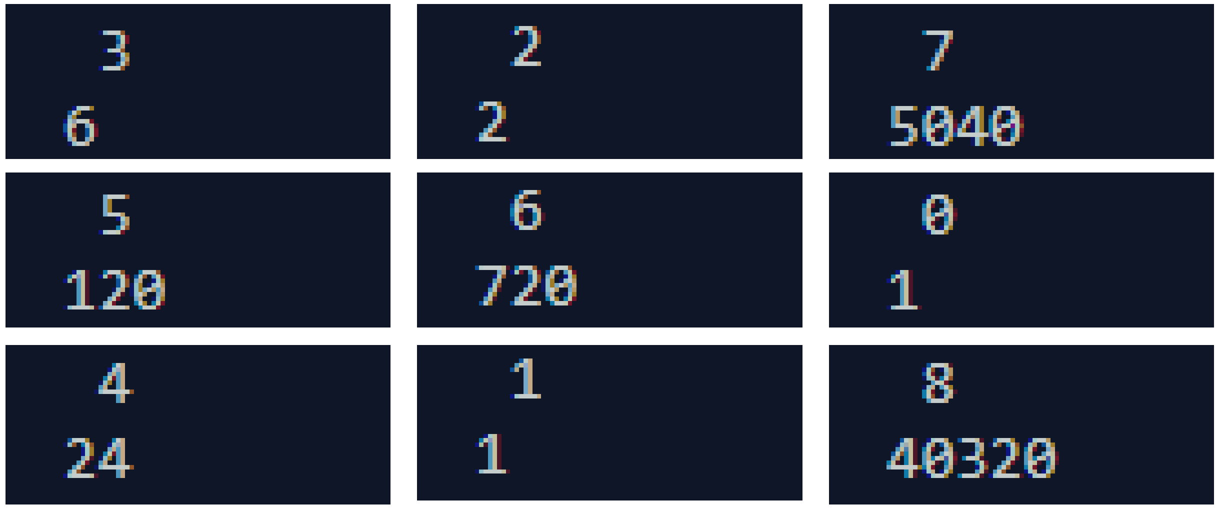


```
 python
n
o
h
t
y
p
```

Problem 5
Factorial

- Take as an input a number n then implement recursive function which calculate factorial of n to use it in our problem, such that n <= 10 and n is non-negative integer number

- Function Name: factorial
- Parameters: n
- Return: int value represent factorial of n
- Test Cases:



```
 3
6
```

```
 2
2
```

```
 7
5040
```

```
 5
120
```

```
 6
720
```

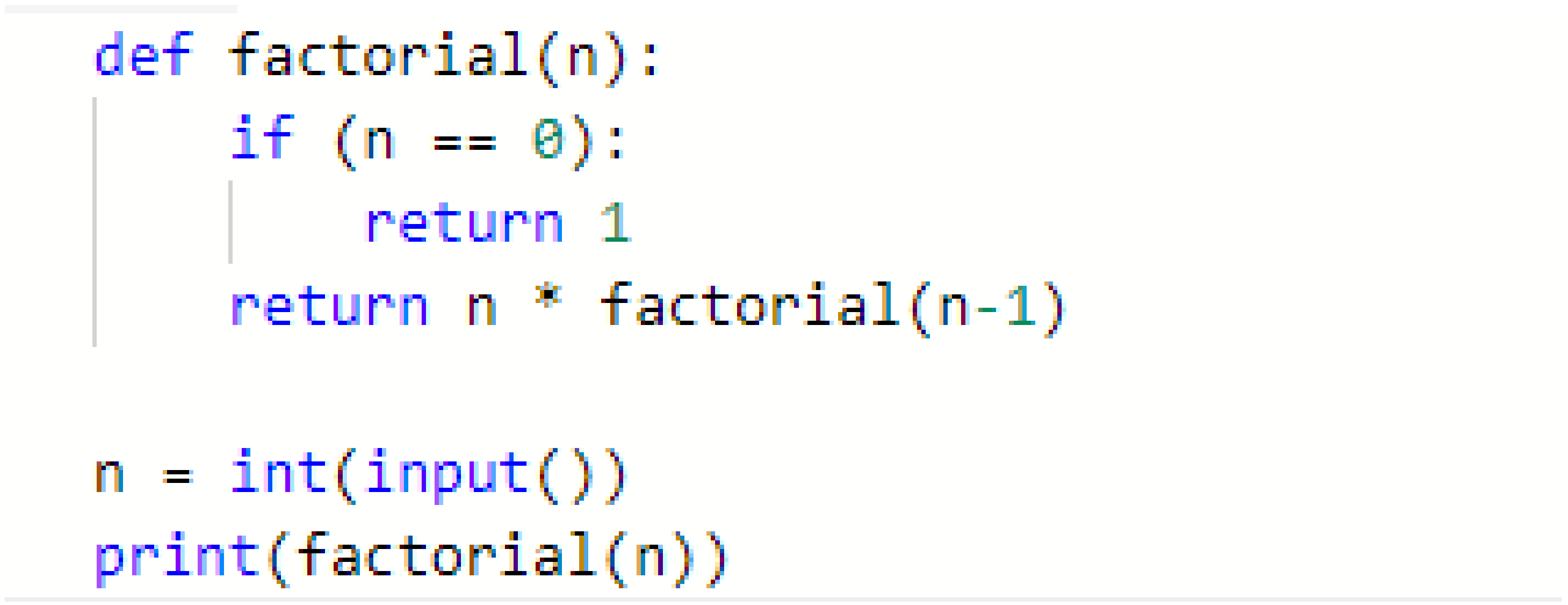
```
 0
1
```

```
 4
24
```

```
 1
1
```

```
 8
40320
```

Problem 5 Solution Factorial



```
def factorial(n):
    if (n == 0):
        return 1
    return n * factorial(n-1)

n = int(input())
print(factorial(n))
```

Outline

✓ 1- Function Definition

✓ 2- Calling a Function

✓ 3- return Statement

✓ 4- Passing by reference & value

✓ 5- Function Argument

✓ 6- Anonymous Function

✓ 7- Global & Local variables

✓ 8- Recursion

Practice

Questions ?

References

[01]	Online Courses YouTube playlists	http://bit.ly/2EcbLvZ
[02]	Programiz Tutorial	https://bit.ly/2paX2xA
[03]	Python 3 Tutorial Point PDF	http://bit.ly/2xcqp66
[04]	Python Guru Tutorial	https://bit.ly/28TtLRr
[05]	Python for Everybody PDF	http://bit.ly/2gSWkSS
[06]	A Byte of Python PDF	http://bit.ly/1l9Yqp9
[07]	Python Course Tutorial	https://bit.ly/2I076hU
[08]	Python Basics Tutorial PDF	http://bit.ly/1jMlluB
[09]	Learning to Program Using Python	http://bit.ly/2zO4f91
[10]	Think Python PDF	http://bit.ly/2cTgLIk
[11]	Python 3 Patterns, Recipes and Idioms	http://bit.ly/2by2bsN