CHAPTER

1

INTRODUCTION TO PYTHON

Programming languages like C, Pascal or FORTRAN concentrate more on the functional aspects of programming. In these languages, there will be more focus on writing the code using functions. For example, we can imagine a C program as a combination of several functions. Computer scientists thought that programming will become easy for human beings to understand if it is based on real life examples. Hence, they developed Object Oriented Programming languages like Java and .NET where programming is done through classes and objects. Programmers started migrating from C to Java and Java soon became the most popular language in the software community.

In Java, a programmer should express his logic through classes and objects only. It is not possible to write a program without writing at least one class! This makes programming lengthy. For example, a simple program to add two numbers in Java looks like this:

```
//Java program to add two numbers
class Add  //create a class
{
    public static void main(String args[])  //start execution
    {
        int a, b;  //take two variables
        a = b = 10;  //store 10 in to a, b
        System.out.println("Sum= "+ (a+b));  //display their sum
    }
}
```

Programmers understood that in certain cases where there is no need to go for classes or objects, this type of coding is consuming more time. In such cases, they do not want to create classes or objects; rather they want to write C style coding. The same program to add two numbers can be written in C as:

```
/* C program to add two numbers */
#include<stdio.h>  /* include standard header file */
void main()  /* start execution */
{
    int a, b;  /* take two variables */
    a = b = 10;   /* store 10 in to a, b */
```

```
    printf("Sum= %d", (a+b));   /* display their sum */
}
```

Of course, the preceding program is almost same as that of Java. There is no improvement in the length of the code. Another problem is that if the programmers go for C language, they will miss the object orientation which is lacking in C. Object orientation becomes an advantage when they want to deal with heavy projects.

Nowadays, programmers want C style coding as well as the Java style object orientation. When they want to develop functional aspects like calculations or processing, they want to use C style coding and when they are in need of going for classes and objects, they will use Java style coding. The only answer for their requirement is Python!

Python

Python is a programming language that combines the features of C and Java. It offers elegant style of developing programs like C. When the programmers want to go for object orientation, Python offers classes and objects like Java. In Python, the program to add two numbers will be as follows:

```
# Python program to add two numbers
a = b = 10  # take two variables and store 10 in to them
print("Sum= ", (a+b))  # display their sum
```

The preceding code is easy to understand and develop. Hence, Python is gaining popularity among the programming folks. Of course, there are several other features of Python which we will discuss in future which make it the preferred choice of most programmers.

Coming to a bit of history, Python was developed by *Guido Van Rossum* in the year 1991 at the Center for Mathematics and Computer Science managed by the Dutch Government. Van Rossum was working on a project to develop system utilities in C where he had to interact with the Bourne shell available in UNIX. He felt the necessity of developing a language that would fill the gap between C and the shell. This has led to the creation of Python.

Van Rossum picked the name *Python* for the new language from the TV show, *Monty Python's Flying Circus*. Python's first working version was ready by early 1990 and Van Rossum released it for the public on February 20, 1991. The logo of Python shows two intertwined snakes as shown in Figure 1.1:

Figure 1.1: Python Official Logo

Python is open source software, which means anybody can freely download it from *www.python.org* and use it to develop programs. Its source code can be accessed and modified as required in the projects.

Features of Python

There are various reasons why Python is gaining good popularity in the programming community. The following are some of the important features of Python:

- **Simple:** Python is a simple programming language. When we read a Python program, we feel like reading English sentences. It means more clarity and less stress on understanding the syntax of the language. Hence, developing and understanding programs will become easy.
- **Easy to learn:** Python uses very few keywords. Its programs use very simple structure. So, developing programs in Python become easy. Also, Python resembles C language. Most of the language constructs in C are also available in Python. Hence, migrating from C to Python is easy for programmers.
- **Open source:** There is no need to pay for Python software. Python can be freely downloaded from www.python.org website. Its source code can be read, modified and can be used in programs as desired by the programmers.
- **High level language:** Programming languages are of two types: low level and high level. A low level language uses machine code instructions to develop programs. These instructions directly interact with the CPU. Machine language and assembly language are called low level languages.

  High level languages use English words to develop programs. These are easy to learn and use. Like COBOL, PHP or Java, Python also uses English words in its programs and hence it is called high level programming language.
- **Dynamically typed:** In Python, we need not declare anything. An assignment statement binds a name to an object, and the object can be of any type. If a name is assigned to an object of one type, it may later be assigned to an object of a different type. This is the meaning of the saying that Python is a dynamically typed language. Languages like C and Java are statically typed. In these languages, the variable names and datatypes should be mentioned properly. Attempting to assign an object of the wrong type to a variable name triggers error or exception.
- **Platform independent:** When a Python program is compiled using a Python compiler, it generates byte code. Python's byte code represents a fixed set of instructions that run on all operating systems and hardware. Using a Python Virtual Machine (PVM), anybody can run these byte code instructions on any computer system. Hence, Python programs are not dependent on any specific operating system. We can use Python on almost all operating systems like UNIX, Linux, Windows,

Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, etc. This makes Python an ideal programming language for any network or Internet.

- **Portable:** When a program yields the same result on any computer in the world, then it is called a portable program. Python programs will give the same result since they are platform independent. Once a Python program is written, it can run on any computer system using PVM. However, Python also contains some system dependent modules (or code), which are specific to operating system. Programmers should be careful about such code while developing the software if they want it to be completely portable.

- **Procedure and object oriented:** Python is a procedure oriented as well as an object oriented programming language. In procedure oriented programming languages (e.g. C and Pascal), the programs are built using functions and procedures. But in object oriented languages (e.g. C++ and Java), the programs use classes and objects.

Let's get some idea on objects and classes. An object is anything that exists physically in the real world. Almost everything comes in this definition. Let's take a dog with the name Snoopy. We can say *Snoopy* is an object since it physically exists in our house. Objects will have behavior represented by their attributes (or properties) and actions. For example, Snoopy has attributes like height, weight, age and color. These attributes are represented by variables in programming. Similarly, Snoopy can perform actions like barking, biting, eating, running, etc. These actions are represented by methods (functions) in programming. Hence, an object contains variables and methods.

A class, on the other hand, does not exist physically. A class is only an abstract idea which represents common behavior of several objects. For example, *dog* is a class. When we talk about dog, we will have a picture in our mind where we imagine a head, body, legs, tail, etc. This imaginary picture is called a class. When we take Snoopy, she has all the features that we have in our mind but she exists physically and hence she becomes the object of dog class. Similarly all the other dogs like Tommy, Charlie, Sophie, etc. exhibit same behavior like Snoopy. Hence, they are all objects of the same class, i.e. dog class. We should understand the point that the object Snoopy exists physically but the class dog does not exist physically. It is only a picture in our mind with some attributes and actions at abstract level. When we take Snoopy, Tommy, Charlie and Sophie, they have these attributes and actions and hence they are all objects of the *dog* class.

As we described in the preceding paragraph, a class indicates common behavior of objects. This common behavior is represented by attributes and actions. Attributes are represented by variables and actions are performed by methods (functions). So, a class also contains variables and methods just like an object does. Figure 1.2 shows relationship between a class and its object:

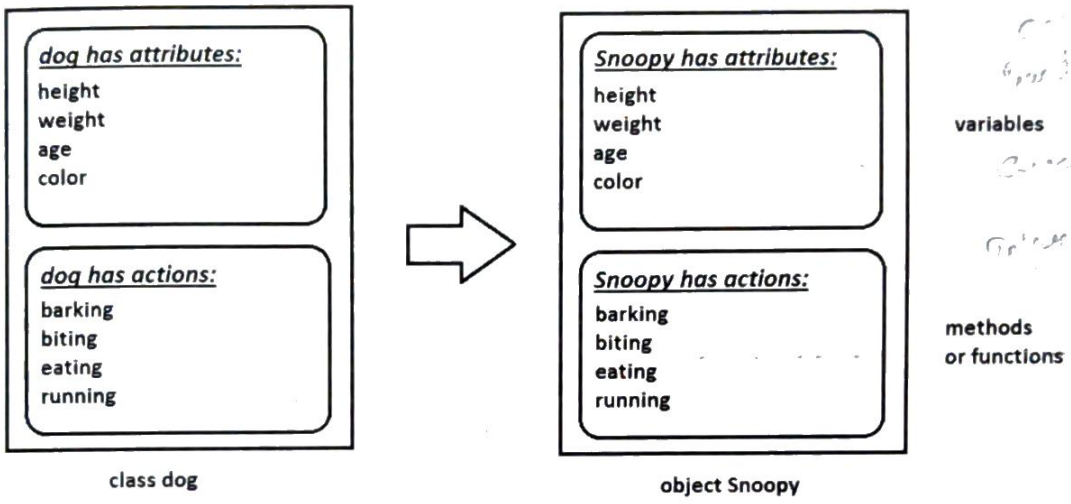

Figure 1.2: A Class and its object

Similarly, parrot, sparrow, pigeon and crow are objects of the *bird* class. We should understand that bird (class) is only an idea that defines some attributes and actions. A parrot and sparrow have the same attributes and actions but they exist physically. Hence, they are objects of the bird class.

Object oriented languages like Python, Java and .NET use the concepts of classes and objects in their programs. Since class does not exist physically, there will not be any memory allocated when the class is created. But, object exists physically and hence, a separate block of memory is allocated when an object is created. In Python language, everything like variables, lists, functions, arrays etc. are treated as objects.

- **Interpreted:** A program code is called *source code*. After writing a Python program, we should compile the source code using Python compiler. Python compiler translates the Python program into an intermediate code called *byte code*. This byte code is then executed by PVM. Inside the PVM, an interpreter converts the byte code instructions into machine code so that the processor will understand and run that machine code to produce results.

- **Extensible:** The programs or pieces of code written in C or C++ can be integrated into Python and executed using PVM. This is what we see in standard Python that is downloaded from *www.python.org*. There are other flavors of Python where programs from other languages can be integrated into Python. For example, Jython is useful to integrate Java code into Python programs and run on JVM (Java Virtual Machine). Similarly, Iron Python is useful to integrate .NET programs and libraries into Python programs and run on CLR (Common Language Runtime).

- **Embeddable:** We can insert Python programs into a C or C++ program. Several applications are already developed in Python which can be integrated into other programming languages like C, C++, Delphi, PHP, Java and .NET. It means

programmers can use these applications for their advantage in various software projects.

- **Huge library:** Python has a big library which can be used on any operating system like UNIX, Windows or Macintosh. Programmers can develop programs very easily using the modules available in the Python library.
- **Scripting language:** A scripting language is a programming language that does not use a compiler for executing the source code. Rather, it uses an interpreter to translate the source code into machine code on the fly (while running). Generally, scripting languages perform supporting tasks for a bigger application or software. For example, PHP is a scripting language that performs supporting task of taking input from an HTML page and send it to Web server software. Python is considered as a scripting language as it is interpreted and it is used on the Internet to support other software.
- **Database connectivity:** A database represents software that stores and manipulates data. For example, Oracle is a popular database using which we can store data in the form of tables and manipulate the data. Python provides interfaces to connect its programs to all major databases like Oracle, Sybase or MySql.
- **Scalable:** A program would be scalable if it could be moved to another operating system or hardware and take full advantage of the new environment in terms of performance. Python programs are scalable since they can run on any platform and use the features of the new platform effectively.
- **Batteries included:** The huge library of Python contains several small applications (or small packages) which are already developed and immediately available to programmers. These small packages can be used and maintained easily. Thus the programmers need not download separate packages or applications in many cases. This will give them a head start in many projects. These libraries are called 'batteries included'. Some interesting batteries or packages are given here:
  - *argparse* is a package that represents command-line parsing library
  - *boto*is Amazon web services library
  - *CherryPy* is anobject-oriented HTTP framework
  - *cryptography* offers cryptographic techniques for the programmers
  - *Fiona* reads and writes big data files
  - *jellyfish* is a library for doing approximate and phonetic matching of strings
  - *mysql-connector-python*is a driver written in Python to connect to MySQL database
  - *numpy* is a package for processing arrays of single or multidimensional type

- *pandas* is a package for powerful data structures for data analysis, time series and statistics
- *matplotlib* is a package for drawing electronic circuits and 2D graphs.
- *Pillow* is a Python imaging library
- *pyquery* represents jquery-like library for Python
- *scipy* is the scientific library to do scientific and engineering calculations
- *Sphinx* is the Python documentation generator
- *sympy* is a package for Computer algebra system (CAS) in Python
- *w3lib* is a library of web related functions
- *whoosh* contains fast and pure Python full text indexing, search and spell checking library

To know the entire list of packages included in Python, one can visit: https://www.pythonanywhere.com/batteries_included/

Execution of a Python Program

Let's assume that we write a Python program with the name x.py. Here, x is the program name and the .py is the extension name. Every Python program is typed with an extension name .py. After typing the program, the next step is to compile the program using Python compiler. The compiler converts the Python program into another code called byte code.

Byte code represents a fixed set of instructions that represents all operations like arithmetic operations, comparison operations, memory related operations, etc., which run on any operating system and hardware. It means the byte instructions are system independent or platform independent. The size of each byte code instruction is 1 byte and hence they are called with the name *byte code*. These byte code instructions are contained in the file x.pyc. Here, the x.pyc file represents a python compiled file.

The next step is to run the program. If we directly give the byte code to the computer, it cannot execute them. Any computer can execute only binary code which comprises 1s and 0s. Since the binary code is understandable to the machine (computer), it is also called machine code. It is therefore necessary to convert the byte code into machine code so that our computer can understand and execute it. For this purpose, we should use PVM (Python Virtual Machine).

PVM uses an interpreter which understands the byte code and converts it into machine code. PVM first understands the processor and operating system in our computer. Then it converts the byte code into machine code understandable to that processor and into that format understandable to that operating system. These machine code instructions are then executed by the processor and results are displayed.

An interpreter translates the program source code line by line. Hence, it is slow. The interpreter that is found inside the PVM runs the Python program slowly. To rectify this problem, in some flavors of Python, a compiler is added to the PVM. This compiler also converts the byte code into machine code but faster than the interpreter. This compiler is called JIT (Just In Time) compiler. The advantage of JIT compiler is to improve speed of execution of a Python program and thus improving the performance.

We should remember that JIT compiler is not available in all Python environments. For example, the standard Python software is called CPython which is created using C language that does not include a JIT compiler. But, PyPy, which is created in Python language itself uses a JIT compiler in addition to an interpreter in the PVM. So, the PyPy flavor of Python definitely offers faster execution of the Python programs than CPython. Figure 1.3 shows the steps for executing a Python program:

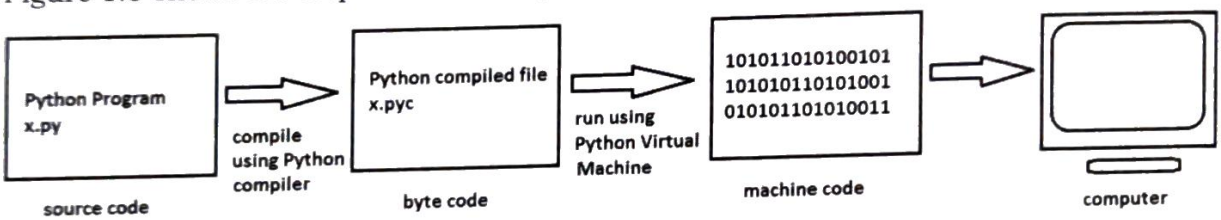

Figure 1.3: Steps of Execution of a Python Program

Normally, when we compile a Python program, we cannot see the .pyc file produced by the Python compiler and the machine code generated by the PVM. This is done internally in the memory and the output is finally visible. For example, if our Python program name is x.py, we can use Python compiler to compile it as:

```
C:\>python x.py
```

In the preceding statement, *python* is the command for calling the Python compiler. The compiler should convert the x.py file into its byte code equivalent file, x.pyc. Instead of doing this, the compiler directly displays the output or result.

If we observe, we cannot find any files in the directory with the extension .pyc. The reason is that all the steps in the sequence: **source code →byte code → machine code → output** are internally completed and then the final result is displayed. Hence, after execution, we cannot find any .pyc file in the directory.

To separately create .pyc file from the source code, we can use the following command:

```
C:\>python -m py_compile x.py
```

In the preceding command, we are calling the Python compiler with *-m* option. *-m* represents module and the module name is *py_compile*. This module generates the .pyc file for the specified .py file. The compiler creates a separate directory in the current directory by the name *__pycache__* where it stores the .pyc file. The .pyc file name may be something like: x.cpython-34.pyc. The word *cpython* here indicates that we are using the Python compiler that was created using C. This is of course, the standard Python compiler.

So the question is 'What is the use of the .pycfiles'? We know that the .pyc files contain byte code. We get these files after the compilation is completed. Hence, the first step is over. The next step is to interpret them using PVM. This can be done by calling the Python compiler as:

```
C:\>python x.cpython-34.pyc
```

In this case, we are supplying .pyc file to the Python compiler. Now, Python compiler will skip the first step where it has to convert the source code into byte code as already it sees the byte code inside this .pyc file. This file is executed directly by the PVM to produce the output. Hence, the program takes less time to run and the performance will be improved. This is the reason that after the completion of the project, the .pyc files are distributed to the user who can directly run these files using PVM and view the output.

Viewing the Byte Code

Let's consider the following Python program:

```
# Python program to add two numbers
a = b = 10  # take two variables and store 10 in to them
print("Sum= ", (a+b))  # display their sum
```

We can type this program in a text editor like Notepad and then save it as 'first.py'. It means, the first.py file contains the source code.

Now, let's compile the program using Python compiler as:

```
C:\>python  first.py
```

It will display the result as:

```
Sum= 20
```

That is ok. But we do not want the output of the program. We want to see the byte code instructions that were created internally by the Python compiler before they are executed by the PVM. For this purpose, we should specify the *dis* module while using python command as:

```
C:\>python  -m  dis  first.py
```

It will produce the following output:

```
  2           0 LOAD_CONST               0 (10)
              3 DUP_TOP
              4 STORE_NAME               0 (a)
              7 STORE_NAME               1 (b)

  3          10 LOAD_NAME                2 (print)
             13 LOAD_CONST               1 ('Sum= ')
             16 LOAD_NAME                0 (a)
             19 LOAD_NAME                1 (b)
             22 BINARY_ADD
             23 CALL_FUNCTION            2 (2 positional, 0 keyword pair)
             26 POP_TOP
             27 LOAD_CONST               2 (None)
             30 RETURN_VALUE
```

The preceding byte code is displayed by the dis module, which is also known as 'disassembler' that displays the byte code in the human understandable format. If we observe the preceding code, we can find 5 columns. The left-most or first column represents the line number in our source program (first.py). The second column represents the offset position of the byte code. The third column shows the name of the byte code instruction. The 4th column represents instruction's argument and the last column represents constants or names as specified by 4th column. For example, see the following instructions:

```
10 LOAD_NAME          2 (print)
13 LOAD_CONST         1 ('Sum= ')
16 LOAD_NAME          0 (a)
19 LOAD_NAME          1 (b)
22 BINARY_ADD
```

The LOAD_NAME specifies that this byte code instruction has 2 arguments. The name that has 2 arguments is (print) function. Since there are 2 arguments, the next byte code instructions will represent those 2 arguments as LOAD_CONST represents the string constant name ('Sum= ') and (a) and (b) as names involved in the second argument for the print function. Then BINARY_ADD instruction adds the previous (i.e. a and b) values.

Flavors of Python

Flavors of Python refer to the different types of Python compilers. These flavors are useful to integrate various programming languages into Python. The following are some of them:

- **CPython**: This is the standard Python compiler implemented in C language. This is the Python software being downloaded and used by programmers directly from *https://www.python.org/downloads/*. In this, any Python program is internally converted into byte code using C language functions. This byte code is run on the interpreter available in Python Virtual Machine (PVM) created in C language. The advantage is that it is possible to execute C and C++ functions and programs in CPython.

- **Jython**: This is earlier known as JPython. This is the implementation of Python programming language which is designed to run on Java platform. Jython compiler first compiles the Python program into Java byte code. This byte code is executed by Java Virtual Machine (JVM) to produce the output. Jython contains libraries which are useful for both Python and Java programmers. This can be downloaded from *http://www.jython.org/*.

- **IronPython**: This is another implementation of Python language for .NET framework. This is written in C# (C Sharp) language. The Python program when compiled gives an intermediate language (IL) which runs on Common Language Runtime (CLR) to produce the output. This flavor of Python gives flexibility of using both the .NET and Python libraries. IronPython can be downloaded from *http://ironpython.net/*.

Python program → IL → runs → CLR → O/P

- **PyPy**: This is Python implementation using Python language. Actually, PyPy is written in a language called RPython which was created in Python language. RPython is suitable for creating language interpreters. PyPy programs run very fast since there is a JIT (Just In Time) compiler added to the PVM. PyPy can be downloaded by visiting the page: *http://pypy.org/download.html.*

  Since the original Python uses only an interpreter in the Python Virtual Machine (PVM), the Python programs run slowly. To improve the speed of execution, a compiler called JIT (Just In Time) is introduced into the PVM of PyPy. Hence, PyPy programs run faster than those of Python. We can download PyPy from *http://pypy.org/download.html.*

- **RubyPython**: This is a bridge between the Ruby and Python interpreters. It encloses a Python interpreter inside Ruby applications. This flavor of Python can be downloaded from *https://rubygems.org/gems/rubypython/versions/0.6.3.*

- **StacklessPython**: Small tasks which should run individually are called tasklets. Tasklets run independently on CPU and can communicate with others via channels. A channel is a manager that takes care of scheduling the tasklets, controlling them and suspending them. A thread is a process which runs hundreds of such tasklets. We can create threads and tasklets in StacklessPython which is reimplementation of original Python language. This can be downloaded from *https://pypi.python.org/pypi/stackless-python/10.0.*

- **Pythonxy**: This is pronounced as Python xy and written as Python(X,Y). This is the Python implementation that we get after adding scientific and engineering related packages. We can download Pythonxy from *https://python-xy.github.io/downloads.html.*

- **AnacondaPython**: When Python is redeveloped for handling large-scale data processing, predictive analytics and scientific computing, it is called Anaconda Python. This implementation mainly focuses on large scale of data. This can be downloaded from *https://www.continuum.io/downloads.*

Python Virtual Machine (PVM)

We know that computers understand only machine code that comprises 1s and 0s. Since computer understands only machine code, it is imperative that we should convert any program into machine code before it is submitted to the computer for execution. For this purpose, we should take the help of a compiler. A compiler normally converts the program source code into machine code.

A Python compiler does the same task but in a slightly different manner. It converts the program source code into another code, called byte code. Each Python program statement is converted into a group of byte code instructions. Then what is byte code? Byte code represents the fixed set of instructions created by Python developers representing all types of operations. The size of each byte code instruction is 1 byte (or 8

bits) and hence these are called byte code instructions. Python organization says that there may be newer instructions added to the existing byte code instructions from time to time. We can find byte code instructions in the .pyc file. Figure 1.4 shows the role of virtual machine in converting byte code instructions into machine code:

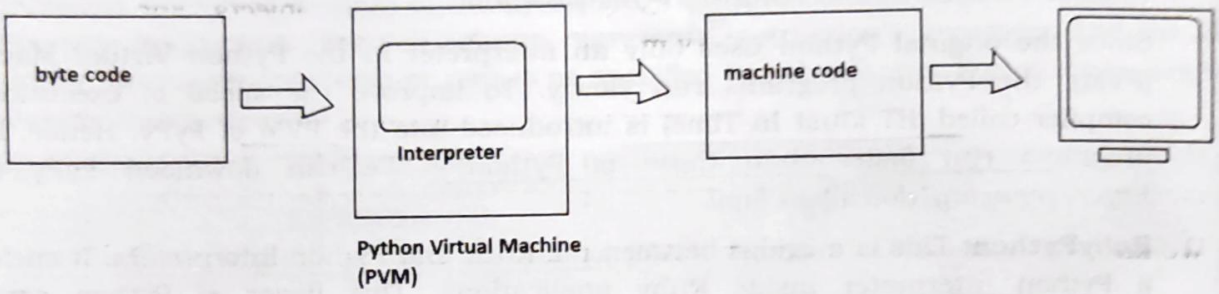

Figure 1.4: The Python Virtual Machine

The role of Python Virtual Machine (PVM) is to convert the byte code instructions into machine code so that the computer can execute those machine code instructions and display the final output. To carry out this conversion, PVM is equipped with an interpreter. The interpreter converts the byte code into machine code and sends that machine code to the computer processor for execution. Since interpreter is playing the main role, often the Python Virtual Machine is also called an interpreter.

Frozen Binaries

When a software is developed in Python, there are two ways to provide the software to the end user. The first way is to provide the .pyc files to the user. The user will install PVM in his computer and run the byte code instructions of the .pyc files.

The other way is to provide the .pyc files, PVM along with necessary Python library. In this method, all the .pyc files, related Python library and PVM will be converted into a single executable file (generally with .exe extension) so that the user can directly execute that file by double clicking on it. In this way, converting the Python programs into true executables is called *frozen binaries*. But frozen binaries will have more size than that of simple .pyc files since they contain PVM and library files also.

For creating frozen binaries, we need to use other party software. For example, py2exe is a software that produces frozen binaries for Windows operating system. We can use pyinstaller for UNIX or LINUX. Freeze is another program from Python organization to generate frozen binaries for UNIX.

Memory Management in Python

In C or C++, the programmer should allocate and deallocate (or free) memory dynamically, during runtime. For example, to allocate memory, the programmer may use malloc() function and to deallocate the memory, he may use the free() function. But in

Python, memory allocation and deallocation are done during runtime automatically. The programmer need not allocate memory while creating objects or deallocate memory when deleting the objects. Python's PVM will take care of such issues.

Everything is considered as an object in Python. For example, strings are objects. Lists are objects. Functions are objects. Even modules are also objects. For every object, memory should be allocated. Memory manager inside the PVM allocates memory required for objects created in a Python program. All these objects are stored on a separate memory called *heap*. Heap is the memory which is allocated during runtime. The size of the heap memory depends on the Random Access Memory (RAM) of our computer and it can increase or decrease its size depending on the requirement of the program.

We know that the actual memory (RAM) for any program is allocated by the underlying Operating system. On the top of the Operating system, a raw memory allocator oversees whether enough memory is available to it for storing objects. On the top of the raw memory allocator, there are several object-specific allocators operate on the same heap. These memory allocators will implement different types of memory management policies depending on the type of the objects. For example, an integer number should be stored in memory in one way and a string should be stored in a different way. Similarly, when we deal with tuples and dictionaries, they should be stored differently. These issues are taken care of by object-specific memory allocators. Figure 1.5 shows the allocation of memory by Python's Virtual Machine:

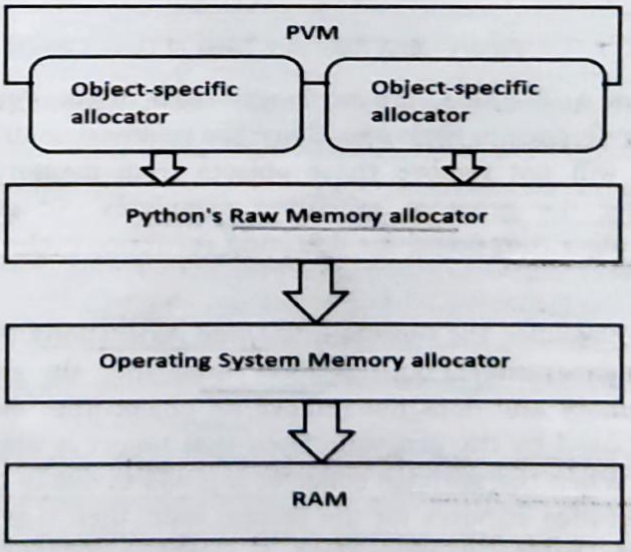

Figure 1.5: Allocation of memory by Python's Virtual Machine (PVM)

Garbage Collection in Python

A module represents Python code that performs a specific task. Garbage collector is a module in Python that is useful to delete objects from memory which are not used in the program. The module that represents the garbage collector is named as *gc*. Garbage collector in the simplest way to maintain a count for each object regarding how many times that object is referenced (or used). When an object is referenced twice, its reference count will be 2. When an object has some count, it is being used in the program and hence garbage collector will not remove it from memory. When an object is found with a reference count 0, garbage collector will understand that the object is not used by the program and hence it can be deleted from memory. Hence, the memory allocated for that object is deallocated or freed.

Garbage collector can detect reference cycles. A reference cycle is a cycle of references pointing to the first object from last object. For example, take three objects A, B and C. The object A refers to the object B whereas the object B holds a reference to the object C. Now if the object C refers to the first object A, it will form a reference cycle. Figure 1.6 shows the reference cycle of three objects:

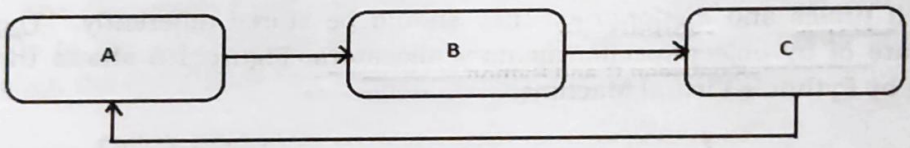

Figure 1.6: A Reference Cycle of Three Objects

Even if the objects A, B and C are no longer used in the Python program, still these objects contain 1 reference to each one. Since the reference count for each object is 1, the garbage collector will not remove these objects from memory. These objects stay in memory even after the program execution completes. To get around this, garbage collector uses an algorithm (logic) for detecting reference cycles and removing objects in the cycle.

Garbage collector classifies the objects into three generations. The newly created objects are considered as generation 0 objects. First time, when the garbage collector examines the objects in memory and does not remove an object from memory due to the reason that the object is used by the program, then that object is placed into next generation, say generation 1. When the garbage collector intends to delete the objects for the second time and the object also survives for the second time, then it is placed into generation 2. Thus, older objects belong to generation 2. Garbage collector tries to delete younger objects which are not referenced in the program rather than the old objects.

Garbage collector runs automatically. Python schedules garbage collector depending upon a number called *threshold*. This number represents the frequency of how many times the garbage collector removed (or collected) the objects. When the number of allocations minus the number of de-allocations is greater than the *threshold* number, the

garbage collector will run automatically. One can know the threshold number by using the method *get_threshold()* of *gc* module.

When more and more objects are created and if the system runs out of memory, then the automatic garbage collector will not run. Instead, the Python program will throw exception (runtime error). When the programmer is sure that his program does not contain any reference cycles, then automatic garbage collector is best suitable.

In some cases, where reference cycles are found in the program, it is better to run the garbage collector manually. For this purpose, *collect()* method of *gc* module can be used. Manual garbage collection can be done in two ways: time-based and event-based. If the garbage collector is called in certain intervals of time, it is called time-based garbage collection. If the garbage collector is called on the basis of an event, for example, when the user disconnects from an application, it is called event-based garbage collection. However, running the garbage collector too frequently will slow down the program execution.

Comparisons between C and Python

In Table 1.1, we will compare some of the important features of C and Python languages:

Table 1.1: Differences between C and Python

C	Python
C is procedure-oriented programming language. It does not contain the features like classes, objects, inheritance, polymorphism, etc.	Python is object-oriented oriented language. It contains features like classes, objects, inheritance, polymorphism, etc.
C programs execute faster.	Python programs are slower compared to C. PyPy flavor or Python programs run a bit faster but still slower than C.
It is compulsory to declare the datatypes of variables, arrays etc. in C.	Type declaration is not required in Python.
C language type discipline is static and weak.	Python type discipline is dynamic and strong.
Pointers concept is available in C.	Python does not use pointers.
C does not have exception handling facility and hence C programs are weak.	Python handles exceptions and hence Python programs are robust.
C has do... while, while and for loops.	Python has while and for loops.
C has switch statement.	Python does not have switch statement.
The variable in for loop does not increment automatically.	The variable in the for loop increments automatically.

C	Python
The programmer should allocate and deallocate memory using malloc(), calloc(), realloc() or free() functions.	Memory allocation and deallocation is done automatically by PVM.
C does not contain a garbage collector.	Automatic garbage collector is available in Python.
C supports single and multi-dimensional arrays.	Python supports only single dimensional arrays. To work with multi-dimensional arrays, we should use third party applications like numpy.
The array index should be positive integer.	Array index can be positive or negative integer number. Negative index represents locations from the end of the array.
Checking the location outside the allocation of an array is not supported in C.	Python performs checking outside an array for all iterations while looping.
Indentation of statements is not necessary in C.	Indentation is required to represent a block of statements.
A semicolon is used to terminate the statements in C and comma is used to separate expressions.	New line indicates end of the statements and semicolon is used as an expression separator.
C supports in-line assignments.	Python does not support in-line assignments.

Separate the commands on single line.

Comparisons between Java and Python

In Table 1.2, we will compare some of the important features of Java and Python languages:

Table 1.2: Differences between Java and Python

Java	Python
Java is object-oriented programming language. Functional programming features are introduced into Java 8.0 through lambda expressions.	Python blends the functional programming with object-oriented programming features. Lambdas are already available in Python.
Java programs are verbose. It means they contain more number of lines.	Python programs are concise and compact. A big program can be written using very less number of lines.
It is compulsory to declare the datatypes of variables, arrays etc. in Java.	Type declaration is not required in Python.
Java language type discipline is static and weak.	Python type discipline is dynamic and strong.

Java	Python
Java has do... while, while, for and for each loops.	Python has while and for loops.
Java has switch statement.	Python does not have switch statement.
The variable in for loop does not increment automatically. But in for each loop, it will increment automatically.	The variable in the for loop increments automatically.
Memory allocation and deal location is done automatically by JVM (Java Virtual Machine).	Memory allocation and deal location is done automatically by PVM (Python Virtual Machine).
Java supports single and multi-dimensional arrays.	Python supports only single dimensional arrays. To work with multi-dimensional arrays, we should use third party applications like numpy.
The array index should be a positive integer.	Array index can be positive or negative integer number. Negative index represents locations from the end of the array.
Checking the location outside the allocation of an array is not supported in Java.	Python performs checking outside an array for all iterations while looping.
Indentation of statements is not necessary in Java.	Indentation is required to represent a block of statements.
A semicolon is used to terminate the statements and comma is used to separate expressions.	New line indicates end of the statements and semicolon is used as an expression separator.
In Java, the collection objects like Stack, LinkedList or Vector store only objects but not primitive datatypes like integer numbers.	Python collection objects like lists and dictionaries can store objects of any type, including numbers and lists.

Points to Remember

- Python was developed by Guido Van Rossum in the year 1991.
- Python is a high level programming language that contains features of functional programming language like C and object oriented programming language like Java.
- In object oriented terminology, an object represents a physical entity that contains behavior. The behavior of an object is represented by attributes (or properties) and actions. The attributes are represented by variables and actions are performed by functions or methods.
- In object oriented terminology, a class is an abstract idea which represents common behavior of several objects. Class represents behavior and does not exist physically.

Behavior is represented by attributes (variables) and actions (functions). So, a class also contains variables and functions.

- A group of objects having same behavior comes under the same class.
- The standard Python compiler is written in C language and hence called CPython.
- There are other flavors of Python, namely Jython, IronPython and PyPy.
- A Python program contains source code that is first compiled by Python compiler to produce byte code. This byte code is given to Python Virtual Machine (PVM) which converts the byte code to machine code so that the processor will execute it and display the results.
- Python's byte code is a set of instructions created by the Python development team to represent all type of operations. Each byte code occupies 1 byte of memory and hence the name byte code.
- Python Virtual Machine (PVM) is the software containing an interpreter that converts the byte code into machine code depending on the operating system and hardware of the computer system where the Python program runs.
- The standard PVM contains only an interpreter and hence Python is called an interpreted language.
- PVM is most often called Python interpreter.
- The PVM of PyPy contains a compiler in addition to the interpreter. This compiler is called Just In Time (JIT) compiler which is useful to speed up the execution of the Python program.
- The programmer need not allocate or deallocate memory in Python. It is the duty of the PVM to allocate or deallocate memory for Python programs.
- Memory manager is a module (or sub program) in PVM which will allocate memory for objects. Garbage collector is another module in PVM that will deallocate (or free) memory for the unused objects.
- The programmer need not call the garbage collector. It will execute automatically when the Python program is running in memory. In addition, the programmer can also call the garbage collector whenever needed.
- The files that contain Python programs along with Python compiler and libraries that can be executed directly are called frozen binaries.
- The 'py_compile' module converts a Python source file into a .pyc file that contains byte code instructions. Generally, the .pyc files are provided to the end user.
- When the Python source file is given, the 'dis' module displays the equivalent byte code instructions in human readable format.