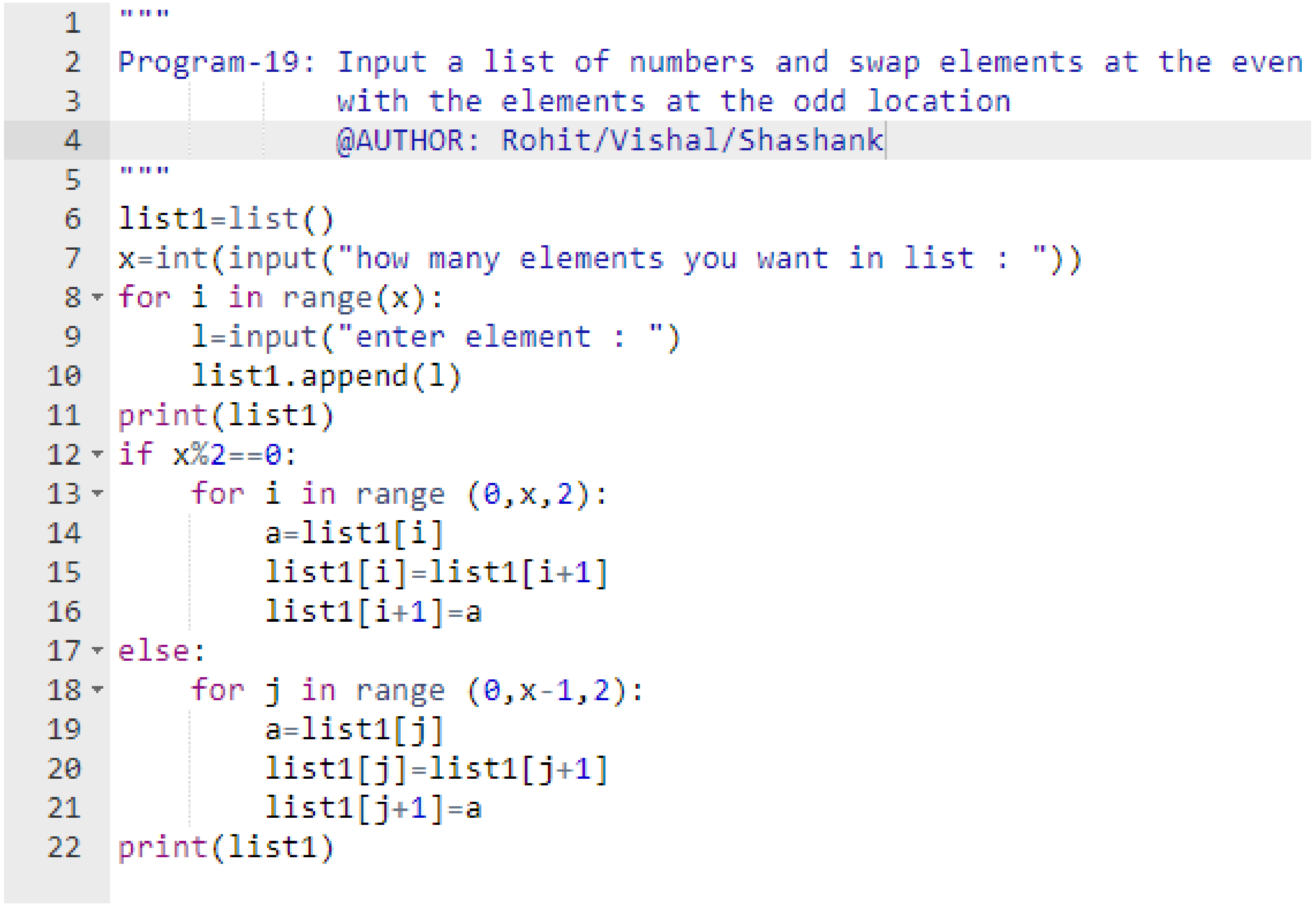
FUNCTIONS in Python

Introduction

Till now we have written some programs and might have realized that as the problem gets complex, the number of lines in a program increase, which makes the program look bulky and difficult to manage.



```python
"""
Program-19: Input a list of numbers and swap elements at the even
            with the elements at the odd location
            @AUTHOR: Rohit/Vishal/Shashank
"""
list1=list()
x=int(input("how many elements you want in list : "))
for i in range(x):
    l=input("enter element : ")
    list1.append(l)
print(list1)
if x%2==0:
    for i in range (0,x,2):
        a=list1[i]
        list1[i]=list1[i+1]
        list1[i+1]=a
else:
    for j in range (0,x-1,2):
        a=list1[j]
        list1[j]=list1[j+1]
        list1[j+1]=a
print(list1)
```

Introduction

- Function can be defined as a named group of instructions that accomplish a specific task when it is invoked.
- In programming, the use of function is one of the means to achieve modularity and reusability.

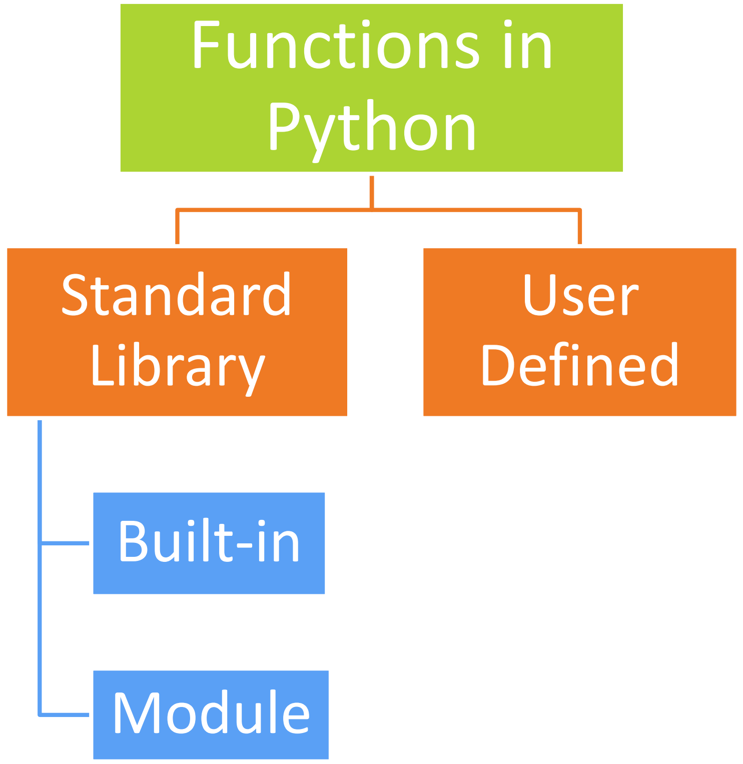
The process of dividing a computer program into separate independent blocks of code or separate sub-problems with different names and specific functionalities is known as modular programming

Advantages of function

- Increases readability, particularly for longer code as by using functions, the program is better organised and easy to understand
- Reduces code length as same code is not required to be written at multiple places in a program. This also makes debugging easier.
- Increases reusability, as function can be called from another function or another program. Thus, we can reuse or build upon already defined functions and avoid repetitions of writing the same piece of code.
- Work can be easily divided among team members and completed in parallel.

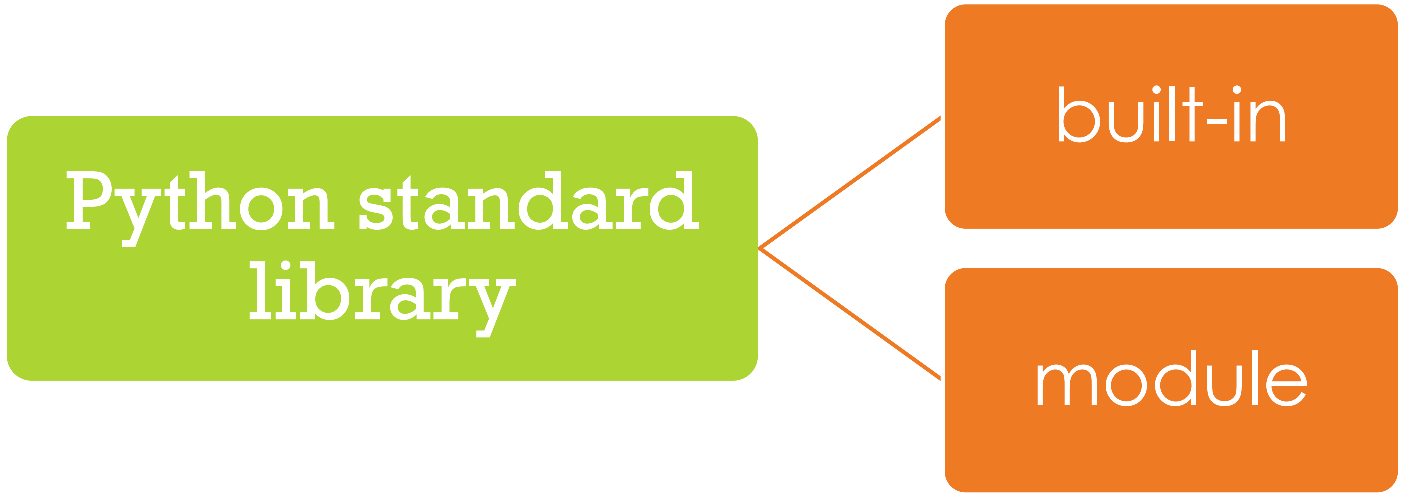
Types of function

- If we talk about functions in Python, it can be categorised as



Python Standard Library

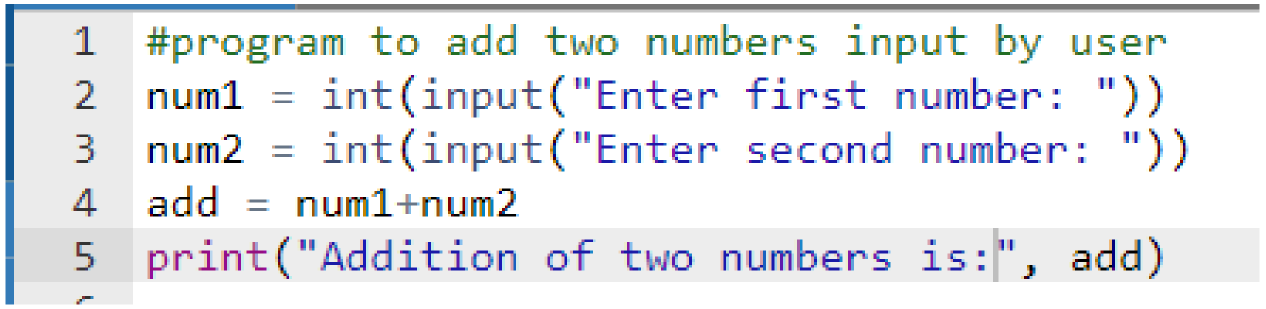
- Python has a very extensive standard library. It is a collection of many built-in functions that can be called in the program whenever required, thus saving programmer's time of creating those commonly used functions everytime.



Built-in functions

- Built-in functions are the ready-made functions in Python that are frequently used in programs.

  for e.g.)



```
#program to add two numbers input by user
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
add = num1+num2
print("Addition of two numbers is:", add)
```

- In the above program, input(), int() and print() are built-in functions
- The set of instructions to be executed for these built-in functions are already defined in the python interpreter.

Built-in functions

- Now, let's consider the below statement:

```
num = input("Enter your number: ")
```

Prompt

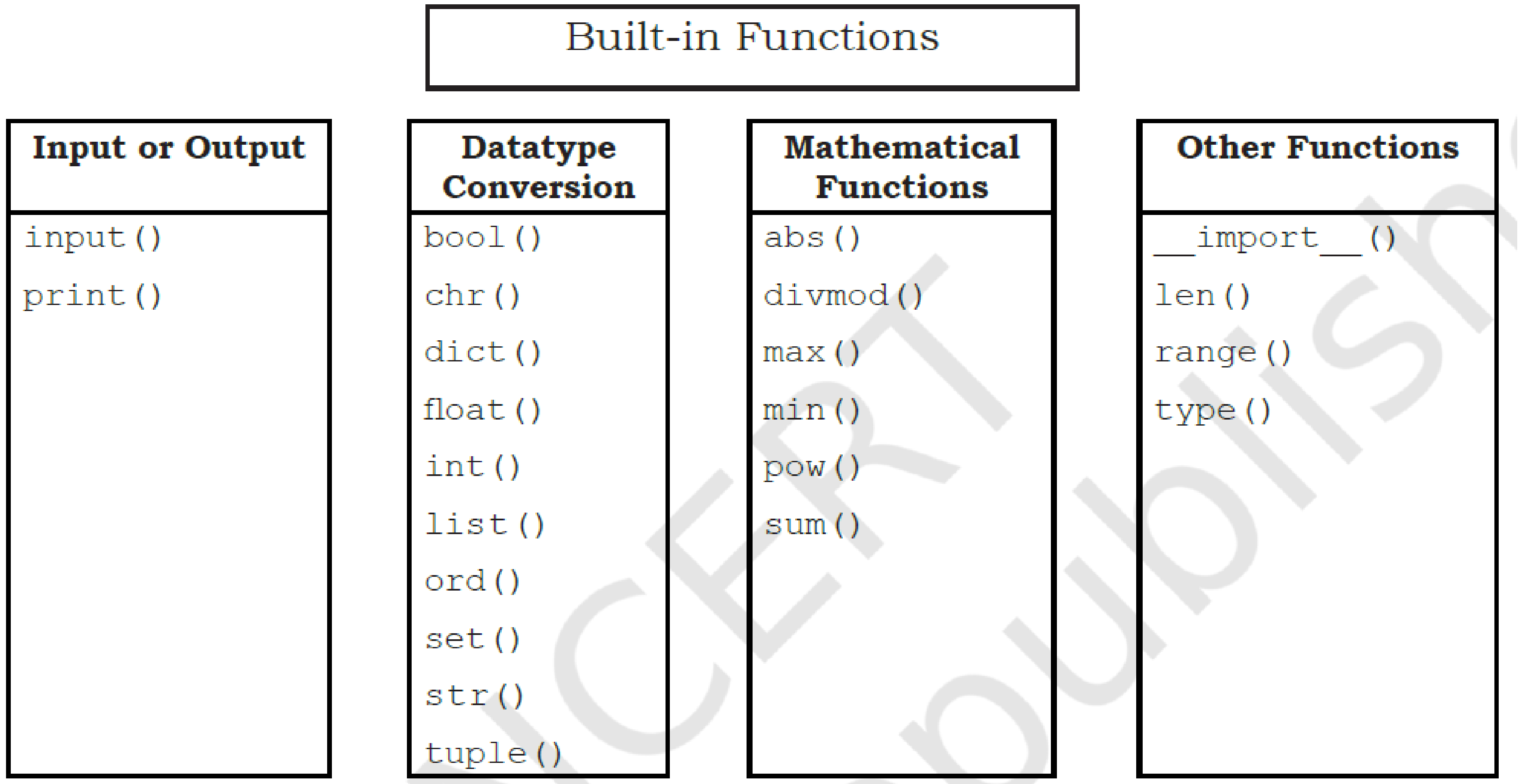
- Let's try to answer these questions:

  name of the function:

  does this function accept a value or argument?

  Does this function return a value?

Built-in function



Built-in Functions

Input or Output	Datatype Conversion	Mathematical Functions	Other Functions
`input()`	`bool()`	`abs()`	`__import__()`
`print()`	`chr()`	`divmod()`	`len()`
	`dict()`	`max()`	`range()`
	`float()`	`min()`	`type()`
	`int()`	`pow()`	
	`list()`	`sum()`	
	`ord()`		
	`set()`		
	`str()`		
	`tuple()`		

Table 7.1 Commonly used built-in functions

Function Syntax	Arguments	Returns	Example Output

abs(x)	x may be an integer or floating point number	Absolute value of x	`>>> abs(4)`
			`4`
			`>>> abs(-5.7)`
			`5.7`

divmod(x,y)	x and y are integers	A tuple: (quotient, remainder)	`>>> divmod(7,2)`
			`(3, 1)`
			`>>> divmod(7.5,2)`
			`(3.0, 1.5)`
			`>>> divmod(-7,2)`
			`(-4, 1)`

max(sequence) or max(x,y,z,...)	x,y,z,.. may be integer or floating point number	Largest number in the sequence/ largest of two or more arguments	`>>> max([1,2,3,4])`
			`4`
			`>>> max("Sincerity")`
			`'y' #Based on ASCII value`

min(sequence)
or
min(x,y,z,...)
	x, y, z,.. may be integer or floating point number	Smallest number in the sequence/ smallest of two or more arguments	`>>> min([1,2,3,4])`
			`1`
			`>>> min("Sincerity")`
			`'S'`
			`#Uppercase letters have lower ASCII values than lowercase letters.`
			`>>> min(23,4,56)`
			`4`

pow(x,y[,z])	x, y, z may be integer or floating point number	x^y (x raised to the power y)
		if z is provided, then: (x^y) % z
			`>>> pow(5,2)`
			`25.0`
			`>>> pow(5.3,2.2)`
			`39.2`
			`>>> pow(5,2,4)`
			`1`

sum(x[,num])	x is a numeric sequence and num is an optional argument	Sum of all the elements in the sequence from left to right.
		if given parameter, num is added to the sum
			`>>> sum([2,4,7,3])`
			`16`
			`>>> sum([2,4,7,3],3)`
			`19`
			`>>> sum((52,8,4,2))`
			`66`

Module

- Other than the built-in functions, the Python standard library also consists of a number of modules. While a function is a grouping of instructions, a module is a grouping of functions.

- As we know that when a program grows, function is used to simplify the code and to avoid repetition.

- For a complex problem, it may not be feasible to manage the code in one single file. Then, the program is divided into different parts under different levels, called modules

Module

- Suppose we have created some functions in a program and we want to reuse them in another program. In that case, we can save those functions under a module and reuse them.

- A module is created as a python (.py) file containing a collection of function definitions.

- To use a module, we need to import the module. Once we import a module, we can directly use all the functions of that module.

Import a Module and using a function

- The syntax of import statement to import a module is as follows:

  ```
  import modulename1 [,modulename2, …]
  ```

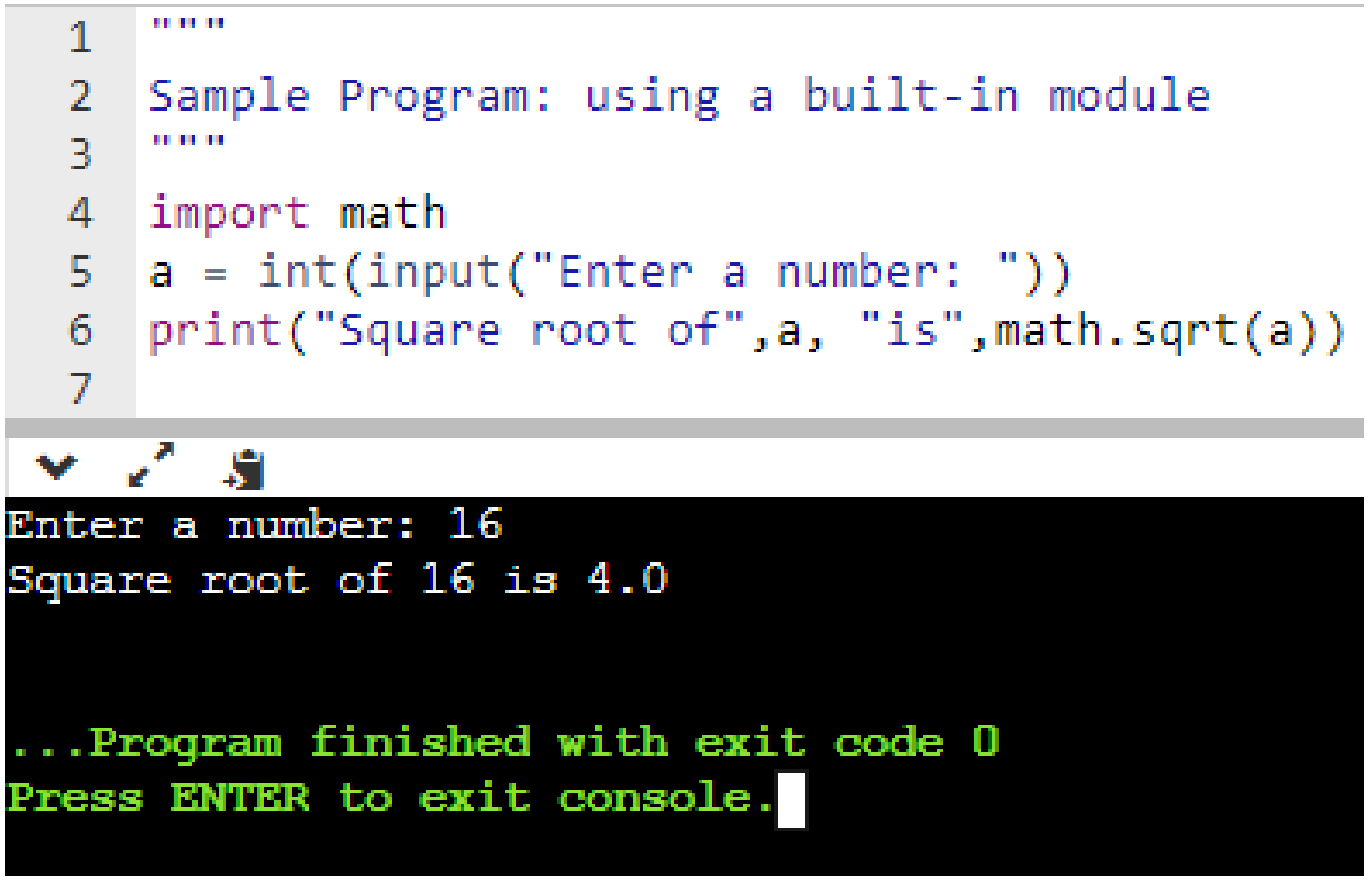
  This gives us access to all the functions in the module(s).

- To call a function of a module, the function name should be preceded with the name of the module with a dot(.) as a separator.

  The syntax for calling a function of particular module is as shown below:

  ```
  modulename.functionname()
  ```

Import a Module and using a function



```
"""
Sample Program: using a built-in module
"""
import math
a = int(input("Enter a number: "))
print("Square root of",a, "is",math.sqrt(a))

```

```
Enter a number: 16
Square root of 16 is 4.0


...Program finished with exit code 0
Press ENTER to exit console.
```

Module

- Python library has many built-in modules that are really handy to programmers. Let us explore some commonly used modules and the frequently used functions that are found in those modules:

  1. math
  2. random
  3. statistics

Note:

All the module names are in lowercase.

Built-in module: math

- math module contains different types of mathematical functions. Most of the functions in this module return a float value.

  In order to use the math module we need to import it using the following statement:

  ```
  import math
  ```
- Some of the commonly used functions in math module are:

  sqrt, ceil, floor, pow, fabs, sin, cos, tan

Function Syntax	**Arguments**	**Returns**	**Example Output**

math.ceil(x)	x may be an integer or floating point number	ceiling value of x	`>>> math.ceil(-9.7)`
			`-9`
			`>>> math.ceil (9.7)`
			`10`
			`>>> math.ceil(9)`
			`9`

math.floor(x)	x may be an integer or floating point number	floor value of x	`>>> math.floor(-4.5)`
			`-5`
			`>>> math.floor(4.5)`
			`4`
			`>>> math.floor(4)`
			`4`

math.fabs(x)	x may be an integer or floating point number	absolute value of x	`>>> math.fabs(6.7)`
			`6.7`
			`>>> math.fabs(-6.7)`
			`6.7`
			`>>> math.fabs(-4)`
			`4.0`

math.factorial(x)	x is a positive integer	factorial of x	`>>> math.factorial(5)`
			`120`

math.fmod(x,y)	x and y may be an integer or floating point number	x % y with sign of x	`>>> math.fmod(4,4.9)`
			`4.0`
			`>>> math.fmod(4.9,4.9)`
			`0.0`
			`>>> math.fmod(-4.9,2.5)`
			`-2.4`
			`>>> math.fmod(4.9,-4.9)`
			`0.0`

math.gcd(x,y)	x, y are positive integers	gcd (greatest common divisor) of x and y	`>>> math.gcd(10,2)`
			`2`

math.pow(x,y)	x, y may be an integer or floating point number	x^y (x raised to the power y)	`>>> math.pow(3,2)`
			`9.0`
			`>>> math.pow(4,2.5)`
			`32.0`
			`>>> math.pow(6.5,2)`
			`42.25`
			`>>> math.pow(5.5,3.2)`
			`233.97`

math.sqrt(x)	x may be a positive integer or floating point number	square root of x	`>>> math.sqrt(144)`
			`12.0`
			`>>> math.sqrt(.64)`
			`0.8`

math.sin(x)	x may be an integer or floating point number in radians	sine of x in radians	`>>> math.sin(0)`
			`0`
			`>>> math.sin(6)`
			`-0.279`

Built-in module: random

- random module contains functions that are used for generating random numbers.

  For using this module, we can import it using the following statement:

  ```
  import random
  ```

- Some of the commonly used functions in random module are:

  random, randint, randrange

Function Syntax	Argument	Return	Example Output

random.random()	No argument (void)	Random Real Number (float) in the range 0.0 to 1.0	`>>> random.random()`
			`0.65333522`

random. randint(x,y)	x, y are integers such that x <= y	Random integer between x and y	`>>> random.randint(3,7)`
			`4`
			`>>> random.randint(-3,5)`
			`1`
			`>>> random.randint(-5,-3)`
			`-5.0`

random. randrange(y)	y is a positive integer signifying the stop value	Random integer between 0 and y	`>>> random.randrange(5)`
			`4`

random. randrange(x,y)	x and y are positive integers signifying the start and stop value	Random integer between x and y	`>>> random.randrange(2,7)`
			`2`

Built-in module: statistics

- statistics module contains functions that are used for calculating statistics of numeric (Real-valued) data.

  For using this module, we can import it using the following statement:

  ```
  import statistics
  ```

- Some of the commonly used functions in statistics module are:

  mean, median, mode

Function Syntax	Argument	Return	Example Output

statistics.mean(x)	x is a numeric sequence	arithmetic mean	`>>> statistics.`
			`mean([11,24,32,45,51])`
			`32.6`

statistics.median(x)	x is a numeric sequence	median (middle value) of x	`>>>statistics.`
			`median([11,24,32,45,51])`
			`32`

statistics.mode(x)	x is a sequence	mode (the most repeated value)	`>>> statistics.`
			`mode([11,24,11,45,11])`
			`11`
			`>>> statistics.`
			`mode(("red","blue","red"))`
			`'red'`

Some important points regarding module

- import statement can be written anywhere in the program
- Module must be imported only once
- In order to get a list of modules available in Python, we can use the following statement:

```
>>> help("module")
```

- To view the content of a module say math, type the following:

```
>>> help("math")
```

An alternative to use module functions

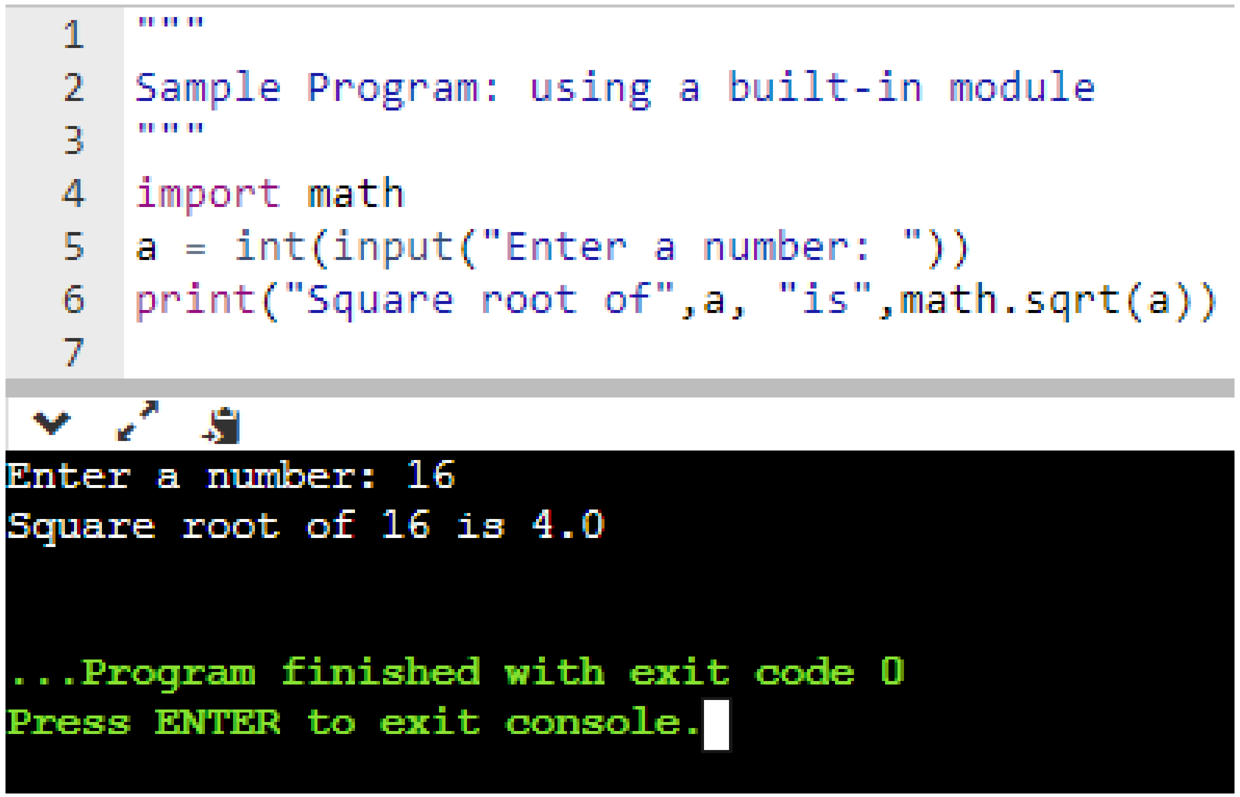
- Instead of loading all the functions into memory by importing a module, we can access the required functions from a particular module using from statement. Its syntax is

```
>>> from modulename import functionname [,functionname,...]
```

- To use the function when imported using "from statement" we do not need to precede it with the module name.

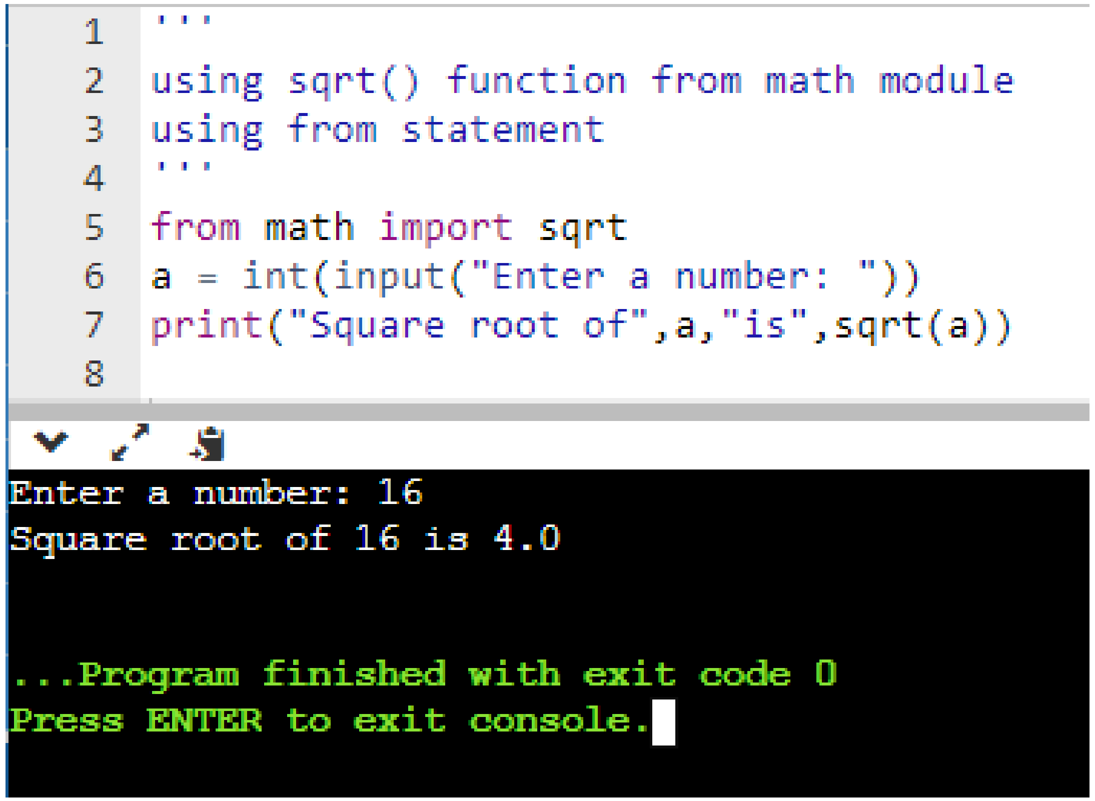
- It loads only the specified function(s) instead of all the functions in a module, which saves memory

An alternative to use module functions



```python
"""
Sample Program: using a built-in module
"""
import math
a = int(input("Enter a number: "))
print("Square root of",a, "is",math.sqrt(a))

```

```
Enter a number: 16
Square root of 16 is 4.0


...Program finished with exit code 0
Press ENTER to exit console.
```



```python
'''
using sqrt() function from math module
using from statement
'''
from math import sqrt
a = int(input("Enter a number: "))
print("Square root of",a,"is",sqrt(a))

```

```
Enter a number: 16
Square root of 16 is 4.0


...Program finished with exit code 0
Press ENTER to exit console.
```

User defined functions

- With the help of built-in functions and module, we can use these functions directly in our program without defining them (i.e., without writing code for these functions)
- However, in addition to the standard library functions, we can define our own functions while writing the program. Such functions are called user defined functions.
- Thus, a function defined to achieve some task as per the programmer's requirement is called a user defined function.

User defined functions

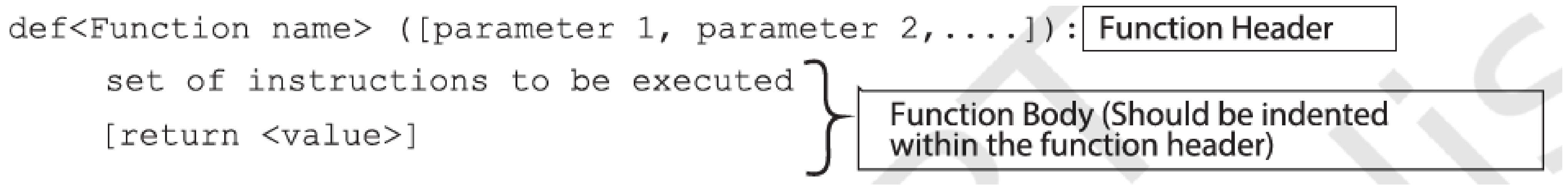
Note:

Once defined, a function can be called repeatedly from different places of the program without writing all the codes of that function everytime, or it can be called from inside another function, by simply writing the name of the function and passing the required parameters, if any.

The programmer can define as many functions as desired while writing the code.

Creating a User defined function

- A function definition begins with def (short for define). The syntax for creating a user defined function is as follows:



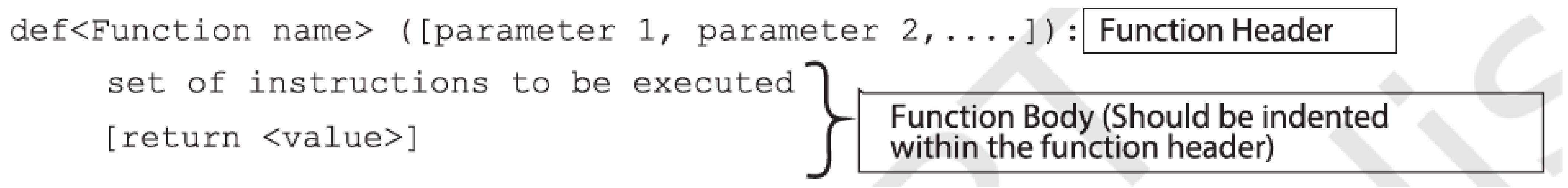
```
def<Function name> ([parameter 1, parameter 2,....]):
      set of instructions to be executed
      [return <value>]
```

Function Header

Function Body (Should be indented within the function header)

  - The items enclosed in "[ ]" are called parameters and they are optional. Hence, a function may or may not have parameters.
  - Similarly, a function may or may not return a value.

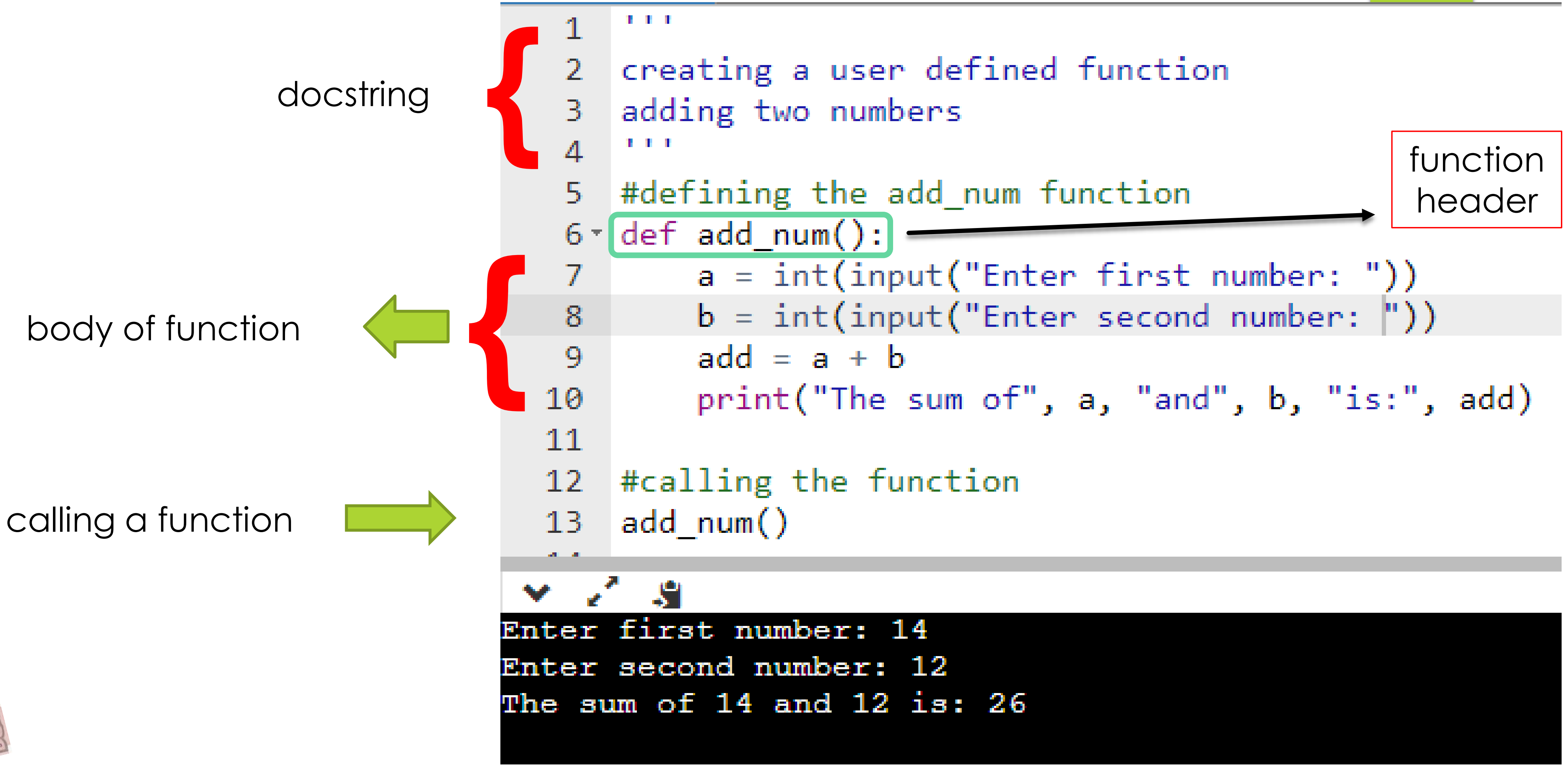
Creating a User defined function



```
def<Function name> ([parameter 1, parameter 2,....]):
     set of instructions to be executed
     [return <value>]
```

Function Header
Function Body (Should be indented within the function header)

- Function header always end with a colon (:)
- Function name should be unique. Rules for naming identifiers also applies for function naming.
- The statements outside the function indentation are not considered as part of the function.



```python
'''
creating a user defined function
adding two numbers
'''
#defining the add_num function
def add_num():
    a = int(input("Enter first number: "))
    b = int(input("Enter second number: "))
    add = a + b
    print("The sum of", a, "and", b, "is:", add)

#calling the function
add_num()
```

```
Enter first number: 14
Enter second number: 12
The sum of 14 and 12 is: 26
```

Note:

If you have noticed, there is no parameters in the function header and the function is not returning any value

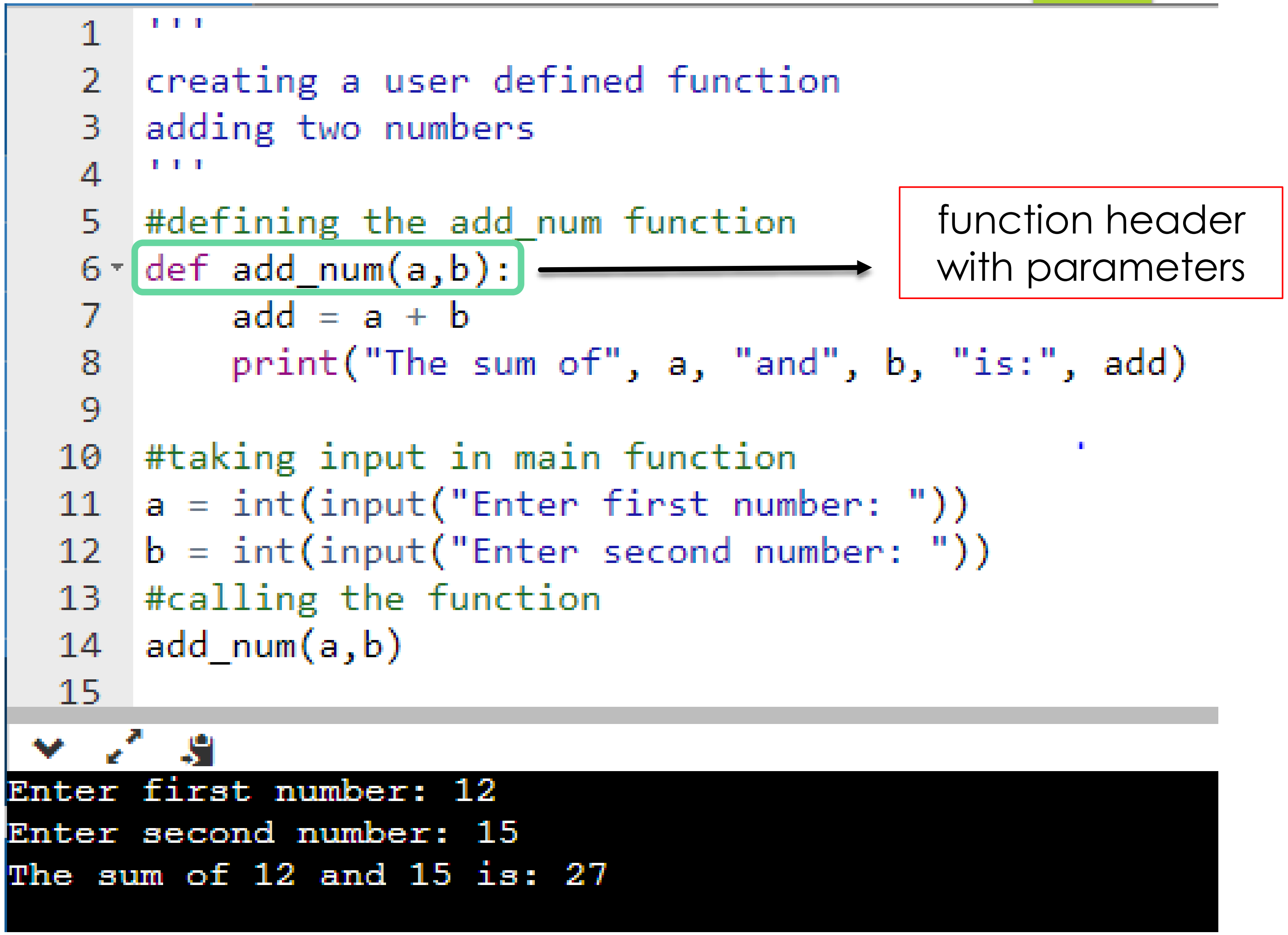
Arguments and parameters

- In the previous example, we have taken the inputs for both the numbers in the function add_num itself.

- However, we could have taken inputs in the main function and pass these values as arguments while calling the function add_num, like this:

```
add_num(a,b)
```

and these arguments will be received as corresponding parameters in the function header



```
'''
creating a user defined function
adding two numbers
'''
#defining the add_num function
def add_num(a,b):
    add = a + b
    print("The sum of", a, "and", b, "is:", add)

#taking input in main function
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
#calling the function
add_num(a,b)
```

```
Enter first number: 12
Enter second number: 15
The sum of 12 and 15 is: 27
```

calling a function
with arguments

Arguments and parameters

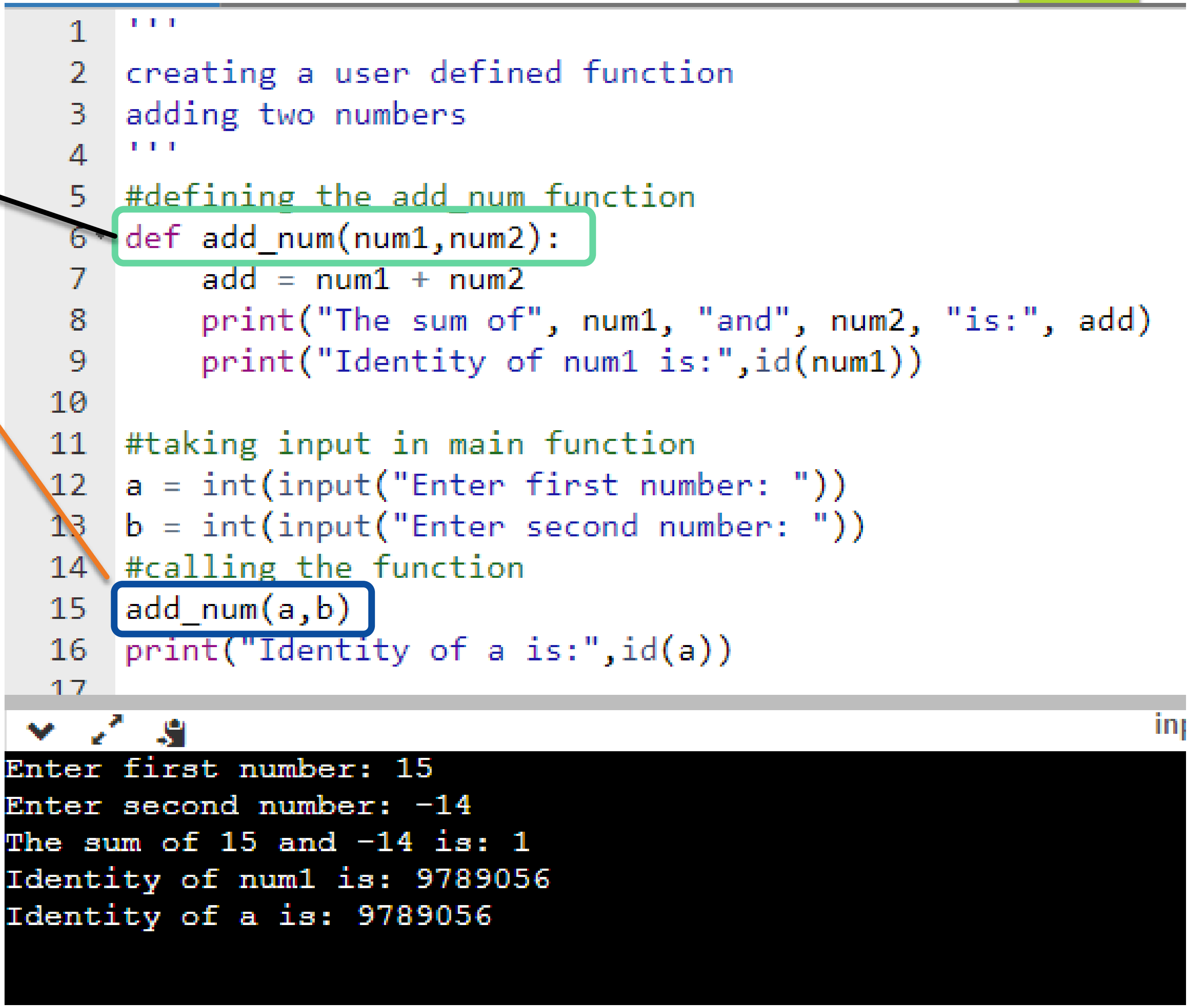
- It is not necessary that arguments and parameters should be of same name, it could be of different names.

- However, since both arguments and parameters are referring to same value, they are bound to have the same identity.

Note:

We can pass any data type as argument while calling the function

Parameters and arguments have different names



```
'''
creating a user defined function
adding two numbers
'''
#defining the add_num function
def add_num(num1,num2):
    add = num1 + num2
    print("The sum of", num1, "and", num2, "is:", add)
    print("Identity of num1 is:",id(num1))

#taking input in main function
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
#calling the function
add_num(a,b)
print("Identity of a is:",id(a))
```

```
Enter first number: 15
Enter second number: -14
The sum of 15 and -14 is: 1
Identity of num1 is: 9789056
Identity of a is: 9789056
```

argument a and parameter num1 refer to same value, hence same identity

Arguments and parameters

Note:

A function argument can also be an expression, such as
add_num(a+5, b+5)

In such a case, the argument is evaluated before calling the function so that a valid value can be assigned to the parameter.

The parameters should be in the same order as that of the arguments.

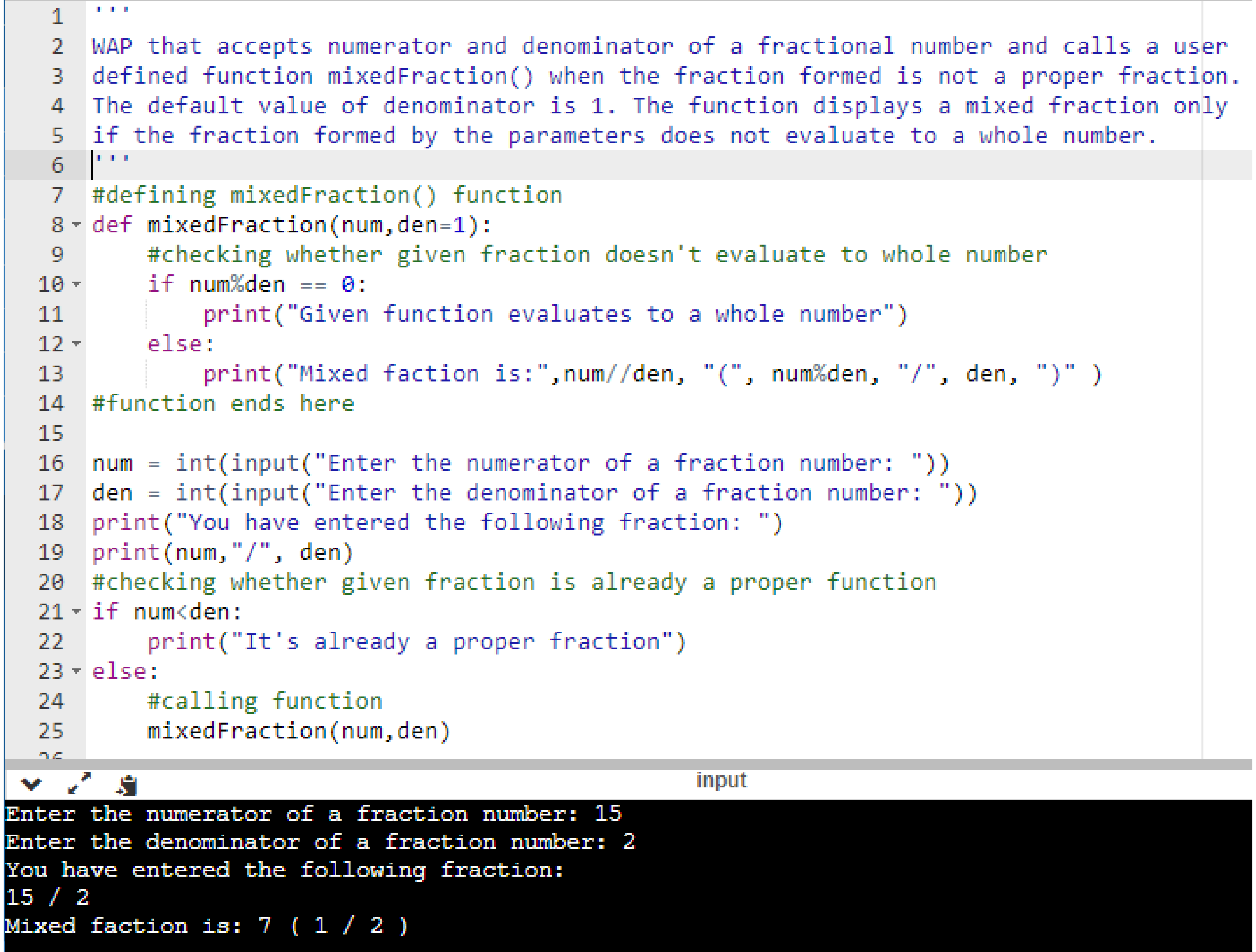
Default parameter

- Python allows assigning a default value to the parameter. A default value is a value that is predecided and assigned to the parameter when the function call does not have its corresponding argument.

Note:

while calling a function if we pass the argument for default parameter, then default value of parameter will be overwritten by that argument

Code:



```
'''
WAP that accepts numerator and denominator of a fractional number and calls a user
defined function mixedFraction() when the fraction formed is not a proper fraction.
The default value of denominator is 1. The function displays a mixed fraction only
if the fraction formed by the parameters does not evaluate to a whole number.
'''
#defining mixedFraction() function
def mixedFraction(num,den=1):
    #checking whether given fraction doesn't evaluate to whole number
    if num%den == 0:
        print("Given function evaluates to a whole number")
    else:
        print("Mixed faction is:",num//den, "(", num%den, "/", den, ")" )
#function ends here

num = int(input("Enter the numerator of a fraction number: "))
den = int(input("Enter the denominator of a fraction number: "))
print("You have entered the following fraction: ")
print(num,"/", den)
#checking whether given fraction is already a proper function
if num<den:
    print("It's already a proper fraction")
else:
    #calling function
    mixedFraction(num,den)
```

input

Output:

```
Enter the numerator of a fraction number: 15
Enter the denominator of a fraction number: 2
You have entered the following fraction:
15 / 2
Mixed faction is: 7 ( 1 / 2 )
```

Default parameter

- The default parameters must be the trailing parameters in the function header that means if any parameter is having default value then all the other parameters to its right must also have default values.

  for e.g.)

```
def mixedFraction(num, deno = 1):                      #correct
def mixedFraction(num = 2, deno = 1):                  #correct
def calcInterest(principal = 1000, rate, time = 5):    #incorrect
def calcInterest(rate, principal = 1000, time = 5):    #correct
```

Function returning value(s)

- A function may or may not return a value when called. Functions which don't return any value is called void functions.

- But a situation may arise, wherein we need to send value(s) from the function to its calling function. This is done using return statement.

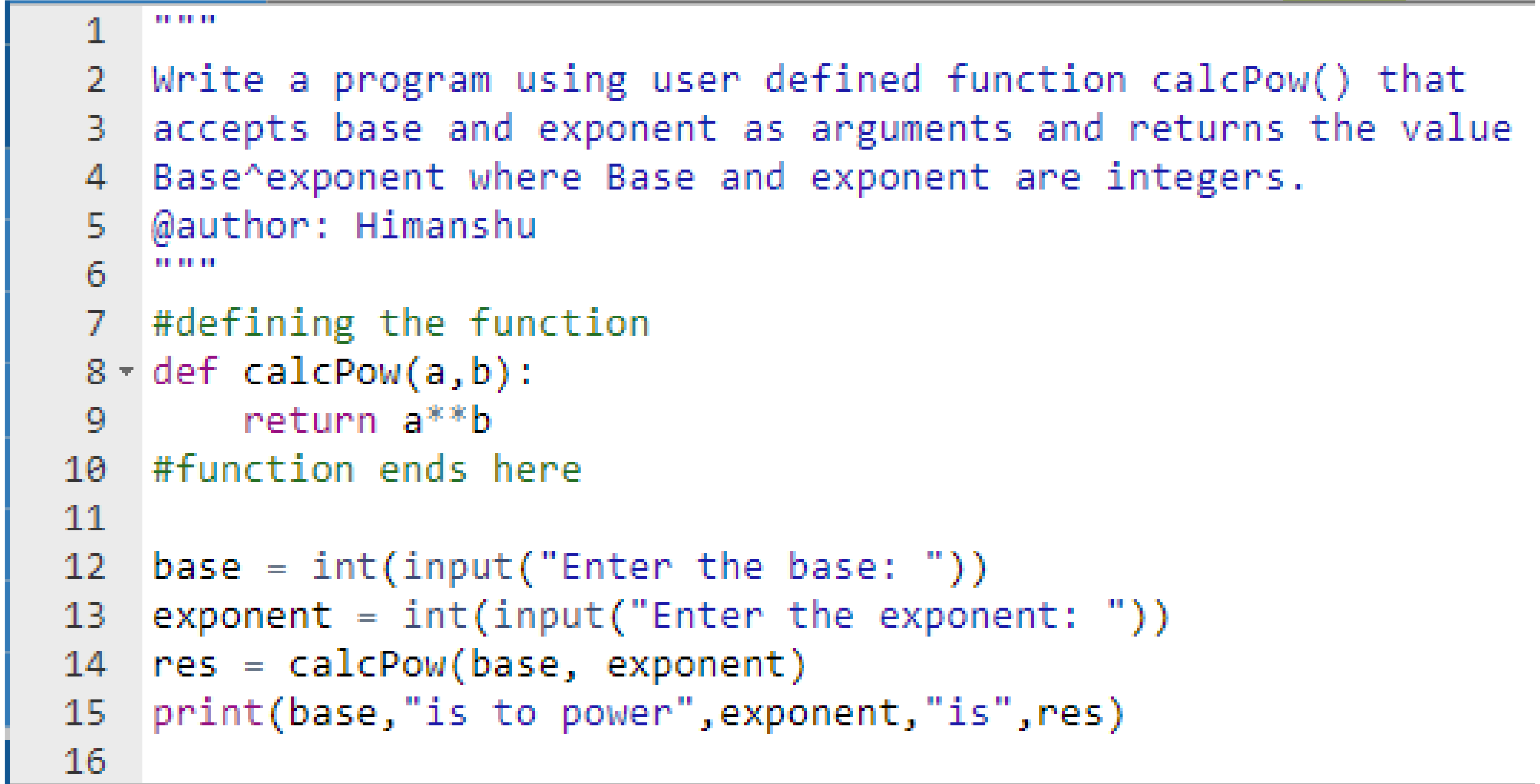
  The return statement does the following:

  - returns the control to the calling function.
  - return value(s) or None.

Function returning value(s)

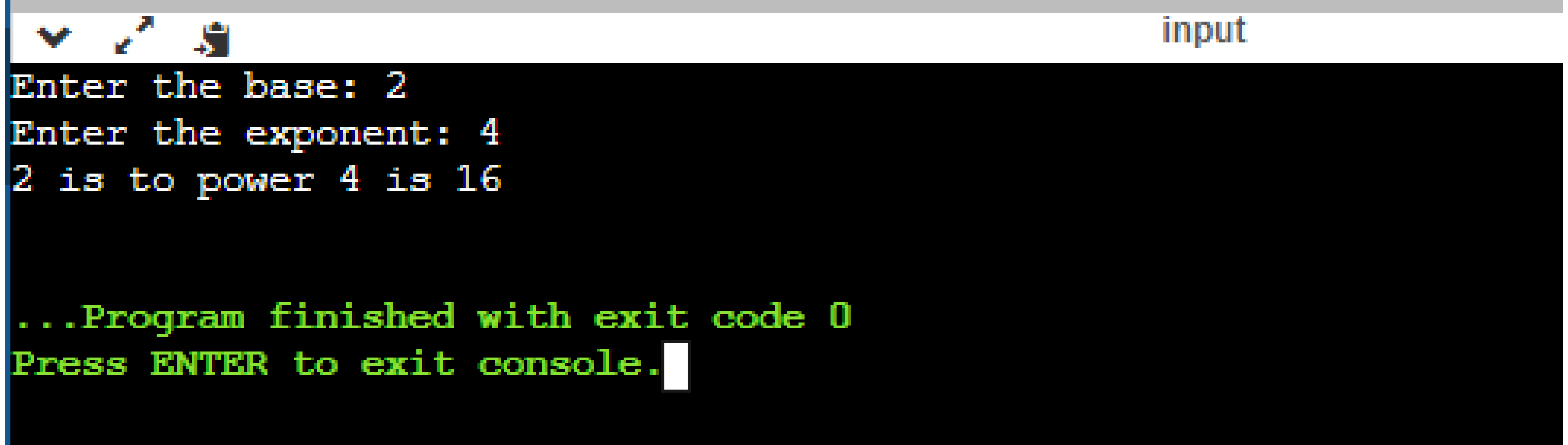
- Sample program: Write a program using user defined function calcPow() that accepts base and exponent as arguments and returns the value $Base^{exponent}$ where Base and exponent are integers.

Code:



```
"""
Write a program using user defined function calcPow() that
accepts base and exponent as arguments and returns the value
Base^exponent where Base and exponent are integers.
@author: Himanshu
"""
#defining the function
def calcPow(a,b):
    return a**b
#function ends here

base = int(input("Enter the base: "))
exponent = int(input("Enter the exponent: "))
res = calcPow(base, exponent)
print(base,"is to power",exponent,"is",res)

```

Output:



```
Enter the base: 2
Enter the exponent: 4
2 is to power 4 is 16


...Program finished with exit code 0
Press ENTER to exit console.
```

Function returning value(s)

- So far we have learnt that a function may or may not have parameter(s) and a function may or may not return any value(s).

- In Python, as per our requirements, we can have the function in either of the following ways:
    - Function with no argument and no return value
    - Function with no argument and with return value(s)
    - Function with argument(s) and no return value
    - Function with argument(s) and return value(s)

Flow of Execution

- Flow of execution can be defined as the order in which the statements in a program are executed.

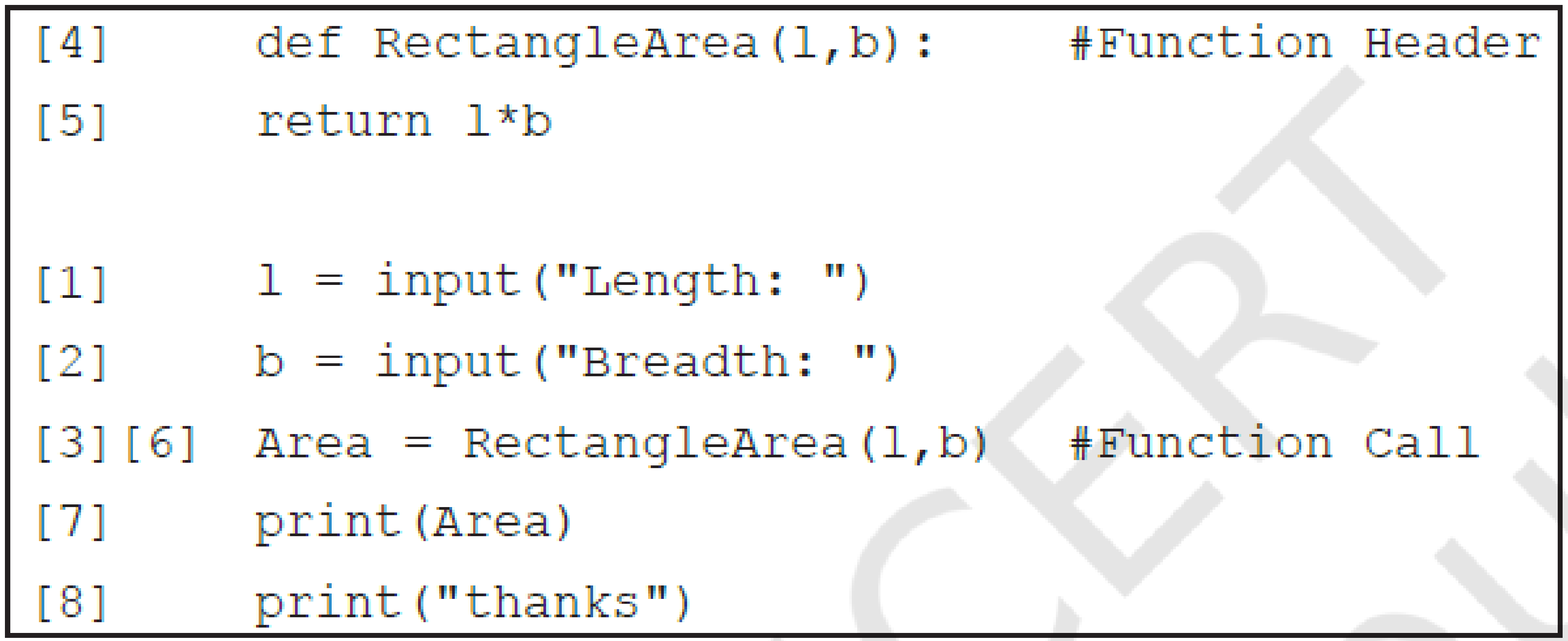
- The Python interpreter starts executing the instructions in a program from the first statement. The statements are executed one by one, in the order of appearance from top to bottom.

- When the interpreter encounters a function definition, the statements inside the function are not executed until the function is called.

Flow of Execution

- Later, when the interpreter encounters a function call, there is a little deviation in the flow of execution. In that case, instead of going to the next statement, the control jumps to the called function and executes the statement of that function.
- After that, the control comes back the point of function call so that the remaining statements in the program can be executed.
- Therefore, when we read a program, we should not simply read from top to bottom. Instead, we should follow the flow of control or execution.
- It is also important to note that a function must be defined before its call within a program.

Flow of Execution



```
[4]      def RectangleArea(l,b):     #Function Header
[5]      return l*b


[1]      l = input("Length: ")
[2]      b = input("Breadth: ")
[3][6]   Area = RectangleArea(l,b)   #Function Call
[7]      print(Area)
[8]      print("thanks")
```

Figure 7.5: Order of execution of statements

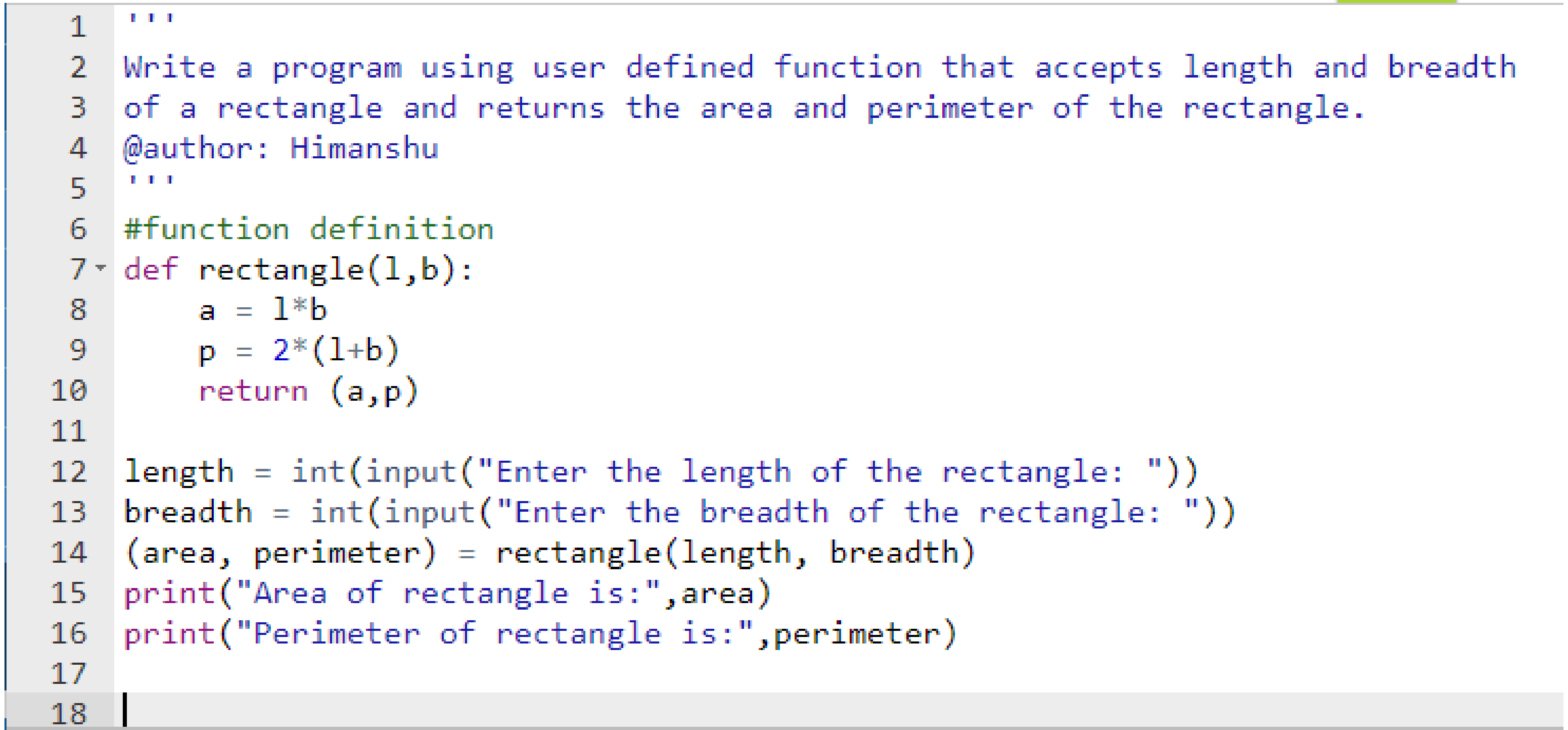
Flow of Execution

- Sometime, a function needs to return multiple values which may be returned using tuple.

  for e.g.)

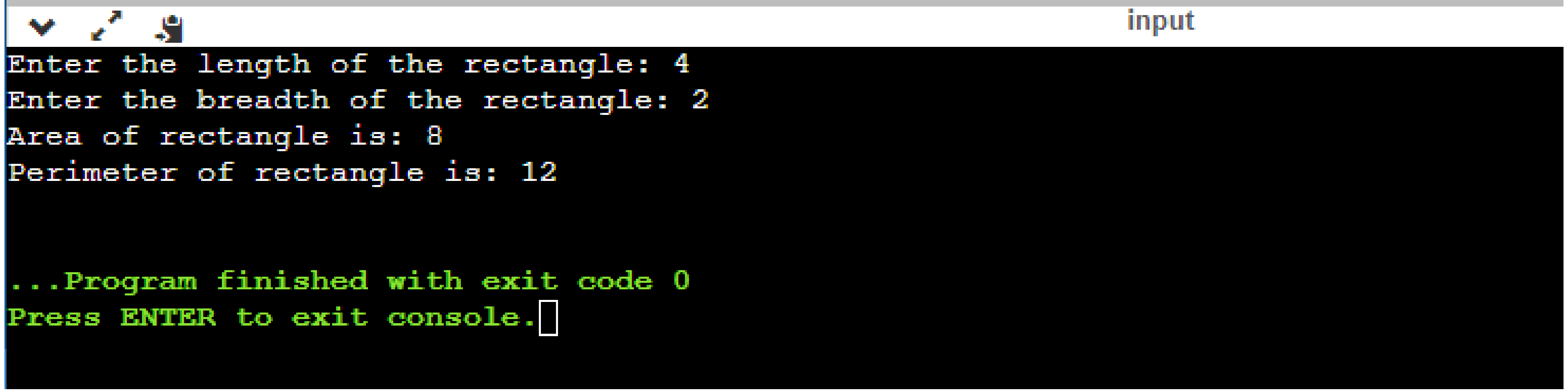
  Sample program: Write a program using user defined function that accepts length and breadth of a rectangle and returns the area and perimeter of the rectangle.

Code:



```python
'''
Write a program using user defined function that accepts length and breadth
of a rectangle and returns the area and perimeter of the rectangle.
@author: Himanshu
'''
#function definition
def rectangle(l,b):
    a = l*b
    p = 2*(l+b)
    return (a,p)

length = int(input("Enter the length of the rectangle: "))
breadth = int(input("Enter the breadth of the rectangle: "))
(area, perimeter) = rectangle(length, breadth)
print("Area of rectangle is:",area)
print("Perimeter of rectangle is:",perimeter)

```

Output:



```
Enter the length of the rectangle: 4
Enter the breadth of the rectangle: 2
Area of rectangle is: 8
Perimeter of rectangle is: 12


...Program finished with exit code 0
Press ENTER to exit console.
```

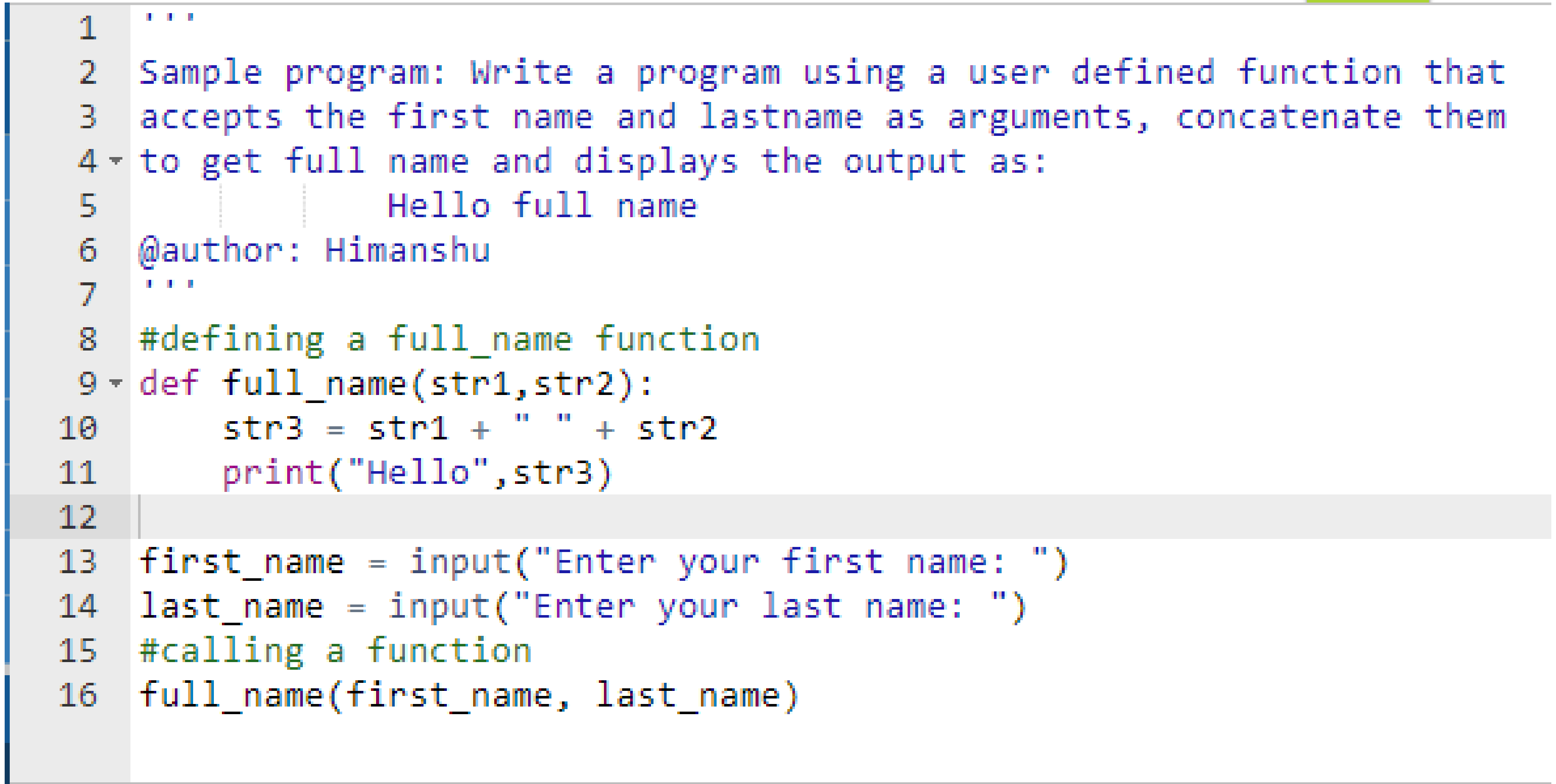
Program on user defined functions

- Sample program:

  Write a program using a user defined function that accepts the first name and lastname as arguments, concatenate them to get full name and displays the output as:

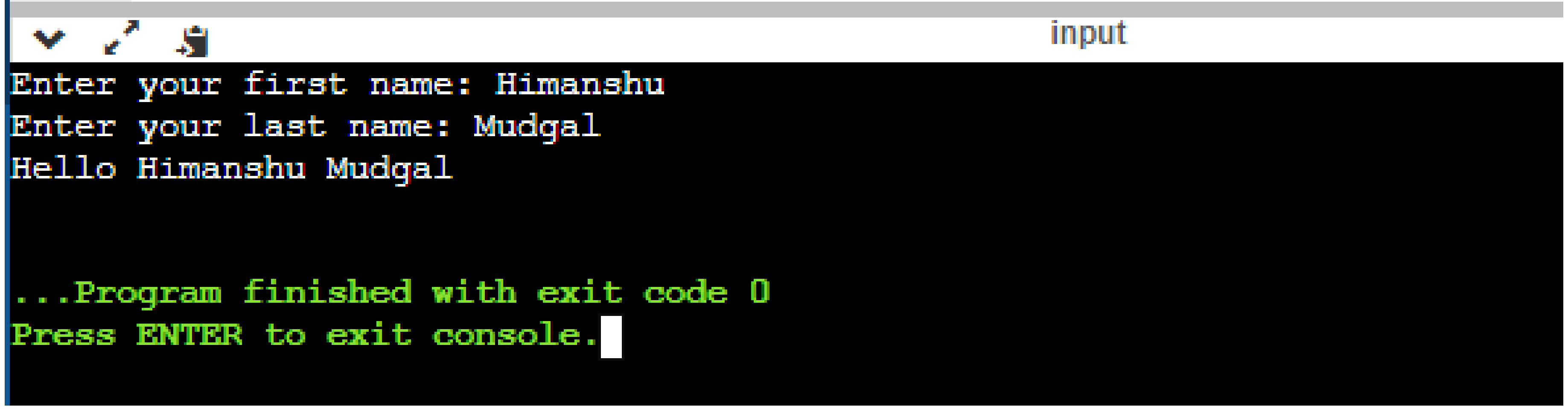
  ```
  Hello full name
  ```

Code:



```python
'''
Sample program: Write a program using a user defined function that
accepts the first name and lastname as arguments, concatenate them
to get full name and displays the output as:
            Hello full name
@author: Himanshu
'''
#defining a full_name function
def full_name(str1,str2):
    str3 = str1 + " " + str2
    print("Hello",str3)

first_name = input("Enter your first name: ")
last_name = input("Enter your last name: ")
#calling a function
full_name(first_name, last_name)
```

Output:



```
Enter your first name: Himanshu
Enter your last name: Mudgal
Hello Himanshu Mudgal


...Program finished with exit code 0
Press ENTER to exit console.
```

Scope of a Variable

- We have already seen that a variable defined inside a function cannot be accessed outside it.

- Every variable has a well-defined accessibility. The part of the program where a variable is accessible can be defined as the scope of that variable.

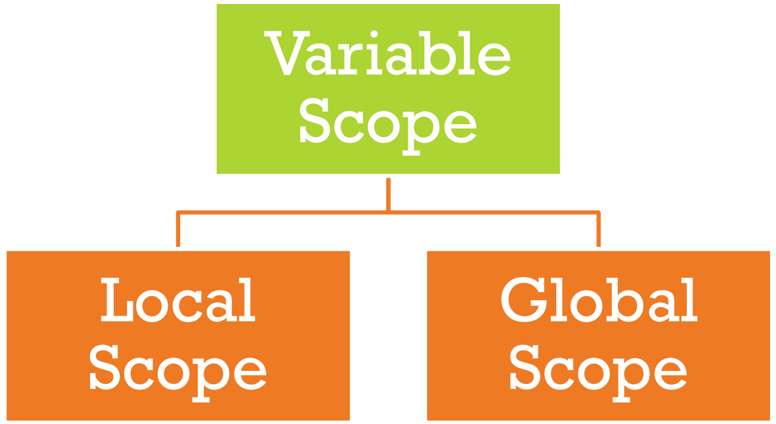
- A variable can have one of the following two scopes:

  a.) Global scope

  b.) Variable scope

Local and global Variable

- A variable that has global scope is known as a global variable and a variable that has a local scope is known as a local variable.



Local and global Variable

- A variable that is defined inside any function or a block is known as a local variable. It can be accessed only in the function or a block where it is defined. It exists only till the function executes.

- In Python, a variable that is defined outside any function or any block is known as a global variable. It can be accessed in any functions defined onwards.

Note:

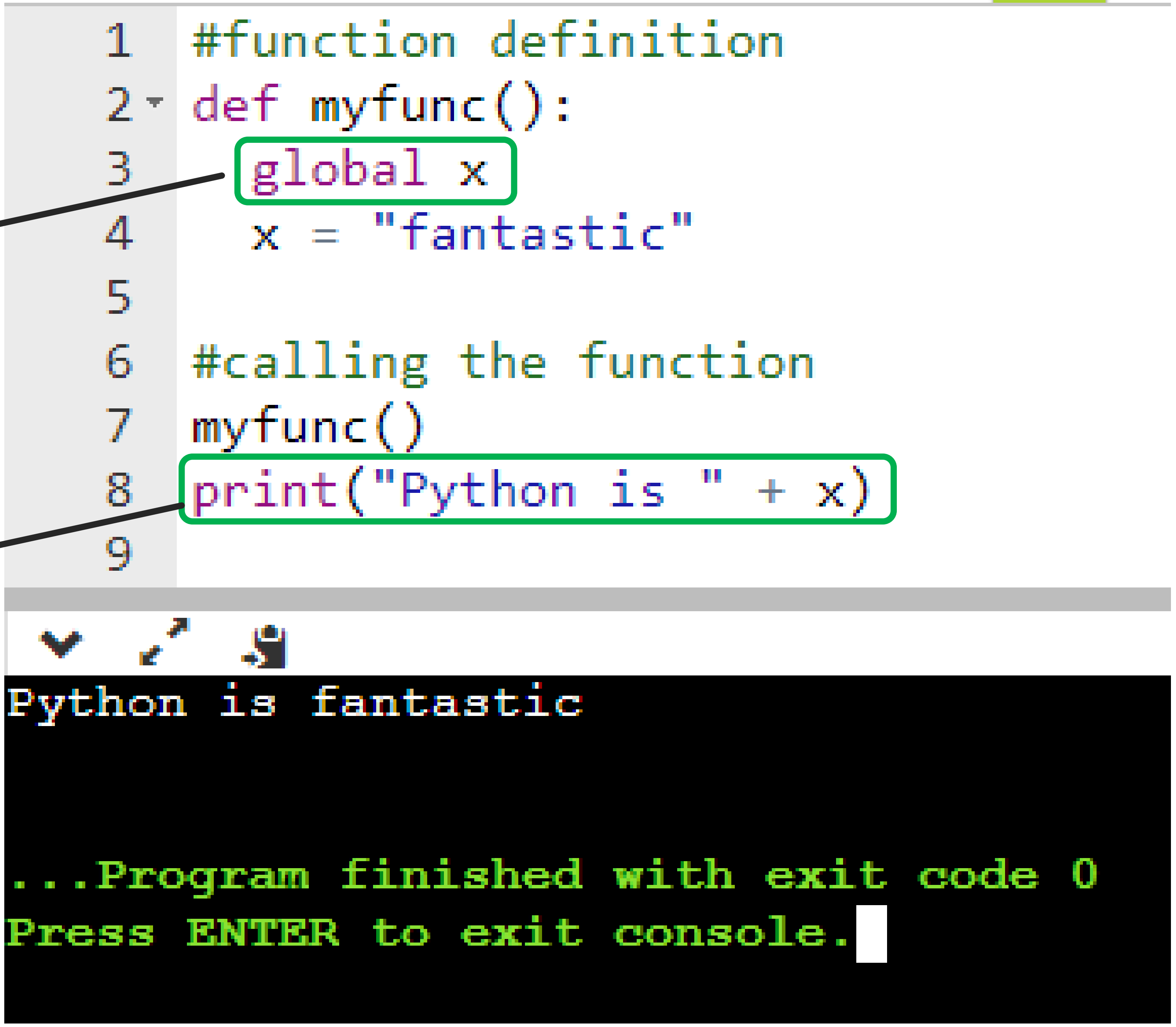
Any change made to the global variable will impact all the functions in the program where that variable can be accessed.

Local and global Variable

- Normally, when we create a variable inside a function, that variable is local, and can only be used inside that function.

- To create a global variable inside a function, we can use the global keyword creating a global variable using global keyword

accessing global variable x, outside the function



```python
#function definition
def myfunc():
    global x
    x = "fantastic"

#calling the function
myfunc()
print("Python is " + x)

```

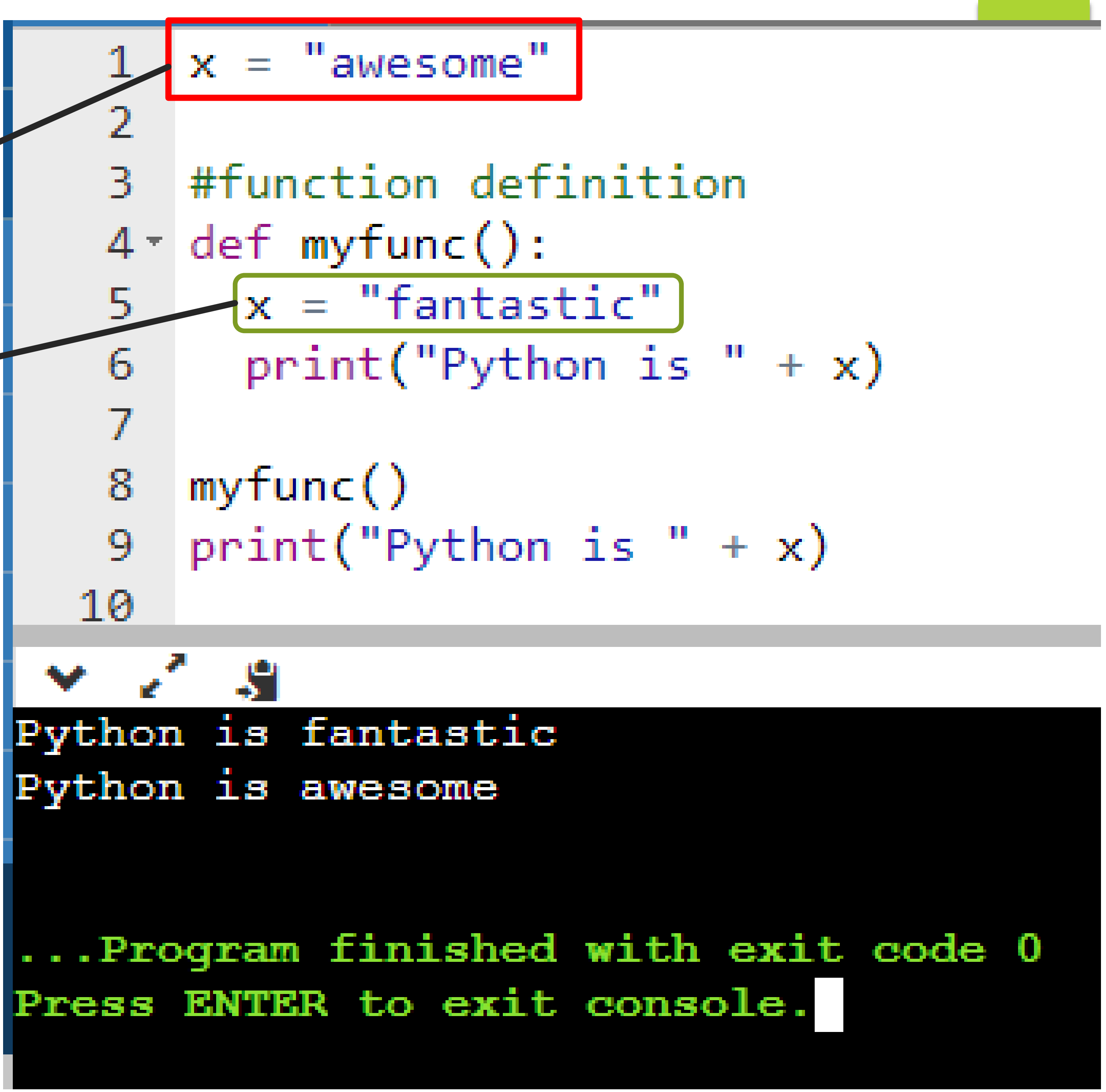
```
Python is fantastic


...Program finished with exit code 0
Press ENTER to exit console.
```

Local and global Variable

Note:

If a variable with the same name as the global variable is defined inside a function, then it is considered local to that function and hides the global variable global variable

local variable with same name as of global variable



```python
x = "awesome"

#function definition
def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()
print("Python is " + x)
```

```
Python is fantastic
Python is awesome


...Program finished with exit code 0
Press ENTER to exit console.
```

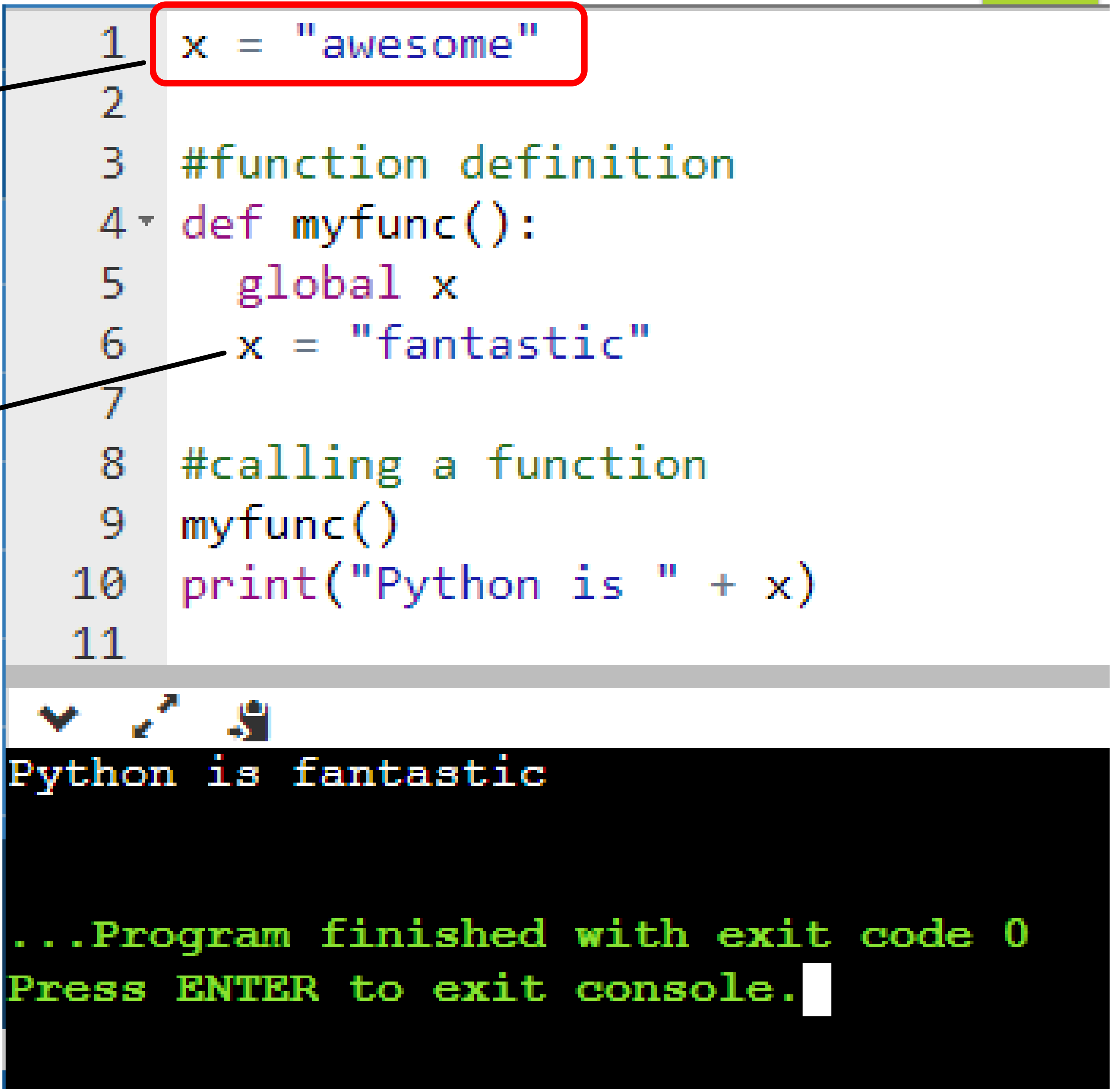
Local and global Variable

Note:

If the modified value of a global variable is to be used outside the function, then the keyword global should be prefixed to the variable name in the function.

defining a global variable

changing a global variable inside a function using global prefix



```python
x = "awesome"

#function definition
def myfunc():
  global x
  x = "fantastic"

#calling a function
myfunc()
print("Python is " + x)

```

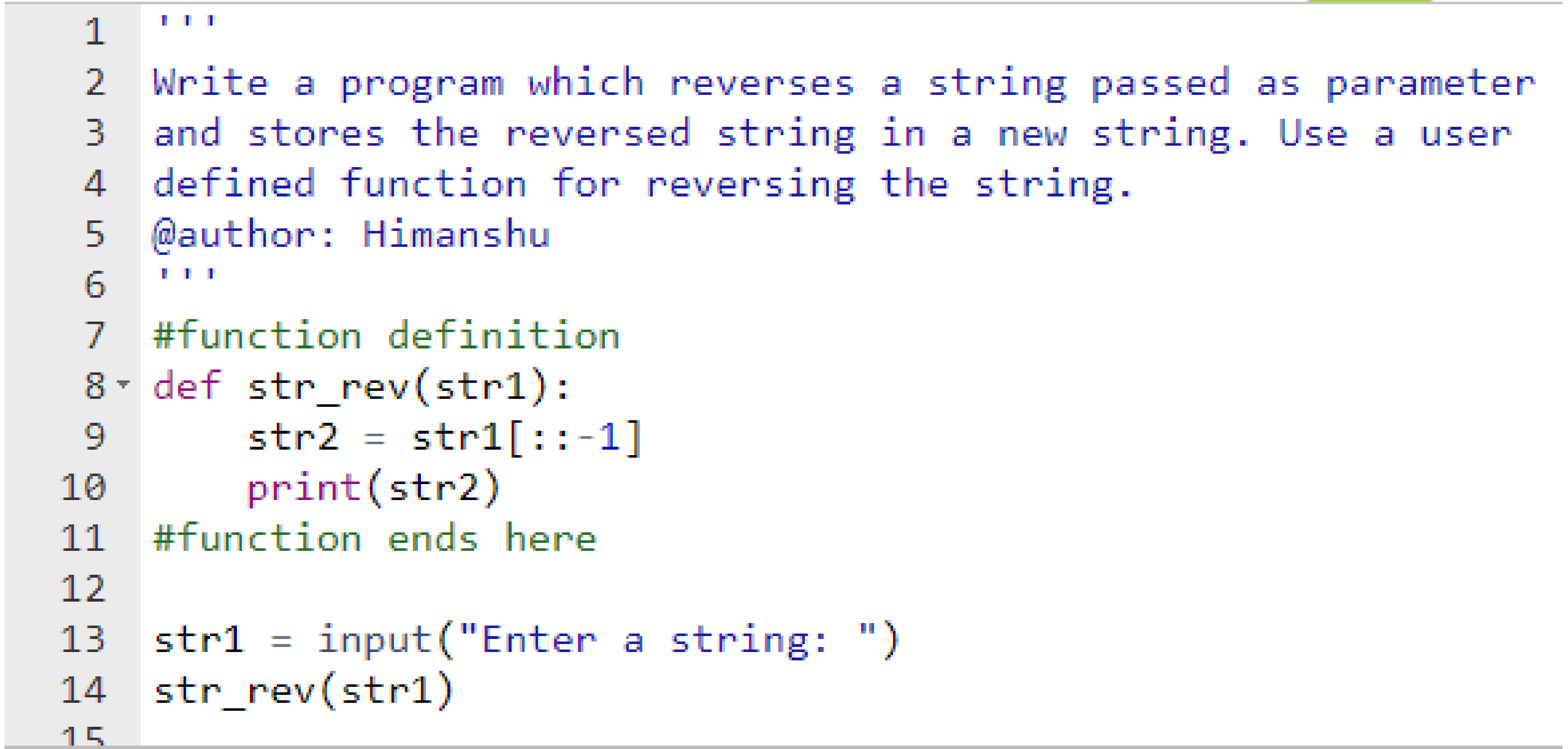
```
Python is fantastic


...Program finished with exit code 0
Press ENTER to exit console.
```

Program on user defined functions

- Sample program:

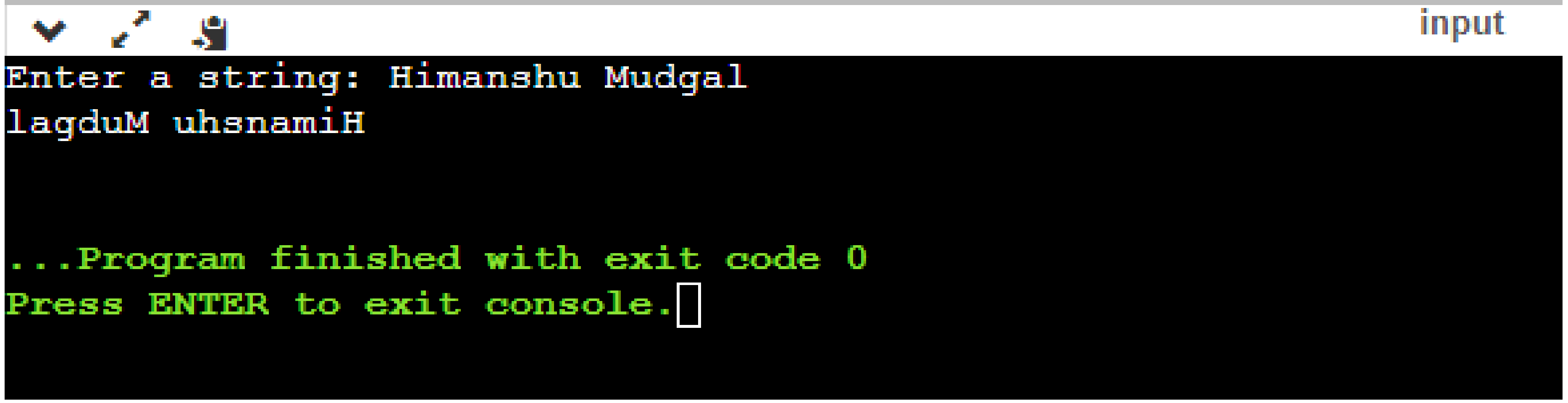
  Write a program which reverses a string passed as parameter and stores the reversed string in a new string. Use a user defined function for reversing the string.

Code:



```python
'''
Write a program which reverses a string passed as parameter
and stores the reversed string in a new string. Use a user
defined function for reversing the string.
@author: Himanshu
'''
#function definition
def str_rev(str1):
    str2 = str1[::-1]
    print(str2)
#function ends here

str1 = input("Enter a string: ")
str_rev(str1)
```

Output:



```
Enter a string: Himanshu Mudgal
lagduM uhsnamiH


...Program finished with exit code 0
Press ENTER to exit console.
```

Program on user defined functions

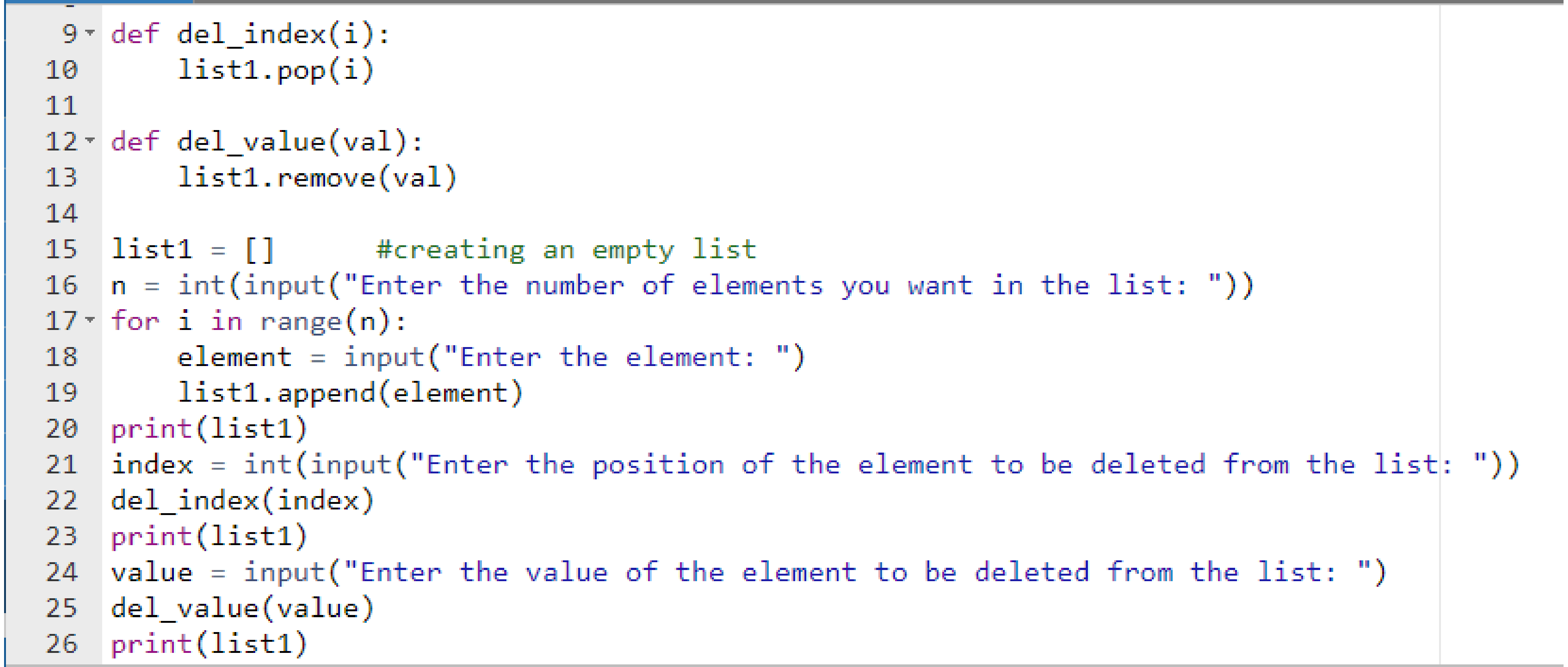
- Sample program:

  Write a program to read elements of a list.

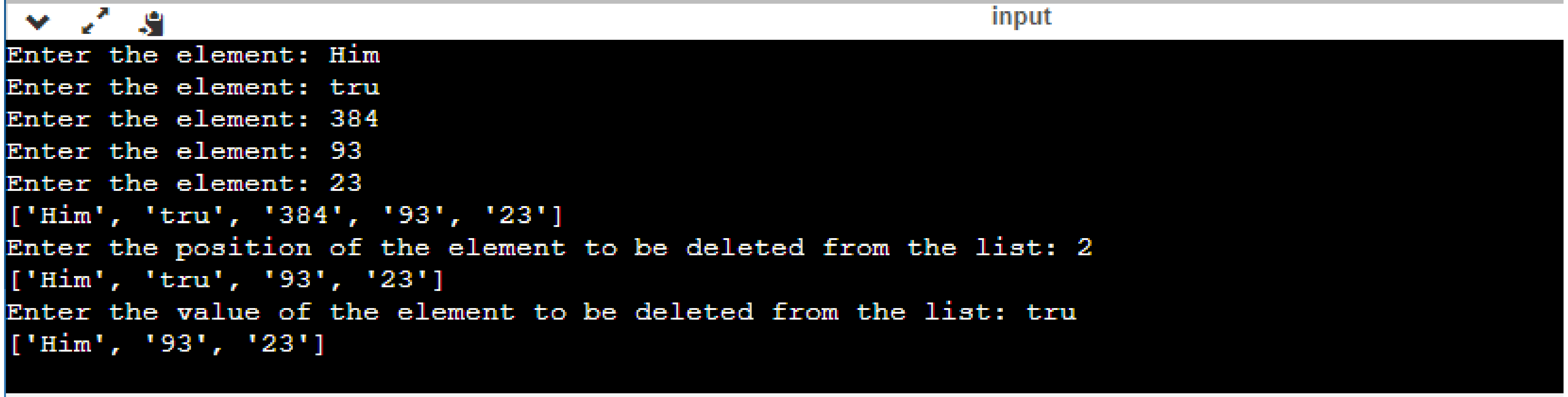
  a) The program should ask for the position of the element to be deleted from the list. Write a function to delete the element at the desired position in the list.

  b) The program should ask for the value of the element to be deleted from the list. Write a function to delete the element of this value from the list.

Code:


```python
def del_index(i):
    list1.pop(i)

def del_value(val):
    list1.remove(val)

list1 = []       #creating an empty list
n = int(input("Enter the number of elements you want in the list: "))
for i in range(n):
    element = input("Enter the element: ")
    list1.append(element)
print(list1)
index = int(input("Enter the position of the element to be deleted from the list: "))
del_index(index)
print(list1)
value = input("Enter the value of the element to be deleted from the list: ")
del_value(value)
print(list1)
```

Output:


```
Enter the element: Him
Enter the element: tru
Enter the element: 384
Enter the element: 93
Enter the element: 23
['Him', 'tru', '384', '93', '23']
Enter the position of the element to be deleted from the list: 2
['Him', 'tru', '93', '23']
Enter the value of the element to be deleted from the list: tru
['Him', '93', '23']
```

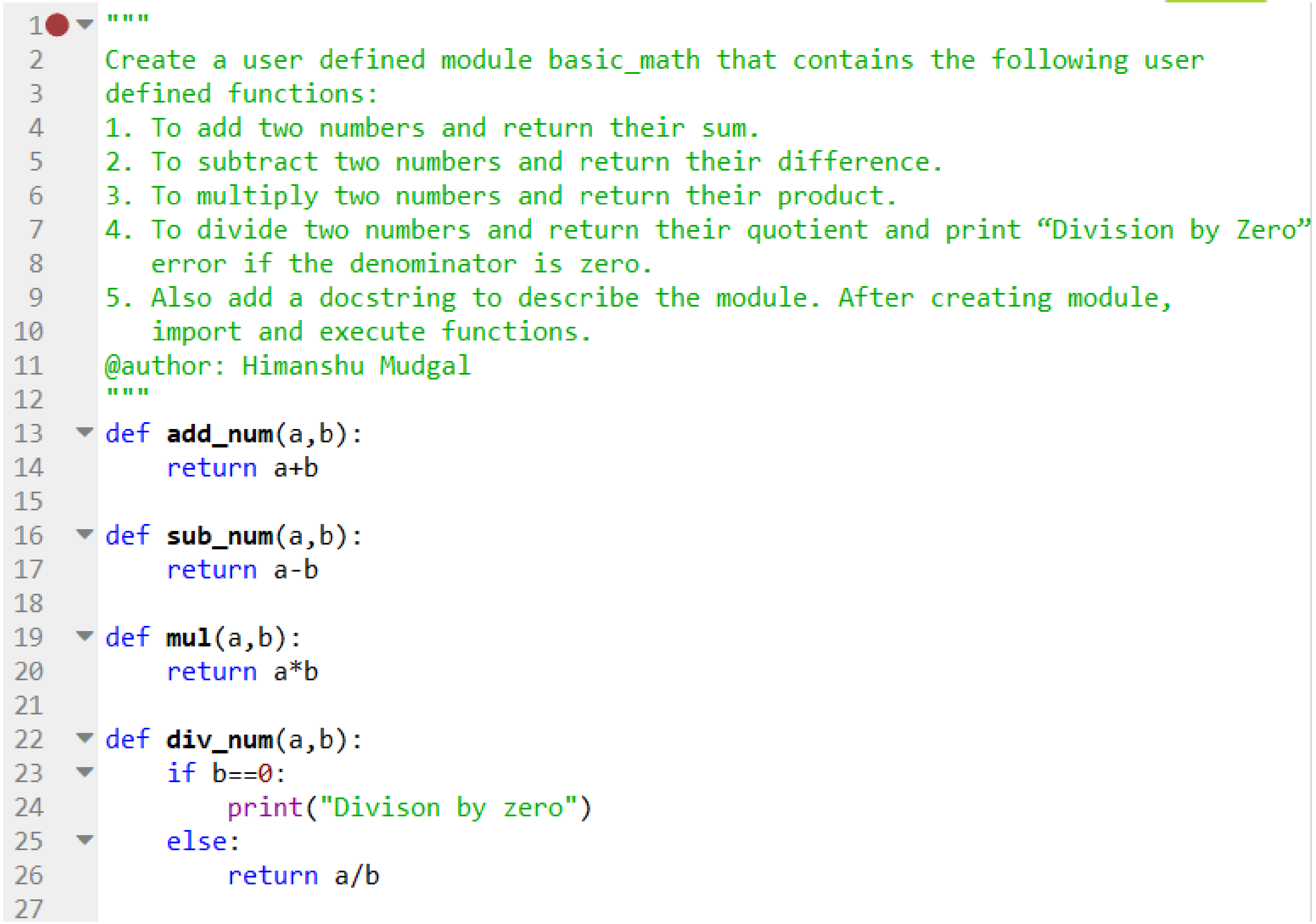
Program on user defined module

- Sample program:

  Create a user defined module basic_math that contains the following user defined functions:

  1. To add two numbers and return their sum.
  2. To subtract two numbers and return their difference.
  3. To multiply two numbers and return their product.
  4. To divide two numbers and return their quotient and print "Division by Zero" error if the denominator is zero.
  5. Also add a docstring to describe the module. After creating module, import and execute functions.

Code:



```
"""
Create a user defined module basic_math that contains the following user
defined functions:
1. To add two numbers and return their sum.
2. To subtract two numbers and return their difference.
3. To multiply two numbers and return their product.
4. To divide two numbers and return their quotient and print “Division by Zero”
   error if the denominator is zero.
5. Also add a docstring to describe the module. After creating module,
   import and execute functions.
@author: Himanshu Mudgal
"""
def add_num(a,b):
    return a+b

def sub_num(a,b):
    return a-b

def mul(a,b):
    return a*b

def div_num(a,b):
    if b==0:
        print("Divison by zero")
    else:
        return a/b

```

Summary

- Introduction to Functions
- Advantages of functions
- Types of functions:
  - Standard Library:
    - built-in
    - module
  - user defined functions

Summary

- Creating a user defined function
- Arguments and parameters
- Default parameter
- Function returning value(s)
- flow of execution
- scope of a variable: local, global
- programs on user defined functions

Assignment - 7

- Note down the summary points in your notebook (page 169)
- Q1-Q7, page 170

Programming Assignment - 7

- Q1-Q8 (NCERT, page 172, 173)
- Input a string having some digits. Write a function to return the sum of digits present in this string.