Recall (from 'Literals/Values' discussion in chapter 6) that fractional numbers can be written in two forms :

(i) **Fractional Form** (*Normal Decimal Notation*) *e.g.*, 3500.75, 0.00005, 147.9101 etc.

(ii) **Exponent Notation** *e.g.*, 3500.75E03, 0.5E-04, 1.479101E02 etc.

Floating point variables represent real numbers, which are used for measurable quantities like distance, area, temperature etc. and typically have a fractional part.

Floating point numbers have *two* **advantages** over integers :

- They can represent values between the integers.
- They can represent a much greater range of values.

But floating-point numbers suffer from one disadvantage also :

- Floating-point operations are usually slower than integer operations.

In Python, floating point numbers represent machine-level **double precision floating point numbers**[2] **(15 digit precision)**. The range of these numbers is limited by underlying machine architecture subject to available (virtual) memory.

NOTE

In Python, the floating point numbers have precision of 15 digits (double-precision).

8.1C Complex Numbers

Python is a versatile language that offers you a numeric type to represent *Complex Numbers* also. Complex Numbers ? Hey, don't you know about *Complex numbers* ? Uhh, I see. You are going to study about *Complex numbers* in class XI Mathematics book. Well, if you don't know anything about complex numbers, then for you to get started, I am giving below brief introduction of *Complex numbers* and then we shall talk about *Python's representation of Complex numbers*.

Mathematically, a complex number is a number of the form $A + Bi$ where i is the imaginary number, ***equal to the square root of*** -1 i.e., $\sqrt{-1}$.

A complex number is made up of both **real and imaginary components**. In complex number $A + Bi$, A and B are real numbers and i is imaginary. If we have a complex number z, where $z = a + bi$ then a would be the *real component* and b would represent the *imaginary component* of z, *e.g.*, real component of $z = 4 + 3i$ is 4 and the imaginary component would be 3.

NOTE

A complex number is in the form $A + Bi$ where i is the imaginary number, equal to the square root of -1 i.e., $\sqrt{-1}$, that is $i^2 = -1$

Check Point 8.1

1. What are the built-in core data types of Python ?
2. What do you mean by Numeric types ? How many numeric data types does Python provide ?
3. What will be the data types of following two variables ?

   A = 2147483647

   B = A + 1

   (Hint. Carefully look the values they are storing. You can refer to range of Python number table.)
4. What are Boolean numbers ? Why are they considered as a type of integers in Python ?

2. As per Python documentation, "Python does not support single-precision floating point numbers ; the savings in processor and memory usage that are usually the reason for using these is dwarfed by the overhead of using objects in Python, so there is no reason to complicate the language with two kinds of floating point numbers".

Complex Numbers in Python

Python represents complex numbers in the form $A + Bj$. That is, to represent imaginary number Python uses j (or J) in place of traditional i. So in Python $j = \sqrt{-1}$. Consider following examples where a and b are storing two complex numbers in Python

```
a = 0 + 3.1j
b = 1.5 + 2j
```

The above complex number a has real component as 0 and imaginary component as 3.1. In complex number b, the real part is 1.5 and imaginary part is 2. When you display complex numbers, Python **displays complex numbers in parentheses when they have a nonzero real part** as shown in following examples.

NOTE

Python represents complex numbers as a pair of floating point numbers.

NOTE

Complex numbers are quite commonly used in Electrical Engineering. In the field of electricity, however, because the symbol i is used to represent current, they use the symbol j for the square root of -1. Python adheres to this convention.

```
>>> c = 0 + 4.5j
>>> d = 1.1 + 3.4j
>>> c
4.5j
>>> d
(1.1 + 3.4j)
>>> print (c)
4.5j
>>> print (d)
(1.1 + 3.4j)
```

See, a complex number with non-zero real part is displayed with parentheses around it.

But no parentheses around complex number with real part as zero.

Unlike Python's other numeric types, **complex numbers** are a composite quantity made of two parts : *the real part* and the *imaginary part*, both of which are represented internally as *float* values (floating point numbers).

You can retrieve the two components using attribute references. For a complex number `z` :

- **z.real** gives the *real part*.
- **z.imag** gives the *imaginary part* as a float, not as a complex value.

For example,

```
>>> z = (1 + 2.56j) + (-4 -3.56j)
>>> a
(-3 -1j)
>>> z.real
-3.0
>>> z.imag
-1.0
```

It will display real part of complex number z

It will display imaginary part of complex number z

TIP

The real and imaginary parts of a complex number `z` can be retrieved through the read-only attributes **z.real** and **z.imag**.

Check Point 8.2

1. What are floating point numbers ? When are they preferred over integers ?
2. What are complex numbers ? How would Python represent a complex number with real part as 3 and imaginary part as - 2.5 ?
3. What will be the output of following code ?

```
p = 3j
q = p + (1 + 1.5j)
print (p)
print (q)
```

4. What will be the output of following code ?

```
r = 2.5 + 3.9j
print (r.real)
print (r.imag)
```

5. Why does Python uses symbol `j` to represent imaginary part of a complex number instead of the conventional `i` ?

Hint. *Refer note above.*

The range of numbers represented through Python's numeric data types is given below

Table 8.1 *The Range of Python Numbers*

Data type	Range
Integers	an unlimited range, subject to available (virtual) memory only
Booleans	two values **True** (1), **False** (0)
Floating point numbers	an unlimited range, subject to available (virtual) memory or underlying machine architecture
Complex numbers	Same as floating point numbers because the real and imaginary parts are represented as **floats**

8.2.2 Strings

You already know about strings (as data) in Python. In this section, we shall be talking about Python's data type string. A string data type lets you hold string data, *i.e.*, any number of valid characters into a set of quotation marks.

In Python 3.x, each character stored in a string[1] is a Unicode character. Or in other words, all strings in Python 3.x are sequences of *pure Unicode characters*. *Unicode* is a system designed to represent every character from every language. A string can hold any type of known characters *i.e., letters, numbers,* and *special characters*, of any known scripted language.

NOTE

All Python (3.x) strings store Unicode characters

Following are all legal strings in Python :

"abcd" , "1234", '$%^&', '????', "ŠÆËà" , "??????", '????', "????", '???', "??"

String as a Sequence of Characters

A Python string is a sequence of characters and each character can be individually accessed using its **index**. Let us understand this.

Let us first study the internal structure or composition of Python strings as it will form the basis of all the learning of various string manipulation concepts. Strings in Python are stored as individual characters in contiguous location, with two-way index for each location.

The individual elements of a string are the characters contained in it (stored in contiguous memory locations) and as mentioned the characters of a string are given two-way index for each location. Let us understand this with the help of an illustration as given in Fig. 8.1.

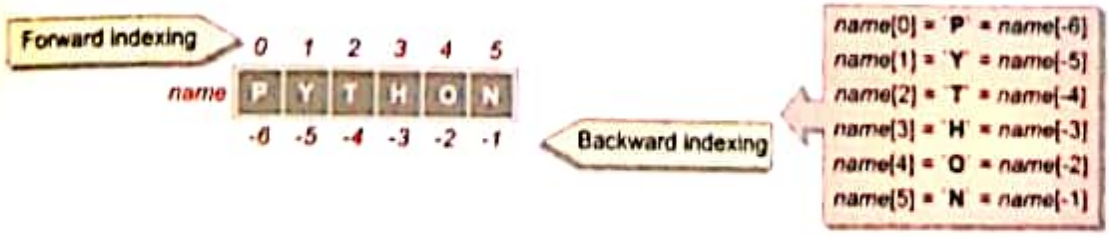

Figure 8.1 Structure of a Python String.

1. Python has no separate **character** datatype, which most other programming languages have – that can hold a single character. In Python, a character is a **string** type only, with single character.