We'll build on the terminology foundation in *What is a Program?* and define a program more completely. This is, after all, our goal, and we'll need to have it clearly defined so we can see how we're closing in on it.

2.1.1 Hardware Terminology

We want to define some terms that we'll be using throughout the book. We're going to build up our Python understanding from this foundational terminology. In the computer world, many concepts are new, and we'll try to make them more familiar to you. Further, some of the concepts are abstract, forcing us to borrow existing words and extend or modify their meanings. We'll also define them by example as we go forward in exploring Python.

This section is a kind of big-picture road map of computers. We'll refer back to these definitions in the sections which follow.

The first set of definitions are things we lump togther under "hardware", since they're mostly tangible things that sit on desks and require dusting. The next section has definitions that will include "software": those intangible things that don't require dusting.

Computer, Computer System Okay, this is perhaps silly, but we want to be very clear. We're talking about the whole *system* of interconnected parts that make up a computer. We're including all the Devices, incluing displays and keyboards and mice. We're drawing a line between our computer and the network that interconnects it to other computers.

A computer is a very generalized appliance. Without software, it's just a lump of parts. Even with the general softare components we'll talk about in *Software Terminology*, it doesn't do anything specific. We reserve the term "application software" for that software that applies this very general system to our specific needs.

Inside a computer system there are numerous electronic components, one of which is the *processor*, which controls most of what a computer does. Other components include *memory*.

It helps to think of two species of computers: your personal computer – desktop or laptop – sometimes called a "client" and shared computers called "servers". When you are surfing a web site, you are using more than one computer: your personal computer is running the web browser, and one or more server computers are responding to your browser's requests. Most of the internet things you see involve your desktop and a server somewhere else.

We do need to note that we're using the principle of *abstraction*. A number of electronic devices are all computers on which we can do Python programming. Laptops, desktops, iMacs, PowerBooks, clients, servers, Dells and HP's are all examples of this abstraction we're calling a computer system.

Device, Peripheral Device We have a number of devices that are part of our computers. Most devices are plugged into the computer box and connected by wires, putting them on the periphery of the computer. A few devices are wireless; they connect using Bluetooth, WiFi (IEEE 802.11) or infrared (IR) signals. We call the connection an *interface*.

The most important devices are hidden within the box, physically adjacent to the central processor. These central items are memory (called random-access memory, RAM) and a disk. The disk, while inside the box, is still considered peripheral because once upon a time, disks were huge and expensive.

The other peripheral devices are the ones we can see: display, keyboard and mouse. After that are other storage devices, including CD's, DVD's, USB drives, cameras, scanners, printers, drawing tablets, etc. Finally we have network connections, which can be Ethernet, wireless or a modem. All devices are controlled by pieces of software called *drivers*.

Note that we've applied the abstraction principle again. We've lumped a variety of components into abstract categories.

Memory, RAM The computer's working memory (*Random-Access Memory*, or RAM) contains two things: our data and the processing instructions (or program) for manipulating that data. Most modern computers are called *stored program digital* computers. The program is stored in memory along with the data. The data is represented as digits, not mechanical analogies. In contrast, an analog computer uses mechanical analogs for numbers, like spinning gears that make an analog speedometer show the speed, or the strip of metal that changes shape to make an analog meat thermometer show the temperature.

The central processor fetches each instruction from the computer's memory and then executes that instruction. We like to call this the *fetch-execute* loop that the processor carries out. The processor chip itself is *hardware*; the instructions in memory are called *software*. Since the instructions are stored in memory, they can be changed. We take this for granted every time we double click an icon and a program is loaded into memory. The data on which the processor is working must also be in memory. When we open a document file, we see it read from the disk into memory so we can work on it.

Memory is *dynamic*: it changes as the software does its work. Memory which doesn't change is called *Read-Only Memory* (ROM).

Memory is *volatile*: when we turn the computer off, the contents vanish. When we turn the computer on, the contents of memory are random, and our programs and data must be loaded into memory from some persistent device. The tradeoff for volatility is that memory is blazingly fast.

Memory is accessed "randomly": any of the 512 million bytes of my computer's memory can be accessed with equal ease. Other kinds of memory have sequential access; for example, magnetic cassette tapes must be accessed sequentially.

For hair-splitters, we recognize that there are special-purpose computing devices which have fixed programs that aren't loaded into memory at the click of a mouse. These devices have their software in read-only memory, and keep only data in working memory. When our program is permanently stored in ROM, we call it *firmware* instead of software. Most household appliances that have computers with ROM.

Disk, Hard Disk, Hard Drive We call these *disk* drives because the memory medium is a spinning magnetizable disk with read-write heads that shuttle across the surface; you can sometimes hear the clicking as the heads move. Individual digits are encoded across the surface of the disk; grouped into blocks of data. Some people are in the habit of calling them "hard" to distinguish them from the obsolete "floppy" disks that were used in the early days of personal computing.

Our various files (or "documents") inluding our programs and our data will – eventually – reside on some kind of disk or disk-like device. However, the operating system interposes some structure, discipline and protocol between our needs for saving files and the vagaries of the disk device. We'll look at this in *Software Terminology* and again in *Working with Files*.

Disk memory is described as "random access", even though it isn't completely random: there are read-write heads which move across the surface and the surface is rotating. There are delays while the computer waits for the heads to arrive at the right position. There are also delays while the computer waits for the disk to spin to the proper location under the heads. At 7200 RPM's, you're waiting less than 1/7200th of a second, but you're still waiting.

Your computer's disk can be imagined as persistent, slow memory: when we turn off the computer, the data remains intact. The tradeoff is that it is agonizingly slow: it reads and writes in milliseconds, close to a million times slower than dynamic memory.

Disk memory is also cheaper than RAM by a factor of at almost 1000: we buy 500 gigabytes (500 billion bytes, or 500,000 megabytes) of disk for $100; the cost of 512 megabytes of memory.

Human Interface, Display, Keyboard, Mouse The human interface to the computer typically consists of three devices: a display, a keyboard and a mouse. Some people use additional devices: a second

display, a microphone, speakers or a drawing tablet are common examples. Some people replace the mouse with a trackball. These are often wired to the computer, but wireless devices are also popular.

In the early days of computers – before the invention of the mouse – the displays and keyboards could only handle characters: letters, numbers and punctuation. When we used computers in the early days, we spelled out each command, one line at a time. Now, we have the addition of sophisticated graphical displays and the mouse. When we use computers now, we point and click, using graphical gestures as our commands. Consequently, we have two kinds of human interfaces: the *Command-Line Interface* (CLI), and the *Graphical User Interface* (GUI).

A keyboard and a mouse provide inputs to software. They work by interrupting what the computer is doing, providing the character you typed, or the mouse button you pushed. A piece of software called the Operating System has the job of collecting this stream of input and providing it to the application software. A stream of characters is pretty simple. The mouse clicks, however, are more complex events because they involve the screen location as well as the button information, plus any keyboard shift keys.

A display shows you the outputs from software. The display device has to be shared by a number of application programs. Each program has one or more windows where their output is sent. The Operating System has the job of mediating this sharing to assure that one program doesn't disturb another program's window. Generally, each program will use a series of drawing commands to paint the letters or pictures. There are many, many different approaches to assembling the output in a window. We won't touch on this because of the bewildering number of choices.

Historically, display devices used paper; everything was printed. Then they switched to video technology. Currently, displays use liquid crystal technology. Because displays were once almost entirely video, we sometimes summarize the human interface as the Keyboard-Video-Mouse (KVM).

In order to keep things as simple as possible, we're going to focus on the command-line interface. Our programs will read characters from the keyboard, and display characters in an output window. Even though the programs we write won't respond to mouse events, we'll still use the mouse to interact with the operating system and programs like **IDLE**.

Other Storage, CD, DVD, USB Drive, Camera These storage devices are slightly different from the internal disk drive or hard drive. The differences are the degree of volatility of the medium. Packaged CD's and DVD's are read-only; we call them CD Read-Only Memory (CD-ROM). When we burn our own CD or DVD, we used to call it creating a Write-Once-Read-Many (WORM) device. Now there are CD-RW devices which can be written (slowly) many times, and read (quickly) many times, making the old WORM acronym outdated.

Where does that leave Universal Serial Bus USB drives (known by a wide variety of trademarked names like Thumb Drive™or Jump Drive™) and the memory stick in our camera? These are just like the internal disk drive, except they don't involve a spinning magnetized disk. They are slower, have less capacity and are slightly more expensive than a disk.

Our operating system provides a single abstraction that makes our various disk drives and "other storage" all appear to be very similar. When we look at these devices they all appear to have folders and documents. We'll return to this unification in *Files III : The Grand Unification*.

Scanner, Printer These are usually USB devices; they are unique in that they send data in one direction only. Scanners send data into our computer; our computer sends data to a printer. These are a kind of storage, but they are focused on human interaction: scanning or printing photos or documents.

The scanner provides a stream of data to an application program. Properly interpreted, this stream of data is a sequence of picture elements (called "pixels") that show the color of a small section of the document on the scanner. Getting input from the scanner is a complex sequence of operations to reset the apparatus and gather the sequence of pixels.

A printer, similarly, accepts a stream of data. Properly interpreted, this stream of data is a sequence

of commands that will draw the appropriate letters and lines in the desired places on the page. Some printers require a sequence of pixels, and the printer uses this to put ink on paper. Other printers use a more sophisticated page description language, which the printer processes to determine the pixels, and then deposits ink on paper. One example of these sophisticated graphic languages is PostScript.

Network, Ethernet, Wireless, WiFi, Dial-up, Modem A network is built from a number of cooperating technologies. Somewhere, buried under streets and closeted in telecommunications facilities is the global Internet: a collection of computers, wires and software that cooperates to route data. When you have a cable-modem, or use a wireless connection in a coffee shop, or use the Local Area Network (LAN) at school or work, your computer is (indirectly) connected to the Internet. There is a physical link (a wire or an antenna), there are software protocols for organizing the data and sharing the link properly. There are software libraries used by the programs on our computer to surf web pages, exchange email or purchase MP3's.

While there are endless physical differences among network devices, the rules, protocols and software make these various devices almost interchangeable. There is stack of technology that uses the principle of abstraction very heavily to minimize the distinctions among wireless and wired connections. This kind of abstraction assures that a program like a web browser will work precisely the same no matter what the physical link really is. The people who designed the Internet had abstraction very firmly in mind as a way to allow the Internet to expand with new technology and still work consistently.

2.1.2 Software Terminology

Hardware terminology is pretty simple. You can see and touch the hardware. You're rarely confused by the difference between a scanner and a printer.

Software, on the other hand, is less tangible. Programming is the act of creating new software. This terminology is perhaps more important than the hardware terminology above.

Note that Software is essential for making our computer do anything. The varius components and devices – without software – are inert lumps of plastic and metal.

Operating System The Operating System (OS) ties all of the computer's devices together to create a usable, integrated computer system. The operating system includes the software called *device drivers* that make the various devices work consistently. It manages scarce resources like memory and time by assuring that all the programs share those resources. The operating system also manages the various disk drives by imposing some organizing rules on the data; we call the organizing rules and the related software the *file system*.

The operating system creates the desktop metaphor that we see. It manages the various windows; it directs mouse clicks and keyboard characters to the proper application program. It depicts the file system with a visual metaphor of folders (directories) and documents (files). The desktop is the often shown to you by a program called the "finder" or "explorer"; this program draws the various icons and the dock or task bar.

In addition to managing devices and resources, the OS starts programs. Starting a program means allocating memory, loading the instructions from the disk, allocating processor time to the program, and allocating any other resources in the processor chip.

Finally, we have to note that it is the OS that provides most of the abstractions that make modern computing possible. The idea that a variety of individual types of devices and components could be summarized by a single abstraction of "storage" allows disk drives, CD-ROM's, DVD-ROM's and thumb drives to peacefully co-exist. It allows us to run out and buy a thumb drive and plug it into our computer and have it immediately available to store the pictures of our trip to Sweden.

Program, Application, Software A program is started by the operating system to do something useful. We'll look at this in depth in *What is a Program?* and *What Happens When a Program "Runs?"*.

Since we will be writing our own programs, we need to be crystal clear on what programs really are and how they make our computer behave.

There isn't a useful distinction between words like "program", "command", "application", "application program", and "application system". Some vendors even call their programs "solutions". We'll try to stick to the word program. A program is rarely a single thing, so we'll try to identify a program with the one file that contains the main part of the program.

File, Document, Data, Database, the "File System" The data you want to keep is saved to the disk in *files*. Sometimes these are called *documents*, to make a metaphorical parallel between a physical paper document and a disk file. Files are collected into *directories*, sometimes depicted as metaphorical *folders*. A paper document is placed in a folder the same way a file is placed in a directory. Computer folders, however, can have huge numbers of documents. Computer folders, also, can contain other folders without any practical limit. The document and folder point of view is a handy visual metaphor used to clarify the file and directory structure on our disk.

This is so important that *Working with Files* is devoted to how our programs can work with files.

Boot Not footwear. Not a synonym for kick, as in "booted out the door." No, boot is used to describe a particular disk as the "boot disk". We call one disk the boot disk because of the way the operating system starts running: it pulls itself up by it's own bootstraps. Consider this quote from James Joyce's Ulysses: "There were others who had forced their way to the top from the lowest rung by the aid of their bootstraps."

The operating system takes control of the computer system in phases. A disk has a *boot sector* (or *boot block*) set aside to contain a tiny program that simply loads other programs into memory. This program can either load the expected OS, or it can load a specialized boot selection program (examples include BootCamp, GRUB, or LiLo.) The boot program allows you to control which OS is loaded. Either the boot sector directly loads the OS, or it loads and runs a boot program which loads the OS.

The part of the OS that is loaded into memory is just the kernel. Once the kernel starts running, it loads a few handy programs and starts these programs running. These programs then load the rest of the OS into memory. The device drivers must be added to the kernel. Once all of the device drivers are loaded, and the devices configured, then the user interface components can be loaded and started. At this point, the "desktop" appears.

Note that part of the OS (the kernel) loads other parts of the operating system into memory and starts them running. It pulls itself up by its own bootstraps. They call this bootstrapping, or booting. The kernel will also load our software into memory and start it running. We'll depend heavily on this central feature of an OS.

2.1.3 What is a Program?

In *Software Terminology* we provided a kind of road map to computers. Here, we're going to look a little more closely at these things called "programs".

What – Exactly – is the Point? The essence of a program is the following: *a program sets up a computer to do a specific task*. We could say that it is a program which *applies* a general-purpose computer to a specific problem. That's why we call them "application programs"; the programs apply this generalized computer appliance to definite data processing needs.

There is a kind of parallel between a computer system running programs and a television playing a particular TV show. Without the program, the computer is just a pile of inert electronics. Similarly, if there is no TV show, the television just sits there showing a blank screen. (When I was a kid, a TV with no program showed a flickering "noise" pattern. Modern TV's don't do this, they just sit there.)

We're going to focus on two parts of a program: data and processing. We'll be aiming at programs which read and write files of data, much like our ordinary desktop tools open and save files. We aren't excluding

game programs or programs that control physical processes. A game's data is the control actions from the player plus the description of the game's levels and environments. The processing that a game does matches the inputs, the current state and the level to determine what happens next. An interactive game, however, is considerably more complex than a program to evaluate a file that has a list of our stocks.

Program Varietals. At this point, we need to make a distinction between some varieties of programs: specifically, a *binary executable* and a *script*. A binary executable or *binary application* is a program that takes direct control computer's processor. We call it binary because it uses the binary codes specific to the processor chip inside the computer. If you haven't encountered "binary" before, see *Binary Codes*. Most programs that you buy or download fit this description. Most of the office applications you use are binary executables. A web browser, for example, is a binary executable, as is the `python` program (named `python.exe` in Windows.)

Your operating system (for example, Windows or GNU/Linux or MacOS) is a complex collection of binary executables. These operating system programs don't solve any particular problem, but they enable the computer to be used by non-engineers.

A binary executable's direct control over the processor is beneficial because it gives the best speed and uses the fewest resources. However, the cost of this control is the relative opacity of the coded instructions that control the processor chip. The processor instruction codes are focused on the electronic switching arcana of gates, flip-flops and registers. They are not focused on data processing at a human level. If you want to see how complex and confusing the processor chip can be, go to Intel or AMD's web site and download the technical specifications for one of their processors.

One subtlety that we have to acknowledge is that even the binary applications don't have *complete* control over the entire computer system. Recall that the computer system loads a kernel of software when it starts. All of the binary applications outside this kernel do parts of their work by using program fragments provided by the kernel. This important design feature of the operating system assures that all of the application programs share resources politely. One of the kernel's two jobs is to coordinate among the application programs. If every binary application simply grabbed resources willy-nilly, one badly behaved program could stop all other programs from working. Imagine the tedium of quitting your browser to make notes in your word processor, then quitting your word processor to go back to your web browser.

The other of the kernel's two jobs is to embody the abstraction principle and make a wide variety of processors have a nearly identical set of features.

Layers of Abstraction. Let's take a close look at our metaphor again. We said there is a strong parallel between a computer running a program and a TV playing a particular TV show. We now have two layers of meaning here:

- The whole computer system running programs – in a very broad sense – is like a TV playing a particular show. This is the most abstract view, combining many concepts together.
- At a more detailed view, we have a composite concept of computer system plus Operating System. It is this hardware-plus-software device which runs our application programs. Here, our TV metaphor starts to break down because we don't have to get a kernel TV show that allows our TV to watch a specific channel. Our TV is complete by itself. Our computer, however, can't do anything without some software. And we need some kernel of OS software to help us run our desired application software.

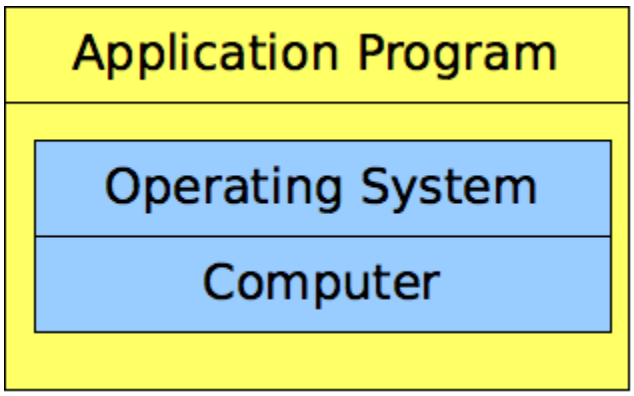

Figure 2.1: Layers of Abstraction

Binary Codes

Binary codes were invented by the inhabitants of the planet Binome, the Binome Individual uniTs, or BITs. These creates had two hands of four fingers each, giving them eight usable digits instead of the ten that most Earthlings have. Unlike Earthlings, who use their ten fingers to count to ten, the BITs use only their right hands and can only count to one.

If their hand is down, that's zero. If they raise their hand, that's one. They don't use their left hands or their fingers. It seems like such a waste, but the BITs have a clever work-around

If a BIT want to count to a larger number, say ten, they recruit three friends. Four BITs can then chose positions and count to ten with ease. The right-most position is worth 1. The next position to the left is worth 2. The next position is worth 4, and the last position is worth 8.

The final answer is the sum of the positions with hands in the air.

Say we have BITs named Alpha, Bravo, Charlie and Delta standing around. Alpha is in the first position, worth only 1, and Delta is in the fourth position, worth 8. If Alpha and Charlie raise their hands, this is positions worth 1 and 4. The total is 5. If all four BITs raise their hands, it's 8+4+2+1, which is 15. Four BITs have 16 different values, from zero (all hands down) to 15 (all hands up).

Delta (8)	Charlie (4)	Bravo (2)	Alpha (1)	total
down	down	down	down	0
down	down	down	up	1
down	down	up	down	2
down	down	up	up	2 + 1 = 3
down	up	down	down	4
down	up	down	up	4 + 1 = 5
down	up	up	down	4 + 2 = 6
down	up	up	up	4 + 2 + 1 = 7
up	down	down	down	8
up	down	down	up	8 + 1 = 9
up	down	up	down	8 + 2 = 10

A party of eight BITs can show 256 different values from zero to 255. A group of thirty-two BITs can count to over 4 billion.

The reason this scheme works is that we only have two values: on and off. This two valued (binary) system is easy to build into electronic circuits: a component is either on or off. Internally, our processor chip works in this binary arithmetic scheme because its fast and efficient.

2.1.4 Where Is The Program?

Our programs (and our data) reside in two places. When we're using a program, it must be stored in memory. However, memory is volatile, so when we're not using a program, it must reside on a disk somewhere. Since our disks are organized into file systems, we find these programs residing in files.

When we look at the various files on our computer, we'll see a number of broad categories.

1. Applications or Programs. These are executable files, they will control the computer-plus-operating system abstract machine. There are two kinds of programs:
   (a) Binary Executable programs use the processor chip's binary codes. We use these, but won't be building them.
   (b) Script programs use a script language like Python. We'll build these.
2. Documents. Our OS associates each document with a program. This is a convenient short-cut for us, and allows us to double-click the document and have the proper program start running.

When we use the Finder's **Get Info** to look at the detailed information for an application icon in MacOS or GNU/Linux, we can see that our application program icons are marked "executable" and the file type will be "application" . In Windows, a binary executable program must have a file name that ends with `.exe` (or `.com`, but this is rare).

Starting A Program. Our various operating systems give us several user interface actions that will load a program into memory so that we can start to use it. Since starting a program is the primary purpose of an operating system, there are many ways to accomplish this.

- Double click an application icon
- Double click a document icon
- Single click something in the dock or task bar
- Click on a **run...** menu item in the **Start...** menu
- Use the Windows **Command Prompt**; in GNU/Linux or the MacOS it is called a **terminal**. Through the terminal window we interact with a shell program that allows us to type the name of another program to have that started.

All of these actions are just different ways to get the operating system to locate the binary executable, load it into memory and give it the resources to do its unique task.

All of these choices boil down to two overlapping paths to a starting a binary executable:

- **From the application icon**. In this case, we clicked the icon that represents the binary application itself. The OS loaded the binary application and started it. Once started, some programs will open a blank document, some will give you a window that lets you pick what you want to do. Others may have saved a preferences file that identifies what document you last worked with and open that document for you when they start. This varies a great deal, there is no single rule to capture the wide variations in start-up behavior of programs.
- **From a document icon**. In this case, we clicked on an icon that represents a document; the document is associated with a specific binary program. The OS uses this association to find and start the appropriate program. The OS also provides the binary program with the name of the document file we clicked so that the program can open the document for you. This provides the easy-to-use experience of clicking on your document and being able to make changes to it.

2.1.5 Concepts Exercises

1. **Inventory Your System**.