1

Introduction

AIM

Setup a Python development environment and understand the importance of Open source software and its different licenses.

LEARNING OUTCOMES

At the end of the chapter, you are expected to

- Identify various domains where Python plays a significant role.
- Install and run the Python interpreter.
- Create and execute Python programs using PyCharm IDE and Jupyter Notebook.
- Understand the meaning of Open Source Software and its different licenses.

Python is a free general-purpose programming language with beautiful syntax. It is available across many platforms including Windows, Linux and Mac OS. Due to its inherently easy to learn nature along with Object Oriented features, Python is used to develop and demonstrate applications quickly. Python has the "batteries included" philosophy wherein the standard programming language comes with a rich set of built-in libraries. It's a known fact that developers spend most of the time reading the code than writing it and Python can speed up software development. Hosting solutions for Python applications are also very cheap. Python Software Foundation (PSF) nurtures the growth of Python programming language. A versatile language like Python can be used not only to write simple scripts for handling file operations but also to develop massively trafficked websites for corporate IT organizations.

1.1 What Is a Program?

The ability to code computer programs is an important part of literacy in today's society. A program is a set of instructions instructing a computer to do specific tasks. "Software" is a generic term used to describe computer programs. Scripts, applications, programs and a set of instructions are all terms often used to describe software. The software can be categorized into three categories,

System software includes device drivers, operating systems (OSs), compilers, disk formatters, text editors and utilities helping the computer to operate more efficiently. System software serves as a base for application software. It is also responsible for managing hardware components.

Programming software is a set of tools to aid developers in writing programs. The various tools available are compilers, linkers, debuggers, interpreters and text editors.

Application software is intended to perform certain tasks. Examples of application software include office suites, gaming applications, database systems and educational software. Application software can be a single program or a collection of small programs. This type of software is what consumers most typically think of as "Software."

There are myriad of areas where programs are used like supermarkets, banks, insurance industries, process control, hospitals, offices, government institutions, education, research, telecommunication, transport industry, police, defense, multimedia applications, entertainment systems, library services and many more.

1.2 Programming Languages

Computers cannot write programs on their own as they don't understand human needs unless we communicate with the computer through programming languages. A programming language is a computer language engineered to communicate instructions to a machine. Programs are created through programming languages to control the behavior and output of a machine through accurate algorithms, similar to the human communication process.

1.2.1 Machine Language

Machine language, also called machine code, is a low-level computer language that is designed to be directly understandable by a computer and it is the language into which all programs must be converted before they can be run. It is entirely comprised of binary, 0's and 1's. In machine language, all instructions, memory locations, numbers and characters are represented in 0's and 1's. For example, a typical piece of machine language might look like, 00000100 10000000.

The main advantage of machine language is that it can run and execute very fast as the code will be directly executed by a computer and the programs efficiently utilize memory.

Some of the disadvantages of machine language are,

- Machine language is almost impossible for humans to use because it consists entirely of numbers.
- Machine language programs are hard to maintain and debug.
- Machine language has no mathematical functions available.
- Memory locations are manipulated directly, requiring the programmer to keep track of every memory location.

1.2.2 Assembly Language

Machine language is extremely difficult for humans to read because it consists merely of patterns of bits (i.e., 0's and 1's). Thus, programmers who want to work at the machine language level instead usually use assembly language, which is a human-readable notation for the machine language. Assembly language replaces the instructions represented by patterns of 0's and 1's with alphanumeric symbols also called as mnemonics in order to make it easier to remember and work with them including reducing the chances of making errors. For example, the code to perform addition and subtraction is,

```
ADD 3, 5, result
SUB 1, 2, result
```

Because of alphanumeric symbols, assembly language is also known as *Symbolic Programming Language*. The use of mnemonics is an advantage over machine language. Since the computer cannot understand assembly language, it uses another program called assembler. Assembler is a program that is used to convert the alphanumeric symbols written in assembly language to machine language and this machine language can be directly executed on the computer.

Some of the disadvantages of Assembly language are,

- There are no symbolic names for memory locations.
- It is difficult to read.
- Assembly language is machine-dependent making it difficult for portability.

1.2.3 High-Level Language

High-level language is more like human language and less like machine language. High-level languages are written in a form that is close to our human language, enabling programmers to just focus on the problem being solved. High-level languages are platform independent which means that the programs written in a high-level language can be executed on different types of machines. A program written in the high-level language is called source program or source code and is any collection of human-readable computer instructions. However, for a computer to understand and execute a source program written in high-level language, it must be translated into machine language. This translation is done using either compiler or interpreter.

Advantages

- Easier to modify, faster to write code and debug as it uses English like statements.
- Portable code, as it is designed to run on multiple machines.

A compiler is a system software program that transforms high-level source code written by a software developer in a high-level programming language into a low-level machine language. The process of converting high-level programming language into machine language is known as compilation. Compilers translate source code all at once and the computer then executes the machine language that the compiler produced. The generated machine language can be later executed many times against different data each time. Programming languages like C, C++, C# and Java use compilers. Compilers can be classified into native-code compilers and cross compilers based on their input language, output language and the platform they run on. A compiler that is intended to produce machine language to run on the same platform that the compiler itself runs on is called a native-code compiler. A cross compiler produces machine language that is intended to run on a different platform than it runs on.

Not all source code is compiled. With some programming languages like Python, Ruby and Perl the source code is frequently executed directly using an interpreter rather than first compiling it and then executing the resulting machine language. An interpreter is a program that reads source code one statement at a time, translates the statement into machine language, executes the machine language statement, then continues with the next statement. It is generally faster to run compiled code than to run a program under an interpreter. This is largely because the interpreter must analyze each statement in the source code each time the program is executed and then perform the desired conversion, whereas this is not necessary with compiled code because the source code was fully analyzed during compilation. However, it can take less time to interpret source code than the total time needed to both compile and run it, and thus interpreting is frequently used when developing and testing source code for new programs.

A programming paradigm is a style, or "way" of programming. Major programming paradigms are,

- Imperative
- Logical
- Functional
- Object-Oriented

It can be shown that anything solvable using one of these paradigms can be solved using the others; however, certain types of problems lend themselves more naturally to specific paradigms and there will be some overlap between different paradigms.

Imperative

Imperative programming is a paradigm of computer programming in which the program describes a sequence of steps that change the state of the computer as each one is executed in turn. Imperative programming explicitly tells the computer "how" to accomplish a certain goal. Structured programming, on the other hand, is a subset of Imperative programming, which emerged to remove the reliance on the GOTO statement by introducing looping structures. Then you also have Procedural programming, which is another subset of Imperative programming, where you use procedures to describe the commands the computer should perform. Procedural programming refers to the ability to combine a sequence of instructions into a procedure so that these instructions can be invoked from many places without resorting to duplicating the same instructions. The difference between procedure and function is that functions return a value, and procedures do not. An assembly language is an imperative language which is NOT structured or procedural. Popular programming language like C is imperative and structured in nature.

Logical

The logical paradigm fits exceptionally well when applied to problem domains that deal with the extraction of knowledge from basic facts and rules. Rules are written as logical clauses with a head and a body; for instance, "Y is true if X1, X2, and X3 are true." Facts are expressed similar to rules, but without a body; for instance, "Y is true." The idea in logical programming is that instead of telling the computer how to calculate things, you tell it what things are. Example: PROLOG.

Functional

In functional programming languages, functions are treated as first-class objects. In other words, you can pass a function as an argument to another function, or a function may return another function. Examples of functional programming languages are F#, LISP, Scheme, and Haskel.

Object-Oriented

The Object-oriented paradigm has gained enormous popularity in the recent decade. Object-oriented is the term used to describe a programming approach based on objects and classes. The object-oriented paradigm allows us to organize software as a collection of objects that consist of both data and behavior. This lets you have nice things like encapsulation, inheritance, and polymorphism. These properties are very important when programs become larger and larger. The object-oriented paradigm provides key benefits of reusable code and code extensibility. Examples of object-oriented languages are Python, C++, Java and C#.

It is important to note that many languages, such as Python and C++, support multiple paradigms. It is also true that even when a language is said to support a particular paradigm, it may not support all the paradigm's features. Not to mention that there is a lot of disagreement as to which features are required for a particular paradigm.

1.3 Software Development

Software development is a process by which stand-alone or individual software is created using a specific programming language. It involves writing a series of interrelated programming code, which provides the functionality of the developed software. Software development may also be called application development.

The process of software development goes through a series of stages in stepwise fashion known as the Software Development Life Cycle (SDLC). It is a systematic approach to develop software (FIGURE 1.1). It creates a structure for the developer to design, create and deliver high-quality software according to the requirements of the customer. It also provides a methodology for improving the quality of the desired product.

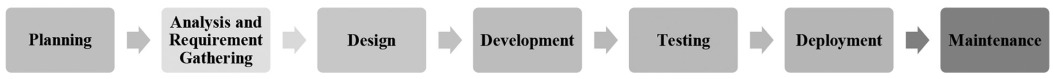


FIGURE 1.1
Different stages of Software Development Life Cycle.

The purpose of the SDLC process is to provide help in producing a product that is cost effective and of high quality. Different stages of the Software Development Life Cycle are,

Planning of Project. At this stage, the total number of resources required to implement this project is determined by estimating the cost and size of the software product.

Analysis and Requirement Gathering. At this stage, the maximum amount of information is collected from the client about the kind of software product he desires. Different questions are posed to the client like: Who is going to use the product? How will they use the product? What kind of data is given as input to the product? What kind of data is expected as output from the product? Questionnaires enable the development team to gather overall specification of the product in good detail. The software development team then analyzes all these requirements of the client, keeping in mind the design constraints, coding standards and its validity. The aim of analysis and requirements gathering stage is to understand the requirements of the client by all the members of the software development team and see how these requirements can be implemented.

Design. At this stage, the software development team analyzes whether the software can be implemented with all the features as specified by the client. Also, the development team has to convince the client about the financial feasibility and technological viability. The software development team has to select the programming language and the platform to implement the software that is best suited to satisfy the requirements of the client. Software design helps the development team to define and understand the overall architecture required for the software product and the approach is captured in detail in a design document.

Development. At this stage, the development team starts building the software according to the design document. The development team translates the design into a set of programs that adhere to coding standards of their organization. Coding is done by dividing the specification mentioned in the design document into different modules to provide a working and reliable product. This is the longest phase of SDLC.

Testing. At this stage, the software product is tested against the requirements specified by the client to ensure the product is working as expected. The testing team is mainly responsible for checking the system to weed out bugs and to verify that the software product is working as expected. Any bugs that are found in the process or any shortcomings in the features of the software product is conveyed to the software development to rectify. This is the last stage of overall software development before handing over the product to the client.

Deployment. At this stage, the product is released to the client to use after testing the product thoroughly to match the requirements of the client. The client needs to be trained in using the software and documents should be provided containing instructions on how to operate the software in a user-friendly language.

Maintenance. The process of taking care of the developed and deployed software product is known as Maintenance. When the customer starts using the deployed software product, unforeseen problems may come up and these need to be solved. Also, new requirements may come up at the client's workplace and the software needs to be updated and upgraded to accommodate these changes.

1.4 History of Python Programming Language

The history of the Python programming language dates back to the late 1980s. Python was conceived in the late 1980s and its implementation was started in December 1989 by Guido van Rossum (FIGURE 1.2) at CWI in the Netherlands as a successor to the ABC programming language capable of exception handling and interfacing with the Amoeba operating system. Van Rossum is Python's principal author, and his continuing central role in deciding the direction of Python is reflected in the title given to him by the Python community. He is the "Benevolent Dictator For Life" (BDFL), which means he continues to oversee Python development and retains the final say in disputes or arguments arising within the community.

FIGURE 1.2
Guido van Rossum—Creator and BDFL of Python Programming Language. (Image courtesy of Wikipedia.org)

Often people assume that the name Python was written after a snake. Even the logo of Python programming language (FIGURE 1.3) depicts the picture of two snakes, blue and yellow. But, the story behind the naming is somewhat different.

FIGURE 1.3
Python logo. (Image courtesy of Python.org)

Back in the 1970s, there was a popular BBC comedy TV show called "Monty Python's Flying Circus" and Van Rossum happened to be a big fan of that show. At the time when he began implementing Python, Guido van Rossum was also reading the published scripts from "Monty Python's Flying Circus." It occurred to him that he needed a name that was short, unique, and slightly mysterious, so he decided to call the language "Python."

Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code that would not be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale. Python is a multi-paradigm programming language having full support for Object-oriented programming and Structured programming and there are a number of language features which support Functional programming.

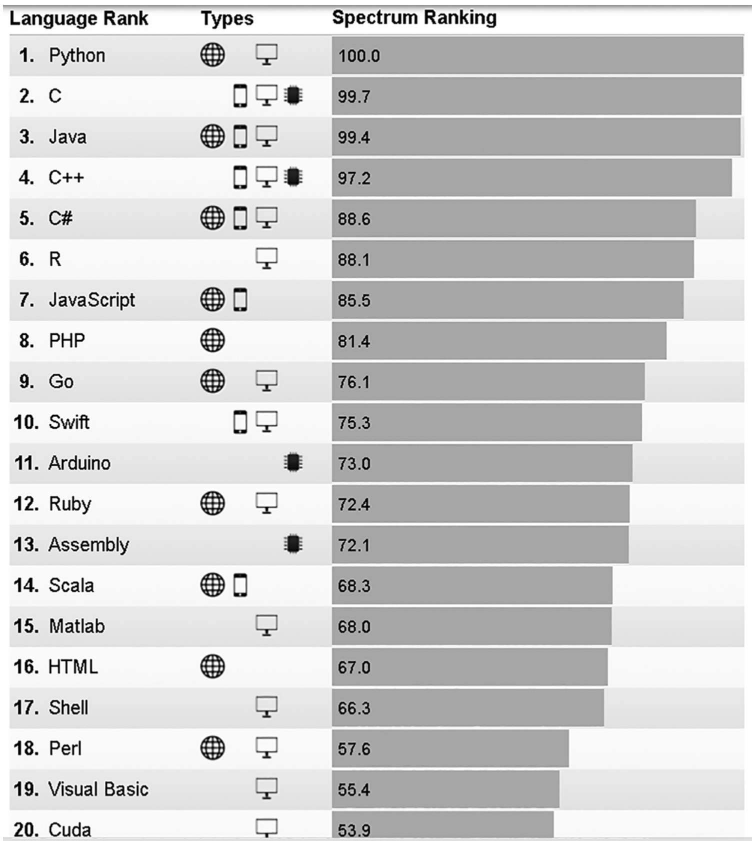
The first ever version of Python (i.e., Python 1.0) was introduced in 1991. Since its inception and introduction of Version 1, the evolution of Python has reached up to Version 3.x (till 2018). Python 2.0 was released on 16 October 2000 and had many major new features, including a cycle-detecting garbage collector and support for Unicode. With this release, the development process became more transparent and community-backed. Python 3.0 (initially called Python 3000 or py3k) was released on 3 December 2008 after a long testing period. It is a major revision of the language that is not completely backward-compatible with previous versions.

The language's core philosophy is summarized in the document *The Zen of Python*, which includes principles such as,

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one—and preferably only one—obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea—let's do more of those!

1.5 Thrust Areas of Python

Python is a top marketable professional skill known for its simplicity and developer friendliness. Python has a solid claim to being the fastest-growing major programming language. Since 2003, Python has been consistently ranked in the top ten most popular programming languages as measured by the TIOBE Programming Community Index. As of April 2018, it is in the fourth position. Python is ranked at first position by IEEE Spectrum ranking of top programming languages for the year 2017 (FIGURE 1.4) and RedMonk Programming language rankings for the year 2018 has listed Python at the third position.



Language Rank	Types	Spectrum Ranking
1. Python	🌐 🖥	100.0
2. C	📱 🖥 ▪	99.7
3. Java	🌐 📱 🖥	99.4
4. C++	📱 🖥 ▪	97.2
5. C#	🌐 📱 🖥	88.6
6. R	🖥	88.1
7. JavaScript	🌐 📱	85.5
8. PHP	🌐	81.4
9. Go	🌐 🖥	76.1
10. Swift	📱 🖥	75.3
11. Arduino	▪	73.0
12. Ruby	🌐 🖥	72.4
13. Assembly	▪	72.1
14. Scala	🌐 📱	68.3
15. Matlab	🖥	68.0
16. HTML	🌐	67.0
17. Shell	🖥	66.3
18. Perl	🌐 🖥	57.6
19. Visual Basic	🖥	55.4
20. Cuda	🖥	53.9

FIGURE 1.4
Ranking of programming languages by IEEE. (Image courtesy of *IEEE Spectrum*, New York.)

1.5.1 Academia

Python is being offered as the introductory programming language in the majority of the computer science departments at various American universities. Python is being adapted by academia for research purposes at an accelerated rate and is competing with Matlab for the coveted title of most preferred language for research. There are a few advantages of Python over Matlab, like Matlab is not a real programming language but Python is. Python has lots of scientific tools which are almost as good as Matlab modules. Developers nowadays need to work in multiple languages and the majority of the languages, including Python, have their index starting from zero while Matlab index starts from 1, which may lead to more syntactical errors. Also, Matlab uses parentheses both for indexing and functions, while Python uses the square brackets for indexing and parentheses for functions which brings more clarity in the code. Matlab is closed and proprietary with a very expensive licensing agreement, while Python is free and open source. Raspberry Pi project was started by Raspberry Pi foundation which aims to bring computer knowledge to children, elderly people and the lower strata of the society who are deprived of computer education. This foundation brings out Raspberry Pi devices, which are tiny, low-cost microcomputers that are powerful enough to do most of the work that can be done using a desktop. Python, due to its ease of learning, is recommended as the preferred programming language for Raspberry Pi.

1.5.2 Scientific Tools

Scientific tools are essential for simulating and analyzing complex systems. The Python ecosystem consists of these core scientific packages, namely SciPy library, NumPy, Jupyter, Sympy and Matplotlib. Most of these tools are available under Berkeley Software Distribution (BSD) license and can be used without any restrictions. SciPy library is mainly used for numerical integration and optimization. NumPy provides N-dimensional array objects which can be used to perform linear algebra, Fourier transform and other mathematical operations. Jupyter has revolutionized the way programming is done in Python. Jupyter provides an interactive web-based interface which can be invoked from a browser. Jupyter is used to write Python programs and create embeddable plots to visualize data. SymPy library is used to generate symbolic mathematics. Matplotlib is the oldest and most popular plotting library available for Python. With these tools, we have better chances of solving scientific problems and create working prototypes more quickly than any other competing tools.

1.5.3 Machine Learning

Machine Learning is an effective and adaptive tool to learn from experience and by a dataset. Many machine-learning algorithms and techniques have been developed that allow computers to learn. Machine Learning has its origin in Computer Science and Statistics. Scikit-Learn is a well-known Machine Learning tool built on top of other Python scientific tools like NumPy, SciPy and Matplotlib. This allows Scikit-Learn to be easily extended to implement new models. Scikit-Learn supports various models for Classification, Regression, Clustering, Model Selection, Dimensionality Reduction and Preprocessing. Some of the advantages of Scikit-Learn are integration of parallelization, consistent APIs, good documentation and it is available under BSD license as well as commercial license with full support.

1.5.4 Natural Language Processing

Natural language processing is used to read and understand the text. Natural Language Toolkit (NLTK) is the popular library used for natural language processing in Python. NLTK has numerous trained algorithms to understand the text. NLTK has huge corpora of datasets and lexical resources like journals, chat logs, movie reviews and many more. NLTK is available under Apache License V2.0.

1.5.5 Data Analysis

Pandas library changed the landscape of data analysis in Python altogether and is available under BSD license. Pandas is built on top of NumPy and has two important data structures, namely Series and DataFrame. It can hold any type of data like integers, floats, strings, objects and others. Each of the data stored in series is labeled after the index. DataFrame is a tabular data structure with labeled rows and columns similar to Excel spreadsheet. In the real world, data is never in order and pandas can be used to fill in missing data, reshaping of datasets, slicing, indexing, merging, and joining of datasets. Pandas can be used to read Comma-Separated Values (CSV) files, Microsoft Excel, Structured Query Language (SQL) database and Hierarchical Data Format (HDF5) format files.

1.5.6 Statistics

Statsmodels is a Python library used for statistical analysis. It supports various models and features like linear aggression models, generalized linear models, discrete choice models and functions for time series analysis. To ensure the accuracy of results Statsmodels is tested thoroughly by comparing it with other statistical packages. Statsmodels can also be used along with Pandas to fit statistical models. This package is available under modified BSD license. Statsmodels is used across various fields like economics, finance and engineering.

1.5.7 Hypertext Transfer Protocol (HTTP) Library

The Requests HTTP library is popularly referred to as the library written for humans. Python has a standard HTTP library called urllib.request to carry out most of the HTTP operations. But the Application Programming Interfaces (APIs) of urllib.request are not easy to use and are verbose. To overcome these problems Requests was created as a standalone library. Common HTTP verbs like POST, GET, PUT and DELETE which correspond to create, read, update and delete operations are fully supported. Also, Requests provides features like thread-safety, support for International Domains, Cookie Persistence and Connection Timeouts. Requests library is available under Apache license 2.0.

1.5.8 Database Connectors/ORM/NoSQL Connectors

Database connectors are drivers that allows us to query the database from the programming language itself. MySQL and PostgreSQL are the popular open source databases. MySQL-Python-Connector for Python from Oracle is the most popular Python connector available for MySQL and Psycopg2 is the Python connector widely used for PostgreSQL.

Object Role Modeling (ORM) is a powerful way of querying the database to achieve persistence so that data can live beyond the application process. There is a mismatch between the Object-oriented language models and the Relational databases leading to several problems like granularity, inheritance, identity, associations and navigations. ORM helps in mapping the data from Object-oriented languages to the Relational databases and follows the business layer logic. SQLAlchemy is a highly recommended ORM toolkit for Python applications to be deployed at the enterprise level. Python connectors are also available for popular NoSQL databases like MongoDB and Cassandra.

1.5.9 Web Frameworks

Django and Flask are the two most popular web frameworks. Both have different purposes. While Django is a full-fledged framework, Flask is a microframework that is used to build small applications with minimal requirements. Django has built-in support for various web-related services like caching, internationalization, serialization, ORM support and automatic admin interface, while Flask allows users to configure web services according to your needs by installing external libraries. Both of these frameworks are available under BSD derived licenses.

1.5.10 Cloud Computing

OpenStack is entirely written in Python and is used to create a scalable private and public cloud. OpenStack Foundation oversees the development of OpenStack software. OpenStack has decent load balancing, is highly reliable, vendor independent and has built-in security. OpenStack uses the dashboard as a central unit to manage network resources, processing power and storage in a data center. Linux distributions like Fedora and Ubuntu include

OpenStack as part of their package. Hosting of Python applications on a cloud platform is well supported by various cloud service providers like Google App Engine, Amazon Web Services (AWS), Heroku and Microsoft Azure.

1.5.11 Python Distributions

Python Software Foundation releases Python interpreter with standard libraries. But in order to use Python in a scientific or enterprise environment other packages needs to be installed. Having these packages tested for compatibility with the latest release of Python is cumbersome and time-consuming. Anaconda and Enthought Canopy Express are two popular distributions that come with core Python interpreter and popular scientific tools to help us start working out of the box.

1.5.12 IDE Available

Integrated Development Environments (IDEs) help in the rapid development of the software and increase in productivity. PyCharm is the most popular IDE for Python programming. PyCharm comes in three flavors namely, Professional Edition, Community Edition and Educational Edition. PyCharm has advanced features like auto code completion, code highlighting, refactoring, remote development capabilities and support for various web frameworks. PyCharm is available for various platforms like Windows, Linux and OS X. Microsoft has released an extension for Visual Studio called Python Tools for Visual Studio (PTVS) which transforms Visual Studio IDE into a full-fledged Python IDE. Spyder is another IDE that comes as part of Anaconda distribution itself.

1.5.13 Community

Community is what really defines the success of Open Source projects. Development of projects is taken forward by adding new features and Community members play an important role in testing the software, recommending it to others and in documenting the software. Python community members are expected to follow the Python Code of Conduct and the Python community is generally considered very helpful. The Python community is very active and cordial in accommodating newbies. Python conferences are held regularly across the world wherein the core Python developers are invited to share their experience with other developers thus paving the way for more Python adaption across the horizon. Python language documentation is renowned for its depth and completeness.

1.5.14 Python Stack in Industry

Various companies use Python stack to power up their infrastructure. The popular online photo sharing service Instagram uses the Django framework for application development. At Mozilla, which develops the popular Firefox web browser, the majority of the web development is done using the Django framework. PayPal and eBay, where transactions worth billions of dollars take place every year, swear by the security features provided by Python libraries. Companies like Pinterest and Twilio have adapted Flask as their web development framework. Requests library is used in major projects of companies like Amazon, Google, Washington Post, Twitter and others. Python Scientific and data analysis tools are being used at LinkedIn, Boeing and NASA. Dropbox has hired Guido van Rossum, Father of Python programming language, to add new features to their existing

Python stack. Even though this is not a complete list of companies using Python, it surely indicates industry interest in using Python to solve challenging problems.
[*Source*: CSI Communications, Vol. 40, April 2016.]

1.6 Installing Anaconda Python Distribution

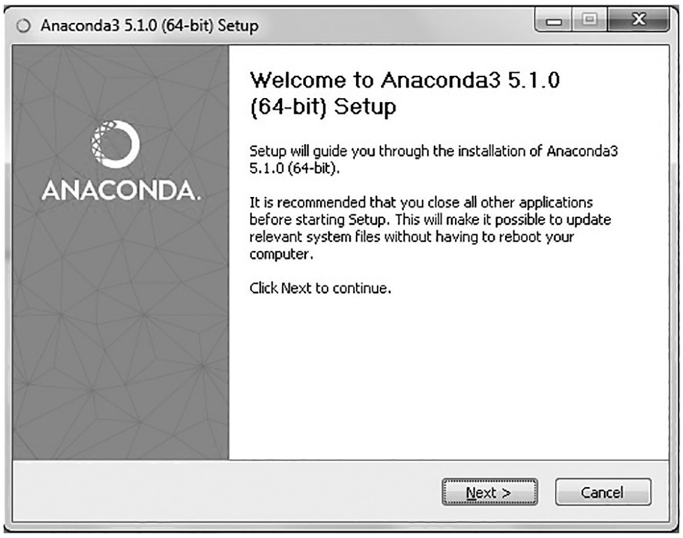
Anaconda is a free and open source distribution of the Python programming language for data science and machine-learning related applications such as large-scale data processing, predictive analytics, and scientific computing, that aims to simplify package management and deployment. Package versions are managed by the package management system *conda*, which makes it quite simple to install, run, and update complex data science and machine learning software libraries like Scikit-learn, PyTorch, TensorFlow, and SciPy. Anaconda Distribution is used by over 6 million users, and it includes more than 250 popular data science packages suitable for Windows, Linux, and MacOS.

The steps described here work on the Windows 10 OS.

Step 1: Go to the link https://www.continnum.io/downloads. You have the option to download the 32-bit or the 64-bit version of either Python 2.7 or Python 3.6 supported Anaconda distribution. At the time of writing this book, Anaconda supported Python 3.6 version. As and when a new version of Python is released, Anaconda distribution will be updated to newer releases. In this book 64-bit Anaconda distribution supporting Python 3.6 is used to execute programs, so download the same version.

Step 2: Click on the executable file of Anaconda Python distribution which you have downloaded and the setup screen will start loading.

Step 3: You will get a welcome screen as shown in FIGURE 1.5. Click on *Next* button.


Anaconda3 5.1.0 (64-bit) Setup

Welcome to Anaconda3 5.1.0 (64-bit) Setup

Setup will guide you through the installation of Anaconda3 5.1.0 (64-bit).

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

FIGURE 1.5
Welcome screen of Anaconda installation.