MSME

MICRO, SMALL & MEDIUM ENTERPRISES

सूक्ष्म, लघु एवं मध्यम उद्यम

OUR STRENGTH • हमारी शक्ति

सत्यमेव जयते

Ministry of MSME, Govt. of India

PPDC

AICTE TRAINING & LEARNING ACADEMY

ATAL

National Career Service

सही अवसर, सही समय

Right Opportunities, Right Time

5DAY DATA SCIENCE FACULT Y DEVELOPEMENT PROGRAM FROM 4TH MAY TILL 8TH MAY, 2020

ARTIFICIAL INTELLIGENCE FOR CAREER ENRICHMENT

(HANDS ON SESSIONS WITH INDUSTRY APPLICATION)

ENABLED THROUGH
ONLINE GOTO MEETING PLATFROM

HTTPS://GLOBAL.GOTOMEETING.COM/INSTALL/732061941

Semantic & Syntax in Python

Ms Ratna

Ratna bring in 20 years of diversified experience , worked for a decade in IT industry in Financial domain in technical and functional role

- She moved on to academia with the mission to create young mind focused on developing both professionally and spiritually , a combination which is a foundation for a future leader. She has dedicated a decade in Academia

- **Ratna has been associated with Ramco System, Microsoft, Infosys**
- **BE in Electronics & Communication (BIT , Mesra, Ranchi),**
- **MBA Finance (BIT , Mesra, Ranchi),**
- **Pursuing Phd in Financial Derivatives (BIT , Mesra, Ranchi),**

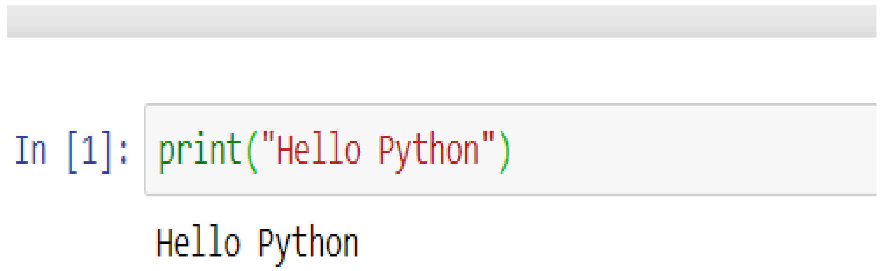
Features of Python

- Simple
- Easy to Learn
- Free and Open Source
- High-level Language
- Portable
- Interpreted
- Object Oriented
- Extensible
- Embeddable

Basic Syntax

Basic syntax of a python program is too simple than other languages.

Let's take an example, here the following program prints "Hello Python”

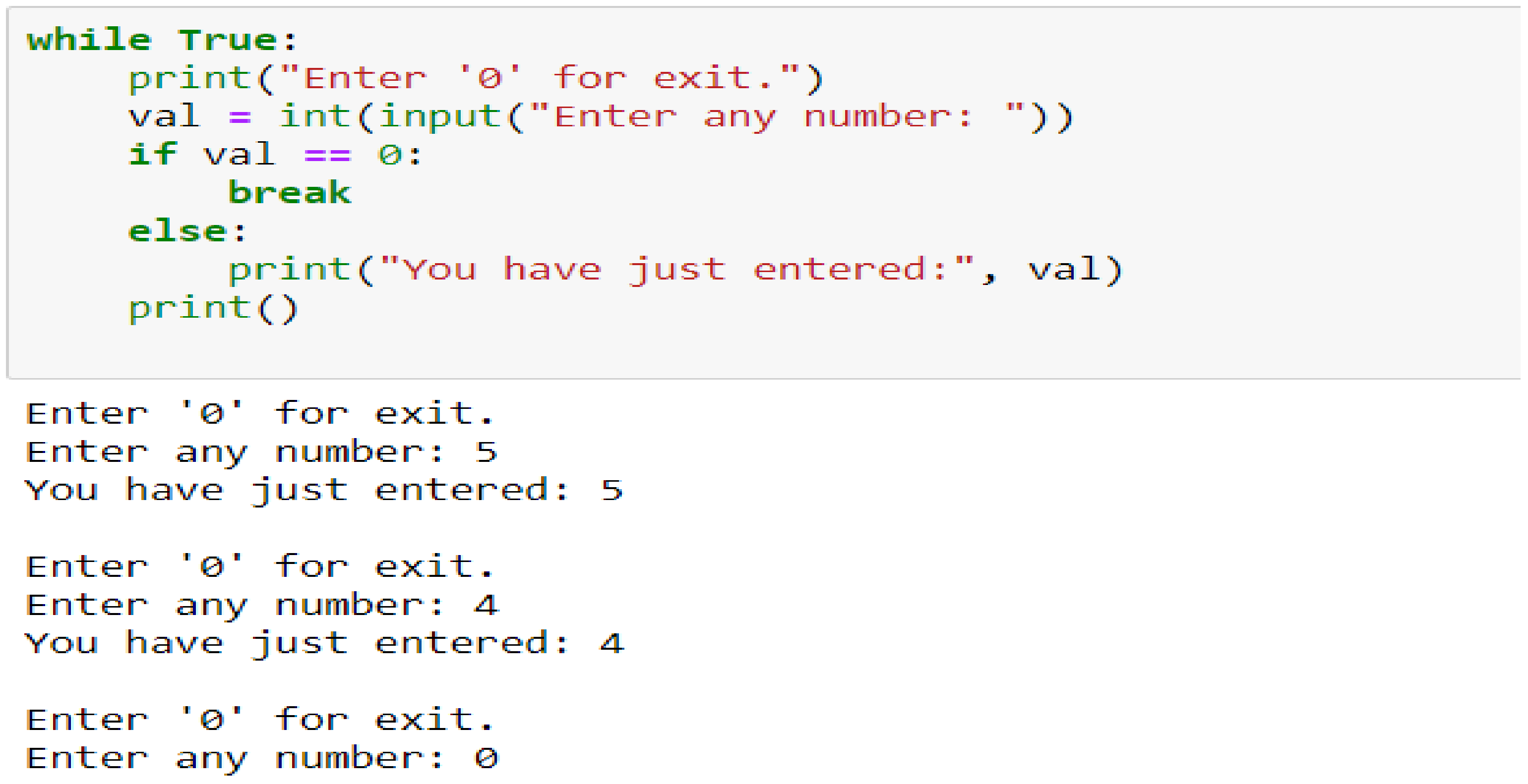


```
In [1]: print("Hello Python")

        Hello Python
```

Read User Input from Keyboard in Python

Get Integer Input from User

# Python Program - Get Integer Input from User



```
while True:
    print("Enter '0' for exit.")
    val = int(input("Enter any number: "))
    if val == 0:
        break
    else:
        print("You have just entered:", val)
    print()
```

```
Enter '0' for exit.
Enter any number: 5
You have just entered: 5

Enter '0' for exit.
Enter any number: 4
You have just entered: 4

Enter '0' for exit.
Enter any number: 0
```

Read User Input from Keyboard in Python

Get Integer Input from User

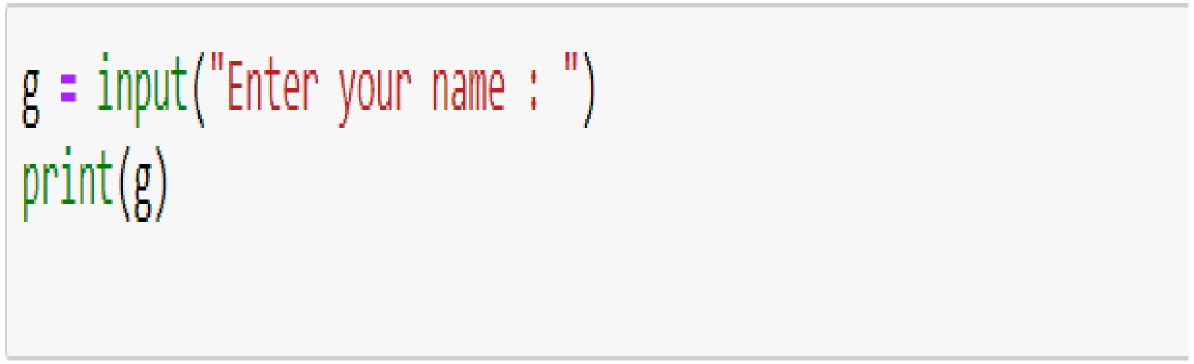
```
# Python Program - Get Integer Input from User
```

```
a = int(input("Enter an Integer: "))
b = int(input("Enter an Integer: "))
print("Sum of a and b:",a + b)
print("Multiplication of a and b:",a * b)
```

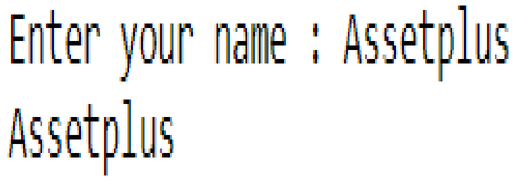
```
Enter an Integer: 6 Enter an Integer: 7 Sum of a and b: 13
Multiplication of a and b: 42
```

Get String Input from User

# Python Program - Get String Input from User



```python
g = input("Enter your name : ")
print(g)
```

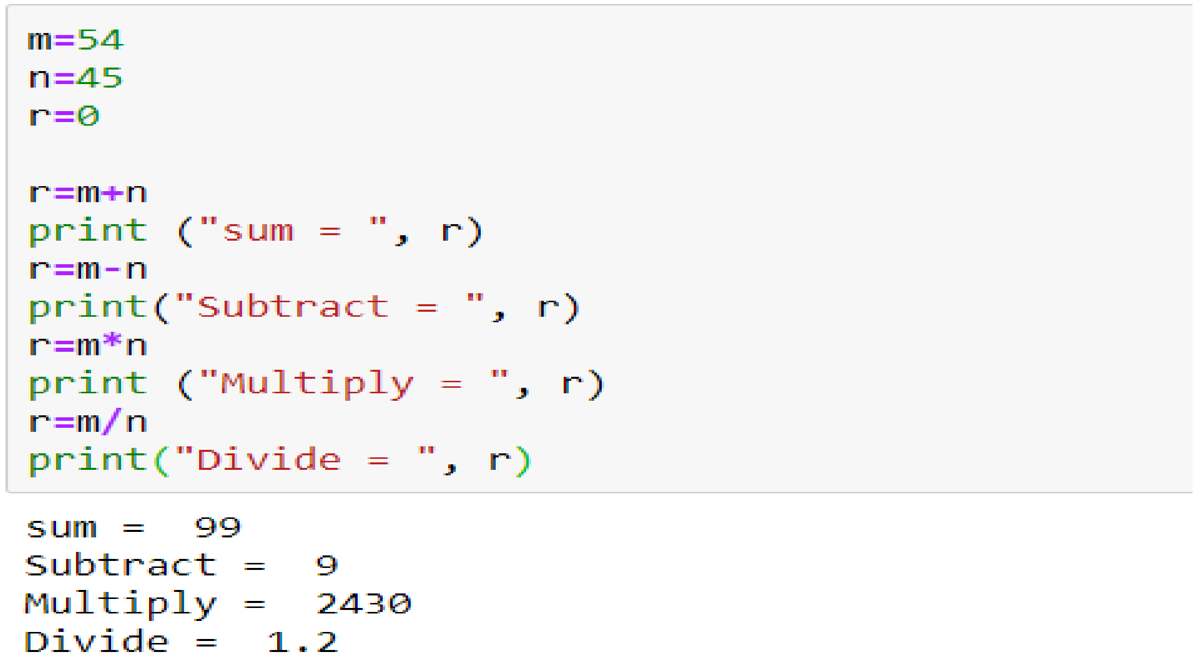


```
Enter your name : Assetplus
Assetplus
```

Variables in Python

❖ Variables in Python, are the reserved memory locations to store the values in a Python program.

Python Variables - Example Program



```
m=54
n=45
r=0

r=m+n
print ("sum = ", r)
r=m-n
print("Subtract = ", r)
r=m*n
print ("Multiply = ", r)
r=m/n
print("Divide = ", r)
```

```
sum =  99
Subtract =  9
Multiply =  2430
Divide =  1.2
```

Data types in Python

Everything in Python, is simply an object and every object has an identity, a type and a value. There are following five standard data types available in Python programming:

I. Numbers type
II. Strings type
III. List type
IV. Tuple type
V. Dictionary type

Number type

- Number data types store numeric values.
- Number objects are created when you assign a value to them.

For example
```
var1 = 1
var2 = 10
```

- You can delete a single object or multiple objects by using the del statement.

For example
```
del var
del var1, var2
```

Python supports four different numerical types

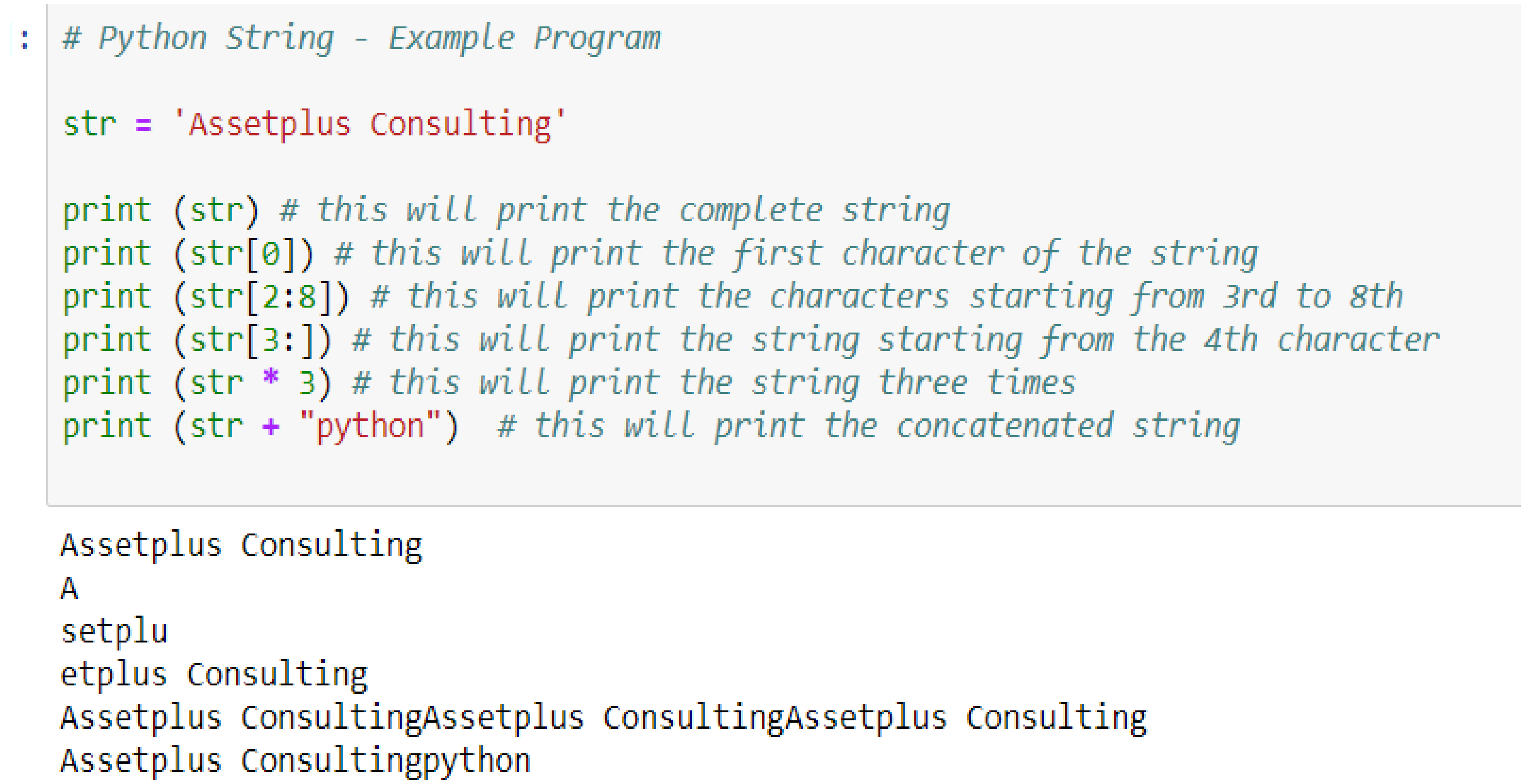
- ✓ int (signed integers)
- ✓ long (long integers, they can also be represented in octal and hexadecimal)
- ✓ float (floating point real values)
- ✓ complex (complex numbers)

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

String Data Type

- Strings in Python, are contiguous set of characters between quotation marks.

String Examples

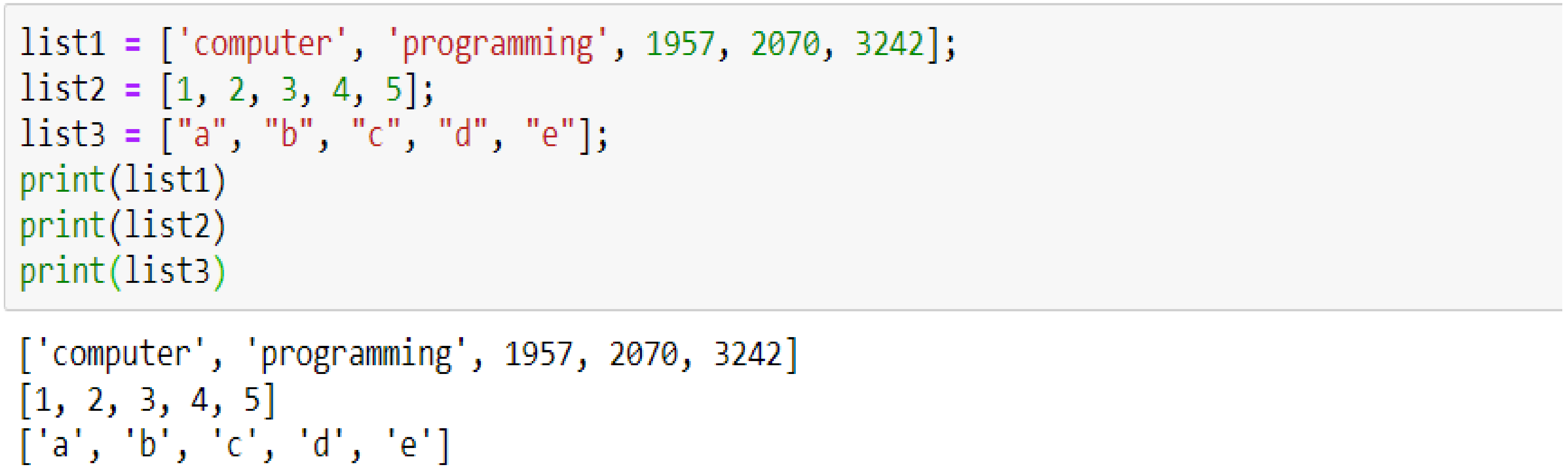


```
# Python String - Example Program

str = 'Assetplus Consulting'

print (str) # this will print the complete string
print (str[0]) # this will print the first character of the string
print (str[2:8]) # this will print the characters starting from 3rd to 8th
print (str[3:]) # this will print the string starting from the 4th character
print (str * 3) # this will print the string three times
print (str + "python")  # this will print the concatenated string
```

```
Assetplus Consulting
A
setplu
etplus Consulting
Assetplus ConsultingAssetplus ConsultingAssetplus Consulting
Assetplus Consultingpython
```

List Data type

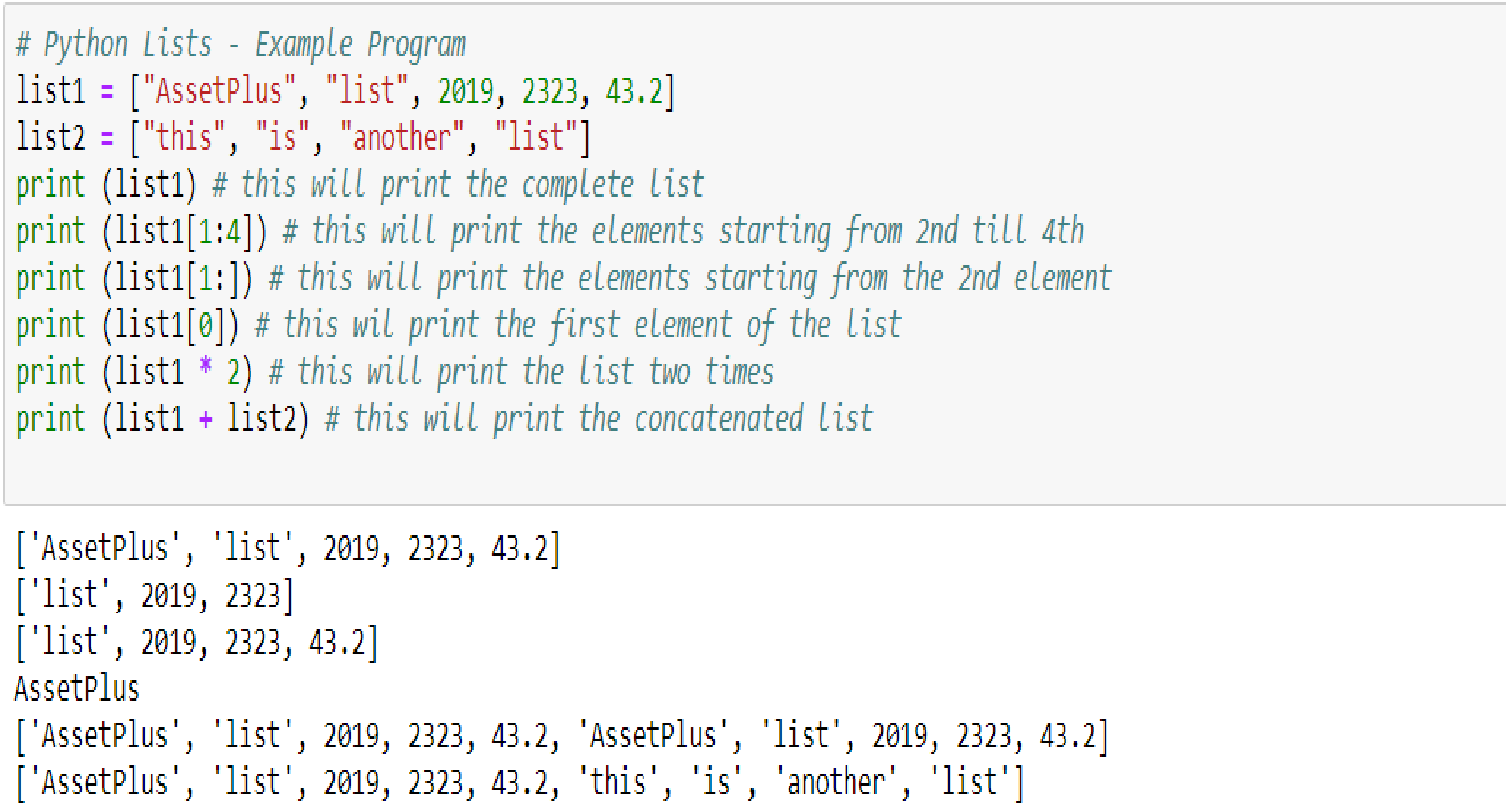
- ➢ A list in Python, contains items separated by commas and enclosed within square brackets.
- ➢ A list basically contains items separated by commas and enclosed within the square brackets [ ]. Items in the list needn't be of the same type.



```python
list1 = ['computer', 'programming', 1957, 2070, 3242];
list2 = [1, 2, 3, 4, 5];
list3 = ["a", "b", "c", "d", "e"];
print(list1)
print(list2)
print(list3)
```

```
['computer', 'programming', 1957, 2070, 3242]
[1, 2, 3, 4, 5]
['a', 'b', 'c', 'd', 'e']
```

Python List Example

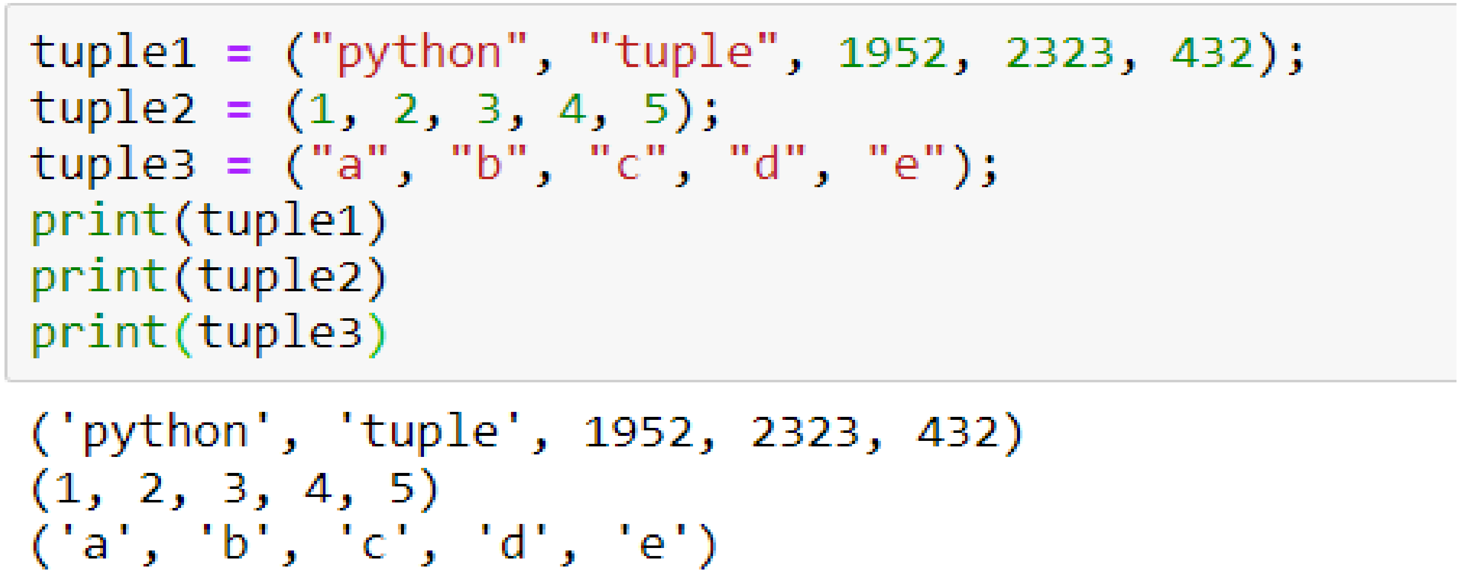


```python
# Python Lists - Example Program
list1 = ["AssetPlus", "list", 2019, 2323, 43.2]
list2 = ["this", "is", "another", "list"]
print (list1) # this will print the complete list
print (list1[1:4]) # this will print the elements starting from 2nd till 4th
print (list1[1:]) # this will print the elements starting from the 2nd element
print (list1[0]) # this wil print the first element of the list
print (list1 * 2) # this will print the list two times
print (list1 + list2) # this will print the concatenated list
```

```
['AssetPlus', 'list', 2019, 2323, 43.2]
['list', 2019, 2323]
['list', 2019, 2323, 43.2]
AssetPlus
['AssetPlus', 'list', 2019, 2323, 43.2, 'AssetPlus', 'list', 2019, 2323, 43.2]
['AssetPlus', 'list', 2019, 2323, 43.2, 'this', 'is', 'another', 'list']
```

Tuples Data Type

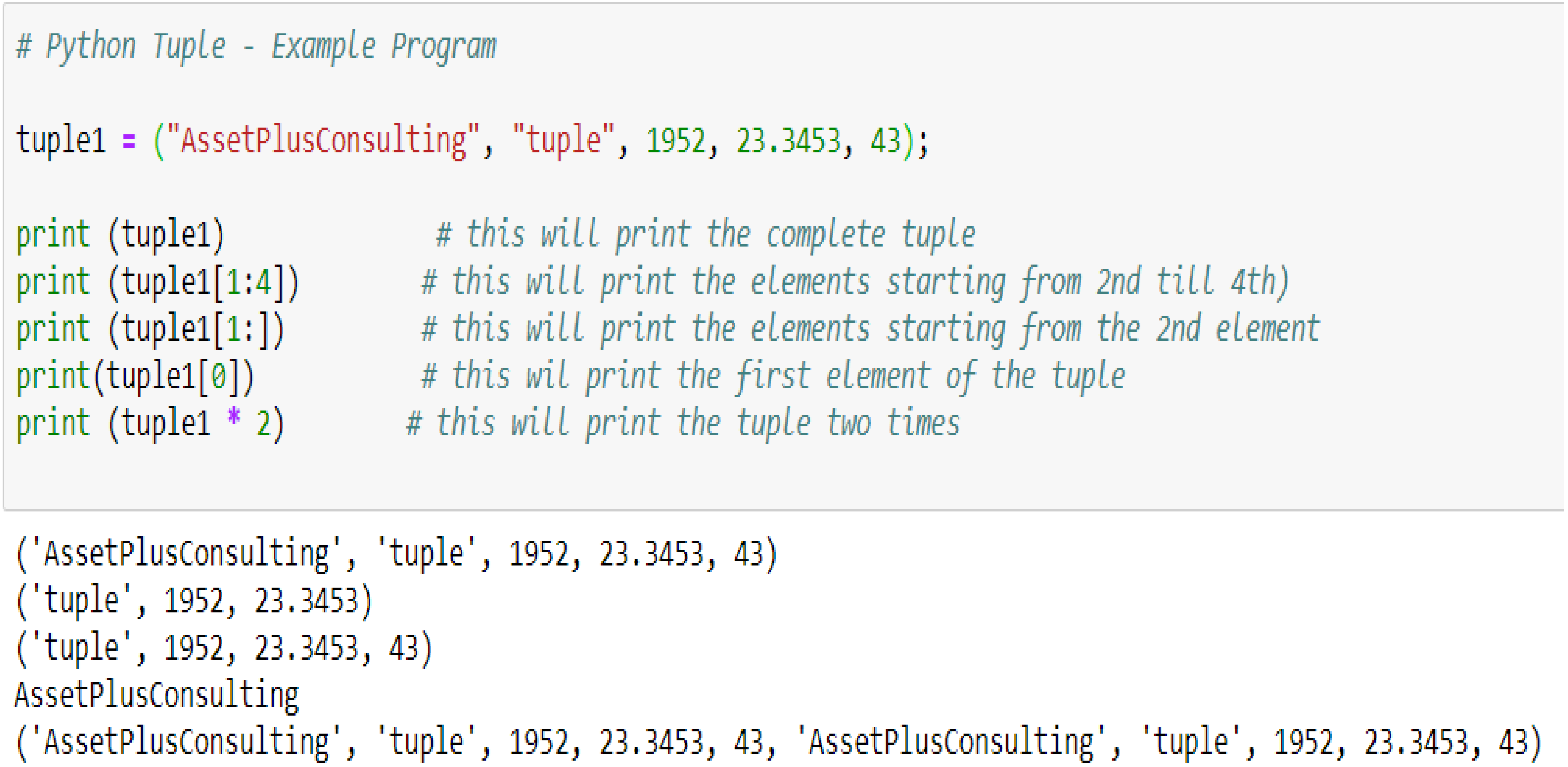
- ✓ A tuple is another sequence data type that is similar to the list.
- ✓ A tuple consists of a number of values separated by commas.
- ✓ The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) Tuples are enclosed in parentheses (( )) and cannot be updated.



```python
tuple1 = ("python", "tuple", 1952, 2323, 432);
tuple2 = (1, 2, 3, 4, 5);
tuple3 = ("a", "b", "c", "d", "e");
print(tuple1)
print(tuple2)
print(tuple3)
```

```
('python', 'tuple', 1952, 2323, 432)
(1, 2, 3, 4, 5)
('a', 'b', 'c', 'd', 'e')
```

Tuple Example



```python
# Python Tuple - Example Program

tuple1 = ("AssetPlusConsulting", "tuple", 1952, 23.3453, 43);

print (tuple1)              # this will print the complete tuple
print (tuple1[1:4])        # this will print the elements starting from 2nd till 4th)
print (tuple1[1:])         # this will print the elements starting from the 2nd element
print(tuple1[0])           # this wil print the first element of the tuple
print (tuple1 * 2)        # this will print the tuple two times
```

```
('AssetPlusConsulting', 'tuple', 1952, 23.3453, 43)
('tuple', 1952, 23.3453)
('tuple', 1952, 23.3453, 43)
AssetPlusConsulting
('AssetPlusConsulting', 'tuple', 1952, 23.3453, 43, 'AssetPlusConsulting', 'tuple', 1952, 23.3453, 43)
```

Difference Between List and Tuple in Python:

SR.NO.	LIST	TUPLE

1	Lists are mutable
	a = ["apples", "bananas", "oranges"]
	Let’s change “apples” to “berries”.
	a[0] = "berries“
	Print(a)
	['berries', 'bananas', 'oranges']
		Tuple are immutable
		a = ["apples", "bananas", "oranges"]
		Let’s change “apples” to “berries”.
		a[0] = "berries“
		Print(a)
		Traceback (most recent call last): File "", line 1, in Type Error: 'tuple' object does not support item assignment

2	The list is better for performing operations, such as insertion and deletion.	Tuple data type is appropriate for accessing the elements

SR.NO.	LIST	TUPLE

4	Lists consume more memory
	a = [1]
	b = [1]
	id(a)
	o/p:4305324416
	id(b)
	o/p:4305324416
	Each values in the list takes separate memory
		Tuple consume less memory as compared to the list
		a = (1, 2)
		b = (1, 2)
		id(a)
		o/p:4364806856
		id(b)
		o/p:4364806920
		But not in lists

5	Lists have several built-in methods	Tuple does no have much built-in methods.

List Built_In_Methods

Method	Description
List append() list.append(item)	Add Single Element to The List
List extend() list1.extend(list2)	Add Elements of a List to Another List
List insert() list.insert(index, element) index - position where an element needs to be inserted element - this is the element to be inserted in the list	Inserts Element to The List
List remove() list.remove(element)	Removes Element from the List
List index() list.index(element) element - element that is to be searched.	returns smallest index of element in list
List count() list.count(element)	returns occurrences of element in a list

Method	Description
List pop() list.pop(index) The pop() method removes the item at the given index from the list.	Removes Element at Given Index
List reverse() list.reverse()	Reverses a List

List copy()
list = ['cat', 0, 6.7]
new_list = list.copy()
	Returns Shallow Copy of a List

List clear() list.clear()	Removes all Items from the List

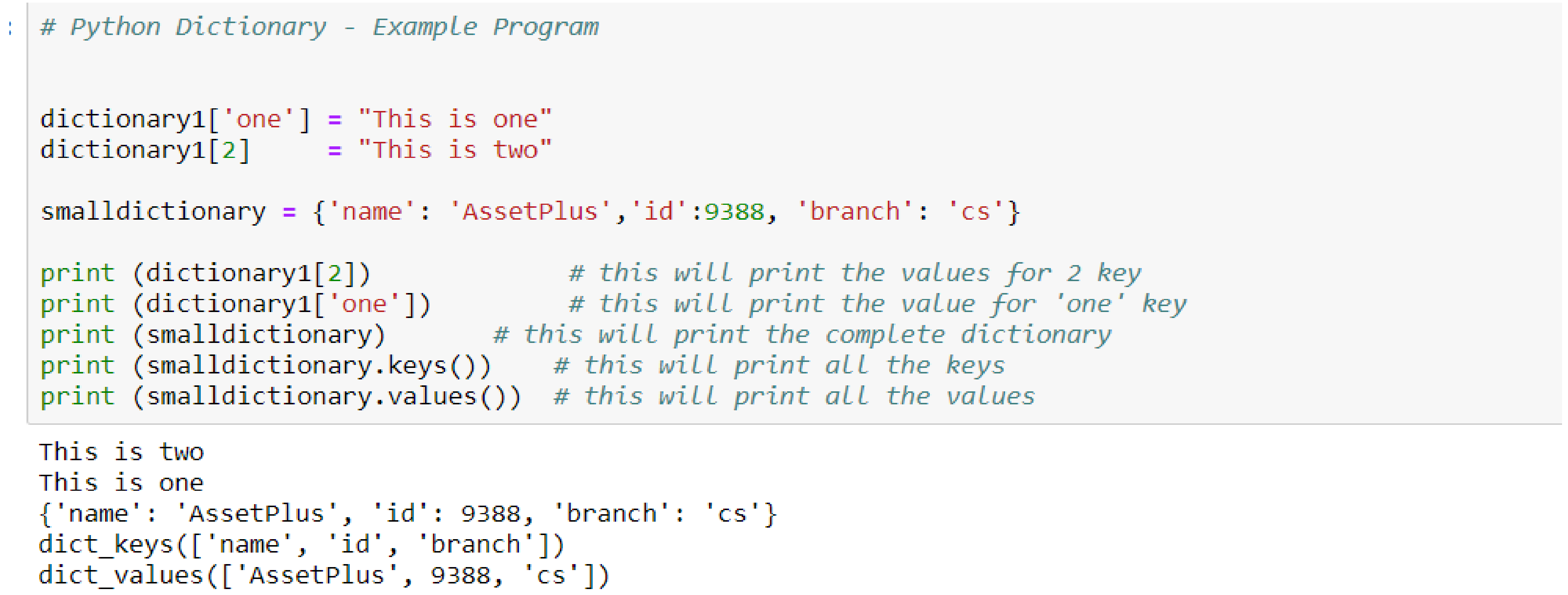
Tuples Built_In_Methods

Method	Description
Tuple count() tuple.count(element)	returns occurrences of element in a tuple
Tuple index() tuple.index(element) The index method returns the position/index of the given element in the tuple.	returns smallest index of element in tuple

Dictionary Data type

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

Dictionary Example

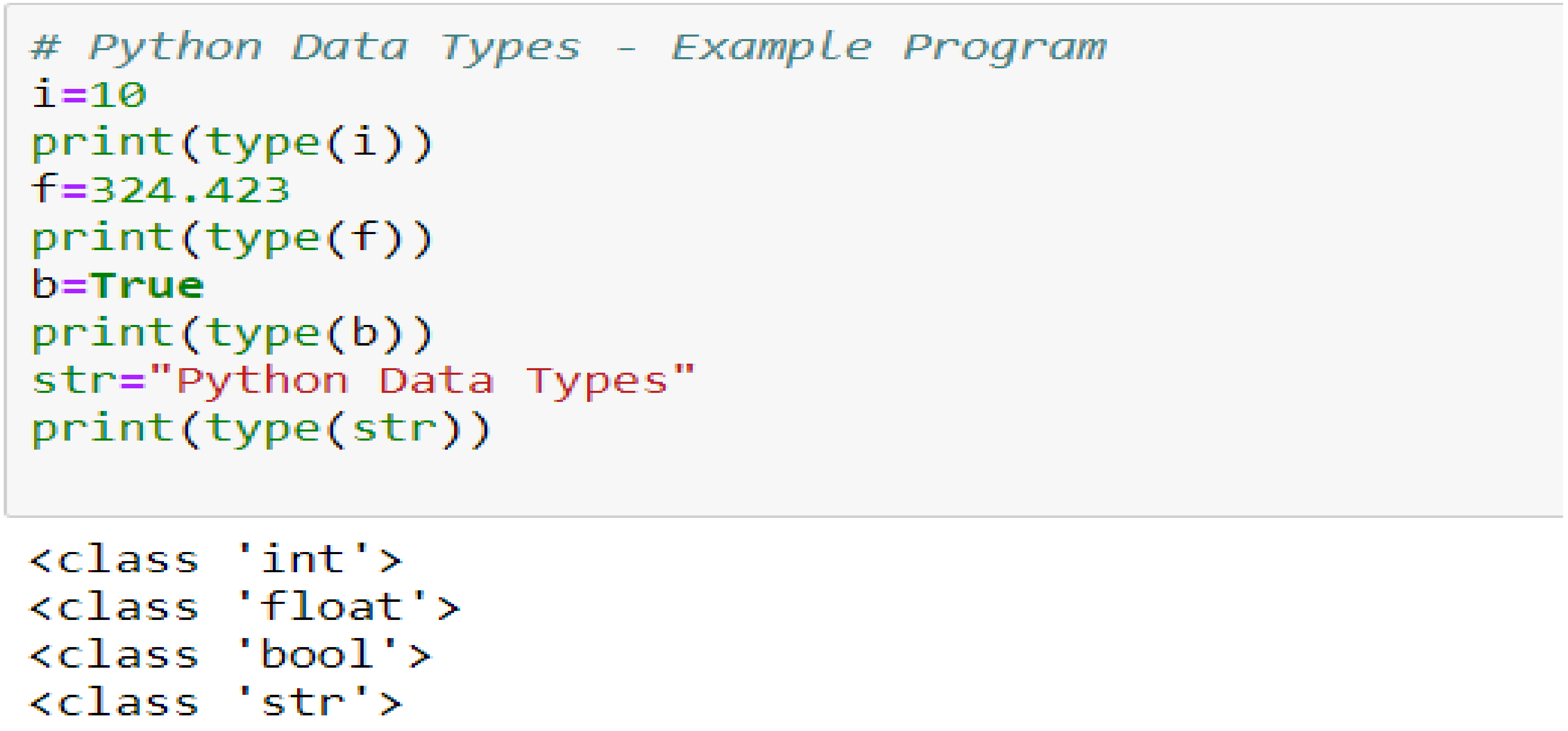


```
# Python Dictionary - Example Program


dictionary1['one'] = "This is one"
dictionary1[2]     = "This is two"

smalldictionary = {'name': 'AssetPlus','id':9388, 'branch': 'cs'}

print (dictionary1[2])            # this will print the values for 2 key
print (dictionary1['one'])        # this will print the value for 'one' key
print (smalldictionary)       # this will print the complete dictionary
print (smalldictionary.keys())    # this will print all the keys
print (smalldictionary.values())  # this will print all the values
```

```
This is two
This is one
{'name': 'AssetPlus', 'id': 9388, 'branch': 'cs'}
dict_keys(['name', 'id', 'branch'])
dict_values(['AssetPlus', 9388, 'cs'])
```

Determine Variable's Type in Python

- You can use the function type() available in Python, to determine the type of variable in Python.



```python
# Python Data Types - Example Program
i=10
print(type(i))
f=324.423
print(type(f))
b=True
print(type(b))
str="Python Data Types"
print(type(str))
```

```
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'str'>
```

Operators in Python

- Operators in Python, are used to perform mathematical and logical operations.

❖ Arithmetic Operators

❖ Logical Operators

❖ Comparison (Relational) Operators

❖ Assignment Operators

❖ Bitwise Operators

❖ Membership Operators

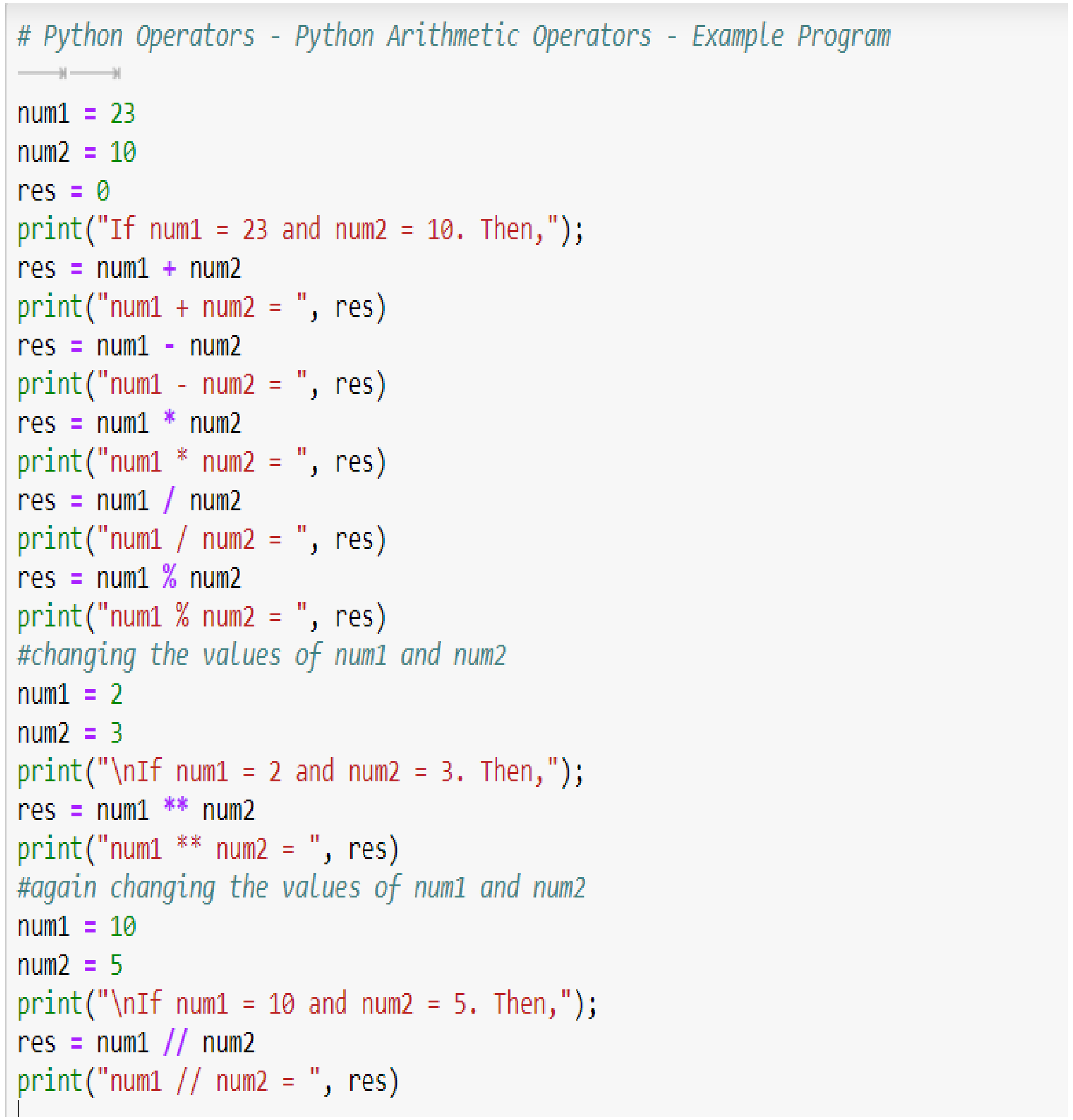
❖ Identity Operators

Arithmetic operators in Python

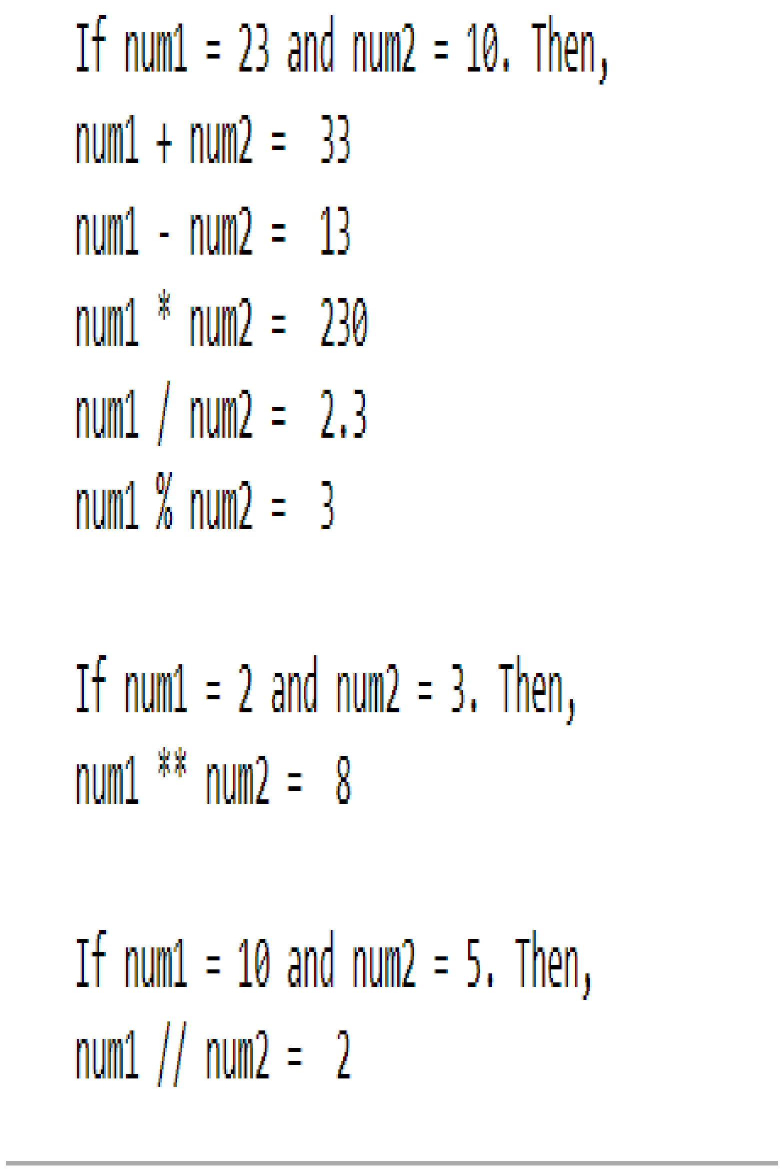
Assume variable a holds 10 and variable b holds 20

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
*Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero.	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

Python Arithmetic Operators



```python
# Python Operators - Python Arithmetic Operators - Example Program

num1 = 23
num2 = 10
res = 0
print("If num1 = 23 and num2 = 10. Then,");
res = num1 + num2
print("num1 + num2 = ", res)
res = num1 - num2
print("num1 - num2 = ", res)
res = num1 * num2
print("num1 * num2 = ", res)
res = num1 / num2
print("num1 / num2 = ", res)
res = num1 % num2
print("num1 % num2 = ", res)
#changing the values of num1 and num2
num1 = 2
num2 = 3
print("\nIf num1 = 2 and num2 = 3. Then,");
res = num1 ** num2
print("num1 ** num2 = ", res)
#again changing the values of num1 and num2
num1 = 10
num2 = 5
print("\nIf num1 = 10 and num2 = 5. Then,");
res = num1 // num2
print("num1 // num2 = ", res)
```

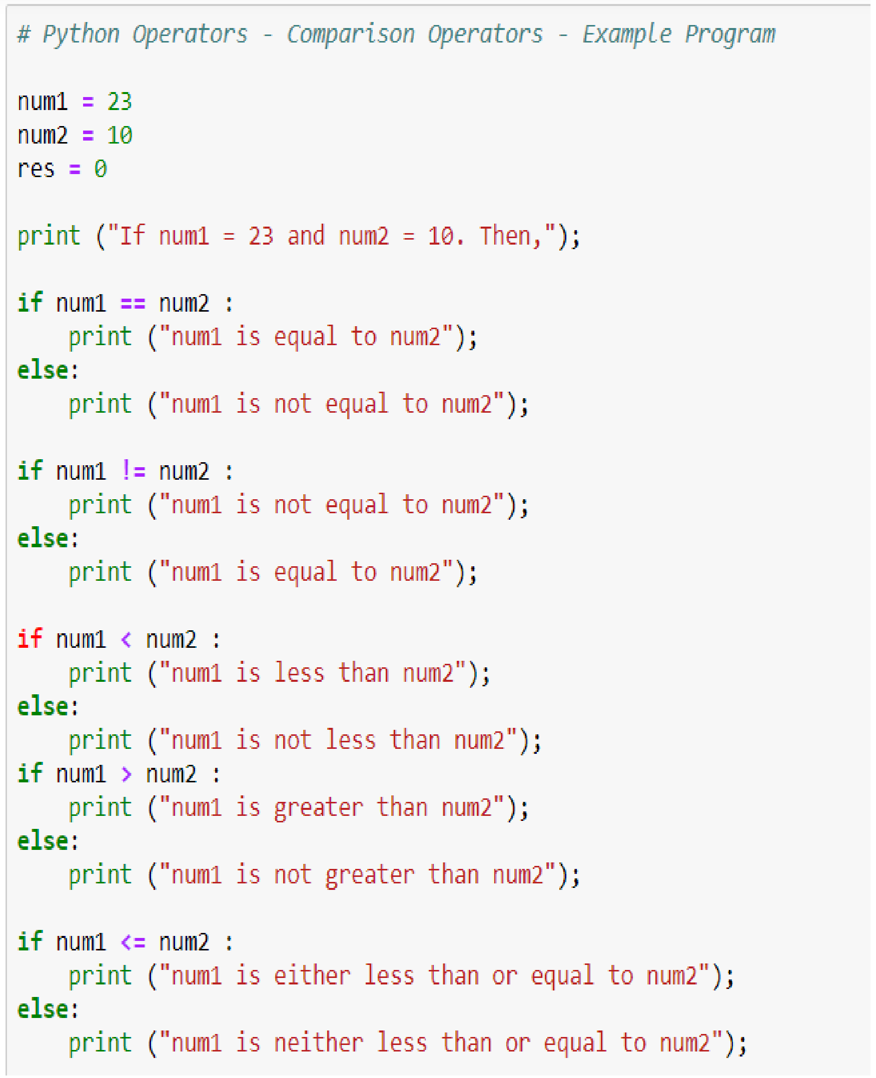


```
If num1 = 23 and num2 = 10. Then,
num1 + num2 =  33
num1 - num2 =  13
num1 * num2 =  230
num1 / num2 =  2.3
num1 % num2 =  3

If num1 = 2 and num2 = 3. Then,
num1 ** num2 =  8

If num1 = 10 and num2 = 5. Then,
num1 // num2 =  2
```

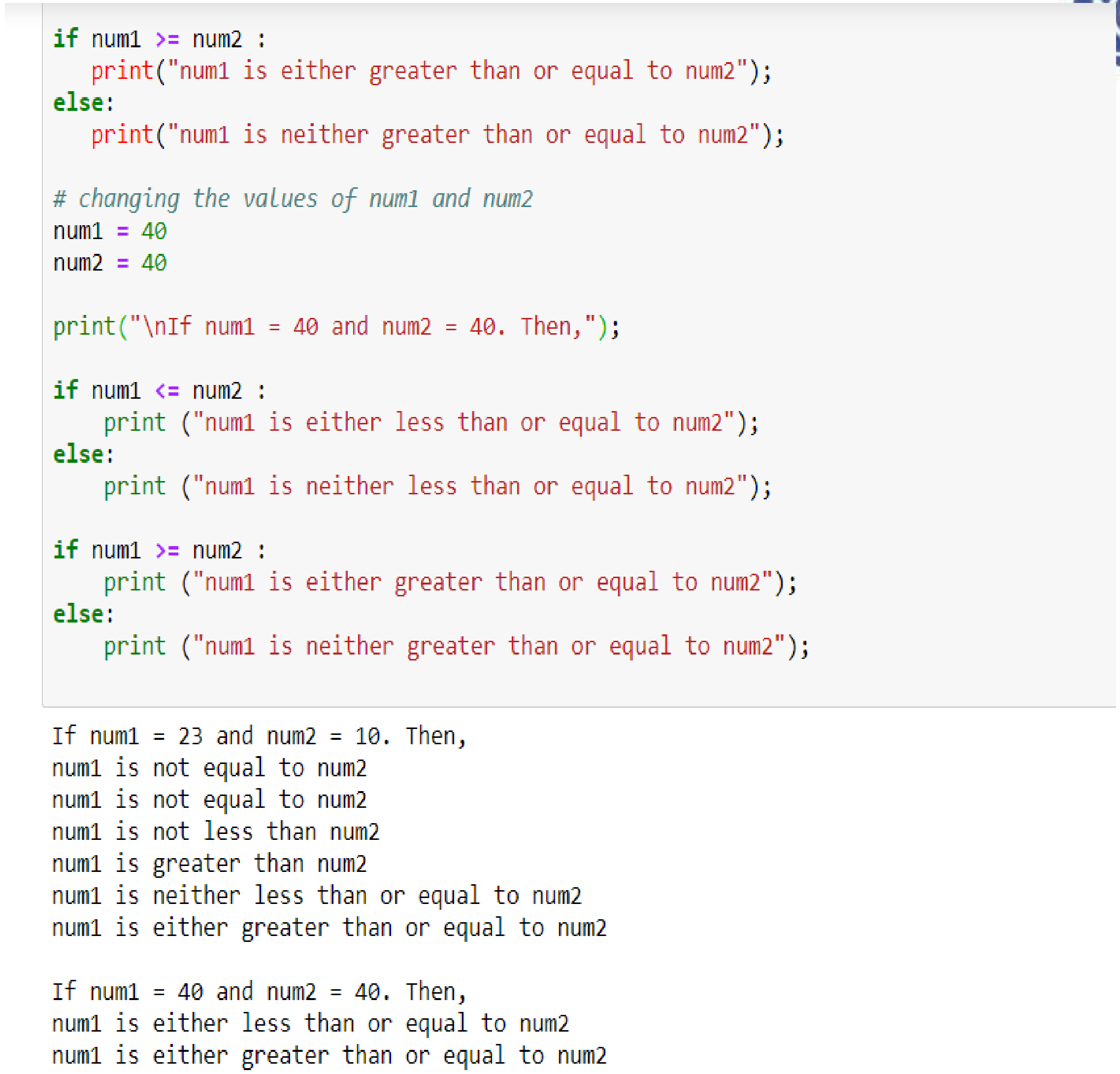
Comparison operators in Python

Operator	Meaning
==	This operator checks if the value of the two operands are equal or not. If equal, then the condition becomes true, otherwise false
!=	This operator checks if the value of the two operands are equal or not. If not equal, then the condition becomes true, otherwise false
>	This operator checks if the value of the left operand is greater than the value of the right operand or not. If yes, then the condition becomes true, otherwise false
<	This operator checks if the value of the left operand is less than the value of the right operand or not. If yes, then the condition becomes true, otherwise false
>=	This operator checks if the value of the left operand is greater than or equal to the value of the right operand or not. If yes, then the condition becomes true
<=	This operator checks if the value of the left operand is less than or equal to the value of the right operand or not. If yes, then the condition becomes true

Comparison Operators - Example Program



```python
# Python Operators - Comparison Operators - Example Program

num1 = 23
num2 = 10
res = 0

print ("If num1 = 23 and num2 = 10. Then,");

if num1 == num2 :
    print ("num1 is equal to num2");
else:
    print ("num1 is not equal to num2");

if num1 != num2 :
    print ("num1 is not equal to num2");
else:
    print ("num1 is equal to num2");

if num1 < num2 :
    print ("num1 is less than num2");
else:
    print ("num1 is not less than num2");
if num1 > num2 :
    print ("num1 is greater than num2");
else:
    print ("num1 is not greater than num2");

if num1 <= num2 :
    print ("num1 is either less than or equal to num2");
else:
    print ("num1 is neither less than or equal to num2");
```



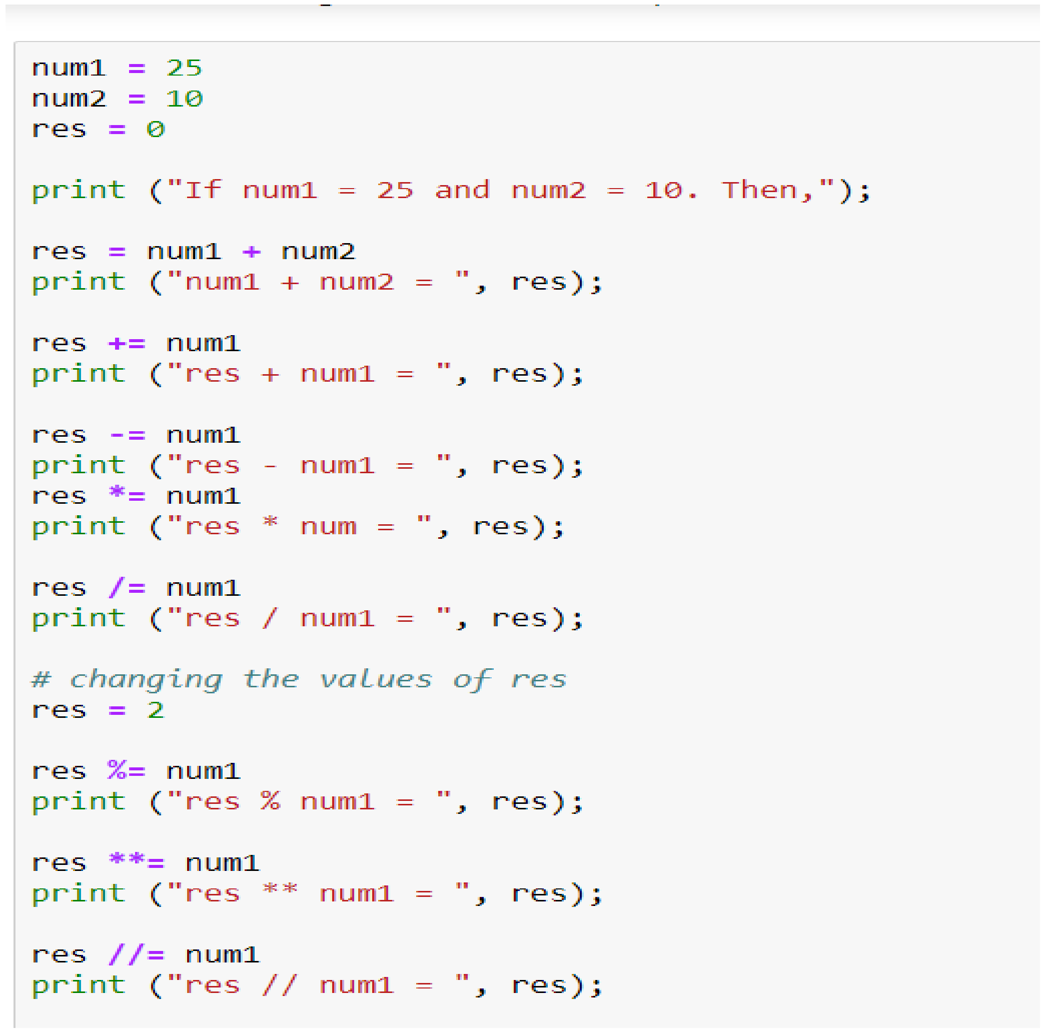
```python
if num1 >= num2 :
   print("num1 is either greater than or equal to num2");
else:
   print("num1 is neither greater than or equal to num2");

# changing the values of num1 and num2
num1 = 40
num2 = 40

print("\nIf num1 = 40 and num2 = 40. Then,");

if num1 <= num2 :
    print ("num1 is either less than or equal to num2");
else:
    print ("num1 is neither less than or equal to num2");

if num1 >= num2 :
    print ("num1 is either greater than or equal to num2");
else:
    print ("num1 is neither greater than or equal to num2");
```

```
If num1 = 23 and num2 = 10. Then,
num1 is not equal to num2
num1 is not equal to num2
num1 is not less than num2
num1 is greater than num2
num1 is neither less than or equal to num2
num1 is either greater than or equal to num2

If num1 = 40 and num2 = 40. Then,
num1 is either less than or equal to num2
num1 is either greater than or equal to num2
```

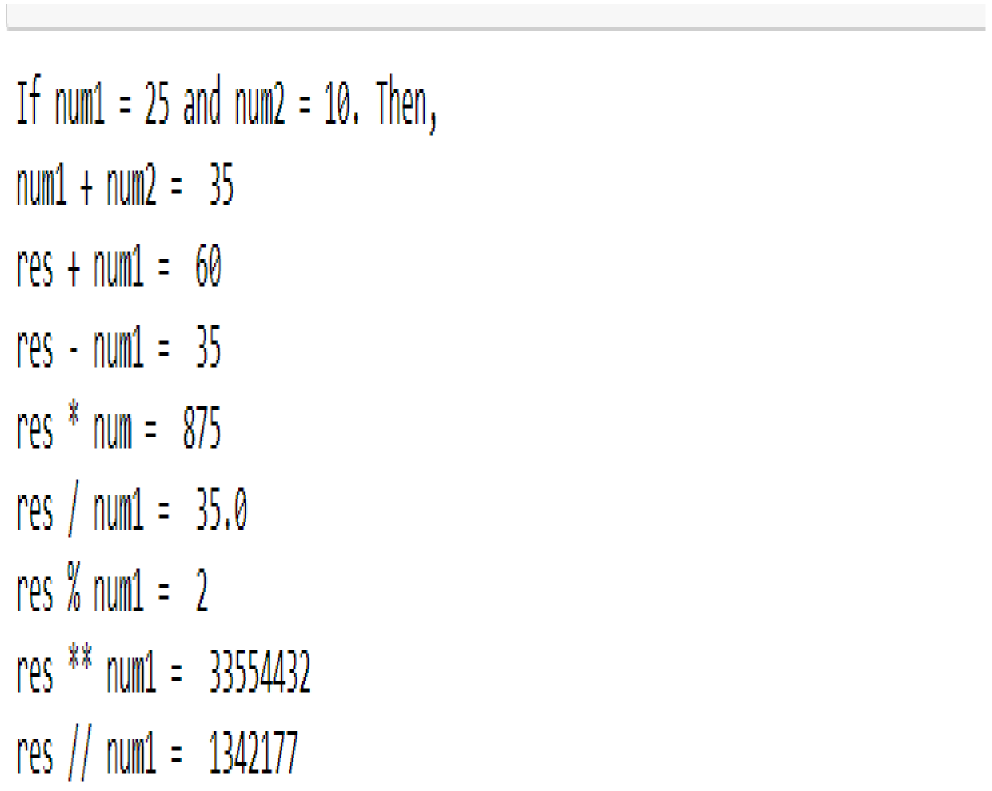
Python Assignment Operators

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Python Assignment Operators - Example Program



```python
num1 = 25
num2 = 10
res = 0

print ("If num1 = 25 and num2 = 10. Then,");

res = num1 + num2
print ("num1 + num2 = ", res);

res += num1
print ("res + num1 = ", res);

res -= num1
print ("res - num1 = ", res);
res *= num1
print ("res * num = ", res);

res /= num1
print ("res / num1 = ", res);

# changing the values of res
res = 2

res %= num1
print ("res % num1 = ", res);

res **= num1
print ("res ** num1 = ", res);

res //= num1
print ("res // num1 = ", res);
```



```
If num1 = 25 and num2 = 10. Then,
num1 + num2 =  35
res + num1 =  60
res - num1 =  35
res * num =  875
res / num1 =  35.0
res % num1 =  2
res ** num1 =  33554432
res // num1 =  1342177
```

Python Bitwise Operators

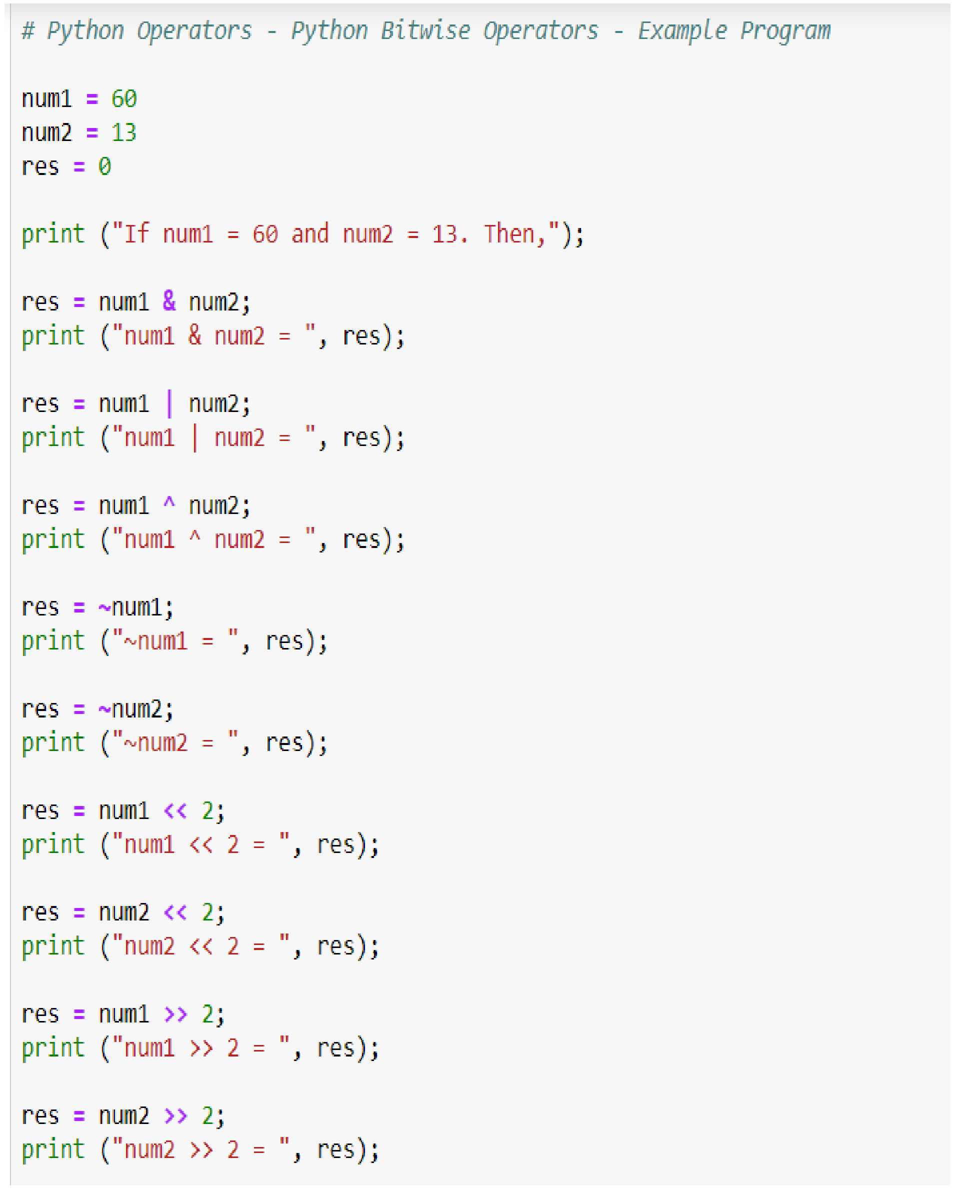
Bitwise operator works on bits and performs bit by bit operation.

Assume if a = 60; and b = 13; Now in binary format they will be as follows −

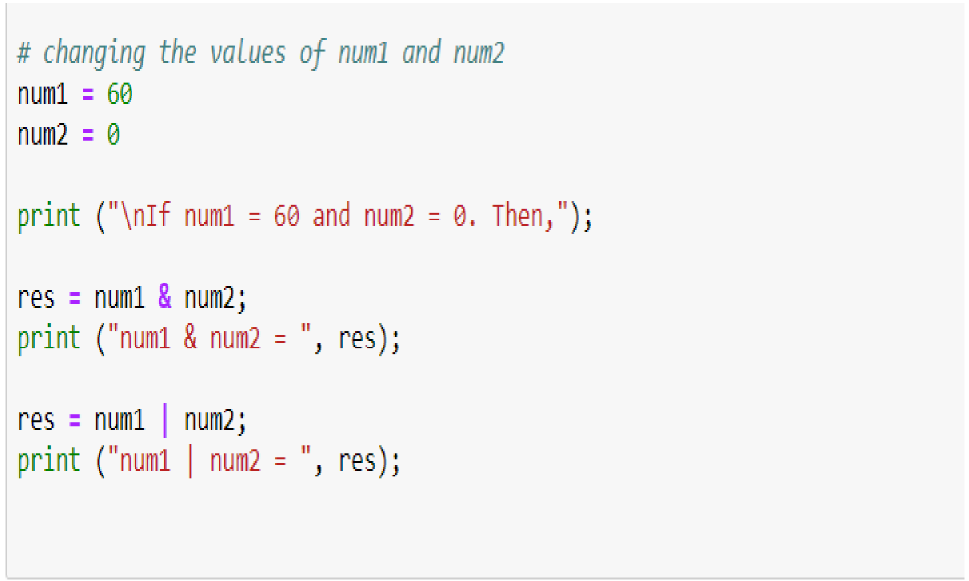
```
a = 0011 1100
b = 0000 1101
-----------------

a&b = 0000 1100
a|b = 0011 1101
a^b = 0011 0001
~a  = 1100 0011
```

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

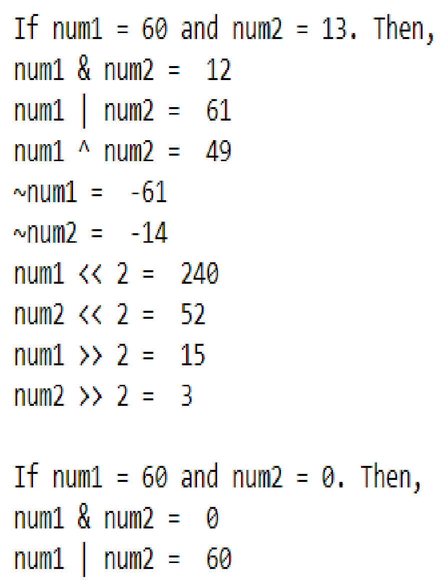
Bitwise operators



```python
# Python Operators - Python Bitwise Operators - Example Program

num1 = 60
num2 = 13
res = 0

print ("If num1 = 60 and num2 = 13. Then,");

res = num1 & num2;
print ("num1 & num2 = ", res);

res = num1 | num2;
print ("num1 | num2 = ", res);

res = num1 ^ num2;
print ("num1 ^ num2 = ", res);

res = ~num1;
print ("~num1 = ", res);

res = ~num2;
print ("~num2 = ", res);

res = num1 << 2;
print ("num1 << 2 = ", res);

res = num2 << 2;
print ("num2 << 2 = ", res);

res = num1 >> 2;
print ("num1 >> 2 = ", res);

res = num2 >> 2;
print ("num2 >> 2 = ", res);
```



```python
# changing the values of num1 and num2
num1 = 60
num2 = 0

print ("\nIf num1 = 60 and num2 = 0. Then,");

res = num1 & num2;
print ("num1 & num2 = ", res);

res = num1 | num2;
print ("num1 | num2 = ", res);
```



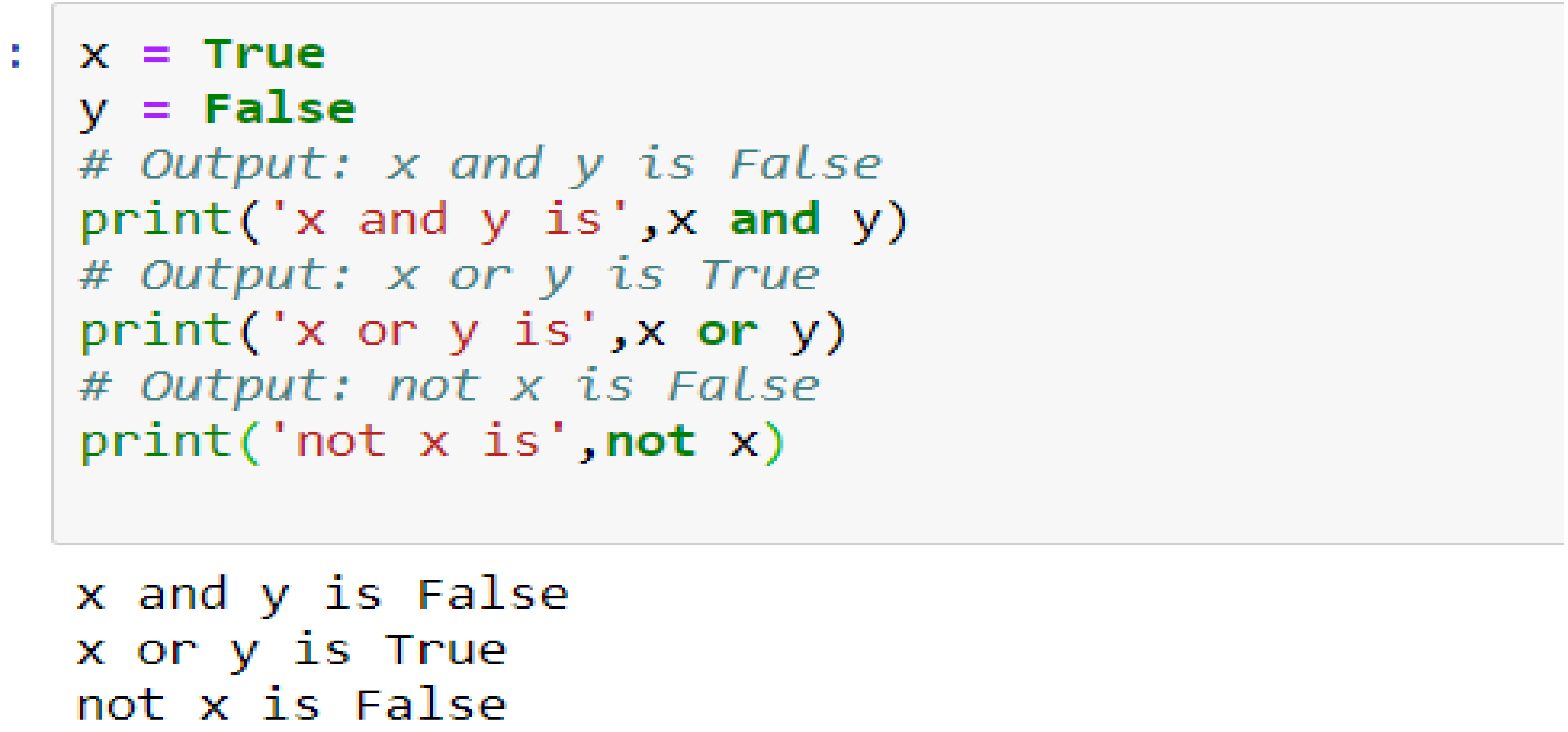
```
If num1 = 60 and num2 = 13. Then,
num1 & num2 =  12
num1 | num2 =  61
num1 ^ num2 =  49
~num1 =  -61
~num2 =  -14
num1 << 2 =  240
num2 << 2 =  52
num1 >> 2 =  15
num2 >> 2 =  3

If num1 = 60 and num2 = 0. Then,
num1 & num2 =  0
num1 | num2 =  60
```

Python Logical Operators

The logical operators and, or and not are also referred to as boolean operators, which may evaluate true or false.

OPERATOR	DESCRIPTION	SYNTAX
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if operand is false	not x

Logical operators



```
x = True
y = False
# Output: x and y is False
print('x and y is',x and y)
# Output: x or y is True
print('x or y is',x or y)
# Output: not x is False
print('not x is',not x)
```

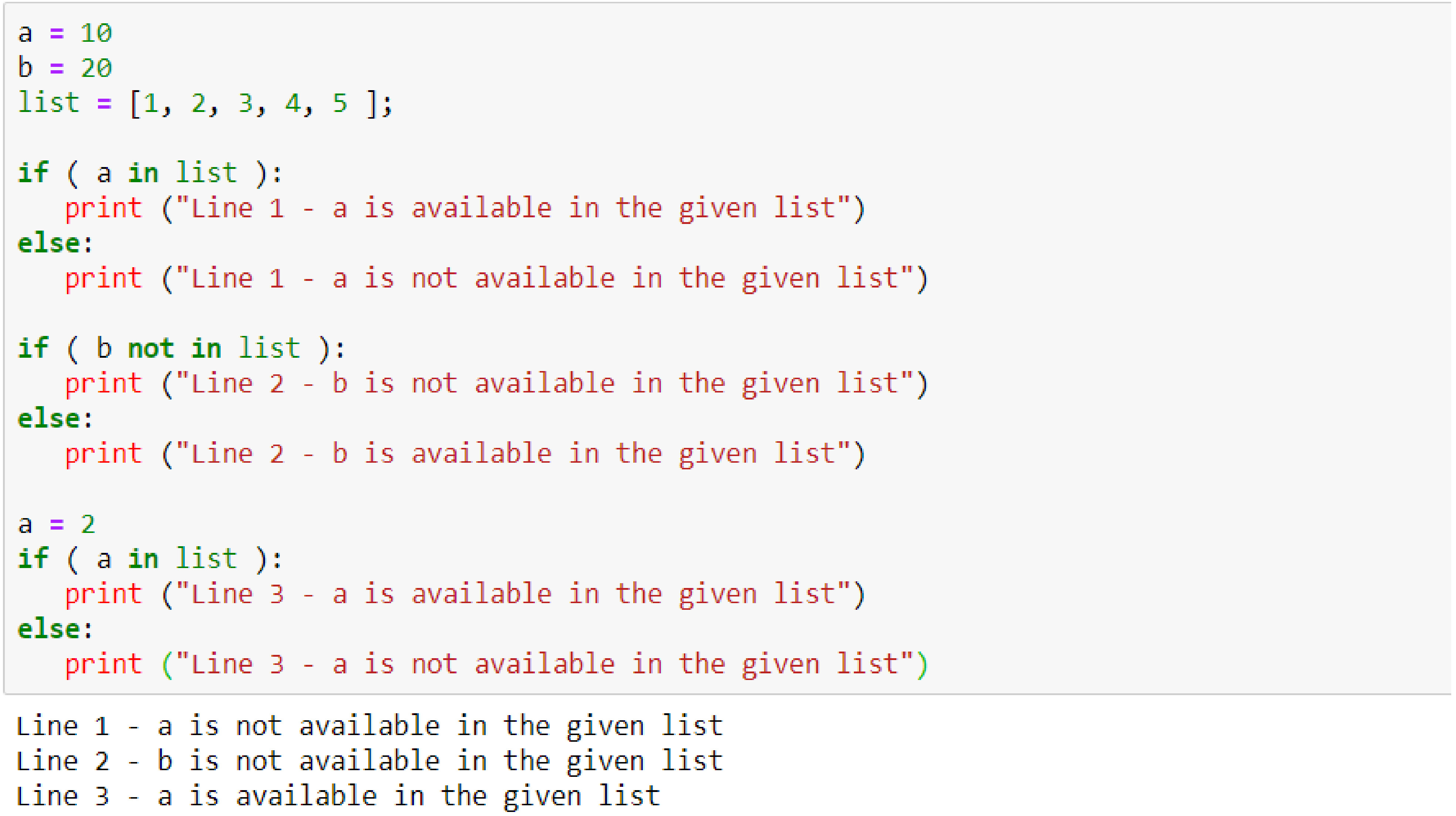
```
x and y is False
x or y is True
not x is False
```

Python Membership Operators

- Python’s membership operators test for membership in a sequence, such as strings, lists, or tuples.
- There are two membership operators as explained below

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Python Membership Operators Example Program



```python
a = 10
b = 20
list = [1, 2, 3, 4, 5 ];

if ( a in list ):
   print ("Line 1 - a is available in the given list")
else:
   print ("Line 1 - a is not available in the given list")

if ( b not in list ):
   print ("Line 2 - b is not available in the given list")
else:
   print ("Line 2 - b is available in the given list")

a = 2
if ( a in list ):
   print ("Line 3 - a is available in the given list")
else:
   print ("Line 3 - a is not available in the given list")
```

```
Line 1 - a is not available in the given list
Line 2 - b is not available in the given list
Line 3 - a is available in the given list
```

Python Identity Operators

- The identity opeartors in Python are used to determine whether a value is of a certain class or type.
- They are usually used to determine the type of data a certain variable contains.

Two Identity Opeartors are –

is – returns TRUE if the type of the value in the right operand points to the same type in the left operand

Is not – returns TRUE if the type of the value in the right operand points to a different type than the value in the left operand.

Python Identity Operators Example Program

```
x = 5
type(x) is int

True

type(x) is not float
True

y=5.5
type(y) is not float

False

type(y) is int

False
```

Functions

What is Function?

- In Python, function is a group of related statements that perform a specific task.
- Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

Types of Functions:

a) **Built-in Functions**: Functions that are predefined. We have used many predefined functions in Python.

b) **User- Defined**: Functions that are created according to the requirements.

Defining a Function

1) Keyword def is used to start the Function Definition. Def specifies the starting of Function block.

2) def is followed by function-name followed by parenthesis.

3) Parameters are passed inside the parenthesis. At the end a colon is marked.

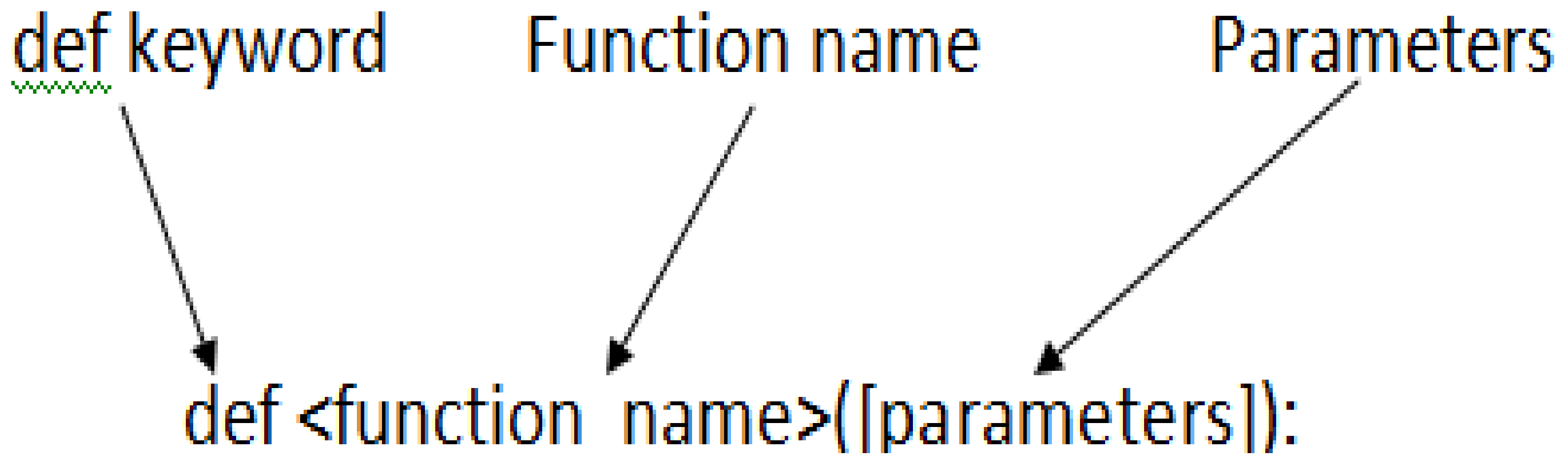
Syntax:

```
def <function_name>([parameters]):

</function_name>
```

E.g. def sum(a,b):

4) Before writing a code, an Indentation is provided before every statement. It should be same for all statements inside the function.

Function Syntax



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

Calling a function

- To execute a function it needs to be called. This is called function calling.

Syntax:

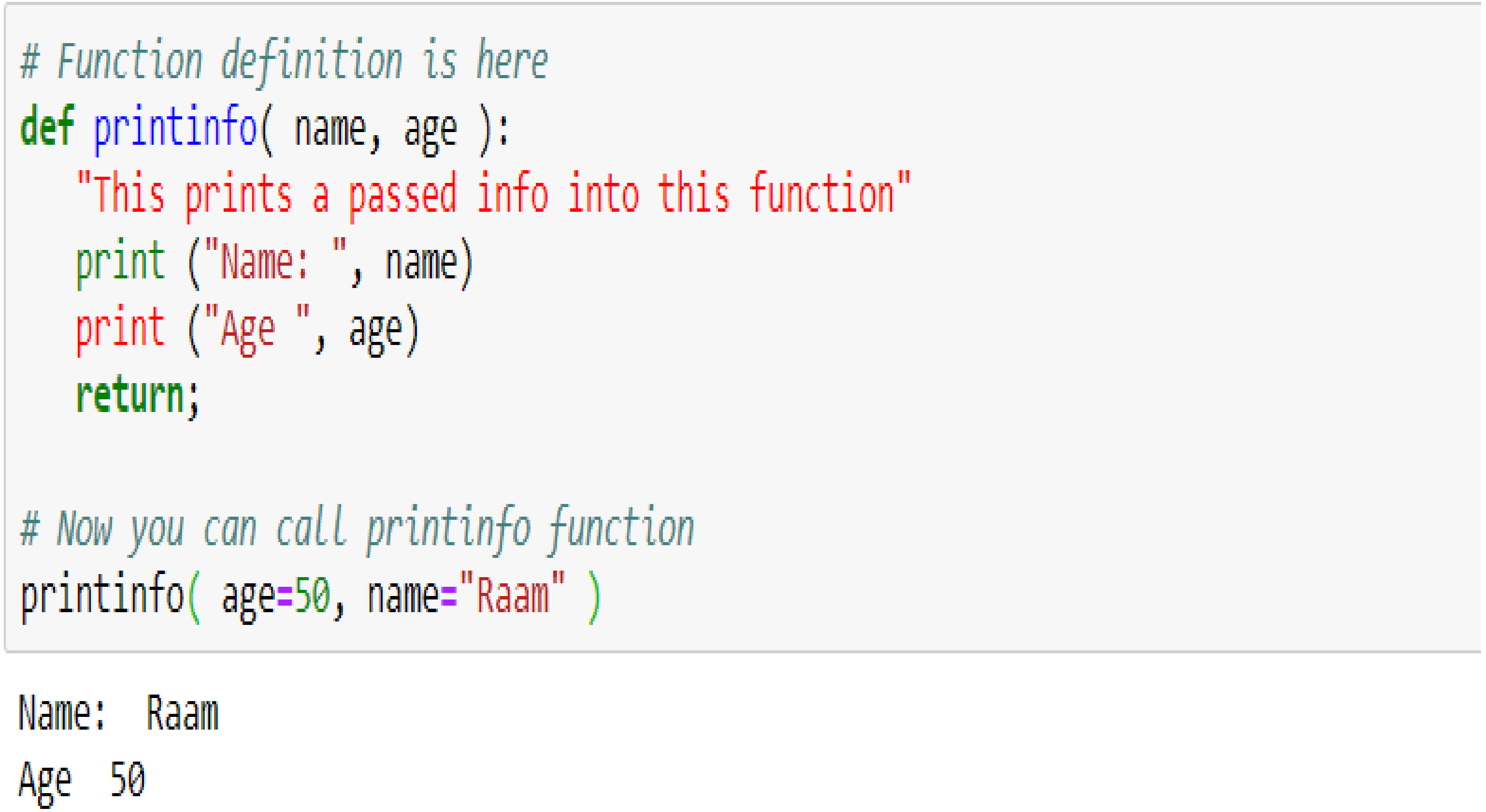
<function name>(parameters)

</function name>

eg:

sum(a,b)

Example program



```python
# Function definition is here
def printinfo( name, age ):
   "This prints a passed info into this function"
   print ("Name: ", name)
   print ("Age ", age)
   return;

# Now you can call printinfo function
printinfo( age=50, name="Raam" )
```

```
Name:  Raam
Age  50
```

AICTE TRAINING & LEARNING ACADEMY
ATAL

MICRO, SMALL & MEDIUM ENTERPRISES
सूक्ष्म, लघु एवं मध्यम उद्यम
OUR STRENGTH • हमारी शक्ति
Ministry of MSME, Govt. of India

Q&A