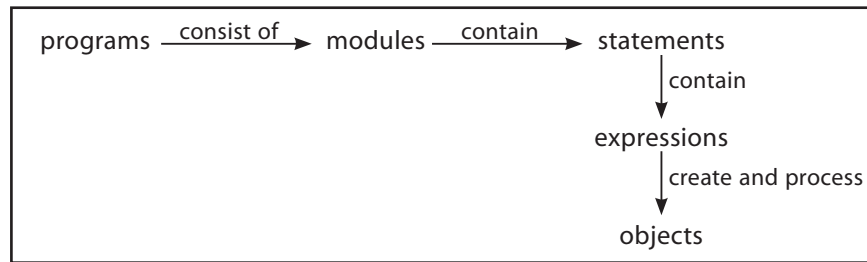
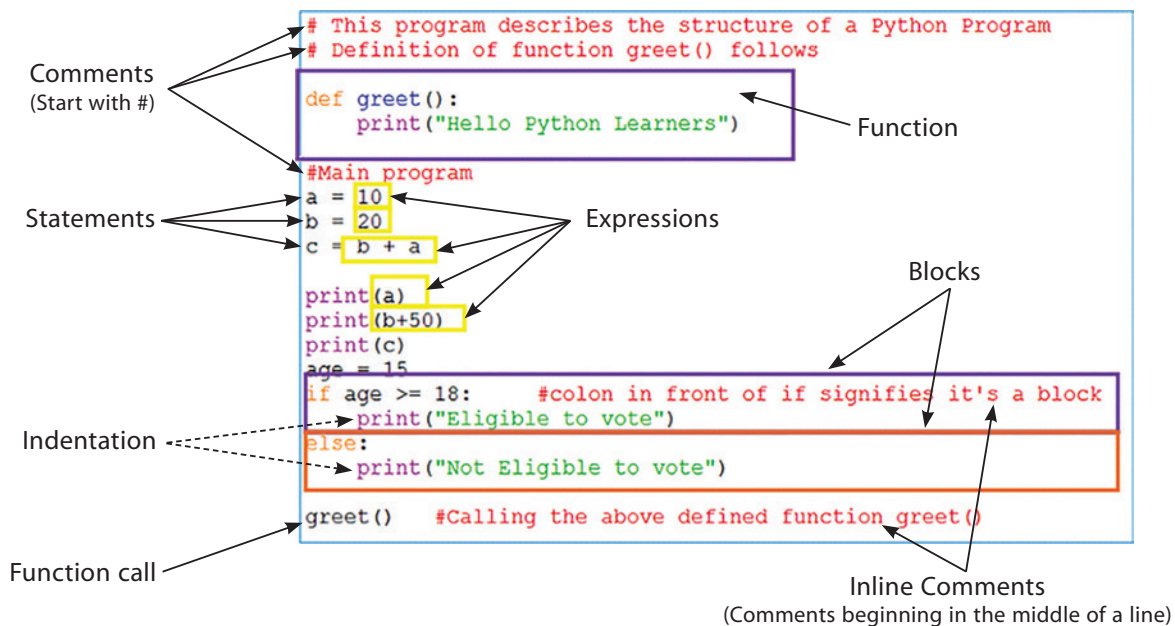


## 1.2 STRUCTURE OF A PYTHON PROGRAM

A Python program constitutes several elements such as statements, expressions, functions, comments, etc., which are synchronized in the manner as shown below:



Let us see the several components of a basic Python program.



As shown in the snippet given above, the several components that a Python program holds are:

- **Expressions:** An expression **represents** something, like a number, a string, or an element. Any value is an expression.
- **Statements:** Anything that **does something** is a statement. Any assignment to a variable or function call is a statement. Any value contained in that statement is an expression.
- **Comments:** Comments are the additional information provided against a statement or a chunk of code for the better clarity of the code. Interpreter ignores the comments and does not count them in commands.

Symbols used for writing comments include Hash (#) or Triple Double Quotation marks (""").

**Hash (#)** is used in writing **single-line comments** that do not span multiple lines. **Triple Quotation Marks (""" or """)** are used to write **multiple-line comments**. Three triple quotation marks to start the comment and again three quotation marks to end the comment.

- **Functions:** Function is a set of instructions defined under a particular name, which once written can be called whenever and wherever required.
- **Block(s):** A block refers to a group of statements which are part of another statement or function. All statements inside a block are indented at the same level.

## 1.3 VARIABLES AND DATA TYPES

A variable is like a container that stores values you can access or change. The purpose of using variables is to allow the stored values to be used later on. We have learnt that any object or variable in Python is a name that refers to a value at a particular memory location and possesses three components:

- **A Value:** It represents any number or a letter or a string. To assign any value to a variable, we use assignment operator (=).
- **An Identity:** It refers to the address of the variable in memory which does not change once created. To retrieve the address (identity) of a variable, the command used is:

```
>>>id(variable_name)
```

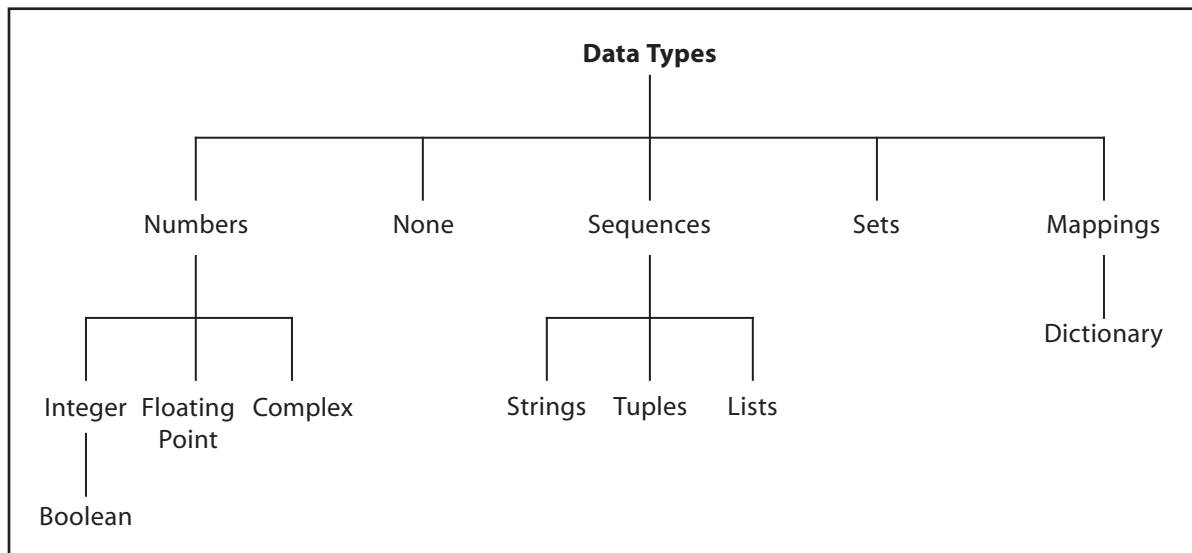
- **A Type:** We are not required to explicitly declare a variable with its type. Whenever we declare a variable with some value, Python automatically allocates the relevant data type associated with it.

Hence, the data type of a variable is according to the value it holds.

For example, `>>> x = 20`

The above statement signifies 'x' to be of integer type since it has been assigned an integer value 20.

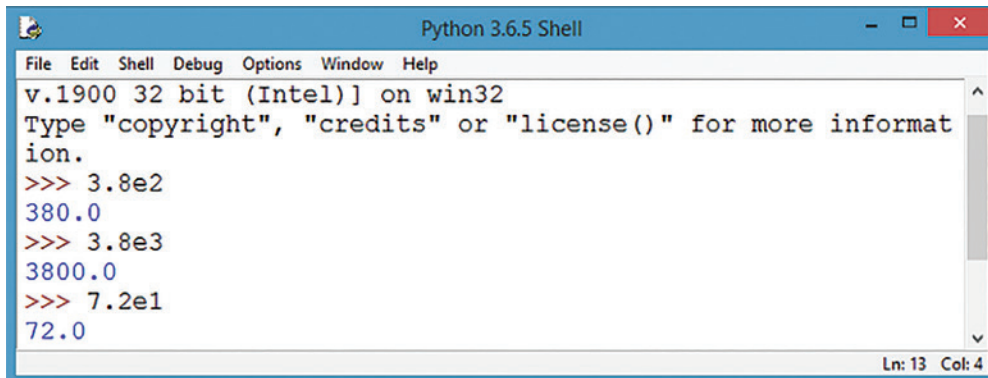
Data types are classified as follows (Fig.1.2):



**Fig. 1.2:** Classification of Data Types in Python

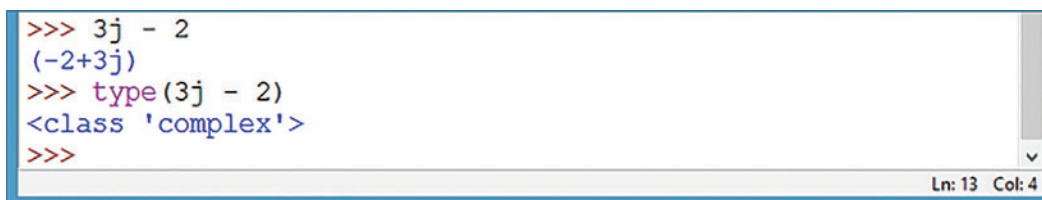
1. **Number or Numeric Data Type:** Numeric data type is used to store numeric values. It is further classified into three subtypes:
  - (a) **Integer and Long:** To store whole numbers, *i.e.*, decimal digits without a fraction part. They can be positive or negative. **Examples:** 566, -783, -3, 44, etc.

- (b) **Float/Floating Point:** Floating point numbers signify real numbers. This data type is used to store numbers with a fraction part. They can be represented in scientific notation where the uppercase or lowercase letter 'e' signifies the 10<sup>th</sup> power:



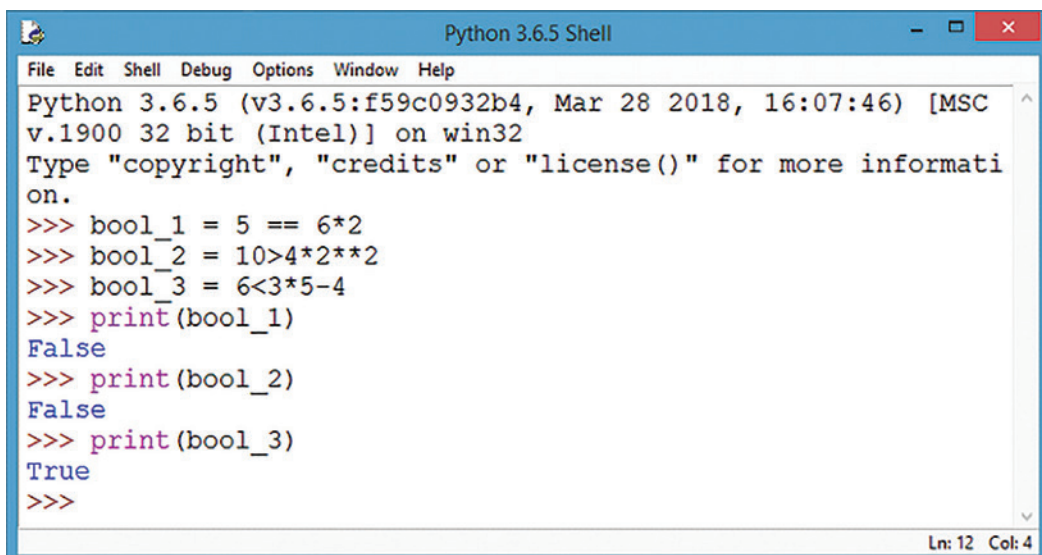
```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more informat
ion.
>>> 3.8e2
380.0
>>> 3.8e3
3800.0
>>> 7.2e1
72.0
Ln: 13 Col: 4
```

- (c) **Complex Numbers:** Complex numbers are pairs of real and imaginary numbers. They take the form 'a + bj', where 'a' is the float and 'b' is the real part of the complex number.



```
>>> 3j - 2
(-2+3j)
>>> type(3j - 2)
<class 'complex'>
>>>
Ln: 13 Col: 4
```

- (d) **Boolean:** Boolean data type is used in situations where comparisons to be made always result in either a true or a false value.



```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC
v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more informati
on.
>>> bool_1 = 5 == 6*2
>>> bool_2 = 10>4*2**2
>>> bool_3 = 6<3*5-4
>>> print(bool_1)
False
>>> print(bool_2)
False
>>> print(bool_3)
True
>>>
Ln: 12 Col: 4
```

2. **None:** This is a special data type with an unidentified value. It signifies the absence of value in a situation, represented by None. Python doesn't display anything when we give a command to display the value of a variable containing value as None.
3. **Sequence:** A sequence is an ordered collection of items, indexed by integers (both positive as well as negative). The three types of sequence data types available in Python are Strings, Lists and Tuples, which we will discuss in successive topics.



4. **Sets:** Set is an unordered collection of values of any type with no duplicate entry. It is immutable.
5. **Mappings:** This data type is unordered and mutable. Dictionaries in Python fall under Mappings. A dictionary represents data in key-value pairs and accessed using keys, which are immutable. Dictionary is enclosed in curly brackets ({ }).

### 1.3.1 Dynamic Typing



One of the salient features of Python is dynamic typing. It refers to declaring a variable multiple times with values of different data types as and when required. It allows you to redefine a variable with different data types such as numeric, string, etc.

*For example, consider the statement:*

```
x = 20
```

The above statement signifies a variable 'x' of integer type as it holds an integer value. Now, suppose later in the program, you re-assign a value of different type to variable 'x' then, Python shall generate no error and allows the re-assignment with different set of values. *For example,*

```
x = 20
```

```
print(x)
```

```
x = "Computer Science"
```

```
print(x)
```

```
x = 3.276
```

```
print(x)
```

The above code on execution shall display the output as:

```
>>>20
```

```
>>>Computer Science
```

```
>>>3.276
```

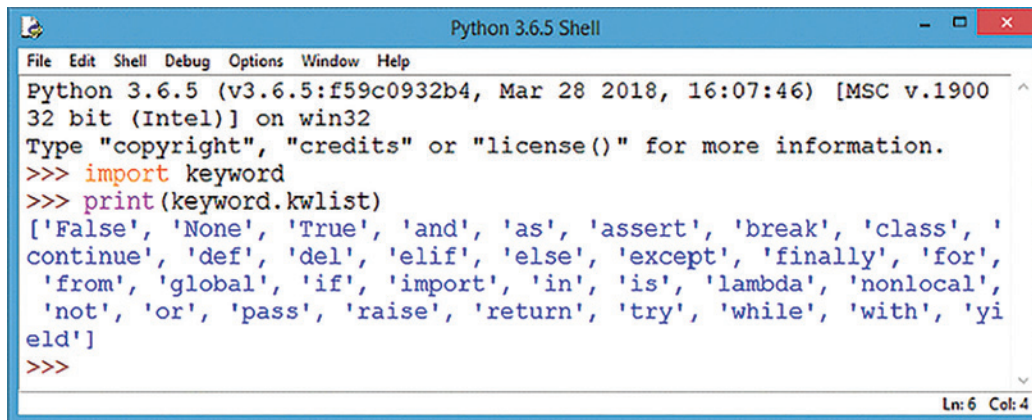
In the above example, we have assigned three different values to the variable 'x' with different types. This process is referred to as **Dynamic typing**.

**CTM:** Each and every element in Python is referred to as an object.

## 1.4 KEYWORDS

Keywords are the reserved words used by a Python interpreter to recognize the structure of a program. As these words have specific meanings for the interpreter, they cannot be used as variable names or for any other purpose. For checking/displaying the list of keywords available in Python, we have to write the following two statements:

```
import keyword  
print(keyword.kwlist)
```



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900
32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import keyword
>>> print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', '
continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for',
'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal',
'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yi
eld']
>>>
```

Fig. 1.3: Keywords in Python

**CTM:** All these keywords are in small letters, except for False, None, True, which start with capital letters.

## 1.5 MUTABLE AND IMMUTABLE TYPES

In certain situations, we may require changing or updating 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 under:

- **Examples of mutable objects:** list, dictionary, set, etc.
- **Examples of immutable objects:** int, float, complex, bool, string, tuple, etc.

For example, int is an immutable type which, once created, cannot be modified.

Consider a variable 'x' of integer type:

```
>>>x = 5
```

This statement will create a value 5 referenced by x.

x → 5

Now, we create another variable 'y' which is copy of the variable 'x'.

```
>>>y = x
```

The above statement will make y refer to value 5 of x. We are creating an object of type int. Identifiers x and y point to the same object.

x → 5  
y → 5

Now, we give another statement as:

```
>>>x = x + y
```

The above statement shall result in adding up the value of x and y and assigning to x.



Thus, x gets rebuilt to 10.

$x \longrightarrow 10$

$y \longrightarrow 5$

The object in which x was tagged is changed. Object x = 5 was never modified. **An immutable object doesn't allow modification after creation. Another example of immutable object is a string.**

```
>>>str = "strings immutable"
>>>str[0] = 'p'
>>>print(str)
```

This statement shall result in TypeError on execution.

**TypeError: 'str' object does not support item assignment.**

This is because of the fact that strings are immutable. On the contrary, a mutable type object such as a list can be modified even after creation, whenever and wherever required.

```
new_list = [10, 20, 30]
print(new_list)
```

Output:

```
[10, 20, 30]
```

Suppose we need to change the first element in the above list as:

```
new_list = [10, 20, 30]
new_list[0] = 100
```

print(new\_list) will make the necessary updating in the list new\_list and shall display the output as:

```
[100, 20, 30]
```

This operation is successful since lists are mutable.

Python handles mutable and immutable objects differently. Immutable objects are quicker to access than mutable objects. Also, immutable objects are fundamentally expensive to “change” because doing so involves creating a copy. Changing mutable objects is cheap.

## 1.6 OPERATORS AND OPERANDS

Python allows programmers to manipulate data or operands through operators. Operators are the symbols that operate upon these operands to form an expression. Operators available in Python are categorized as follows:

**Table 1.1:** Operators in Python

|                                    |
|------------------------------------|
| Arithmetic Operators               |
| Assignment Operators               |
| Relational or Comparison Operators |
| Logical Operators                  |
| Identity Operators                 |
| Bitwise Operators                  |
| Membership Operators               |



- **Arithmetic/Mathematical Operators:** + (addition), - (subtraction), \* (multiplication), / (Division), \*\* (Exponent), % (Modulus), // (Floor Division).

| Operator   | Description  | Example(s)  |
|------------|--|---|
| + (unary)  | To explicitly express a positive number  | Value of +3 is +3   |
| - (unary)  | To represent a negative number   | Value of -3 is -3   |
| + (binary) | To add two numbers   | Value of 23+3.5 is 26.5   |
| - (binary) | To subtract one value from the other   | Value of 45 - 32 is 13  |
| *          | To find the product of two numbers   | Value of 3.2*6 is 19.2  |
| /          | To find the quotient when one value is divided by the other.   | <ul style="list-style-type: none"> <li>Value of 3/2 is 1.5</li> <li>Value of -3/2 is -1.5</li> <li>Value of 10.0/3 is 3.3333333333333335</li> </ul>                           |
| //         | (Floor division) To find the integer part of the quotient when one number is divided by the other. The result is always the largest integer less than or equal to the actual quotient. | <ul style="list-style-type: none"> <li>Value of 3//2 is 1</li> <li>Value of -3//2 is -2</li> <li>Value of 10.0//3 is 3.0</li> </ul>   |
| %          | (Remainder) To find the remainder when one value is divided by the other.  | <ul style="list-style-type: none"> <li>Value of 3%2 is 1</li> <li>Value of 10%6 is 4</li> <li>Value of 6%10 is 6</li> <li>Value of 10.3%3.2 is 0.70000000000000002</li> </ul> |
| **         | (Exponent) To raise a number to the power of another number. The expression a**b is used to find the value of a <sup>b</sup> .   | <ul style="list-style-type: none"> <li>Value of 3**2 is 9</li> <li>Value of 3**-2 is 0.11111111111111111</li> <li>Value of 10.2**3.1 is 1338.6299344200029</li> </ul>         |

- **Assignment Operators:** = (Assignment), += (Add and Assign), -= (Subtract and Assign), \*= (Multiply and Assign), /= (Divide and Assign Quotient), \*\*= (Exponent and Assign), %= (Divide and Assign Remainder), //= (Floor division and Assign).

| Operator                    | Description  | Example  |
|-----------------------------|--|--|
| +=<br>(Add and Assign)      | Evaluates R-value and adds it to L-value. The final result is assigned to L-value.         | <pre>x=5 y=10 x+=2*y print("x=",x,"and y=",y) gives the result: x= 25 and y= 10</pre>  |
| -=<br>(Subtract and Assign) | Evaluates R-value and subtracts it from L-value. The final result is assigned to L-value.  | <pre>x=5 y=10 x-=2*y print("x=",x,"and y=",y) gives the result: x= -15 and y= 10</pre> |
| *=<br>(Multiply and Assign) | Evaluates R-value and multiplies it with L-value. The final result is assigned to L-value. | <pre>x=5 y=10 x*=2*y print("x=",x,"and y=",y) gives the result: x= 100 and y= 10</pre> |





|  |   |  |
|--|---|--|
| <code>/=</code><br>(Divide and Assign Quotient)  | Evaluates R-value and divides the L-value with R-value. The quotient is assigned to L-value.                  | <code>x=5</code><br><code>y = 10</code><br><code>x /= y</code><br><code>print("x=", x, "and y=", y)</code><br>gives the result: <code>x= 0.5</code> and <code>y= 10</code> |
| <code>%=</code><br>(Divide and Assign Remainder) | Evaluates R-value and divides the L-value with R-value. The remainder is assigned to L-value.                 | <code>x=5</code><br><code>x%= 4</code><br><code>print("x=", x)</code><br>gives the result: <code>x=1</code>  |
| <code>//=</code><br>(Floor division and Assign)  | Evaluates R-value and divides (floor division) the L-value with R-value. The quotient is assigned to L-value. | <code>x=5</code><br><code>x //= 4</code><br><code>print("x=", x)</code><br>gives the result: <code>x=1</code>  |
| <code>**=</code><br>(Exponent and Assign)        | Evaluates R-value and calculates (L-value) <sup>R-value</sup> . The final result is assigned to L-value.      | <code>x=5</code><br><code>x **= 4</code><br><code>print("x=", x)</code><br>gives the result: <code>x= 625</code>   |

- **Relational/Comparison Operators:** `>` (greater than), `<` (less than), `>=` (greater than or equal to), `<=` (less than or equal to), `==` (equal to), `!=` (not equal to).

| Operator   | Description   | Example (assuming <code>x=6, y=2</code> ) |
|--|---|---|
| <code>==</code><br>(Equality)                    | Compares two values for equality. Returns True if they are equal, otherwise returns False.                        | <code>(x==y)</code> returns<br>False      |
| <code>!=</code><br>(Inequality)                  | Compares two values for inequality. Returns True if they are unequal, otherwise returns False.                    | <code>(x!=y)</code> returns<br>True       |
| <code>&lt;</code><br>(Less than)                 | Compares two values. Returns True if first value is less than the second, otherwise returns False.                | <code>(x&lt;y)</code> returns<br>False    |
| <code>&lt;=</code><br>(Less than or equal to)    | Compares two values. Returns True if first value is less than or equal to the second, otherwise returns False.    | <code>(x&lt;=y)</code> returns<br>False   |
| <code>&gt;</code><br>(Greater than)              | Compares two values. Returns True if first value is greater than the second, otherwise returns False.             | <code>(x&gt;y)</code> returns<br>True     |
| <code>&gt;=</code><br>(Greater than or equal to) | Compares two values. Returns True if first value is greater than or equal to the second, otherwise returns False. | <code>(x&gt;=y)</code> returns<br>True    |

- **Logical Operators:** `or`, `and`, `not`.

| Operator         | Description  | Example (assuming <code>x=6, y=2</code> )             |
|------------------|--|---|
| <code>not</code> | Negates a condition and returns True if the condition is false, otherwise returns False.                     | <code>not(x &gt; y)</code> returns<br>False           |
| <code>and</code> | Combines two conditions and returns True if both the conditions are true, otherwise returns False.           | <code>(x &gt; 3 and y &lt; 2)</code> returns<br>False |
| <code>or</code>  | Combines two conditions and returns True if at least one of the conditions is true, otherwise returns False. | <code>(x &gt; 3 or y &lt; 2)</code> returns<br>True   |



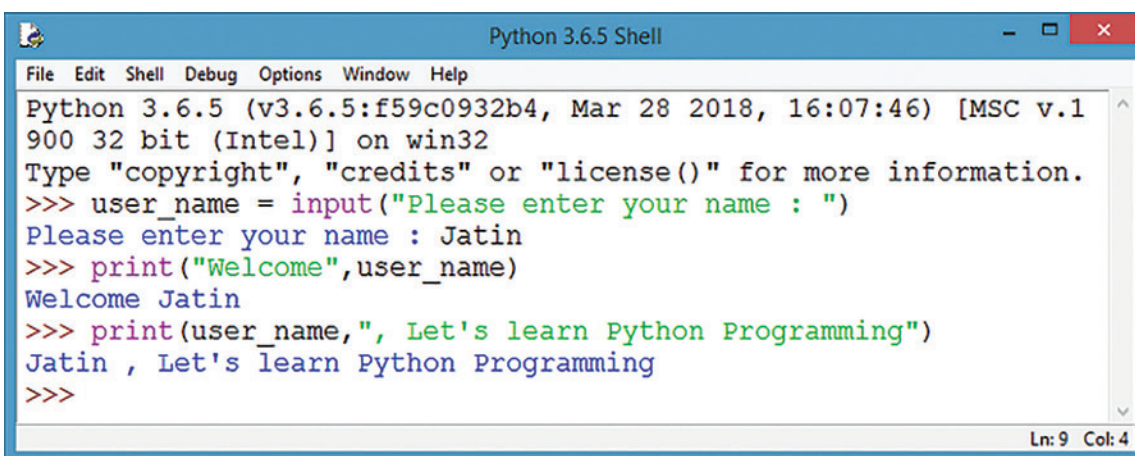
## 1.7 INPUT AND OUTPUT (PYTHON'S BUILT-IN FUNCTIONS)

In order to provide the required set of values, data is to be fetched from a user to accomplish the desired task. Thus, Python provides the following I/O (Input-Output) built-in library functions:

1. **input():** The input() function accepts and returns the user's input as a string and stores it in the variable which is assigned with the assignment operator. It is important to remember that while working with input(), the input fetched is always a string; so, in order to work with numeric values, use of appropriate conversion function (int) becomes mandatory.

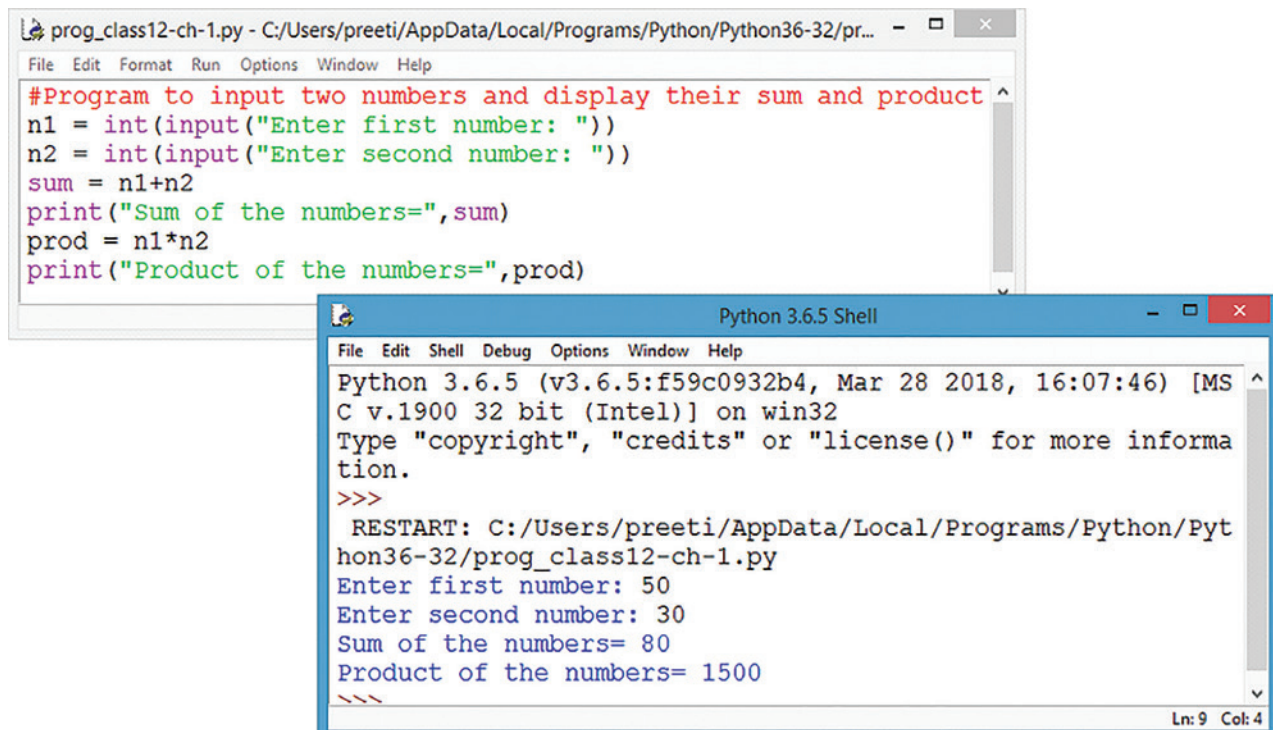
The input() function takes one string argument (called prompt). During execution, input() shows the prompt to the user and waits for the user to input a value from the keyboard. When the user enters a value, input() returns this value to the script. In almost all the cases, we store this value in a variable.

**Example 3:** Display a welcome message to the user.



```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1
900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> user_name = input("Please enter your name : ")
Please enter your name : Jatin
>>> print("Welcome",user_name)
Welcome Jatin
>>> print(user_name," , Let's learn Python Programming")
Jatin , Let's learn Python Programming
>>>
```

**Example 4:** Input two numbers from the user and display their sum and product.



```
prog_class12-ch-1.py - C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/pr...
File Edit Format Run Options Window Help
#Program to input two numbers and display their sum and product
n1 = int(input("Enter first number: "))
n2 = int(input("Enter second number: "))
sum = n1+n2
print("Sum of the numbers=",sum)
prod = n1*n2
print("Product of the numbers=",prod)

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MS
C v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more informa
tion.
>>>
RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Pyt
hon36-32/prog_class12-ch-1.py
Enter first number: 50
Enter second number: 30
Sum of the numbers= 80
Product of the numbers= 1500
>>>
```

In the above program, we have used the **int() method**. `int()` takes a number, expression or a string as an argument and returns the corresponding integer value. `int()` method behaves as per the following criteria:

- (a) If the argument is an integer value, `int()` returns the same integer. *For example, `int(12)` returns 12.*
  - (b) If the argument is an integer expression, the expression is evaluated and `int()` returns this value. *For example, `int(12+34)` returns 46.*
  - (c) If the argument is a float number, `int()` returns the integer part (before the decimal point) of the number. *For example, `int(12.56)` returns 12.*
2. **eval()**: `eval()` method takes a string as an argument, evaluates this string as a number, and returns the numeric result (int or float as the case may be). If the given argument is not a string or if it cannot be evaluated as a number, then `eval()` results in an error.

### 1.7.1 Type Casting (Explicit Conversion)

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 into 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 deliberately, *i.e.*, the programmer forces 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 data loss since we are forcing an expression to be of a specific type.

*For example, converting a floating value of `x = 50.75` into an integer type, *i.e.*, `int(x)` will discard the fractional part .75 and shall return the value as 50.*

```
>>> x = 50.75
>>> print(int(x))
50
```

Following are some of the functions in Python that are used for explicitly converting an expression or a variable into a different type.

**Table 1.2:** Explicit type conversion functions in Python

| Function               | Description                              |
|------------------------|--|
| <code>int(x)</code>    | Converts x into an integer.              |
| <code>float(x)</code>  | Converts x into a floating-point number. |
| <code>str(x)</code>    | Converts x into a string representation. |
| <code>chr(x)</code>    | Converts x into a character.             |
| <code>unichr(x)</code> | Converts x into a Unicode character.     |



- Python supports dynamic typing, *i.e.*, a variable can hold values of different types at different times.
- A function is a named block of statements that can be invoked by its name.
- The input() function evaluates the data input and takes the result as numeric type.
- The if statement is a decision-making statement.
- The looping constructs while and for statements allow sections of code to be executed repeatedly.
- for statement iterates over a range of values or a sequence.
- The statements within the body of a while loop are executed over and over again until the condition of the while becomes false or remains true.
- A string is a sequence of characters.
- We can create strings simply by enclosing characters in quotes (single, double or triple).
- Positive subscript helps in accessing the string from the beginning.
- Negative subscript helps in accessing the string from the end.
- '+' operator joins or concatenates the strings on both sides of the operator.
- The \* operator creates a new string concatenating multiple copies of the same string.
- List is a sequence data type.
- A list is a mutable sequence of values which can be of any type and are indexed by integer.
- A list is created by placing all the items (elements) inside a square bracket [], separated by commas.
- A list can even have another list as an item. This is called nested list.
- Another way of creating tuple is built-in function list().
- Traversing a list means accessing each element of a list. This can be done by using looping statement, either for or while.
- List slicing allows us to obtain a subset of items.
- copy() creates a list from another list. It does not take any parameter.
- A tuple is an immutable sequence of values which can be of any type and are indexed by an integer.
- Creating a tuple is as simple as putting values separated by a comma. Tuples can be created using parentheses ().
- To create a tuple with a single element, the final comma is necessary.
- Python provides various operators like '+', '\*', 'in', 'not in', etc., which can be applied to tuples.
- In a dictionary, each key maps a value. The association of a key and a value is called a key-value pair.
- To create a dictionary, key-value pairs are separated by a comma and are enclosed in two curly braces {}. In key-value pair, each key is separated from its value by a colon (:).
- We can add new elements to an existing dictionary, extend it with single pair of values or join two dictionaries into one.
- We can also update a dictionary by modifying an existing key-value pair or by merging another dictionary with an existing one.
- Python provides us with a number of dictionary methods like: len(), pop(), items(), keys(), values(), get(), etc.
- keys() method in Python dictionary returns an object that displays a list of all the keys in the dictionary.
- Bubble sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order.
- Insertion sort is an in-place sorting algorithm.
- In Insertion sort, an element gets compared and inserted into the correct position in the list.

## OBJECTIVE TYPE QUESTIONS

### 1. Fill in the blanks.

- (a) ..... is the Python operator responsible for declaring variables.
- (b) The built-in function randrange() belongs to ..... module.

- (c) A ..... operator does not directly operate on data but produces a left-to-right evaluation of expression.
- (d) median() method belongs to ..... module in Python.
- (e) The reserved words in Python are called ..... and these cannot be used as names or identifiers.
- (f) An ..... is a symbol used to perform an action on some value.
- (g) A file that contains a collection of related functions and other definitions is called .....
- (h) The modules in Python have the ..... extension.
- (i) A ..... is just a module that contains some useful definitions.
- (j) Each object in Python has three key attributes—a ....., a ..... and an .....
- (k) In Python, the non-zero value is treated as ..... and zero value is treated as .....
- (l) Keys of a dictionary must be .....
- (m) In ....., the adjoining values in a sequence are compared and exchanged repeatedly until the entire array is sorted.
- (n) Logical operators are used to combine two or more ..... expressions.
- (o) The ..... function returns the length of a specified list.

**Answers:** (a) Assignment (=) operator      (b) random      (c) comma (,)  
 (d) statistics      (e) keywords      (f) operator  
 (g) module      (h) .py      (i) library  
 (j) type, value, id      (k) true, false      (l) unique  
 (m) Bubble sort      (n) relational      (o) len()

## 2. State whether the following statements are True or False.

- (a) The two statements `x = int(22.0/7)` and `x = int(22/7.0)` yield the same results.
- (b) The given statement: `x + 1 = x` is a valid statement.
- (c) List slice is a list in itself.
- (d) Relational operators return either true or false.
- (e) `break`, `continue`, `pass` are the three conditional statements.
- (f) The % (modulus) operator cannot be used with the float data type.
- (g) The `range()` function is used to specify the length of a for-in loop.
- (h) Assignment operator can be used in place of equality operator in the test condition.
- (i) Comments in Python begin with a “\$” symbol.
- (j) In `print()` function, if you use a concatenate operator (+) between two strings, both the strings are joined with a space in between them.
- (k) If we execute Python code using prompt “>>>” then we call it an interactive interpreter.
- (l) Lists are immutable while strings are mutable.
- (m) The keys of a dictionary must be of immutable types.
- (n) Lists and strings in Python support two-way indexing.
- (o) Tuples can be nested and can contain other compound objects like lists, dictionaries and other tuples.

**Answers:** (a) True      (b) False      (c) True      (d) True      (e) False      (f) True  
 (g) True      (h) False      (i) False      (j) False      (k) True      (l) False  
 (m) True      (n) True      (o) True

## 3. Multiple Choice Questions (MCQs)

- (a) Which of the following is not considered a valid identifier in Python?
  - (i) two2      (ii) \_main      (iii) hello\_rsp1      (iv) 2 hundred
- (b) What will be the output of the following code— `print("100+200")`?
  - (i) 300      (ii) 100200      (iii) 100+200      (iv) 200





- (c) Which amongst the following is a mutable datatype in Python?  
(i) int (ii) string (iii) tuple (iv) list
- (d) pow() function belongs to which library?  
(i) math (ii) string (iii) random (iv) maths
- (e) Which of the following statements converts a tuple into a list?  
(i) len(string) (ii) list(string) (iii) tup(list) (iv) dict(string)
- (f) The statement: bval = str1 > str2 shall return ..... as the output if two strings str1 and str2 contains "Delhi" and "New Delhi".  
(i) True (ii) Delhi (iii) New Delhi (iv) False
- (g) What will be the output generated by the following snippet?  
a = [5,10,15,20,25]  
k = 1  
i = a[1] + 1  
j = a[2] + 1  
m = a[k+1]  
print(i, j, m)  
(i) 11 15 16 (ii) 11 16 15 (iii) 11 15 15 (iv) 16 11 15
- (h) The process of arranging the array elements in a specified order is termed as:  
(i) Indexing (ii) Slicing (iii) Sorting (iv) Traversing
- (i) Which of the following Python functions is used to iterate over a sequence of numbers by specifying a numeric end value within its parameters?  
(i) range() (ii) len() (iii) substring() (iv) random()
- (j) What is the output of the following?  
d = {0: 'a', 1: 'b', 2: 'c'}  
for i in d:  
    print(i)  
(i) 0 (ii) a (iii) 0 (iv) 2  
    1 b a 2  
    2 c 1 b 2  
    c 2 b 2  
    c c
- (k) What is the output of the following?  
x = 123  
for i in x:  
    print(i)  
(i) 1 2 3 (ii) 123 (iii) error (iv) infinite loop
- (l) Which arithmetic operators cannot be used with strings?  
(i) + (ii) \* (iii) - (iv) All of the above
- (m) What will be the output when the following code is executed?  
>>>str1="helloworld"  
>>>str1[:-1]  
(i) dlrowolleh (ii) hello (iii) world (iv) helloworld
- (n) What is the output of the following statement?  
print("xyyxyzxyzxyy".count('yy', 1))  
(i) 2 (ii) 0 (iii) 1 (iv) Error