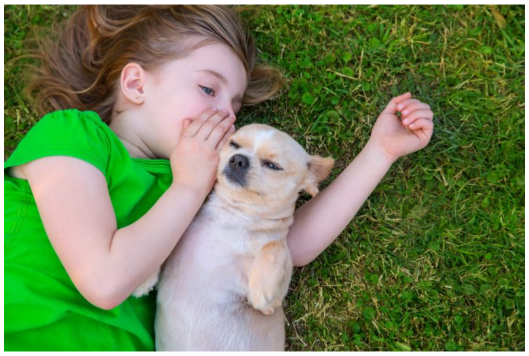
Introduction

Welcome to ***Python for Intelligent Network Operations***. This manuscript is intended for a course in Python programming and problem-solving for network operators and engineers. The course can be designed around 10 networking related practical problems, and accordingly consists of 10 learning modules. In each module, it further consists of three independent but related parts: Technical background; Python programming, and Problem solving.

What is Programming?

Do you have pets? Do you love them? Are you close friends to each other? Have you talked to them? Did you enjoy your conversation?

I am not sure about your experience. I would say (at least in most cases) **Life is sweeter with pets**. Here, we often assume a precondition: you should know your pets and let them understand you as well. In this case, good conversations between you is necessary.



Can a computer become your new pet? Absolutely yes, as long as you know how to communicate with your computers.

Communication can be conducted at different levels: basic, medium, or advanced.

We are actually communicating with computers every day at a basic level via an operating system (OS) of a computer, as discussed later in this chapter. Things like reading your email, sharing your recipe on Facebook, searching for the greatest gift for your best friend, are all examples of communication that occurs between a computer and you. The OS is similar like a pre-built and installed housekeep in a computer, who manages all computer hardware and software resources, and is responsible for handling communication to an user though a well-designed graphic or command line user interface. The communication is claimed to be basic because an user can only use a fixed set of commands/instructions.

Talking directly to a machine is an advanced communication option, and may seem quite odd at first. Having a conversation with your computer might sound like the script of a science fiction movie. After all, the members of the Enterprise on Star Trek regularly talked with their computer. In fact, the computer often talked back. However, with the rise of Apple's Siri (http://www.apple.com/ios/siri/), Amazon's Echo (https://www.amazon.com/dp/B00X4WHP5E/), and other interactive software, perhaps you really don't find a conversation so unbelievable.

Asking the computer for information or doing pre-defined tasks is one thing, but providing it with your own instructions is quite another. Programming is a way to achieve this goal.

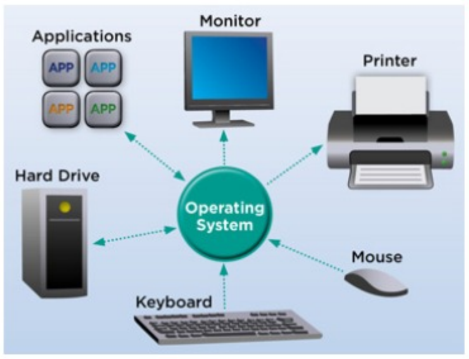
This chapter considers why you want to instruct your computer about anything and what benefit you gain from it. You also discover the need for a special language when performing this kind of communication and why you want to use Python to accomplish it. However, the main thing to get out of this chapter is that programming is simply a kind of communication that is akin to other forms of communication you already have with your computer.

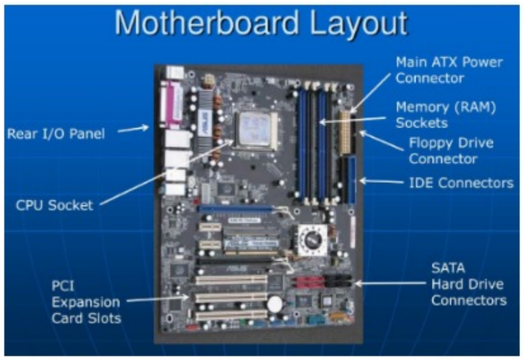
The Structure of a Modern Computer System

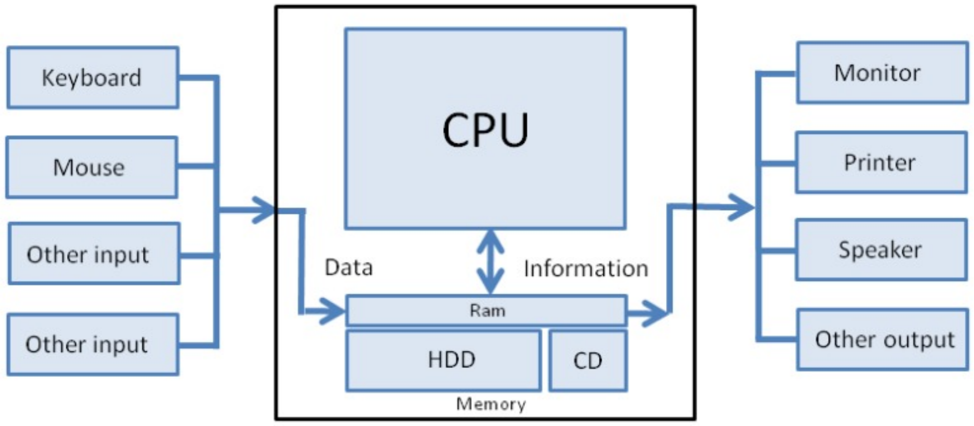
We now give a brief overview of the structure of modern computer systems. A modern computer system consists of **hardware** and **software**. Hardware consists of the physical devices required to execute algorithms. Software is the set of these algorithms, represented as programs in particular programming languages. In the discussion that follows, we focus on the hardware and software found in a typical desktop computer system, although similar components are also found in other computer systems, such as smart phones.

Computer Hardware

The basic hardware components of a computer are memory, a central processing unit (CPU), and a set of input/output devices, as shown in the figure below.







Human users primarily interact with the input and output devices. The input devices include a keyboard, a mouse, and a microphone. Common output devices include a monitor and speakers. Computers can also communicate with the external world through various ports that connect them to networks and to other devices such as handheld music players and digital cameras. The purpose of most of the input devices is to convert information that human beings deal with, such as text, images, and sounds, into information for computational processing. The purpose of most output devices is to convert the results of this processing back to human-usable form.

Computer memory is set up to represent and store information in electronic form. Specifically, information is stored as patterns of binary digits (1s and 0s). We can represent any information, including text, images, and sound, in binary form. The memory is also sometimes called primary or internal or random access memory (RAM).

The information stored in memory can represent any type of data, such as numbers, text, images, or sound, or the instructions of a program. We can also store in memory an algorithm encoded as binary instructions for the computer. Once the information is stored in memory, we typically want to do something with it—that is, we want to process it. The part of a computer that is responsible for processing data is the central processing unit (CPU). This device, which is also sometimes called a processor, consists of electronic switches arranged to perform simple logical, arithmetic, and control operations. The CPU executes an algorithm by fetching its binary instructions from memory, decoding them, and executing them. Executing an instruction might involve fetching other binary information—the data—from memory as well.

The processor can locate data in a computer's primary memory very quickly. However, these data exist only as long as electric power comes into the computer. If the power fails or is turned off, the data in primary memory are lost. Clearly, a more permanent type of memory is needed to preserve data. This more permanent type of memory is called external or secondary memory, and it comes in several forms. Magnetic storage media, such as tapes and hard disks, allow bit patterns to be stored as patterns on a magnetic field. Semiconductor storage media, such as flash memory sticks, perform much the same function with a different technology, as do optical storage media, such as CDs and DVDs. Some of these secondary storage media can hold much larger quantities of information than the internal memory of a computer.

Computer Software

You have learned that a computer is a general-purpose, problem-solving machine. To solve any computable problem, a computer must be capable of executing any algorithm (i.e., the steps to solve a computable problem). Because it is impossible to anticipate all of the problems for which there are algorithmic solutions, there is no way to "hardcode" all potential algorithms into a computer's hardware. Instead, we build some basic operations into the hardware's processor and require any algorithm to use them. The algorithms are converted to binary form and then loaded, with their data, into the computer's memory. The processor can then execute the algorithms' instructions by running the hardware's more basic operations.

Any programs that are stored in memory so that they can be executed later are called software. A program stored in computer memory must be represented in binary digits, which is also known as machine code. Loading machine code into computer memory one digit at a time would be a tedious, error-prone task for human beings. It would be convenient if we could automate this process to get it right every time. For this reason, computer scientists have developed another program, called a **loader**, to perform this task. A loader takes a set of machine language instructions as input and loads them into the appropriate memory locations. When the loader is finished, the machine language program is ready to execute. Obviously, the loader cannot load itself into memory, so this is one of those algorithms that must be hardwired into the computer.

Now that a loader exists, we can load and execute other programs that make the development, execution, and management of programs easier. This type of software is called **system software**. The most important example of system software is a computer's **operating system**.

You are probably already familiar with at least one of the most popular operating systems, such as Linux, Apple's Mac OS, and Microsoft Windows. An operating system is responsible for managing and scheduling several concurrently running programs. It also manages the computer's memory, including the external storage, and manages communications between the CPU, the input/output devices, and other computers on a network. An important part of any operating system is its file system, which allows human users to organize their data and programs in permanent storage. Another important function of an operating system is to provide user interfaces—that is, ways for the human user to interact with the computer's software. A terminal-based interface accepts inputs from a keyboard and displays text output on a monitor screen. A modern graphical user interface (GUI) organizes the monitor screen around the metaphor of a desktop, with windows containing icons for folders, files, and applications. This type of user interface also allows the user to manipulate images with a pointing device such as a mouse.

Another major type of software is called applications software, or simply **applications**. An application is a program that is designed for a specific task, such as editing a document or displaying a Web page. Applications include Web browsers, word processors, spreadsheets, database managers, graphic design packages, music production systems, and games, among many others. As you begin to learn to write computer programs, you will focus on writing simple applications.

As you have learned, computer hardware can execute only instructions that are written in binary form—that is, in machine language. Writing a machine language program, however, would be an extremely tedious, error-prone task. To ease the process of writing computer programs, computer scientists have developed high-level programming languages for expressing algorithms. These languages resemble English and allow the author to express algorithms in a form that other people can understand.

A programmer typically starts by writing high-level language statements in a text editor. The programmer then runs another program called a translator to convert the high-level program code into executable code. Because it is possible for a programmer to make grammatical mistakes even when writing high-level code, the translator checks for syntax errors before it completes the translation process. If it detects any of these errors, the translator alerts the programmer via error messages. The programmer then has to revise the program. If the translation process succeeds without a syntax error, the program can be executed by the run-time system. The run-time system might execute the program directly on the hardware or run yet another program called an interpreter or virtual machine to execute the program.

What is an computer application?

Computer applications provide the means to define express human ideas in a manner that a computer can understand. To accomplish this goal, the application relies on one or more procedures that tell the computer how to perform the tasks related to the manipulation of data and its presentation. What you see onscreen is the text from your word processor, but to see that information, the computer requires procedures for retrieving the data from disk, putting it into a form you can understand, and then presenting it to you.

How to program?

Thinking about procedures you use daily

A procedure is simply a set of steps you follow to perform a task. For example, when making toast, you might use a procedure like this:

1. Get the bread and butter from the refrigerator.
2. Open the bread bag and take out two pieces of toast.
3. Remove the cover from the toaster.
4. Place each piece of bread in its own slot.
5. Push the toaster lever down to start toasting the bread.
6. Wait for the toasting process to complete.
7. Remove toast from the toaster.
8. Place toast on a plate.
9. Butter the toast.

Your procedure might vary from the one presented here, but it should be similar. It’s unlikely that you’d butter the toast before placing it in the toaster. Most people never actually think about the procedure for making toast. However, you use a procedure like this one even though you don’t think about it.

Computers can’t perform tasks without a procedure. You must tell the computer which steps to perform, the order in which to perform them, and any exceptions to the rule that could cause failure. All this information (and more) appears within an application. In short, an application is simply a written procedure that you use to tell the computer what to do, when to do it, and how to do it. Because you’ve been using procedures all your life, all you really need to do is apply the knowledge you already possess to what a computer needs to know about specific tasks.

Seeing computer applications as being like any other procedure

When you write a computer application, you’re writing a procedure that defines a series of steps that the computer should perform to accomplish whatever task you have in mind. If you leave out a step, the results won’t be what you expected. The computer won’t know what you mean or that you intended for it to perform certain tasks automatically. The only thing the computer knows is that you have provided it with a specific procedure and it needs to perform that procedure.

Understanding that computers take things literally

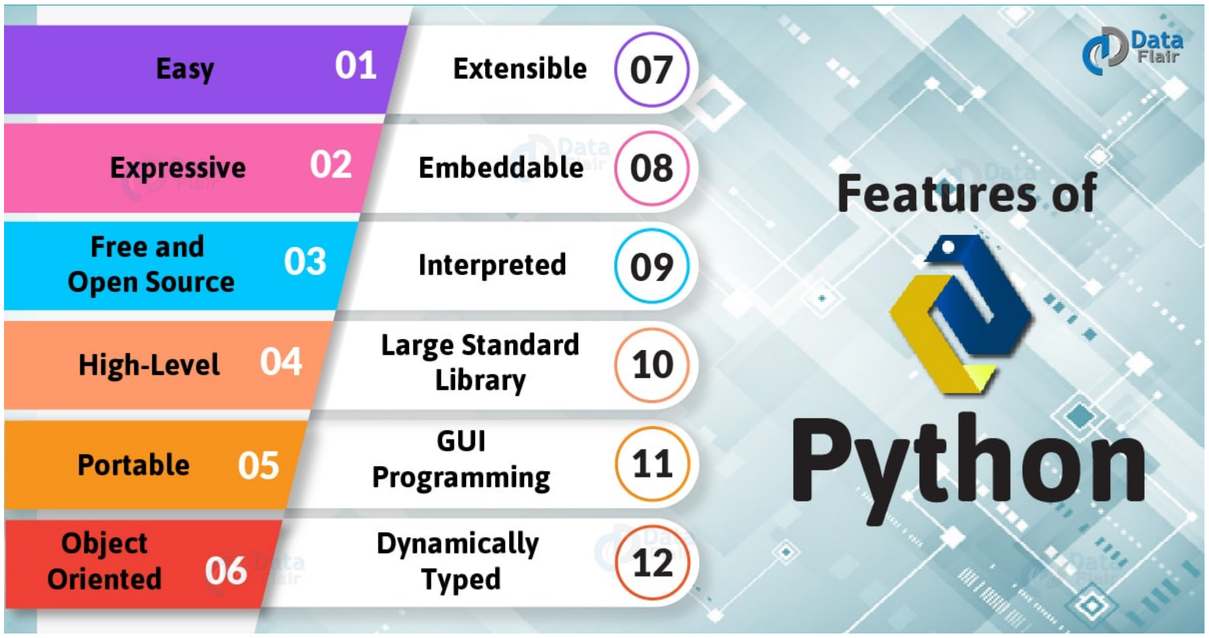
When you begin writing computer programs, you'll get frustrated because computers perform tasks precisely and read your instructions literally. Generally, computers are inflexible, unintuitive, and unimaginative. When you write a procedure for a computer, the computer will do precisely as you ask absolutely every time and never modify your procedure or decide that you really meant for it to do something else.

Why Python?

Programming is a communication between human beings and computers. Many programming languages are available today. Programmers keep creating new languages for good reason. Each language has something special to offer — something it does exceptionally well. In addition, as computer technology evolves, so do the programming languages in order to keep up. Because creating an application is all about efficient communication, many programmers know multiple programming languages so that they can choose just the right language for a particular task. One language might work better to obtain data from a database, and another might create user interface elements especially well.

As with every other programming language, Python does some things exceptionally well, and you need to know what they are before you begin using it. You might be amazed by the really cool things you can do with Python. Knowing a programming language's strengths and weaknesses helps you use it better as well as avoid frustration by not using the language for things it doesn't do well. The following sections help you make these sorts of decisions about Python.

Features of Python



This course uses the Python programming language as a way of making the initial experience of programming a network more manageable and attractive for students and instructors in the Network Engineering major. Python has the following pedagogical benefits:

- Python has simple, conventional syntax. Python statements are very close to those of pseudocode algorithms, and Python expressions use the conventional notation found in algebra. Thus, students can spend less time learning the syntax of a programming language and more time learning to solve interesting problems.
- Python has safe semantics. Any expression or statement whose meaning violates the definition of the language produces an error message.
- Python scales well. It is very easy for beginners to write simple programs (e.g., as a scripting language) in Python. Python also includes all of the advanced features of a modern

programming language, such as support for data structures and object-oriented software development, for use when they become necessary.
- Python is highly interactive. Expressions and statements can be entered at an interpreter's prompts to allow the programmer to try out experimental code and receive immediate feedback. Longer code segments can then be composed and saved in script files to be loaded and run as modules or standalone applications.
- Python is general purpose. In today's context, this means that the language includes resources for contemporary applications, including cloud computing, data science, and machine learning.
- Python is free and is in widespread use in industry. Students can download Python to run on a variety of devices. There is a lot of documentation and a large Python user community; and expertise in Python programming has great resume value.

To summarize these benefits, Python is a comfortable, friendly and flexible vehicle for expressing ideas about computation, both for beginners and for experts as well. If students learn these ideas well in such an initial programming course, they should have no problems making a quick transition to other languages needed for courses later in the curriculum. Most importantly, beginning students will spend less time staring at a computer screen and more time thinking about interesting problems to solve.

Multi-platform

The Python interpreter is available on many platforms (Linux, DOS, Windows, and macOS). The code that we create in Python is translated into bytecode when it is executed for the first time. For that reason, in systems in which we are going to execute our programs or scripts developed in Python, we need the interpreter to be installed.

The Python Programming Environment

As an introductory Python course, we will try using various tools (e.g., command line or Web-based Python shell, IDE) to access Python on different platforms (e.g., Windows, Linux, MacOS). However, we will work most of time on Jupyter Notebook to learn most of the features of the Python programming language.

Organization of the Book

The following book layout is tentative and subject to changes as needed.

The approach of this text is easygoing and problem oriented, with each new concept introduced only when it is needed.