- **Using Single Quotes ('):** For example, a string can be written as `'HELLO'`.
- **Using Double Quotes ("):** Strings in double quotes are exactly same as those in single quotes. Therefore, `'HELLO'` is same as `"HELLO"`.

Note All spaces and tabs within a string are preserved in quotes (single quote as well as double).

- **Using Triple Quotes (''' '''):** You can specify multi-line strings using triple quotes. You can use as many single quotes and double quotes as you want in a string within triple quotes.
  An example of a multi-line string can be given as,

```
'''Good morning everyone.
"Welcome to the world of 'Python'."
Happy reading.'''
```

When you print the above string in the `IDLE`, you will see that the string is printed as it is observing the spaces, tabs, new lines, and quotes (single as well as double).

You can even print a string without using the `print()` function. For this, you need to simply type the string within the quotes (single, double, or triple) as shown below.

`>>> 'Hello'`
`'Hello'`
	`>>> "HELLO"`
	`'HELLO'`
		`>>> '''HELLO'''`
		`'HELLO'`

Now, irrespective of the way in which you specify a string, the fact is that all strings are *immutable*. This means that once you have created a string, you cannot change it.

String literal concatenation

Python concatenates two string literals that are placed side by side. Consider the code below wherein Python has automatically concatenated three string literals.

```
>>> print('Beautiful Weather' '.....' 'Seems it would rain')
Beautiful Weather.....Seems it would rain
```

Unicode Strings

Unicode is a standard way of writing international text. That is, if you want to write some text in your native language like Hindi, then you need to have a Unicode-enabled text editor. Python allows you to specify Unicode text by prefixing the string with a u or U. For example,

Programming Tip: There is no char data type in Python.

```
u"Sample Unicode string."
```

Note The 'U' prefix specifies that the file contains text written in language other than English.

Escape Sequences

Some characters (`like "`, \) cannot be directly included in a string. Such characters must be escaped by placing a backslash before them. For example, let us observe what will happen if you try to print `What's your name?`

```
>>> print('What's your name?')
SyntaxError: invalid syntax
```

Can you guess why we got this error? The answer is simple. Python got confused as to where the string starts and ends. So, we need to clearly specify that this single quote does not indicate the end of the string. This indication can be given with the help of an ***escape sequence.*** You specify the single quote as \' (single quote preceded by a backslash). Let us try again.

```
>>> print('What\'s your name?')
What's your name?
```

Note An *escape sequence* is a combination of characters that is translated into another character or a sequence of characters that may be difficult or impossible to represent directly.

Similarly, to print a double quotes in a string enclosed within double quotes, you need to precede the double quotes with a backslash as given below.

```
>>> print("The boy replies, \"My name is Aaditya.\"")
The boy replies, "My name is Aaditya."
```

In previous section, we learnt that to print a multi-line string, we use triple quotes. There is another way for doing the same. You can use an escape sequence for the newline character (\n). Characters following the \n are moved to the next line. Observe the output of the following command.

```
>>> print("Today is 15th August. \n India became
independent on this day.")
Today is 15th August.
India became independent on this day.
```

Programming Tip: When a string is printed, the quotes around it are not displayed.

Another useful escape sequence is `\t` which inserts tab in a string. Consider the command given below to show how the string gets displayed on the screen.

```
>>> print("Hello All. \t Welcome to the world of Python.")
Hello All.    Welcome to the world of Python.
```

Note that when specifying a string, if a single backslash (\) at the end of the line is added, then it indicates that the string is continued in the next line, but no new line is added otherwise. For example,

```
>>> print("I have studied many programming languages. \
But my best favorite language is Python.")

I have studied many programming languages. But my best favorite language is Python.
```

The different types of escape sequences used in Python are summarized in Table 3.1

Table 3.1 Some of the escape sequences used in Python

Escape Sequence	Purpose	Example	Output
\\	Prints Backslash	`print("\\")`	\
\'	Prints single-quote	`print("\'")`	'
\"	Prints double-quote	`print("\"")`	"

Contd

Table 3.1 *Contd*

Escape Sequence	Purpose	Example	Output
`\a`	Rings bell	`print("\a")`	`Bell rings`
`\f`	Prints form feed character	`print("Hello\fWorld")`	`Hello World`
`\n`	Prints newline character	`print("Hello\nWorld")`	`Hello World`
`\t`	Prints a tab	`print( "Hello\tWorld")`	`Hello    World`
`\o`	Prints octal value	`print("\o56")`	`.`
`\x`	Prints hex value	`print("\x87")`	`+`

Raw Strings

If you want to specify a string that should not handle any escape sequences and want to display exactly as specified, then you need to specify that string as a *raw string*.

A raw string is specified by prefixing `r` or `R` to the string. Consider the code below that prints the string as it is.

```
>>> print(R "What\'s your name?")
What\'s your name?
```

String Formatting

We have already used the built-in `format()` function to format floating point numbers. The same function can also be used to control the display of strings. The syntax of `format()` function is given as,

```
format(value, format_specifier)
```

where, `value` is the value or the string to be displayed, and `format_specifier` can contain a combination of formatting options.

Example 3.2 Commands to display 'Hello' left-justified, right-justified, and center-aligned in a field width of 30 characters.

```
>>>format('Hello', '<30')
'                          Hello'
>>> format('Hello','>30')
'            Hello'

>>> format('Hello','^30')
'Hello
```

Here, the `'<'` symbol means to left justify. Similarly, to right justify the string use the `'>'` symbol and the `'^'` symbol to centrally align the string.

We have seen above that `format()` function uses blank spaces to fill the specified width. But you can also use the `format()` function to fill the width in the formatted string using any other character as shown below.

```
>>> print('Hello', format('-','-<10'),'World')
('Hello', '----------', 'World')
```

We will learn about string operations later in this chapter.