

# B Sc Computer Science – VI Semester

## Elective Papers - CS6PET01: Python and LaTeX

### Module I - Introduction to Python

The Python Programming Language, Variables, Basic expressions and statements, Arithmetic Operators, Data types - Type conversion, Numbers, Floats, String operations

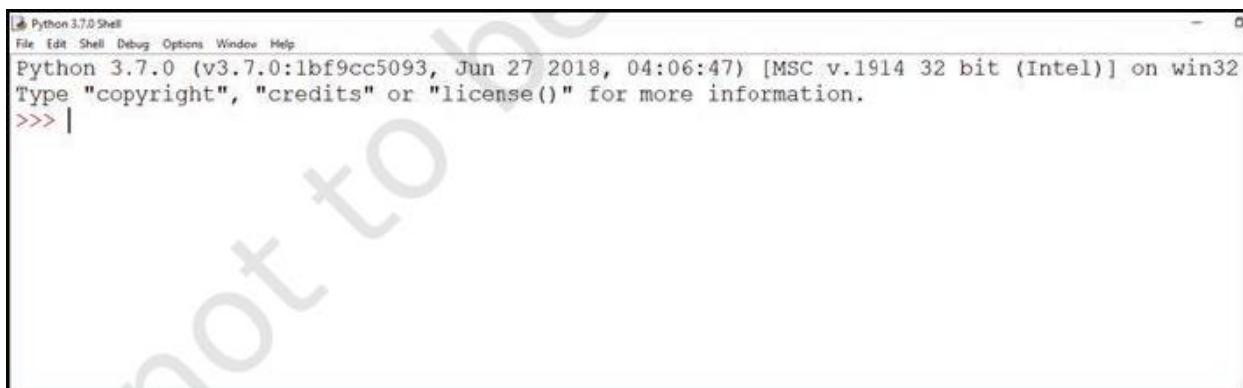
---

#### Features of Python

- Python is a high level language. It is a free and open source language.
- It is an interpreted language, as Python programs are executed by an interpreter.
- Python programs are easy to understand as they have a clearly defined syntax and relatively simple structure.
- Python is case-sensitive. For example, NUMBER and number are not same in Python.
- Python is portable and platform independent, means it can run on various operating systems and hardware platforms.
- Python has a rich library of predefined functions.
- Python is also helpful in web development. Many popular web services and applications are built using Python.
- Python uses indentation for blocks and nested blocks.

#### Working with Python

To write and run (execute) a Python program, we need to have a Python interpreter installed on our computer or we can use any online Python interpreter. The interpreter is also called Python shell. A sample screen of Python interpreter is shown in Figure



*Python interpreter or shell*

#### Execution Modes

There are two ways to use the Python interpreter:

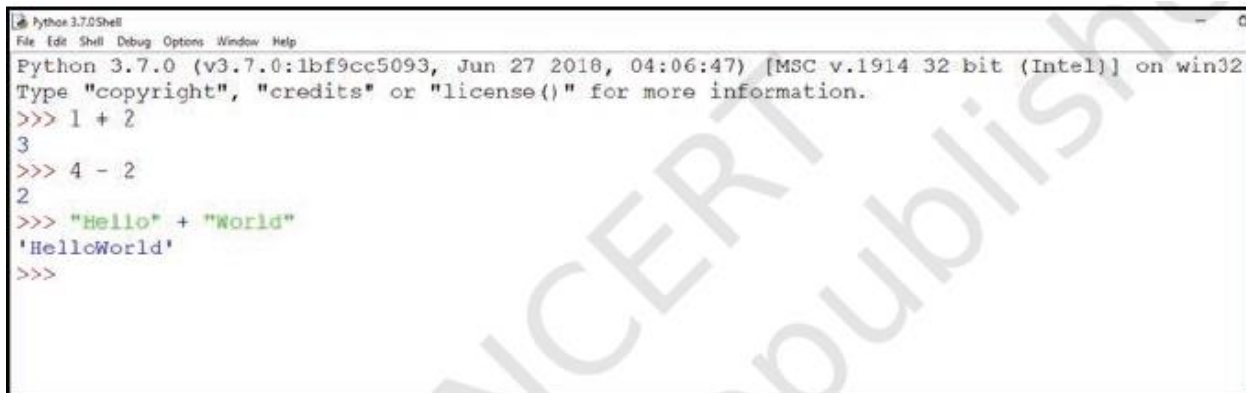
- a) Interactive mode

**b) Script mode**

Interactive mode allows execution of individual statement instantaneously. Whereas, Script mode allows us to write more than one instruction in a file called Python source code file that can be executed.

**(a) Interactive Mode**

To work in the interactive mode, we can simply type a Python statement on the >>> prompt directly. As soon as we press enter, the interpreter executes the statement and displays the result(s), as shown in Figure



```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 1 + 2
3
>>> 4 - 2
2
>>> "Hello" + "World"
'HelloWorld'
>>>
```

*Python interpreter in interactive mode*

Working in the interactive mode is convenient for testing a single line code for instant execution. But in the interactive mode, we cannot save the statements for future use and we have to retype the statements to run them again.

**(B) Script Mode**

In the script mode, we can write a Python program in a file, save it and then use the interpreter to execute it. Python scripts are saved as files where file name has extension “.py”. By default, the Python scripts are saved in the Python installation folder. To execute a script, we can either:

a) Type the file name along with the path at the prompt. For example, if the name of the file is prog5-1.py, we type prog5-1.py. We can otherwise open the program directly from IDLE as shown in Figure



```
prog5-1.py - C:/NCERT/prog5-1.py (3.7.0)
File Edit Format Run Options Window Help
print("Save Earth")
print("Preserve Future")
```

b) While working in the script mode, after saving the file, click [Run]->[Run Module] from the menu as shown in Figure.



## PYTHON KEYWORDS

Keywords are reserved words. Each keyword has a specific meaning to the Python interpreter, and we can use a keyword in our program only for the purpose for which it has been defined. As Python is case sensitive, keywords must be written exactly as given below :

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

## IDENTIFIERS

In programming languages, identifiers are names used to identify a variable, function, or other entities in a program. The rules for naming an identifier in Python are as follows:

- The name should begin with an uppercase or a lowercase alphabet or an underscore sign (\_). This may be followed by any combination of characters a–z, A–Z, 0–9 or underscore (\_). Thus, an identifier cannot start with a digit.
- It can be of any length. (However, it is preferred to keep it short and meaningful).
- It should not be a keyword or reserved word given in the above Table.
- We cannot use special symbols like !, @, #, \$, %, etc., in identifiers.

For example,

1. To find the average of marks obtained by a student in three subjects, we can choose the identifiers as marks1, marks2, marks3 and avg rather than a, b, c, or A, B, C.  

$$\text{avg} = (\text{marks1} + \text{marks2} + \text{marks3})/3$$
2. Similarly, to calculate the area of a rectangle, we can use identifier names, such as area, length, breadth instead of single alphabets as identifiers for clarity and more readability.  

$$\text{area} = \text{length} * \text{breadth}$$

## VARIABLES

A variable in a program is uniquely identified by a name (identifier). Variable in Python refers to an object — an item or element that is stored in the memory. Value of a variable can be a string (e.g., 'a', 'Hello World'), numeric (e.g., 345) or any combination of alphanumeric characters (e.g., CD67). In Python we can use an assignment statement to create new variables and assign specific values to them.

Example : `gender = 'M'`  
`message = "Keep Smiling"`  
`price = 987.9`

Write a program to display values of variables in Python.

```
#To display values of variables
message = " Hello World"
print(message)
userNo = 100
print('User Number is', userNo)
```

Output:  
Hello World  
User Number is 100

The variable message holds string type value and so its content is assigned within double quotes " " (can also be within single quotes ' '), whereas the value of variable userNo is not enclosed in quotes as it is a numeric value.

**Variable declaration is implicit in Python**, means variables are automatically declared and defined when they are assigned a value the first time. Variables must always be assigned values before they are used in expressions as otherwise it will lead to an error in the program. Wherever a variable name occurs in an expression, the interpreter replaces it with the value of that particular variable.

Write a Python program to find the area of a rectangle given that its length is 10 units and breadth is 20 units.

```
#To find the area of a rectangle
length = 10
breadth = 20
area = length * breadth
print(area)
Output:
200
```

### COMMENTS

Comments are used to add a remark or a note in the source code. Comments are not executed by interpreter. They are added with the purpose of making the source code easier for humans to understand. They are used primarily to document the meaning and purpose of source code and its input and output requirements, so that we can remember later how it functions and how to use it. For large and complex software, it may require programmers to work in teams and sometimes, a program written by one programmer is required to be used or maintained by another programmer. In such situations, documentations in the form of comments are needed to understand the working of the program. In Python, a comment starts with # (hash sign). Everything following the # till the end of that line is treated as a comment and the interpreter simply ignores it while executing the statement.

Write a Python program to find the sum of two numbers.

```
#To find the sum of two numbers
```

```
num1 = 10
```

```
num2 = 20
```

```
result = num1 + num2
```

```
print(result)
```

Output:

30

### EVERYTHING IS AN OBJECT

Python treats every value or data item whether numeric, string, or other type (discussed in the next section) as an object in the sense that it can be assigned to some variable or can be passed to a function as an argument. Every object in Python is assigned a unique identity (ID) which remains the same for the lifetime of that object. This ID is akin to the memory address of the object. The function `id()` returns the identity of an object.

#### Example

```
>>> num1 = 20
```

```
>>> id(num1)
```

```
1433920576 #identity of num1
```

```
>>> num2 = 30 - 10
```

```
>>> id(num2)
```

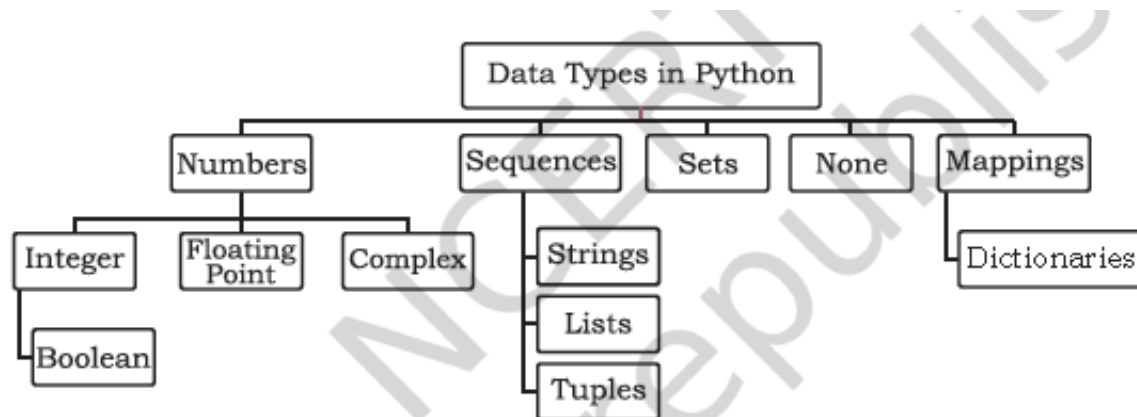
```
1433920576 #identity of num2 and num1
```

```
#are same as both refers to object 20
```

### DATA TYPES

Every value belongs to a specific data type in Python. Data type identifies the type of data values a variable can hold and the operations that can be performed on that data.

The following Figure enlists the data types available in Python.



## 1. Number

Number data type stores numerical values only. It is further classified into three different types: int, float and complex.

Type / Class	Description	Examples
int	integer numbers	-12, -3, 0, 125, 2
float	real or floating point numbers	-2.04, 4.0, 14.23
complex	complex numbers	3 + 4j, 2 - 2j

Boolean data type (bool) is a subtype of integer. It is a unique data type, consisting of two constants, True and False. Boolean True value is non-zero, non-null and non-empty. Boolean False is the value zero.

To determine the data type of the variable using built-in function type().

*Example:*

```
>>> num1 = 10
>>> type(num1)
<class 'int'>
>>> num2 = -1210
>>> type(num2)
<class 'int'>
>>> var1 = True
>>> type(var1)
<class 'bool'>
>>> float1 = -1921.9
>>> type(float1)
<class 'float'>
>>> float2 = -9.8*10**2
>>> print(float2, type(float2))
-980.0000000000001 <class 'float'>
>>> var2 = -3+7.2j
>>> print(var2, type(var2))
(-3+7.2j) <class 'complex'>
```

Variables of simple data types like integers, float, boolean, etc., hold single values. But such variables are not useful to hold a long list of information, for example, names of the months in a year, names of students in a class, names and numbers in a phone book or the list of artefacts in a museum. For this, Python provides data types like **tuples, lists, dictionaries and sets**.

## 2. Sequence

A Python sequence is an ordered collection of items, where each item is indexed by an integer. The three types of sequence data types available in Python are Strings, Lists and Tuples. We will learn about each of them in detail in later chapters. A brief introduction to these data types is as follows:

**(A) String**

String is a group of characters. These characters may be alphabets, digits or special characters including spaces. String values are enclosed either in single quotation marks (e.g., 'Hello') or in double quotation marks (e.g. "Hello"). The quotes are not a part of the string, they are used to mark the beginning and end of the string for the interpreter. For example,

```
>>> str1 = 'Hello Friend'
>>> str2 = "452"
```

We cannot perform numerical operations on strings, even when the string contains a numeric value, as in str2.

**(B) List**

List is a sequence of items separated by commas and the items are enclosed in square brackets [ ].

**Example**

```
#To create a list
>>> list1 = [5, 3.4, "New Delhi", "20C", 45]
#print the elements of the list list1
>>> print(list1)
[5, 3.4, 'New Delhi', '20C', 45]
```

**(C) Tuple**

Tuple is a sequence of items separated by commas and items are enclosed in parenthesis ( ).

This is unlike list, where values are enclosed in brackets [ ]. Once created, we cannot change the tuple.

**Example**

```
#create a tuple tuple1
>>> tuple1 = (10, 20, "Apple", 3.4, 'a')
#print the elements of the tuple tuple1
>>> print(tuple1)
(10, 20, "Apple", 3.4, 'a')
```

**3. Set**

Set is an unordered collection of items separated by commas and the items are enclosed in curly brackets { }. A set is similar to list, except that it cannot have duplicate entries. Once created, elements of a set cannot be changed.

**Example**

```
#create a set
>>> set1 = {10,20,3.14,"New Delhi"}
>>> print(type(set1))
<class 'set'>
>>> print(set1)
{10, 20, 3.14, "New Delhi"}
#duplicate elements are not included in set
>>> set2 = {1,2,1,3}
>>> print(set2)
{1, 2, 3}
```

#### 4. None

None is a special data type with a single value. It is used to signify the absence of value in a situation. None supports no special operations, and it is neither same as False nor 0 (zero).

##### Example

```
>>> myVar = None
>>> print(type(myVar))
<class 'NoneType'>
>>> print(myVar)
None
```

#### 5. Mapping

Mapping is an unordered data type in Python. Currently, there is only one standard mapping data type in Python called dictionary.

##### (A) Dictionary

Dictionary in Python holds data items in key-value pairs. Items in a dictionary are enclosed in curly brackets { }. Dictionaries permit faster access to data. Every key is separated from its value using a colon (:) sign. The key : value pairs of a dictionary can be accessed using the key. The keys are usually strings and their values can be any data type. In order to access any value in the dictionary, we have to specify its key in square brackets [ ].

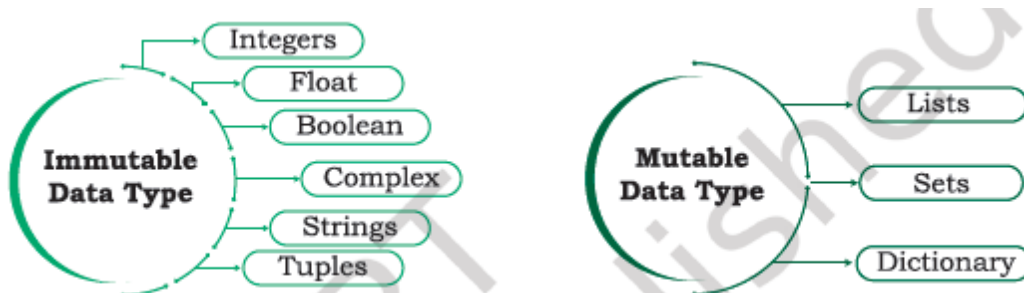
##### Example

```
#create a dictionary
>>> dict1 = {'Fruit':'Apple', 'Climate':'Cold', 'Price(kg)':120}
>>> print(dict1)
{'Fruit': 'Apple', 'Climate': 'Cold', 'Price(kg)': 120}
>>> print(dict1['Price(kg)'])
120
```

#### Mutable and Immutable Data Types

Sometimes we may require to change or update the values of certain variables used in a program. However, for certain data types, Python does not allow us to change the values once a variable of that type has been created and assigned values.

Variables whose values can be changed after they are created and assigned are called mutable. Variables whose values cannot be changed after they are created and assigned are called immutable. When an attempt is made to update the value of an immutable variable, the old variable is destroyed and a new variable is created by the same name in memory. Python data types can be classified into mutable and immutable as shown in Figure



Let us now see what happens when an attempt is made to update the value of a variable.

```
>>> num1 = 300
```



This statement will create an object with value 300 and the object is referenced by the identifier num1 as shown in following Figure I.

```
>>> num2 = num1
```

The statement num2 = num1 will make num2 refer to the value 300, also being referred by num1, and stored at memory location number, say 1000. So, num1 shares the referenced location with num2 as shown in Figure II

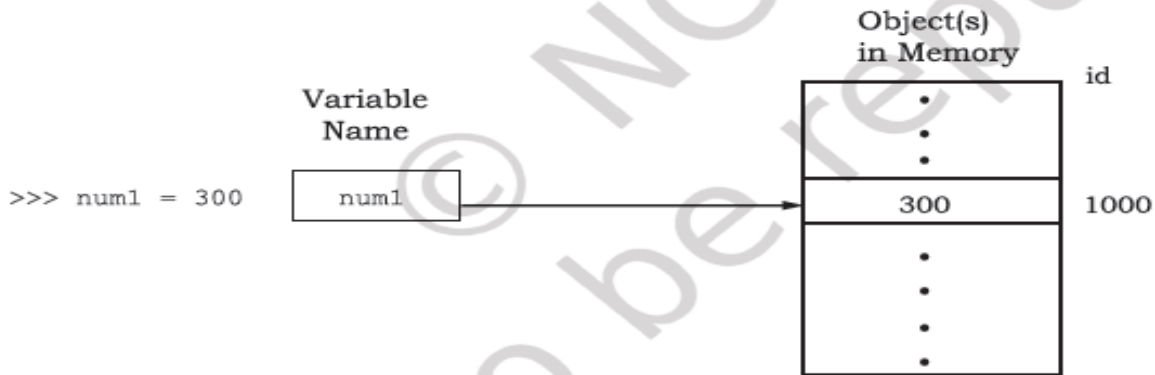


Figure I : Object and its identifier

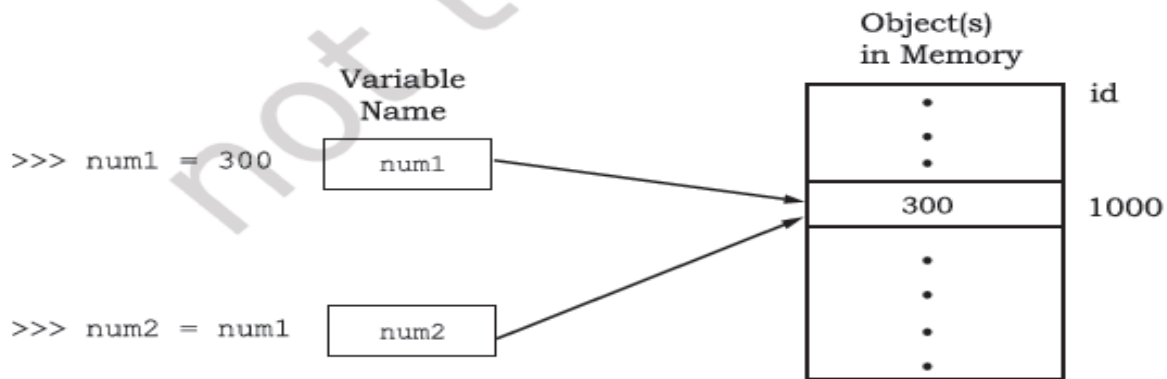
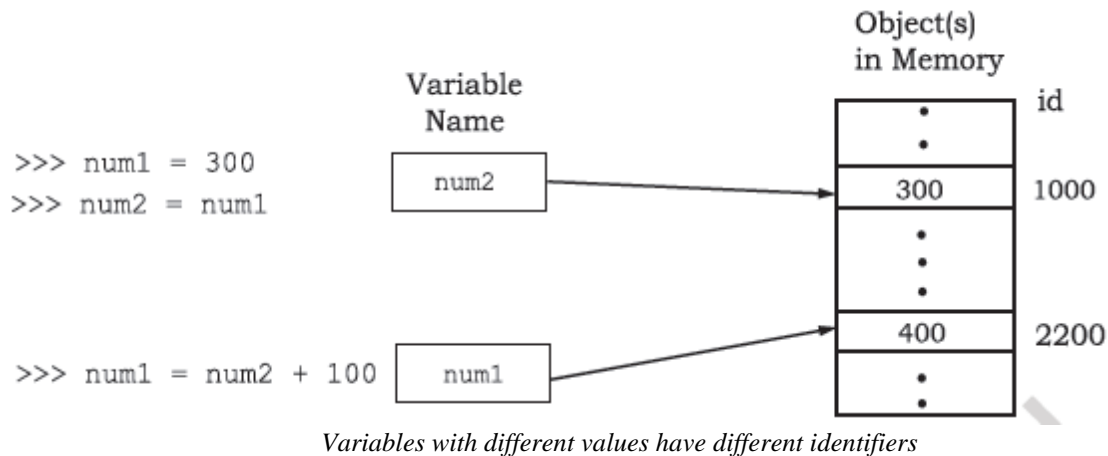


Figure II : Variables with same value have same identifier

In this manner Python makes the assignment effective by copying only the reference, and not the data:

```
>>> num1
= num2 +
100
```

This statement 1 num1 = num2 + 100 links the variable num1 to a new object stored at memory location number say 2200 having a value 400. As num1 is an integer, which is an immutable type, it is rebuilt, as shown in following Figure.



### Deciding Usage of Python Data Types

It is preferred to use lists when we need a simple iterable collection of data that may go for frequent modifications. For example, if we store the names of students of a class in a list, then it is easy to update the list when some new students join or some leave the course. Tuples are used when we do not need any change in the data. For example, names of months in a year. When we need uniqueness of elements and to avoid duplicacy it is preferable to use sets, for example, list of artefacts in a museum. If our data is being constantly modified or we need a fast lookup based on a custom key or we need a logical association between the key : value pair, it is advised to use dictionaries. A mobile phone book is a good application of dictionary.

### OPERATORS

An operator is used to perform specific mathematical or logical operation on values. Python supports several kinds of operators whose categorisation is briefly explained below:

#### 1. Arithmetic Operators

Python supports arithmetic operators that are used to perform the four basic arithmetic operations as well as modular division, floor division and exponentiation.

#### Arithmetic Operators in Python

Operator	Operation	Description	Example (Try in Lab)
+	Addition	Adds the two numeric values on either side of the operator  This operator can also be used to concatenate two strings on either side of the operator	<pre>&gt;&gt;&gt; num1 = 5 &gt;&gt;&gt; num2 = 6 &gt;&gt;&gt; num1 + num2 11 &gt;&gt;&gt; str1 = "Hello" &gt;&gt;&gt; str2 = "India" &gt;&gt;&gt; str1 + str2 'HelloIndia'</pre>
-	Subtraction	Subtracts the operand on the right from the operand on the left	<pre>&gt;&gt;&gt; num1 = 5 &gt;&gt;&gt; num2 = 6 &gt;&gt;&gt; num1 - num2 -1</pre>
*	Multiplication	Multiplies the two values on both side of the operator  Repeats the item on left of the operator if first operand is a string and second operand is an integer value	<pre>&gt;&gt;&gt; num1 = 5 &gt;&gt;&gt; num2 = 6 &gt;&gt;&gt; num1 * num2 30 &gt;&gt;&gt; str1 = 'India' &gt;&gt;&gt; str1 * 2 'IndiaIndia'</pre>
/	Division	Divides the operand on the left by the operand on the right and returns the quotient	<pre>&gt;&gt;&gt; num1 = 8 &gt;&gt;&gt; num2 = 4 &gt;&gt;&gt; num2 / num1 0.5</pre>
%	Modulus	Divides the operand on the left by the operand on the right and returns the remainder	<pre>&gt;&gt;&gt; num1 = 13 &gt;&gt;&gt; num2 = 5 &gt;&gt;&gt; num1 % num2 3</pre>
//	Floor Division	Divides the operand on the left by the operand on the right and returns the quotient by removing the decimal part. It is sometimes also called integer division.	<pre>&gt;&gt;&gt; num1 = 13 &gt;&gt;&gt; num2 = 4 &gt;&gt;&gt; num1 // num2 3 &gt;&gt;&gt; num2 // num1 0</pre>
**	Exponent	Performs exponential [power] calculation on operands. That is, raise the operand on the left to the power of the operand on the right	<pre>&gt;&gt;&gt; num1 = 3 &gt;&gt;&gt; num2 = 4 &gt;&gt;&gt; num1 ** num2 81</pre>

## 2. Relational Operators

Relational operator compares the values of the operands on its either side and determines the relationship among them. Assume the Python variables num1 = 10, num2 = 0, num3 = 10, str1 = "Good", str2 = "Afternoon" for the following examples:

### Relational operators in Python

Operator	Operation	Description	Example (Try in Lab)
==	Equals to	If the values of two operands are equal, then the condition is True, otherwise it is False	<pre>&gt;&gt;&gt; num1 == num2 False &gt;&gt;&gt; str1 == str2 False</pre>
!=	Not equal to	If values of two operands are not equal, then condition is True, otherwise it is False	<pre>&gt;&gt;&gt; num1 != num2 True &gt;&gt;&gt; str1 != str2 True &gt;&gt;&gt; num1 != num3 False</pre>
>	Greater than	If the value of the left-side operand is greater than the value of the right-side operand, then condition is True, otherwise it is False	<pre>&gt;&gt;&gt; num1 &gt; num2 True &gt;&gt;&gt; str1 &gt; str2 True</pre>
<	Less than	If the value of the left-side operand is less than the value of the right-side operand, then condition is True, otherwise it is False	<pre>&gt;&gt;&gt; num1 &lt; num3 False &gt;&gt;&gt; str2 &lt; str1 True</pre>
>=	Greater than or equal to	If the value of the left-side operand is greater than or equal to the value of the right-side operand, then condition is True, otherwise it is False	<pre>&gt;&gt;&gt; num1 &gt;= num2 True &gt;&gt;&gt; num2 &gt;= num3 False &gt;&gt;&gt; str1 &gt;= str2 True</pre>
<=	Less than or equal to	If the value of the left operand is less than or equal to the value of the right operand, then is True otherwise it is False	<pre>&gt;&gt;&gt; num1 &lt;= num2 False &gt;&gt;&gt; num2 &lt;= num3 True &gt;&gt;&gt; str1 &lt;= str2 False</pre>

### 3. Assignment Operators

Assignment operator assigns or changes the value of the variable on its left.

#### Assignment operators in Python

Operator	Description	Example (Try in Lab)
=	Assigns value from right-side operand to left-side operand	<pre>&gt;&gt;&gt; num1 = 2 &gt;&gt;&gt; num2 = num1 &gt;&gt;&gt; num2 2 &gt;&gt;&gt; country = 'India' &gt;&gt;&gt; country 'India'</pre>

+=	It adds the value of right-side operand to the left-side operand and assigns the result to the left-side operand <b>Note:</b> $x += y$ is same as $x = x + y$	<pre>&gt;&gt;&gt; num1 = 10 &gt;&gt;&gt; num2 = 2 &gt;&gt;&gt; num1 += num2 &gt;&gt;&gt; num1 12 &gt;&gt;&gt; num2 2 &gt;&gt;&gt; str1 = 'Hello' &gt;&gt;&gt; str2 = 'India' &gt;&gt;&gt; str1 += str2 &gt;&gt;&gt; str1 'HelloIndia'</pre>
-=	It subtracts the value of right-side operand from the left-side operand and assigns the result to left-side operand <b>Note:</b> $x -= y$ is same as $x = x - y$	<pre>&gt;&gt;&gt; num1 = 10 &gt;&gt;&gt; num2 = 2 &gt;&gt;&gt; num1 -= num2 &gt;&gt;&gt; num1 8</pre>
*=	It multiplies the value of right-side operand with the value of left-side operand and assigns the result to left-side operand <b>Note:</b> $x *= y$ is same as $x = x * y$	<pre>&gt;&gt;&gt; num1 = 2 &gt;&gt;&gt; num2 = 3 &gt;&gt;&gt; num1 *= 3  &gt;&gt;&gt; num1 6 &gt;&gt;&gt; a = 'India' &gt;&gt;&gt; a *= 3 &gt;&gt;&gt; a 'IndiaIndiaIndia'</pre>
/=	It divides the value of left-side operand by the value of right-side operand and assigns the result to left-side operand <b>Note:</b> $x /= y$ is same as $x = x / y$	<pre>&gt;&gt;&gt; num1 = 6 &gt;&gt;&gt; num2 = 3 &gt;&gt;&gt; num1 /= num2 &gt;&gt;&gt; num1 2.0</pre>
%=	It performs modulus operation using two operands and assigns the result to left-side operand <b>Note:</b> $x \% = y$ is same as $x = x \% y$	<pre>&gt;&gt;&gt; num1 = 7 &gt;&gt;&gt; num2 = 3 &gt;&gt;&gt; num1 %= num2 &gt;&gt;&gt; num1 1</pre>
//=	It performs floor division using two operands and assigns the result to left-side operand <b>Note:</b> $x //= y$ is same as $x = x // y$	<pre>&gt;&gt;&gt; num1 = 7 &gt;&gt;&gt; num2 = 3 &gt;&gt;&gt; num1 //= num2 &gt;&gt;&gt; num1 2</pre>
**=	It performs exponential [power] calculation on operators and assigns value to the left-side operand <b>Note:</b> $x ** = y$ is same as $x = x ** y$	<pre>&gt;&gt;&gt; num1 = 2 &gt;&gt;&gt; num2 = 3 &gt;&gt;&gt; num1 **= num2 &gt;&gt;&gt; num1 8</pre>

#### 4. Logical Operators

There are three logical operators supported by Python. These operators (and, or, not) are to be written in lower case only. The logical operator evaluates to either True or False based on the logical operands on either side. Every value is logically either True or False. By default, all values are True except None, False, 0 (zero), empty collections "", (), [], {}, and few other

special values. So if we say num1 = 10, num2 = -20, then both num1 and num2 are logically True.

#### Logical operators in Python

Operator	Operation	Description	Example (Try in Lab)
and	Logical AND	If both the operands are True, then condition becomes True	<pre>&gt;&gt;&gt; True and True True &gt;&gt;&gt; num1 = 10 &gt;&gt;&gt; num2 = -20 &gt;&gt;&gt; bool(num1 and num2) True &gt;&gt;&gt; True and False False &gt;&gt;&gt; num3 = 0 &gt;&gt;&gt; bool(num1 and num3) False &gt;&gt;&gt; False and False False</pre>
or	Logical OR	If any of the two operands are True, then condition becomes True	<pre>&gt;&gt;&gt; True or True True &gt;&gt;&gt; True or False True &gt;&gt;&gt; bool(num1 or num3) True &gt;&gt;&gt; False or False False</pre>
not	Logical NOT	Used to reverse the logical state of its operand	<pre>&gt;&gt;&gt; num1 = 10 &gt;&gt;&gt; bool(num1) True &gt;&gt;&gt; not num1 False &gt;&gt;&gt; bool(num1) True</pre>

### 5. Identity Operators

Identity operators are used to determine whether the value of a variable is of a certain type or not. Identity operators can also be used to determine whether two variables are referring to the same object or not. There are two identity operators.

#### Identity operators in Python

Operator	Description	Example (Try in Lab)
is	Evaluates True if the variables on either side of the operator point towards the same memory location and False otherwise. var1 is var2 results to True if id(var1) is equal to id(var2)	<pre>&gt;&gt;&gt; num1 = 5 &gt;&gt;&gt; type(num1) is int True &gt;&gt;&gt; num2 = num1 &gt;&gt;&gt; id(num1) 1433920576 &gt;&gt;&gt; id(num2) 1433920576 &gt;&gt;&gt; num1 is num2 True</pre>
is not	Evaluates to False if the variables on either side of the operator point to the same memory location and True otherwise. var1 is not var2 results to True if id(var1) is not equal to id(var2)	<pre>&gt;&gt;&gt; num1 is not num2 False</pre>

## 6. Membership Operators

Membership operators are used to check if a value is a member of the given sequence or not

### Membership operators in Python

Operator	Description	Example (Try in Lab)
in	Returns True if the variable/value is found in the specified sequence and False otherwise	<pre>&gt;&gt;&gt; a = [1,2,3] &gt;&gt;&gt; 2 in a True &gt;&gt;&gt; '1' in a False</pre>
not in	Returns True if the variable/value is not found in the specified sequence and False otherwise	<pre>&gt;&gt;&gt; a = [1,2,3] &gt;&gt;&gt; 10 not in a True &gt;&gt;&gt; 1 not in a False</pre>

## EXPRESSIONS

An expression is defined as a combination of constants, variables, and operators. An expression always evaluates to a value. A value or a standalone variable is also considered as an expression but a standalone operator is not an expression. Some examples of valid expressions are given below.

- |                  |                            |
|------------------|----------------------------|
| (i) 100          | (iv) $3.0 + 3.14$          |
| (ii) num         | (v) $23/3 - 5 * 7(14 - 2)$ |
| (iii) num – 20.4 | (vi) "Global" + "Citizen"  |

## Precedence of Operators

Evaluation of the expression is based on precedence of operators. When an expression contains different kinds of operators, precedence determines which operator should be applied first. Higher precedence operator is evaluated before the lower precedence operator. Most of the operators studied till now are binary operators. Binary operators are operators with two operands. The unary operators need only one operand, and they have a higher precedence than the binary operators. The minus (-) as well as + (plus) operators can act as both unary and binary operators, but not is a unary logical operator.

#Depth is using - (minus) as unary operator

Value = -Depth

#not is a unary operator, negates True

print(not(True))

### Note:

- Paranthesis can be used to override the precedence of operators. The expression within () is evaluated first.
- For operators with equal precedence, the expression is evaluated from left to right.

The following table lists precedence of all operators from highest to lowest.



Precedence of all operators in Python

Order of Precedence	Operators	Description
1	**	Exponentiation (raise to the power)
2	~ , +, -	Complement, unary plus and unary minus
3	*, /, %, //	Multiply, divide, modulo and floor division
4	+, -	Addition and subtraction
5	<= , < , > , >=, == , !=	Relational and Comparison operators
6	=, %=, /=, //=, -=, +=, *=, **=	Assignment operators
7	is, is not	Identity operators
8	in, not in	Membership operators
9	not	Logical operators
10	and	
11	or	

## STATEMENT

In Python, a statement is a unit of code that the Python interpreter can execute.

### Example

```
>>> x = 4 #assignment statement
>>> cube = x ** 3 #assignment statement
>>> print(x, cube) #print statement
4 64
```

## INPUT AND OUTPUT

Sometimes, a program needs to interact with the user's to get some input data or information from the end user and process it to give the desired output. In Python, we have the `input()` function for taking the user input. The `input()` function prompts the user to enter data. It accepts all user input as string. The user may enter a number or a string but the `input()` function treats them as strings only. The syntax for `input()` is:

`input ([Prompt])`

Prompt is the string we may like to display on the screen prior to taking the input, and it is optional. When a prompt is specified, first it is displayed on the screen after which the user can enter data. The `input()` takes exactly what is typed from the keyboard, converts it into a string and assigns it to the variable on left-hand side of the assignment operator (`=`). Entering data for the input function is terminated by pressing the enter key.

### Example

```
>>> fname = input("Enter your first name: ")
Enter your first name: Arnab
>>> age = input("Enter your age: ")
Enter your age: 19
>>> type(age)
<class 'str'>
```

The variable `fname` will get the string 'Arbab', entered by the user. Similarly, the variable `age` will get the string '19'. We can typecast or change the datatype of the string data accepted from user to an appropriate numeric value. For example, the following statement will convert the accepted string to an integer. If the user enters any non-numeric value, an error will be generated.

#### Example

```
#function int() to convert string to integer
>>> age = int( input("Enter your age:"))
Enter your age: 19
>>> type(age)
<class 'int'>
```

Python uses the `print()` function to output data to standard output device — the screen. The function `print()` evaluates the expression before displaying it on the screen. The `print()` outputs a complete line and then moves to the next line for subsequent output. The syntax for `print()` is:

```
print(value [, ..., sep = ' ', end = '\n'])
```

- **sep:** The optional parameter `sep` is a separator between the output values. We can use a character, integer or a string as a separator. The default separator is space.
- **end:** This is also optional and it allows us to specify any string to be appended after the last value. The default is a new line.

#### Example

Statement	Output
<code>print("Hello")</code>	Hello
<code>print(10*2.5)</code>	25.0
<code>print("I" + "love" + "my" + "country")</code>	Ilovecountry
<code>print("I'm", 16, "years old")</code>	I'm 16 years old

The third `print` function in the above example is concatenating strings, and we use `+` (plus) between two strings to concatenate them. The fourth `print` function also appears to be concatenating strings but uses commas (,) between strings. Actually, here we are passing multiple arguments, separated by commas to the `print` function. As arguments can be of different types, hence the `print` function accepts integer (16) along with strings here. But in case the `print` statement has values of different types and `+` is used instead of comma, it will generate an error.

### TYPE CONVERSION

Consider the following program

```
num1 = input("Enter a number and I'll double it: ")
num1 = num1 * 2
print(num1)
```

The program was expected to display double the value of the number received and store in variable num1. So if a user enters 2 and expects the program to display 4 as the output, the program displays the following result:

```
Enter a number and I'll double it: 2
22
```

This is because the value returned by *the input function is a string ("2") by default*. As a result, in statement `num1 = num1 * 2`, num1 has string value and `*` acts as repetition operator which results in output as "22". To get 4 as output, we need to convert the data type of the value entered by the user to integer. Thus, we modify the program as follows:

```
num1 = input("Enter a number and I'll double it: ")
num1 = int(num1) #convert string input to integer
num1 = num1 * 2
print(num1)
```

Now, the program will display the expected output as follows:

```
Enter a number and I'll double it: 2
4
```

As and when required, we can change the data type of a variable in Python from one type to another. Such data type conversion can happen in two ways: either explicitly (forced) when the programmer specifies for the interpreter to convert a data type to another type; or implicitly, when the interpreter understands such a need by itself and does the type conversion automatically.

### Explicit Conversion

Explicit conversion, also called type casting happens when data type conversion takes place because the programmer forced it in the program. The general form of an explicit data type conversion is:

(new\_data\_type) (expression)

With explicit type conversion, there is a risk of loss of information since we are forcing an expression to be of a specific type. For example, converting a floating value of `x = 2 0.67` into an integer type, i.e., `int(x)` will discard the fractional part `.67`. Following are some of the functions in Python that are used for explicitly converting an expression or a variable to a different type.

#### Explicit type conversion functions in Python

Function	Description
<code>int(x)</code>	Converts x to an integer
<code>float(x)</code>	Converts x to a floating-point number

<code>str(x)</code>	Converts x to a string representation
<code>chr(x)</code>	Converts ASCII value of x to character
<code>ord(x)</code>	returns the character associated with the ASCII code x

Program of explicit type conversion from int to float.

```
#Explicit type conversion from int to float
```

```
num1 = 10
num2 = 20
num3 = num1 + num2
print(num3)
print(type(num3))
num4 = float(num1 + num2)
print(num4)
print(type(num4))
```

Output:

```
30
<class 'int'>
30.0
<class 'float'>
```

Example of type conversion between numbers and strings.

```
#Type Conversion between Numbers and Strings
```

```
priceIcecream = 25
priceBrownie = 45
totalPrice = priceIcecream + priceBrownie
print("The total is Rs." + totalPrice )
```

On execution, above program gives an informing that the interpreter cannot convert an integer value to string implicitly. It may appear quite intuitive that the program should convert the integer value to a string depending upon the usage. However, the interpreter may not decide on its own when to convert as there is a risk of loss of information. Python provides the mechanism of the explicit type conversion so that one can clearly state the desired outcome. Program works perfectly using explicit type casting:

Program Program to show explicit type casting.

```
#Explicit type casting
```

```
priceIcecream = 25
priceBrownie = 45
totalPrice = priceIcecream + priceBrownie
print("The total in Rs." + str(totalPrice))
```

Output:

```
The total in Rs.70
```

Program to show explicit type conversion.

```
#Explicit type conversion
icecream = '25'
brownie = '45'
#String concatenation
price = icecream + brownie
print("Total Price Rs." + price)
```

```
#Explicit type conversion - string to integer
price = int(icecream)+int(brownie)
print("Total Price Rs." + str(price))
Output:
Total Price Rs.2545
Total Price Rs.70
```

### Implicit Conversion

Implicit conversion, also known as coercion, happens when data type conversion is done automatically by Python and is not instructed by the programmer.

Program to show implicit conversion from int to float.

```
#Implicit type conversion from int to float
num1 = 10 #num1 is an integer
num2 = 20.0 #num2 is a float
sum1 = num1 + num2 #sum1 is sum of a float and an integer
print(sum1)
print(type(sum1))
```

```
Output:
30.0
<class 'float'>
```

In the above example, an integer value stored in variable `num1` is added to a float value stored in variable `num2`, and the result was automatically converted to a float value stored in variable `sum1` without explicitly telling the interpreter. This is an example of implicit data conversion. Why was the float value not converted to an integer instead? This is due to type promotion that allows performing operations (whenever possible) by converting data into a wider-sized data type without any loss of information.