CHAPTER 3

Basics of Python Programming

• Features, History, and Future of Python • Literals, Constants, Variables, Keywords • Data Types, Comments, Indentation • Operators and Expressions • Type Conversion

3.1 FEATURES OF PYTHON

Python is an exciting and powerful language with the right combination of performance and features that makes programming fun and easy. It is a high-level, interpreted, interactive, object-oriented, and a reliable language that is very simple and uses English-like words. It has a vast library of modules to support integration of complex solutions from pre-built components.

Python is an open-source project, supported by many individuals. It is a platform-independent, scripted language, with complete access to operating system APIs. This allows users to integrate applications seamlessly to create high-powered, highly-focused applications. Python is a complete programming language with the following features.

Simple Python is a simple and a small language. Reading a program written in Python feels almost like reading English. This is in fact the greatest strength of Python which allows programmers to concentrate on the solution to the problem rather than the language itself.

Easy to Learn A Python program is clearly defined and easily readable. The structure of the program is very simple. It uses few keywords and a clearly defined syntax. This makes it easy for just anyone to pick up the language quickly.

Versatile Python supports development of a wide range of applications ranging from simple text processing to WWW browsers to games.

Free and Open Source Python is an example of an *open source software*. Therefore, anyone can freely distribute it, read the source code, edit it, and even use the code to write new (free) programs.

Note Python has been constantly improved by a community of users who have always strived hard to take it to the next level.

High-level Language When writing programs in Python, the programmers don't have to worry about the low-level details like managing memory used by the program, etc. They just need to concentrate on writing solutions of the current problem at hand.

Interactive Programs in Python work in interactive mode which allows interactive testing and debugging of pieces of code. Programmers can easily interact with the interpreter directly at the Python prompt to write their programs.

Portable Python is a portable language and hence the programs behave the same on a wide variety of hardware platforms and has the same interface on all platforms. The programs work on any of the operating systems like Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE, and even Pocket PC without requiring any changes.

Note A good Python program must not use any system specific feature.

Object Oriented Python supports object-oriented as well as procedure-oriented style of programming. While object-oriented technique encapsulates data and functionalities within objects, *procedure-oriented* technique, on the other hand, builds the program around procedures or functions which are nothing but reusable pieces of programs. Python is powerful yet a simple language for implementing OOP concepts, especially when compared to languages like C++ or Java.

Interpreted We have already seen the difference between a compiler and a linker in Chapter 1. We know that an interpreted language has a simpler execute cycle and also works faster.

Python is processed at run-time by the interpreter. So, there is no need to compile a program before executing it. You can simply *run* the program. Basically, Python converts the source code into an intermediate form called ***bytecode,*** which is then translated into the native language of your computer so that it can be executed. Bytecodes makes the Python code portable since users just have to copy the code and run it without worrying about compiling, linking, and loading processes.

Note The Python interpreter can run interactively to support program development and testing.

Dynamic Python executes dynamically. Programs written in Python can be copied and used for flexible development of applications. If there is any error, it is reported at run-time to allow interactive program development.

Extensible Since Python is an open source software, anyone can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to work more efficiently. Moreover, if you want a piece of code not to be accessible for everyone, then you can even code that part of your program in C or C++ and then use them from your Python program.

Embeddable Programmers can embed Python within their C, C++, COM, ActiveX, CORBA, and Java programs to give 'scripting' capabilities for users.

Extensive Libraries Python has a huge library that is easily portable across different platforms. These library functions are compatible on UNIX, Windows, Macintosh, etc. and allows programmers to perform a wide range of applications varying from text processing, maintaining databases, to GUI programming.

Besides the above stated features, Python has a big list of good features, such as

Easy Maintenance Code written in Python is easy to maintain.

Secure The Python language environment is secure from tampering. Modules can be distributed to prevent altering the source code. Apart from this, additional security checks can be easily added to implement additional security features.

Robust Python programmers cannot manipulate memory directly. Moreover, errors are raised as exceptions that can be catch and handled by the program code. For every syntactical mistake, a simple and easy to interpret message is displayed. All these things makes the language robust.

Multi-threaded Python supports multi-threading, that is executing more than one process of a program simultaneously. It also allows programmers to perform process management tasks.

Garbage Collection The Python run-time environment handles garbage collection of all Python objects. For this, a reference counter is maintained to assure that no object that is currently in use is deleted. An object that

is no longer used or has gone out of scope are eligible for garbage collection. This frees the programmers from the worry of memory leak (failure to delete) and dangling reference (deleting too early) problems. However, the programmers can still perform memory management functions by explicitly deleting an unused object.

Limitations of Python

- Parallel processing can be done in Python but not as elegantly as done in some other languages (like JavaScript and Go Lang).
- Being an interpreted language, Python is slow as compared to C/C++. Python is not a very good choice for those developing a high-graphic 3d game that takes up a lot of CPU.
- As compared to other languages, Python is evolving continuously and there is little substantial documentation available for the language.
- As of now, there are few users of Python as compared to those using C, C++ or Java.
- It lacks true multiprocessor support.
- It has very limited commercial support point.
- Python is slower than C or C++ when it comes to computation of heavy tasks and desktop applications.
- It is difficult to pack up a big Python application into a single executable file. This makes it difficult to distribute Python to non-technical users.

Note BitTorrent, YouTube, Dropbox, Deluge, Cinema 4D, and Bazaar are a few globally-used applications based on Python.

3.2 HISTORY OF PYTHON

Python was developed by Guido van Rossum in the late 80's and early 90's at the National Research Institute for Mathematics and Computer Science in the Netherlands. It has been derived from many languages such as ABC, Modula-3, C, C++, Algol-68, SmallTalk, Unix shell, and other scripting languages. Since early 90's Python has been improved tremendously. Its version 1.0 was released in 1991, which introduced several new functional programming tools. While version 2.0 included list comprehensions and was released in 2000 by the BeOpen Python Labs team. Python 2.7 which is still used today will be supported until 2020. But there will be no 2.8, instead, support team will continue to support version 2.7 and concentrate further development of Python 3. Currently, Python 3.6.4 is already available. The newer versions have better features like flexible string representation, etc.

The difference between 2.x and 3.x versions of Python are discussed in detail Annexure 3.

Although Python is copyrighted, its source code is available under the GNU General Public License (GPL) like that of Perl. Python is currently maintained by a core development team at the institute which is directed by Guido van Rossum.

These days, from data to web development, Python has emerged as a very powerful and popular language. It would be surprising to know that Python is actually older than Java, R, and JavaScript.

Why is it called 'Python'?

Python language was released by its designer, Guido Van Rossum, in February 1991 while working for CWI (also known as Stichting Mathematisch Centrum). At the time he began implementing this language, he was also reading the published scripts from Monty Python's Flying Circus (a BBC comedy series from the 70's). Rossum wanted a name that was short, unique, and slightly mysterious. Since, he was a fan of the show he thought Python would be the perfect name for the new language.

Applications of Python

Since its origin in 1989, Python has grown to become part of a plethora of web-based, desktop-based, graphic design, scientific, and computational applications. With Python being freely available for Windows,

Mac OS X, and Linux/UNIX, its popularity of use is constantly increasing. Some of the key applications of Python include:

Python is a high-level general purpose programming language that is used to develop a wide range of applications including image processing, text processing, web, and enterprise level applications using scientific and numeric data from network.

- *Embedded scripting language*: Python is used as an embedded scripting language for various testing/ building/ deployment/ monitoring frameworks, scientific apps, and quick scripts.
- *3D Software:* 3D software like Maya uses Python for automating small user tasks, or for doing more complex integration such as talking to databases and asset management systems.
- *Web development:* Python is an easily extensible language that provides good integration with database and other web standards. Therefore, it is a popular language for web development. For example, website *Quora* has a lot of code written in Python. Besides this, *Odoo, a* consolidated suite of business applications and *Google App engine* are other popular web applications based on Python.

For web development, Python has frameworks such as `Django` and `Pyramid`, micro-frameworks such as `Flask` and `Bottle`, and advanced content management systems such as `Plone` and `django CMS`. These frameworks provide libraries and modules which simplifies content management, interaction with database, and interfacing with different internet protocols such as HTTP, SMTP, XML-RPC, FTP, and POP.

- *GUI-based desktop applications:* Simple syntax, modular architecture, rich text processing tools, and the ability to work on multiple operating systems makes Python a preferred choice for developing desktop-based applications. For this, Python has various GUI toolkits like wxPython, PyQt, or PyGtk which help developers create highly functional Graphical User Interface (GUI) including,
- *Image processing and graphic design applications:* Python is used to make 2D imaging software such as `Inkscape`, `GIMP`, `Paint Shop Pro`, and `Scribus`. It is also used to make 3D animation packages, like `Blender`, `3ds Max`, `Cinema 4D`, Houdini, `Lightwave`, and `Maya`.
- *Scientific and computational applications:* Features like high speed, productivity, and availability of tools, such as `Scientific Python` and `Numeric Python`, have made Python a preferred language to perform computation and processing of scientific data. 3D modeling software, such as `FreeCAD`, and finite element method software, like `Abaqus`, are coded in Python.

Moreover, `SciPy` is a collection of packages for mathematics, science, and engineering; Pandas is a data analysis and modeling library and `IPython` is a powerful interactive shell that supports ease of editing and recording a work session. In addition to this, `IPython` supports visualizations and parallel computing.

- *Games:* Python has various modules, libraries, and platforms that support development of games. While `PySoy` is a `3D` game engine, `PyGame` on the other hand provides functionality and a library for game development. Games like `Civilization-IV`, `Disney's Toontown Online`, `Vega Strike`, etc. are coded using Python.
- *Enterprise and business applications:* Simple and reliable syntax, modules and libraries, extensibility, and scalability together make Python a suitable coding language for customizing larger applications. For example, `Reddit` which was originally written in Common Lips, was rewritten in Python in 2005. A large part of `Youtube` code is also written in Python.
- *Operating Systems:* Python forms an integral part of Linux distributions. For example, Ubuntu's Ubiquity Installer, and Fedora's and Red Hat Enterprise Linux's Anaconda Installer are written in Python. Gentoo Linux uses Python for Portage, its package management system.
- *Language Development:* Python's design and module architecture is used to develop other languages. For example, `Boo` language uses an object model, syntax, and indentation, similar to Python. `Apple's Swift`, `CoffeeScript`, `Cobra`, and `OCaml` all have syntax similar to Python.

- *Prototyping:* Since Python is very easy to learn and an open source language, it is widely used for prototype development. Moreover, agility, extensibility, and scalability of code written in Python supports faster development from initial prototype.
- *Network Programming:* Python is used for network programming as it has easy to use socket interface, functions for email processing, and support for `FTP`, `IMAP`, and other Internet protocols.
- *Teaching:* Python is a perfect language for teaching programming skills at the introductory as well as advanced level.

3.3 THE FUTURE OF PYTHON

Python has a huge user base that is constantly growing. It is a stable language that is going to stay for long. The strength of Python can be understood from the fact that this programming language is the most preferred language of companies, such as Nokia, Google, and YouTube, as well as NASA for its easy syntax. Python has a bright future ahead of it supported by a huge community of OS developers. The support for multiple programming paradigms including object-oriented Python programming, functional Python programming, and parallel programming models makes it an ideal choice for the programmers. Based on the data from Google Trends and other relevant websites, Python is amongst the top five most preferred languages in academics as well industry.

Python is a high-speed dynamic language. Therefore, it works well in applications like photo development and has been embedded in programs such as GIMP and Paint Shop Pro. In fact, the YouTube architect, Cuong Do, has appreciated this language for record speed with which the language allows them to work. The best part is that more and more companies have started using Python for a broader range of applications ranging from social networks, through automation to science calculations.

3.4 WRITING AND EXECUTING FIRST PYTHON PROGRAM

Here onwards, we will be using Python, via the Python console. For that you need to first download Python from `www.Python.org`. The codes in this book have been developed on `Python 3.4.1`. But they can also be executed on newer versions like Python 3.5 and 3.6.

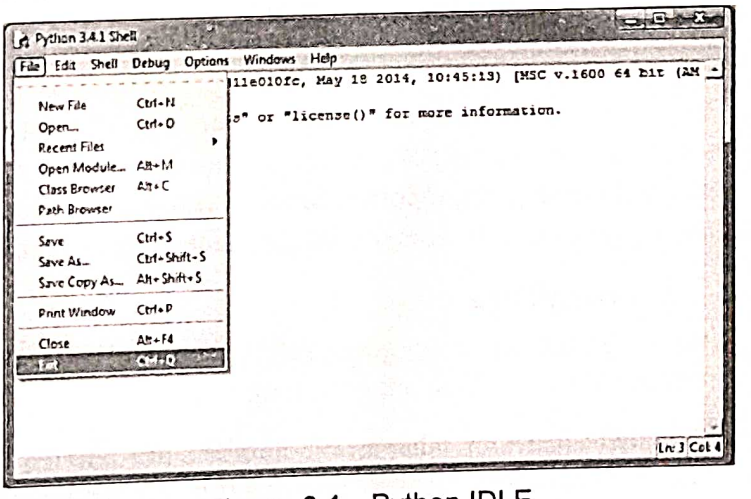

Figure 3.1 Python IDLE

Once installed, the Python console can be accessed in several ways. We will discuss only two of them here. First, using the command line and running the Python interpreter directly. Second, using a GUI software that comes installed with Python called Python's Integrated Development and Learning Environment (`IDLE`), as shown in Figure 3.1.

When you run the IDLE, you get a prompt of three right arrows. Type in your instructions at the prompt and press enter. Let us print `Hello World!!!` on the screen. For this, simply type the following line on the `IDLE`.

Example 3.1 To print a message on the screen

```
>>> print("Hello World!!!")
Hello World!!!
```

Note The >>> symbol denotes the Python prompt.

Python IDLE works on different platforms (like Windows, Unix, and Mac OS X) in almost the same way. It contains the shell window, an interactive interpreter, debugger, and a multi-window text editor that has features like Python colorizing, smart indent, call tips, and auto completion.

Programmers can even use the REPL editor to write Python programs. The REPL editor is same as IDLE. But, you can think of IDLE as Notepad and REPL as the NotePad++ editor.

Writing Python Programs

In general, the standard way to save and run a Python program is as follows:

Step 1: Open an editor.

Step 2: Write the instructions.

Step 3: Save it as a file with the filename having the extension `.py`.

Step 4: Run the interpreter with the command `python program_name.py` or use IDLE to run the programs.

To execute the program at the *command prompt*, simply change your working directory to `C:\Python34` (or move to the directory where you have saved Python) and then type `python program_name.py`.

If you want to execute the program in Python shell, then just press `F5` key or click on `Run` Menu and then select `Run Module`.

Note For exiting from the IDLE, click on `File->Exit`, or, press `Ctrl + Q` keys or type `quit()` at the command prompt.

In the next section, we will read about building blocks (such as constants, variables, data types, operators, etc.) of Python programming language.

3.5 LITERAL CONSTANTS

The word "literal" has been derived from literally. The value of a literal constant can be used directly in programs. For example, `7`, `3.9`, `'A'`, and `"Hello"` are literal constants. The number `7` always represents itself and nothing else. Moreover, it is a constant because its value cannot be changed. Hence, it is known as *literal constant*. In this section, we will read about number and string constants in Python.

3.5.1 Numbers

Number as the name suggests, refers to a numeric value. You can use four types of numbers in Python program. These include integers, long integers, floating point, and complex numbers.

- Numbers like `5` or other whole numbers are referred to as *integers*. Bigger whole numbers are called *long integers*. For example, `535633629843L` is a long integer. Note that a long integer must have `'l'` or `'L'` as the suffix.
- Numbers like are `3.23` and `91.5E-2` are termed as *floating point numbers*.
- Numbers of `a + bi` form (like `-3 + 7i`) are *complex numbers*.

Programming Tip: You can specify integers in octal as well as hexadecimal number system.

Note The `'E'` notation indicates powers of `10`. In this case, `91.5E-2` means `91.5 * 10-2`.

Remember that commas are never used in numeric literals or numeric values. Therefore, numbers like `3,567` `1,23.89` `-8,904`, are not allowed in Python.

Although, there is no limit to the size of an integer that can be represented in Python, floating-point numbers do have a limited *range* and a limited *precision*. In Python, you can have a floating point number in a range of `10-308` to `10308` with `16` to `17` digits of precision. In fact, large floating point numbers are efficiently

represented in scientific notation. For example, `5.0012304*106` (6 digits of precision) can be written as `5.0012304e+6` in scientific notation.

Although floating point numbers are very efficient at handling large numbers, there are some issues while dealing with them as they may produce following errors.

- **The Arithmetic Overflow Problem:** When you multiply two very large floating point numbers you may get an *arithmetic overflow*. Arithmetic overflow is a condition that occurs when a calculated result is too large in magnitude (size) to be represented. For example, just try to multiply `2.7e200 * 4.3e200`. You will get result as `inf`, which means infinity. The result infinity denotes that an arithmetic overflow has occurred.
- **The Arithmetic Underflow Problem:** You can get an arithmetic underflow while doing division of two floating point numbers. Arithmetic underflow is a condition that occurs when a calculated result is too small in magnitude to be represented. For example, just try to divide `3.0e-400/5.0e200`. You will get the result as `0.0`. The value `0.0` indicates that there was an arithmetic underflow in the result.
- **Loss of Precision Problem:** When you divide `1/3` you know that the results is `.33333333...`, where 3 is repeated infinitely. Since any floating-point number has a limited precision and range, the result is just an approximation of the true value.
- Python automatically displays a rounded result to keep the number of digits displayed manageable. For most applications, this slight loss in accuracy is of no practical concern but in scientific computing and other applications in which precise calculations are required, it may be a big issue.

Built-in `format()` Function

Any floating-point value may contain an arbitrary number of decimal places, so it is always recommended to use the built-in `format()` function to produce a string version of a number with a specific number of decimal places. Observe the difference between the following outputs.

```
# Without using format()
>>> float(16/(float(3)))
5.333333333333333
```

```
# Using format()
>>> format(float(16/(float(3))), '.2f')
'5.33'
```

Here, `.2f` in the `format()` function rounds the result to two decimal places of accuracy in the string produced. For very large (or very small) values, 'e' can be used as a *format specifier*.

The `format()` function can also be used to format floating point numbers in scientific notation. Look at the result of the expression given below.

```
>>> format(3**50,'.5e')
'7.17898e+23'
```

The result is formatted in scientific notation with five decimal places of precision. This feature is especially useful when displaying results in which only a certain number of decimal places is needed.

Finally, the `format()` function can also be used to insert a comma in the number as shown below.

```
>>> format(123456,',')
'123,456'
```

Note The `format()` function produces a numeric string of a floating point value rounded to a specific number of decimal places.

Simple Operations on Numbers

Python can carry out simple operations on numbers. To perform a calculation, simply enter the numbers and the type of operations that needs to be performed on them directly into the Python console, and it will print the answer, as shown in the following examples.

```
>>>10 + 7
 17
```

```
>>> 50 + 40 - 35
55
```

```
>>> 12 * 10
120
```

```
>>> 96 / 12
8.0
```

```
>>> (-30 * 4) + 500
380
```

Note The spaces around the plus and minus signs here are optional. They are just added to make the statement more readable. The code will execute even if you remove the spaces.

In the above example, using a single slash to divide numbers produces a decimal or a float point number. Therefore, internally, 96/12 = 8.0.

Division by Zero Dividing a number by zero in Python generates an error, and no output is produced as shown below.

```
>>>15/0      # generates error
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    15/0
ZeroDivisionError: division by zero
```

Thus, we see that the last line of an error message indicates the type of error generated.

Dividing Two Integers We have seen that dividing any two integers produces a floating point number. However, a float is also produced by performing an operation on two floats or on a float and an integer. Observe the following statements. Both these statements when executed results in a floating point number.

```
>>> 5*3.0
15.0
>>> 19 + 3.5
22.5
```

You can easily work with a floating point number and an integer because Python automatically converts the integer to a float. This is known as *implicit conversion* (or type coercion).

Quotient and Remainder When diving two numbers, if you want to know the quotient and remainder, use the floor division (//) and modulo operator (%), respectively. These operators can be used with both floats and integers. Observe the following statements and their output. When we divide 78 by 5 we get a quotient of 15 and a remainder of 3.

```
>>> 78//5
15
```

```
>>> 78 % 5
3
```

```
>>> 152.78 // 3.0
50.0
```

```
>>> 152.78 % 3.0
2.780000000000001
```

Exponentiation Besides, +, - , *, and / Python also supports ** operator. The ** operator is used for exponentiation, i.e., raising of one number to the power of another. Consider the statements given below and observe the output.

```
>>> 5**3
125
>>> 121**0.5
11.0
```

3.5.2 Strings

A *string* is a group of characters. If you want to use text in Python, you have to use a string. We have already printed a string in our first program. You can use a string in the following ways in a Python program.

- **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.

3.6 VARIABLES AND IDENTIFIERS

Using just literal constants (discussed in section 3.5) you cannot do much in your programs. For developing little complex programs, you need to store information and manipulate it as required. This is where *variables* can help.

Variables play a very important role in most programming languages, and Python is no exception. Variable, in simple language, means its value can vary. You can store any piece of information in a variable. Variables are nothing but just parts of your computer's memory where information is stored. To be identified easily, each variable is given an appropriate name. Every variable is assigned a name which can be used to refer to the value later in the program.

Note Variables are reserved memory locations that stores values.

Variables are examples of identifiers. *Identifiers* as the name suggests, are names given to identify something. This something can be a variable, function, class, module, or other object. For naming any identifier, there are some basic rules that you must follow. These rules are:

- The first character of an identifier must be an underscore (`'_'`) or a letter (upper or lowercase).
- The rest of the identifier name can be underscores (`'_'`), letters (upper or lowercase), or digits (`0-9`).
- Identifier names are case-sensitive. For example, `myvar` and `myVar` are **not** the same.
- Punctuation characters such as `@`, `$`, and `%` are not allowed within identifiers.

Examples of valid identifier names are `sum`, `__my_var`, `num1`, `r`, `var_20`, `First`, etc.
Examples of invalid identifier names are `1num`, `my-var`, `%check`, `Basic Sal`, `H#R&A`, etc.

Note Python is a case-sensitive language.

3.7 DATA TYPES

In the previous section, we have seen that variables can hold values of different types called ***data types***. Thus, we need different data types to store different types of values in the variables. For example, a person's age is stored is a number, his name is made of only characters, and his address is a mixture of numbers and characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Programming Tip: Variable names can contain only letters, numbers, and underscores. Also, they can't start with numbers.

Based on the data type of a variable, the interpreter reserves memory for it and also determines the type of data that can be stored in the reserved memory. The basic types are numbers and strings. We can even create our own data types in Python (like classes). The five standard data types supported by Python includes—numbers, string, list, tuple, and dictionary. In this chapter, we will learn about numbers and strings. Other data types will be explored in subsequent chapters.

Note Python is a purely object-oriented language. It refers to everything as an object including numbers and strings.

3.7.1 Assigning or Initializing Values to Variables

In Python, programmers need not explicitly declare variables to reserve memory space. The declaration is done automatically when a value is assigned to the variable using the equal sign (=). The operand on the left side of equal sign is the name of the variable and the operand on its right side is the value to be stored in that variable.

Example 3.3 Program to display data of different types using variables and literal constants

```
num = 7
amt = 123.45
code = 'A'
pi = 3.1415926536
population_of_India = 10000000000
msg = "Hi"

print("NUM = "+str(num))
print("\n AMT = "+ str(amt))
print("\n CODE = " + str(code))
print("\n POPULATION OF INDIA = " + str(population_of_India))
print("\n MESSAGE = "+str(msg))
```

OUTPUT

```
NUM = 7
AMT = 123.45
CODE = A
POPULATION OF INDIA = 10000000000
MESSAGE = Hi
```

To run this program, type the code in IDLE. Save it with a suitable name with an extension `.py`. Press `F5` or click on *Run* and then on *Run Module*. In the above code, the program assigns literal constant `7` to the variable `num` using the assignment operator (=). Similarly, we have assigned literal constants to other variables and then printed their values.

In Python, you can reassign variables as many times as you want to change the value stored in them. You may even store value of one data type in a statement and then a value of another data in a subsequent statement. This is possible because Python variables do not have specific types, so you can assign an integer to a variable, and later assign a string to the same variable.

Example 3.4 Program to reassign values to a variable

```
val = 'Hello '
print(val)
val = 100
print(val)
val = 12.34
print(val)
```

OUTPUT

```
Hello
100
12.34
```

Do you know that Python IDLE remembers variables and their values? For example, type the following command lines in the IDLE and observe the output.

```
>>> x = 5
>>> y = 10
>>> print('Hello')
Hello
>>> print(x+y)
15
```

3.7.2 Multiple Assignment

Python allows programmers to assign a single value to more than one variables simultaneously. For example,

```
sum = flag = a= b = 0
```

In the above statement, all four integer variables are assigned a value 0. You can also assign different values to multiple variables simultaneously as shown below.

```
sum, a, b, mesg = 0, 3, 5, "RESULT"
```

Here, variable `sum`, `a`, and `b` are integers (numbers) and `mesg` is a string. `sum` is assigned a value `0`, `a` is assigned `3`, `b` is assigned `5`, and `mesg` is assigned `"RESULT"`.

Remember that trying to reference a variable that has not been assigned any value causes an *error*. This may happen if you have mistakenly used a variable without assigning it a value prior to its use or have deliberately deleted or removed a variable using the `del` statement and then trying to use it later in your code. The examples given below illustrates this concept.

Note Removing a variable means that the reference from the name to the value has been deleted. However, deleted variables can be used again in the code if and only if you reassign them some value.

Example 3.5 Programs to assign and access variables

```
>>> str = "Hello"
>>> num = 10
>>> print(str)
Hello
>>> print(num)
10
>>> print(age)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in
<module>
    print(age)
NameError: name 'age' is not defined
```

Case 1: variable not declared prior to its use

```
>>> str = "Hello"
>>> num = 10
>>> age = 20
>>> print(str)
Hello
>>> print(num)
10
>>> print(age)
20
>>> del num
>>> print(num)
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in
<module>
  print(num)
NameError: name 'num' is not defined
```

Case 2: variable being used after deleting it

3.7.3 Multiple Statements on a Single Line

There are two types of lines in a program code—the physical line and the logical line. While, physical line is what we see while writing the program, logical line, on the other hand, is what Python *sees* as a single statement. For example, `print("Hello")` is a logical line. If this is the only statement present in a line, it also corresponds to physical line. Python assumes that each physical line corresponds to a logical line. So, it is a good programming habit to specify a single statement on a line. This also makes the code readable and easily understandable.

Programming Tip: Try to avoid the use of semi-colons in Python programs.

However, if you want to specify more than one statements in a single line, then you should use a semi-colon(`;`) to separate the two statements. For example, all the statements given below are equivalent.

```
mesg = "Hello"
print(mesg)
```

```
mesg = "Hello";
print(mesg);
```

```
mesg = "Hello";
print(mesg)
```

3.7.4 Boolean

Boolean is another data type in Python. A variable of Boolean type can have one of the two values—*True* or *False*. Similar to other variables, the Boolean variables are also created while we assign a value to them or when we use a relational operator on them.

Note Boolean variables are also created by comparing values using the == operator.

Example 3.6 Codes to assign and access Boolean variables

```
>>>Boolean_var = True
>>>print(Boolean_var)
True
```

```
>>> 20 == 30
False
```

```
>>>"Python" == "Python"
True
```

```
>>> 20 != 20
False
```

```
>>>"Python"! =
"Python3.4"
True
```

```
>>>30 > 50
False
```

```
>>> 90 <= 90
True
```

```
>>>87 == 87.0
False
```

```
>>>87 > 87.0
False
```

```
>>>87 < 87.0
False
```

```
>>>87 >= 87.0
True
```

```
>>>87 <= 87.0
True
```

Programming Tip: <, > operators can also be used to compare strings lexicographically.

3.8 INPUT OPERATION

Real world programs need to be interactive. With interactive we mean that you need to take some sort of input or information from the user and work on that input.

To take input from the users, Python makes use of the `input()` function. The `input()` function prompts the user to provide some information on which the program can work and give the result. However, we must always remember that the input function takes user's input as a string. So whether you input a number or a string, it is treated as a string only.

Example 3.7 Program to read variables from the user

```
name = input("What's your name?")
age = input("Enter your age : ")
print(name + ", you are " + age + " years old")
```

OUTPUT

```
What's your name? Goransh
Enter your age : 10
Goransh, you are 10 years old
```

Note In the latest 3.x versions of Python, `raw_input()` function has been renamed as `input()`.

3.9 COMMENTS

Comments are the non-executable statements in a program. They are just added to describe the statements in the program code. Comments make the program easily readable and understandable by the programmer as well as other users who are seeing the code. The interpreter simply ignores the comments.

In Python, a hash sign (#) that is not inside a string literal begins a comment. All characters following the # and up to the end of the line are part of the comment.

Example 3.8 Program to use comments

```
# This is a comment
print("Hello")  # to display hello
# Program ends here
```

OUTPUT

```
Hello
```

Note that the three comments in the program are not displayed. You can type a comment in a new line or on the same line after a statement or expression.

Note A program can have any number of comments.

3.10 RESERVED WORDS

In every programming language there are certain words which have a pre-defined meaning. These words which are also known as reserved words or keywords cannot be used for naming identifiers. Table 3.2 shows a list of Python keywords.

Table 3.2 Reserved Words

and	assert	break	class	continue	def	del	elif	else	except
exec	finally	for	from	global	if	import	in	is	lambda
not	or	pass	print	raise	return	try	while	with	yield

Note All the Python keywords contain lowercase letters only.

3.11 INDENTATION

Whitespace at the beginning of the line is called ***indentation***. *These whitespaces or the indentation are very important in Python.* In a Python program, the

Programming Tip: Use a single tab for each indentation level.

leading whitespace including spaces and tabs at the beginning of the logical line determines the indentation level of that logical line.

Note In most programming languages, indentation has no effect on program logic. It is used to align statements to make the code readable. However, in Python, indentation is used to associate and group statements.

Example 3.9 Program to exhibit indentation errors

```
age = 21
    print("You can vote") # Error! Tab at the start of the line
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 2
    print("You can vote")
    ^
IndentationError: unexpected indent
```

The level of indentation groups statements to form a block of statements. This means that statements in a block must have the same indentation level. Python very strictly checks the indentation level and gives an error if indentation is not correct.

Note ^ is a standard symbol that indicates where error has occurred in the program.

In the above code, there is a tab at the beginning of the second line. The error indicated by Python tells us that there is an indentation error. Python does not you to arbitrarily start new blocks of statements.
Like other programming languages, Python does not use curly braces ({...}) to indicate blocks of code for class and function definitions or for flow control (discussed later in the book). It uses only indentation to form a block.

Note All statements inside a block should be at the same indentation level.

3.12 OPERATORS AND EXPRESSIONS

Operators are the constructs that are used to manipulate the value of operands. Some basic operators include +, -, *, and /. In an expression, an operator is used on operand(s) (values to be manipulated). For example, in the expression `sum = a + b`, `a` and `b` are operands and + is the operator.

Python supports different types of operators which are as follows:

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Unary Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

3.12.1 Arithmetic Operators

Some basic arithmetic operators are `+, -, *, /, %, **`, and `//`. You can apply these operators on numbers as well as on variables to perform corresponding operations. For example, if a= 100 and b = 200, then look at the Table 3.3 to see the result of operations.

Table 3.3 Arithmetic Operators

Operator	Description	Example	Output
+	Addition: Adds the operands	>>> print(a + b)	300
-	Subtraction: Subtracts operand on the right from the operand on the left of the operator	>>> print(a - b)	-100
*	Multiplication: Multiplies the operands	>>> print(a * b)	20000
/	Division: Divides operand on the left side of the operator with the operand on its right. The division operator returns the quotient.	>>> print(b / a)	2.0
%	Modulus: Divides operand on the left side of the operator with the operand on its right. The modulus operator returns the remainder.	>>> print(b % a)	0

//	Floor Division: Divides the operands and returns the quotient. It also removes the digits after the decimal point. If one of the operands is negative, the result is floored (i.e.,rounded away from zero towards negative infinity).	>>> print(12//5)
		>>> print(12.0//5.0)
		>>> print(-19//5)
		>>> print(-20.0//3)
			2
			2.0
			-4
			-7.0

**	Exponent: Performs exponential calculation, that is, raises operand on the right side to the operand on the left of the operator.	>>> print(a**b)	100^{200}

3.13.2 Comparison Operators

Comparison operators also known as *relational operators* are used to compare the values on its either sides and determines the relation between them. For example, assuming `a = 100` and `b = 200`, we can use the comparison operators on them as specified in Table 3.4.

Table 3.4 Comparison Operator

Operator	Description	Example	Output
==	Returns True if the two values are exactly equal.	>>> print(a == b)	False
!=	Returns True if the two values are not equal.	>>> print(a != b)	True
>	Returns True if the value at the operand on the left side of the operator is greater than the value on its right side.	>>> print(a > b)	False
<	Returns True if the value at the operand on the right side of the operator is greater than the value on its left side.	>>> print(a < b)	True
>=	Returns True if the value at the operand on the left side of the operator is either greater than or equal to the value on its right side.	>>> print(a >= b)	False
<=	Returns True if the value at the operand on the right side of the operator is either greater than or equal to the value on its left side.	>>> print(a <= b)	True

3.12.3 Assignment and In-place or Shortcut Operators

Assignment operator as the name suggests assigns value to the operand. The in-place operators also known as *shortcut operators* that includes `+=`, `-=`, `*=`, `/=`, `%=`, `//=` and `**=` allow you to write code like `num = num + 10` more concisely, as `num += 3`. Different types of assignment and in-place operators are given in Table 3.5.

Table 3.5 Assignment and in-place Operator

Operator	Description	Example
=	Assign value of the operand on the right side of the operator to the operand on the left.	c = a , assigns value of a to the variable c
+=	Add and assign: Adds the operands on the left and right side of the operator and assigns the result to the operand on the left.	a += b is same as a = a + b
-=	Subtract and assign: Subtracts operand on the right from the operand on the left of the operator and assigns the result to the operand on the left.	a –= b is same as a = a - b
*=	Multiply and assign: Multiplies the operands and assigns result to the operand on the left side of the operator.	a *= b is same as a = a * b
/=	Divide and assign: Divides operand on the left side of the operator with the operand on its right. The division operator returns the quotient. This result is assigned to the operand to the left of the division operator.	a /= b is same as a = a / b
%=	Modulus and assign: Divides operand on the left side of the operator with the operand on its right. The modulus operator returns the remainder which is then assigned to the operand on the left of the operator.	a %= b is same as a = a % b
//=	Floor division: Divides the operands and returns the quotient. It also removes the digits after the decimal point. If one of the operands is negative, the result is floored (rounded away from zero towards negative infinity): the result is assigned to the operand on the left of the operator.	a //= b is same as a = a // b
**=	Exponent and assign: Performs exponential calculation, that is, raises operand on the right side to the operand on the left of the operator and assigns the result in the left operand.	a**= b is same as a = a** b

Note that the in-place operators can also be used on other data types.

Example 3.10 Commands to show the application of the += operator on strings

```
>>> str1 ="Good "
>>> str2 = "Morning"
>>> str1 += str2
>>> print(str1)
Good Morning
```

3.12.4 Unary Operators

Unary operators act on single operands. Python supports unary minus operator. Unary minus operator is strikingly different from the arithmetic operator that operates on two operands and subtracts the second operand from the first operand. When an operand is preceded by a minus sign, the unary operator negates its value.

Programming Tip: Python does not support prefix and postfix increment as well as decrement operators.

For example, if a number is positive, it becomes negative when preceded with a unary minus operator. Similarly, if the number is negative, it becomes positive after applying the unary minus operator. Consider the given example.

```
b = 10
a = -(b);
```

The result of this expression is `a = -10`, because variable `b` has a positive value. After applying unary minus operator `(-)` on the operand `b`, the value becomes `-10`, which indicates it as a negative value.

3.12.5 Bitwise Operators

As the name suggests, bitwise operators perform operations at the bit level. These operators include `bitwise AND`, `bitwise OR`, `bitwise XOR`, and `shift` operators. Bitwise operators expect their operands to be of integers and treat them as a sequence of bits.

Bitwise AND (&)

When we use the bitwise `AND` operator, the bit in the first operand is ANDed with the corresponding bit in the second operand. The `bitwise-AND` operator compares each bit of its first operand with the corresponding bit of its second operand. If both bits are `1`, the corresponding bit in the result is `1` and `0` otherwise. For example,

```
10101010 & 01010101 = 00000000
```

Bitwise OR (|)

When we use the `bitwise OR` operator, the bit in the first operand is ORed with the corresponding bit in the second operand. The truth table is same as we had seen in `logical OR` operation. The `bitwise-OR` operator compares each bit of its first operand with the corresponding bit of its second operand. If one or both bits are 1, the corresponding bit in the result is 1 and 0 otherwise. For example,

```
10101010 | 01010101 = 11111111
```

Bitwise XOR (^)

When we use the `bitwise XOR` operator, the bit in the first operand is XORed with the corresponding bit in the second operand. That is, the bitwise-XOR operator compares each bit of its first operand with the corresponding bit of its second operand. If one of the bits is `1`, the corresponding bit in the result is `1` and `0` otherwise. For example,

```
10101010 ^ 01010101 = 11111111
```

Bitwise NOT (~)

The `bitwise NOT`, or complement, is a unary operation, which performs logical negation on each bit of the operand. By performing negation of each bit, it actually produces the ones' complement of the given binary value. `Bitwise NOT` operator sets the bit to `1`, if it was initially `0` and sets it to `0`, if it was initially 1. For example,

```
~10101011 = 01010100
```

The truth tables of these bitwise operators are summarized in Table 3.6.

Table 3.6 Truth Tables for Bitwise Operators

A	B	A&B	A	B	A|B	A	B	A^B	A	!A
0	0	0	0	0	0	0	0	0	0	1
0	1	0	0	1	1	0	1	1	1	0
1	0	0	1	0	1	1	0	1		
1	1	1	1	1	1	1	1	0		

3.12.6 Shift Operators

Python supports two bitwise shift operators. They are `shift left (<<)` and `shift right (>>)`. These operations are used to shift bits to the left or to the right. The syntax for a shift operation can be given as follows:

```
operand op num
```

where, the bits in operand are shifted left or right depending on the operator (left if the operator is << and right if the operator is>>)by number of places denoted by num. For example,

```
if we have x = 0001 1101, then
x << 1 gives result = 0011 1010
```

When we apply a left shift, every bit in x is shifted to the left by one place. Therefore, the MSB (most significant bit) of x is lost and the LSB of x is set to 0. Therefore, for example,

```
if we have x = 0001 1101, then
x << 4 gives result = 1010 0000
```

If you observe carefully, you will notice that shifting once to the left multiplies the number by 2. Hence, multiple shifts of 1 to the left, results in multiplying the number by2 over and over again.

On the contrary, when we apply a right shift, every bit in x is shifted to the right by one place. Therefore, the LSB (least significant bit) of x is lost and the MSB of x is set to 0. For example,

```
if we have x = 0001 1101, then
x >> 1 gives result = 0000 1110.
Similarly, if we have x = 0001 1101 then
x << 4 gives result = 0000 0001
```

If you observe carefully, you will notice that shifting once to the right divides the number by 2. Hence, multiple shifts of 1 to the right, results in dividing the number by 2 over and over again.

Note Bitwise operators cannot be applied to float or double variables.

3.12.7 Logical Operators

Python supports three logical operators—logical AND (&&), logical OR (||), and logical NOT (!). As in case of arithmetic expressions, the logical expressions are evaluated from left to right.

Logical AND (&&)

Logical AND operator is used to simultaneously evaluate two conditions or expressions with relational operators. If expressions on both the sides (left and right side) of the logical operator are true, then the whole expression is true. For example,

If we have an expression `(a>b) && (b>c)`, then the whole expression is true only if both expressions are true. That is, if `b` is greater than `a` and `c`.

Logical OR (||)

Logical OR operator is used to simultaneously evaluate two conditions or expressions with relational operators. If one or both the expressions of the logical operator is true, then the whole expression is true. For example,

If we have an expression `(a>b) || (b>c)`, then the whole expression is true if either `b` is greater than `a` or `b` is greater than `c`.

Logical NOT (!)

The logical not operator takes a single expression and negates the value of the expression. Logical NOT produces a zero if the expression evaluates to a non-zero value and produces a `1` if the expression produces a zero. In other words, it just reverses the value of the expression. For example,

```
a = 10, b
b = !a;
```

Now, the value of `b = 0`. The value of a is not zero, therefore, `!a = 0`. The value of !a is assigned to b, hence, the result.

Note Truth table of logical AND, OR, and NOT operators is exactly same as that of bitwise AND, OR, and NOT.

It can be noted that the logical expressions operate in a shortcut (or lazy) fashion and stop the evaluation when it knows the final outcome for sure. For example, in a logical expression involving logical AND, if the first operand is false, then the second operand is not evaluated as it is certain that the result will be false. Similarly, for a logical expression involving logical OR, if the first operand is true, then the second operand is not evaluated as it is certain that the result will be true.

3.12.8 Membership Operators

Python supports two types of membership operators—`in` and `not in`. These operators, as the name suggests, test for membership in a sequence such as strings, lists, or tuples that will be discussed in later chapters and are listed below.

`in` Operator: The operator returns True if a variable is found in the specified sequence and False otherwise. For example, `a in nums` returns `1`, if `a` is a member of `nums`.

`not in` Operator: The operator returns True if a variable is not found in the specified sequence and False otherwise. For example, `a not in nums` returns `1`, if `a` is not a member of `nums`.

3.12.9 Identity Operators

Python supports two types of identity operators. These operators compare the memory locations of two objects and are given as follows.

`is` Operator: Returns True if operands or values on both sides of the operator point to the same object and False otherwise. For example, if `a is b` returns `1`, if `id(a)` is same as `id(b)`.

`is not` Operator: Returns True if operands or values on both sides of the operator does not point to the same object and False otherwise. For example, if `a is not b` returns `1`, if `id(a)` is not same as `id(b)`.

3.12.10 Operators Precedence and Associativity

Table 3.7 lists all operators from highest precedence to lowest. When an expression has more than one operator, then it is the relative priorities of the operators with respect to each other that determine the order in which the expression will be evaluated.

Table 3.7 Operator Precedence Chart

Operator	Description
**	Exponentiation
~, +, -	Complement, unary plus (positive), and minus (negative)
*, /, %, //	Multiply, divide, modulo, and floor division
+, -	Addition and subtraction
>>, <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<=, <, >, >=	Comparison operators
<>, ==, !=	Equality operators
=, %=, /=, //=, -=, +=, *=, **=	Assignment operators
is, is not	Identity operators
in, not in	Membership operators
not, or, and	Logical operators

Operator precedence table is important as it affects how an expression is evaluated. For example,

```
>>> 10 + 30 * 5
160
```

This is because * has higher precedence than +, so it first multiplies 30 and 5 and then adds 10. The operator precedence table decides which operators are evaluated in what order. However, if you want to change the order in which they are evaluated, you can use parentheses.

Note Parentheses can change the order in which an operator is applied. The operator in the parenthesis is applied first even if there is a higher priority operator in the expression.

Let us try some more codes to see how operator precedence works in our expressions.

```
>>> (40 + 20) * 30 / 10
180
```

```
>>> ((40 + 20) * 30) / 10
180
```

```
>>> (40 + 20) * (30 / 10)
180
```

```
>>> 40 + (20 * 30) / 10
100
```

Programming Tip: Operators are associated from left to right. This means that operators with same precedence are evaluated in a left to right manner.

Let us take more examples and apply operator precedence on Boolean data types.

```
>>>False == False or True
True
(Because == has a higher
precedence than or)
```

```
>>>False==(False or True)
False
(Parenthesis has changed
the order of operators)
```

```
>>>(False==False) or True
True
```

Note Python performs operations in the same order as that of normal mathematics—BEDMAS. That is, Brackets first, then exponentiation, then division, multiplication, and then addition and finally subtraction.

3.13 EXPRESSIONS IN PYTHON

In any programming language, an expression is any legal combination of symbols (like variables, constants, and operators) that represents a value. Every language has its own set of rules that define whether an expression is valid or invalid in that language. In Python, an expression must have at least one operand (variable or constant) and can have one or more operators. On evaluating an expression, we get a value.

Operand is the value on which operator is applied. These operators use constants and variables to form an expression. A * B + C – 5 is an example of an expression, where, +, *, and - are operators; A, B, and C are variables; and 5 is a constant. Some valid expressions in Python are: `a x = a / b, y = a * b, z = a^ b, x = a > b,` etc. When an expression has more than one operator, then the expression is evaluated using the operator precedence chart.

An example of an illegal expression can be `a+, -b,` or, `<y++`. When the program is compiled, it also checks the validity of all expressions. If an illegal expression is encountered, an error message is displayed.

Types of Expressions

Python supports different types of expressions that can be classified as follows.

Based on the position of operators in an expression: These type of expressions include:

- *Infix Expression: It is the most commonly used type of expression in which the operator is placed in between the operands. Example: a = b – c*
- *Prefix Expression: In this type of expression, the operator is placed before the operands. Example: a = □b c*
- *Postfix Expression: In this type of expression, the operator is placed after the operands. Example: a = b c□*

Prefix and postfix expressions are usually used in computers and can be easily evaluated using stacks. You will read about them in data structures.

Based on the data type of the result obtained on evaluating an expression: These type of expressions include:

- *Constant Expressions*: One that involves only constants. Example: `8 + 9 - 2`
  Integral Expressions: One that produces an integer result after evaluating the expression. Example:
  ```
  a = 10
  b = 5
  c = a * b
  ```
- *Floating Point Expressions:* One that produces floating point results. Example: `a * b / 2`
- *Relational Expressions: One that returns either True or False value.* Example: `c = a>b`
- *Logical Expressions:* One that combines two or more relational expressions and returns a value as *True* or *False*. Example: `a>b && y! = 0`
- *Bitwise Expressions:* One that manipulates data at bit level. Example: `x = y&z`
- *Assignment Expressions:* One that assigns a value to a variable. Example: `c = a + b` or `c = 10`

Program 3.11 Write a program to convert degrees fahrenheit into degrees celsius.

```
fahrenheit = float(input("Enter the temperature in fahrenheit : "))
celsius = (0.56)*(fahrenheit-32)
print("Temperature in degrees celsius = ",celsius)
```

OUTPUT

```
Enter the temperature in fahrenheit : 104.3
Temperature in degrees celsius =  40.488
```

Program 3.12 Write a program to calculate the total amount of money in the piggybank, given the coins of Rs 10, Rs 5, Rs 2, and Re 1.

```
num_of_10_coins = int(input("Enter the number of 10Rs coins in the piggybank : "))
num_of_5_coins = int(input("Enter the number of 5Rs coins in the piggybank : "))
num_of_2_coins = int(input("Enter the number of 2Rs coins in the piggybank : "))
num_of_1_coins = int(input("Enter the number of 1Re coins in the piggybank : "))
total_amt = num_of_10_coins*10+num_of_5_coins*5+num_of_2_coins*2+num_of_1_coins
print("Total amount in the piggybank =",total_amt)
```

OUTPUT

```
Enter the number of 10Rs coins in the piggybank : 6
Enter the number of 5Rs coins in the piggybank : 10
Enter the number of 2Rs coins in the piggybank : 15
Enter the number of 1Re coins in the piggybank : 20
Total amount in the piggybank = 160
```

3.13 Write a Python program to obtain the principal amount, rate of interest, and time from the user and calculate the amount payable after compound interest with the compound interest itself.
CODE:

```
p = float(input("Enter principal: "))
r = float(input("Enter rate: "))
t = float(input("Enter time: "))

amt = p * (1 + r / 100) ** t
CI= amt - p
print("Amount Payable =", amt)
print("Compound interest =", CI)
```

3.14. Write a Python program to compute $(a + b)^3$ using the formula $a^3 + b^3 + 3a^2b + 3ab^2$.
CODE:

```
a = int(input("Enter a: "))
b = int(input("Enter b: "))
res = a ** 3 + b ** 3 + 3 * a ** 2 * b + 3 * a * b ** 2
print("Result =", res)
```

3.15. Write a Python program to calculate the area and circumference of a circle.
CODE:

```
import math

# Taking radius input from the user
radius = float(input("Enter the radius of the circle: "))

# Calculating area and circumference
area = math.pi * radius**2
circumference = 2 * math.pi * radius

# Displaying the results
print("Area of the circle:  ", area)
print("Circumference of the circle: ", circumference)
```