1. Computer Programming and Coding

Think about some of the different ways that people use computers. The uses of computers are almost limitless in our everyday lives. Computers can perform such a wide variety of tasks because they can be programmed. This means that computers are not designed to do just one job, but to do any job that their programs tell them to do. A *program* is a set of instructions that a computer follows to perform a task.

Programs are commonly referred to as *software*. Software is essential to a computer because it controls everything the computer does. All of the software that we use to make our computers useful is created by individuals working as programmers or software developers.
A *programmer*, or *software developer*, is a person with the training and skills necessary to design, create, and test computer programs. Computer programming is an exciting and rewarding career. Today, you will find programmers' work used in business, medicine, government, law enforcement, agriculture, academics, entertainment, and many other fields.

1.1 Hardware and software

Hardware

The term *hardware* refers to all of the physical devices, or *components*, of which a computer is made. A computer is not one single device, but a system of devices that all work together. Like the different instruments in a symphony orchestra, each device in a computer plays its own part.

If you have ever shopped for a computer, you've probably seen sales literature listing components such as microprocessors, memory, disk drives, video displays, graphics cards, and so on. Unless you already know a lot about computers, or at least have a friend that does, understanding what these different components do might be challenging.

As shown in Figure 1-1, a typical computer system consists of the following major components:

- The central processing unit (CPU)
- Main memory
- Secondary storage devices
- Input devices
- Output devices

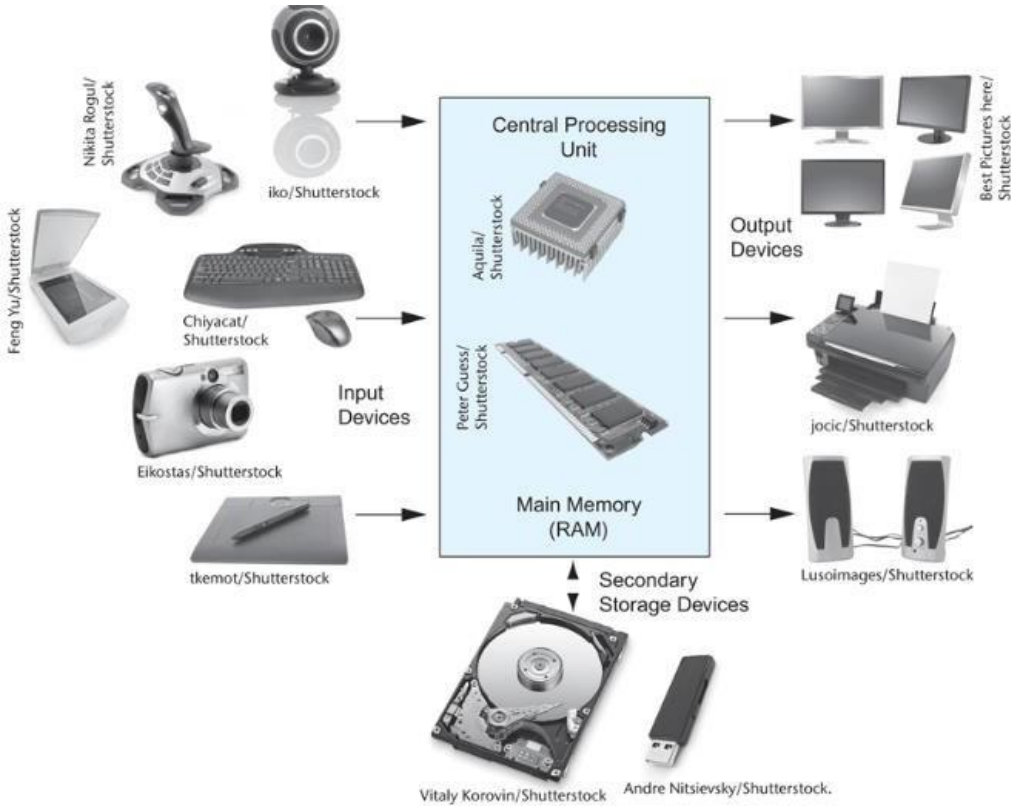


Figure 1-1 Typical components of a computer system

The CPU

When a computer is performing the tasks that a program tells it to do, we say that the computer is *running* or *executing* the program. The *central processing unit*, or *CPU*, is the part of a computer that actually runs programs. The CPU is the most important component in a computer because without it, the computer could not run software.

In the earliest computers, CPUs were huge devices made of electrical and mechanical components such as vacuum tubes and switches. **Figure 1-2** shows such a device. The two women in the photo are working with the historic ENIAC computer. The ENIAC, which is considered by many to be the world's first programmable electronic computer, was built in 1945 to calculate artillery ballistic tables for the U.S. Army. This machine, which was primarily one big CPU, was 8 feet tall, 100 feet long, and weighed 30 tons.

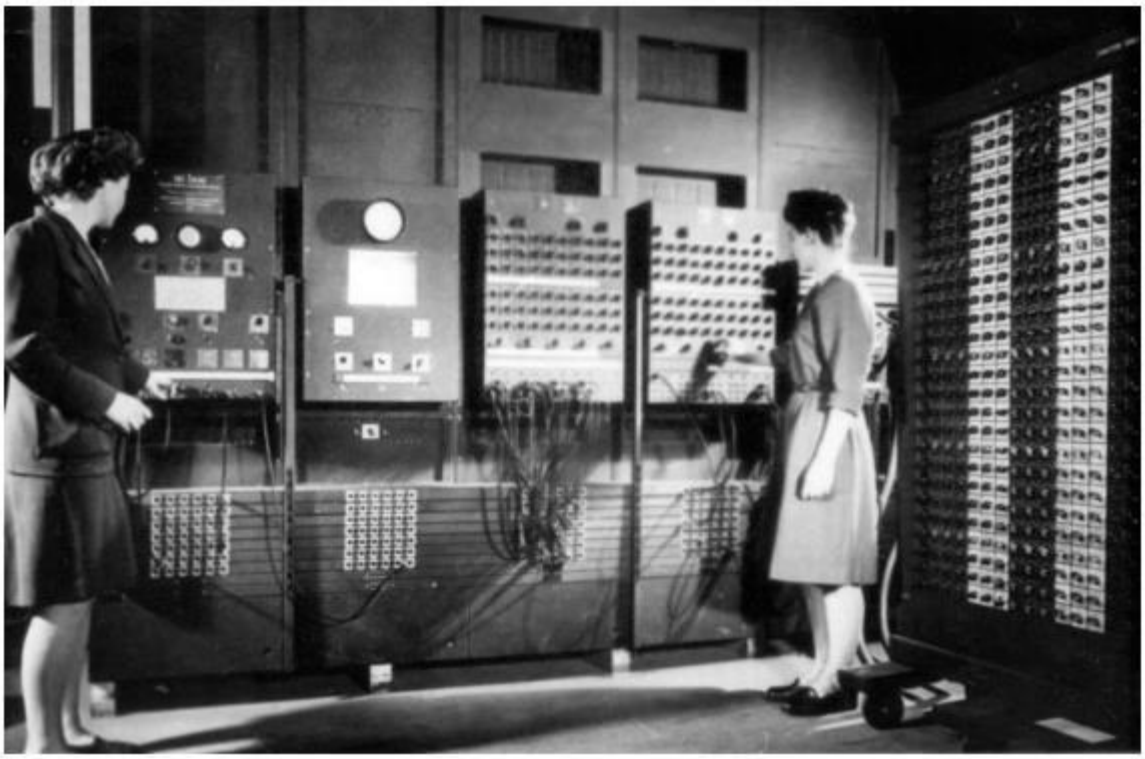


Figure 1-2 The ENIAC computer (*courtesy of U.S. Army Historic Computer Images)*

Today, CPUs are small chips known as *microprocessors*. **Figure 1-3** shows a photo of a lab technician holding a modern microprocessor. In addition to being much smaller than the old electromechanical CPUs in early computers, microprocessors are also much more powerful.

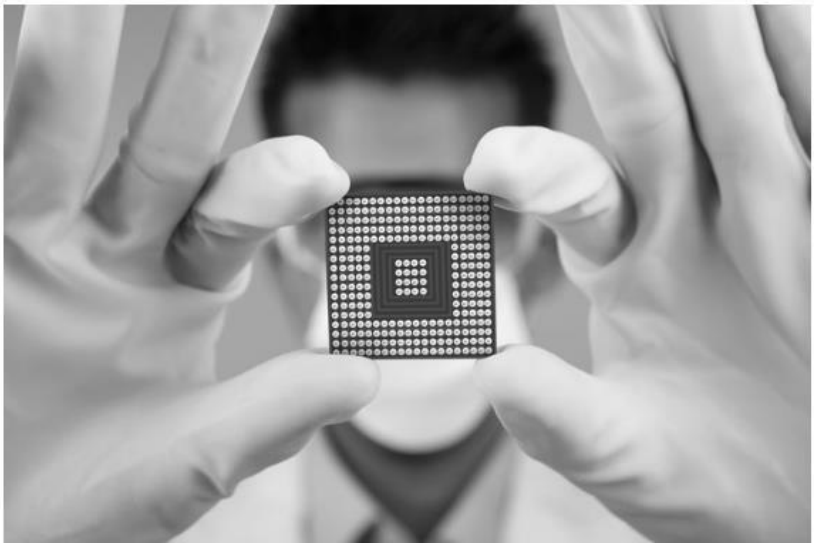


Figure 1-3 A lab technician holds a modern microprocessor (*Creativa/Shutterstock)*

Main Memory

You can think of *main memory* as the computer's work area. This is where the computer stores a program while the program is running, as well as the data that the program is working with. For example, suppose you are using a word processing program to write an essay for one of your classes. While you do this, both the word processing program and the essay are stored in main memory.

Main memory is commonly known as *random-access memory*, or *RAM*. It is called this because the CPU is able to quickly access data stored at any random location in RAM. RAM is usually a *volatile* type of memory that is used only for temporary storage while a program is running. When the computer is turned off, the contents of RAM are erased. Inside your computer, RAM is stored in chips, similar to the ones shown in **Figure 1-4**.

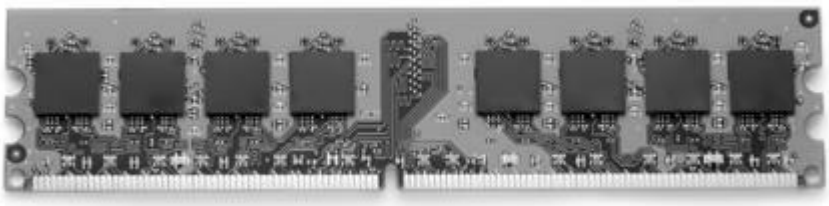


Figure 1-4 Memory chips *(Garsya/Shutterstock)*

Secondary Storage Devices

Secondary storage is a type of memory that can hold data for long periods of time, even when there is no power to the computer. Programs are normally stored in secondary memory and loaded into main memory as needed. Important data, such as word processing documents, payroll data, and inventory records, is saved to secondary storage as well.

The most common type of secondary storage device is the *disk drive*. A traditional disk drive stores data by magnetically encoding it onto a spinning circular disk. *Solid-state drives*, which store data in solid-state memory, are increasingly becoming popular. A solid-state drive has no moving parts and operates faster than a traditional disk drive. Most computers have some sort of secondary storage device, either a traditional disk drive or a solid-state drive, mounted inside their case. External storage devices, which connect to one of the computer's communication ports, are also available. External storage devices can be used to create backup copies of important data or to move data to another computer.

In addition to external storage devices, many types of devices have been created for copying data and for moving it to other computers. For example, *USB drives* are small devices that plug into the computer's USB (universal serial bus) port and appear to the system as a disk drive. These drives do not actually contain a disk, however. They store data in a special type of memory known as *flash memory*. USB drives, which are also known as *memory sticks* and *flash drives*, are inexpensive, reliable, and small enough to be carried in your pocket.

Optical devices such as the *CD* (compact disc) and the *DVD* (digital versatile disc) are also popular for data storage. Data is not recorded magnetically on an optical disc, but is encoded as a series of pits on the disc surface. CD and DVD drives use a laser to detect the pits and thus read the encoded data. Optical discs hold large amounts of data, and for that reason, recordable CD and DVD drives are commonly used for creating backup copies of data.

Input Devices

Input is any data the computer collects from people and from other devices. The component that collects the data and sends it to the computer is called an *input device*. Common input devices are the keyboard, mouse, touchscreen, scanner, microphone, and digital camera. Disk drives and optical drives can also be considered input devices, because programs and data are retrieved from them and loaded into the computer's memory.

Output Devices

Output is any data the computer produces for people or for other devices. It might be a sales report, a list of names, or a graphic image. The data is sent to an *output device,* which formats and presents it. Common output devices are video displays and printers. Disk drives can also be considered output devices because the system sends data to them in order to be saved.

Software

If a computer is to function, software is not optional. Everything computer does, from the time you turn the power switch on until you shut the system down, is under the control of software. There are two general categories of software: system software and application software. Most computer programs clearly fit into one of these two categories. Let's take a closer look at each.

System Software

The programs that control and manage the basic operations of a computer are generally referred to as *system software*. System software typically includes the following types of programs:

Operating Systems

An *operating system* is the most fundamental set of programs on a computer. The operating system controls the internal operations of the computer's hardware, manages all of the devices connected to the computer, allows data to be saved to and retrieved from storage devices, and allows other programs to run on the computer. Popular operating systems for laptop and desktop computers include Windows, macOS, and Linux. Popular operating systems for mobile devices include Android and iOS.

Utility Programs

A *utility program* performs a specialized task that enhances the computer's operation or safeguards data. Examples of utility programs are virus scanners, file compression programs, and data backup programs.

Software Development Tools

Software development tools are the programs that programmers use to create, modify, and test software. Assemblers, compilers, and interpreters are examples of programs that fall into this category.

Application Software

Programs that make a computer useful for everyday tasks are known as *application software*. These are the programs that people normally spend most of their time running on their computers. Two commonly used applications: Microsoft Word, a word processing program, and PowerPoint, a presentation program. Some other examples of application software are spreadsheet programs, email programs, web browsers, and game programs.

1.2 How Computers Store Data

A computer's memory is divided into tiny storage locations known as *bytes*. One byte is only enough memory to store a letter of the alphabet or a small number. In order to do anything meaningful, a computer has to have lots of bytes. Most computers today have millions, or even billions, of bytes of memory.

Each byte is divided into eight smaller storage locations known as bits. The term *bit* stands for *binary digit*. Computer scientists usually think of bits as tiny switches that can be either on or off. Bits aren't actual "switches," however, at least not in the conventional sense. In most computer systems, bits are tiny electrical components that can hold either a positive or a negative charge. Computer scientists think of a positive charge as a switch in the *on* position, and a negative charge as a switch in the *off* position. **Figure 1-5** shows the way that a computer scientist might think of a byte of memory: as a collection of switches that are each flipped to either the on or off position.

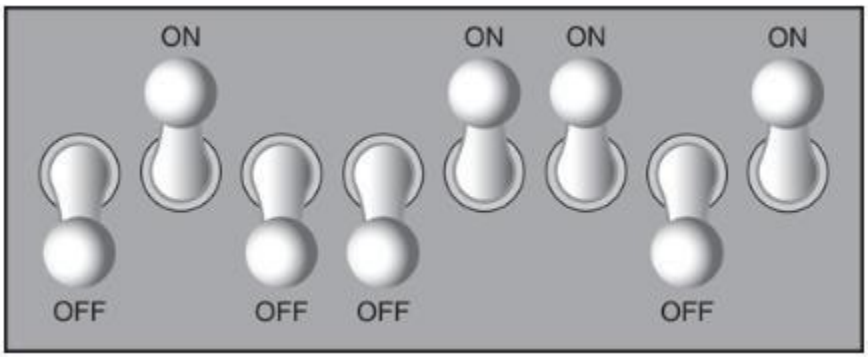


Figure 1-5 Think of a byte as eight switches

When a piece of data is stored in a byte, the computer sets the eight bits to an on/off pattern that represents the data. For example, the pattern on the left in **Figure 1-6** shows how the number 77 would be stored in a byte, and the pattern on the right shows how the letter A would be stored in a byte. We explain below how these patterns are determined.

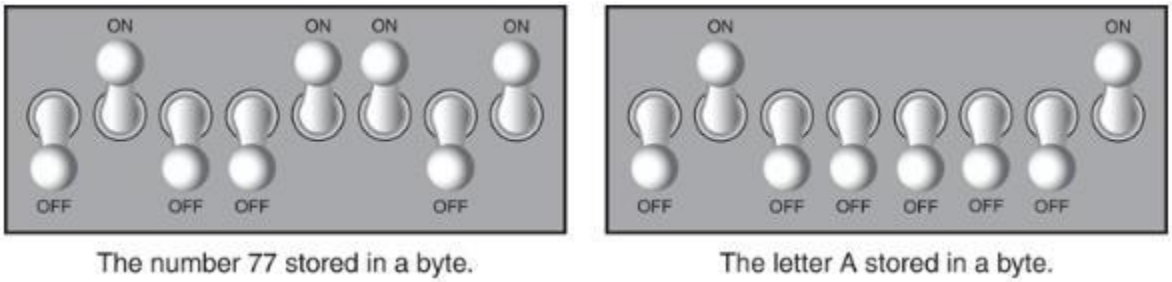


Figure 1-6 Bit patterns for the number 77 and the letter A

Storing Numbers

A bit can be used in a very limited way to represent numbers. Depending on whether the bit is turned on or off, it can represent one of two different values. In computer systems, a bit that is turned off represents the number 0, and a bit that is turned on represents the number 1. This corresponds perfectly to the *binary numbering system*. In the binary numbering system (or *binary*, as it is usually called), all numeric values are written as sequences of 0s and 1s. Here is an example of a number that is written in binary:

10011101

The position of each digit in a binary number has a value assigned to it. Starting with the rightmost digit and moving left, the position values are 2^0, 2^1, 2^2, 2^3, and so forth, as shown in **Figure 1-7**. **Figure 1-8** shows the same diagram with the position values calculated. Starting with the rightmost digit and moving left, the position values are 1, 2, 4, 8, and so forth.

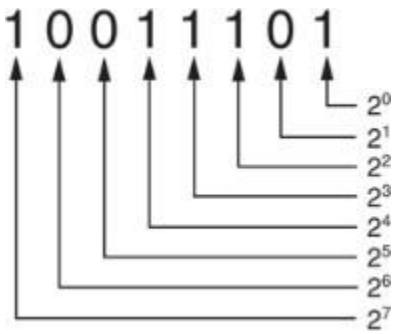


Figure 1-7 The values of binary digits as powers of 2

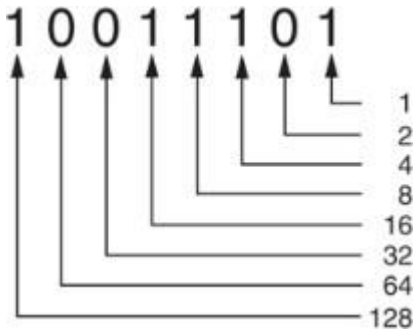


Figure 1-8 The values of binary digits

To determine the value of a binary number, you simply add up the position values of all the 1s. For example, in the binary number 10011101, the position values of the 1s are 1, 4, 8, 16, and 128. This is shown in **Figure 1-9**. The sum of all of these position values is 157. So, the value of the binary number 10011101 is 157.

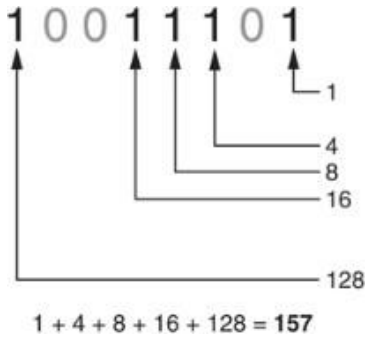


Figure 1-9 Determining the value of 10011101

Figure 1-10 shows how you can picture the number 157 stored in a byte of memory. Each 1 is represented by a bit in the on position, and each 0 is represented by a bit in the off position.

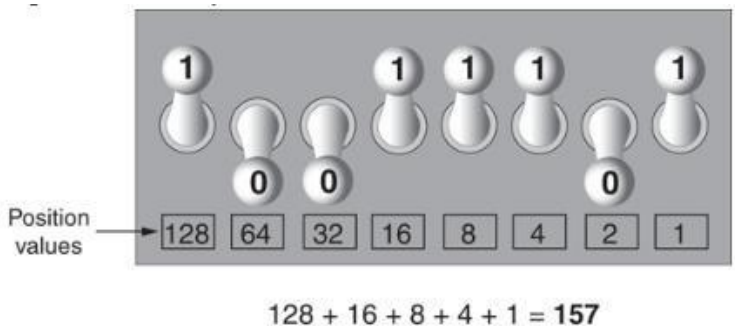


Figure 1-10 The bit pattern for 157

When all of the bits in a byte are set to 0 (turned off), then the value of the byte is 0. When all of the bits in a byte are set to 1 (turned on), then the byte holds the largest value that can be stored in it. The largest value that can be stored in a byte is 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = 255. This limit exists because there are only eight bits in a byte.

What if you need to store a number larger than 255? The answer is simple: use more than one byte. For example, suppose we put two bytes together. That gives us 16 bits. The position values of those 16 bits would be 2^0, 2^1, 2^2, 2^3, and so forth, up through 21^5. As shown in **Figure 1-11**, the maximum value that can be stored in two bytes is 65,535. If you need to store a number larger than this, then more bytes are necessary.

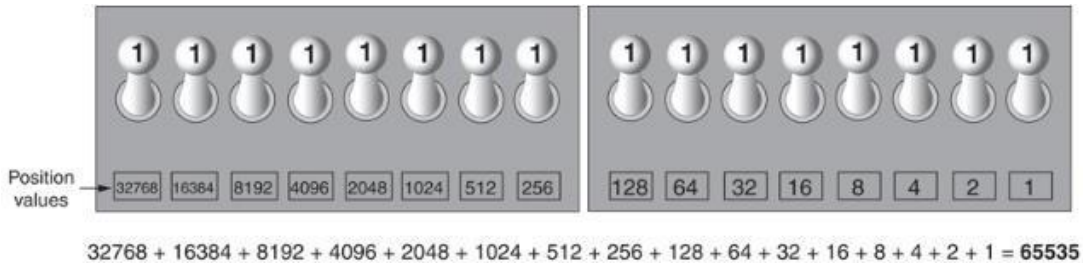


Figure 1-11 Two bytes used for a large number

Storing Characters

Any piece of data that is stored in a computer's memory must be stored as a binary number. That includes characters, such as letters and punctuation marks. When a character is stored in memory, it is first converted to a numeric code. The numeric code is then stored in memory as a binary number.

Over the years, different coding schemes have been developed to represent characters in computer memory. Historically, the most important of these coding schemes is *ASCII*, which stands for the *American Standard Code for Information Interchange*. ASCII is a set of 128 numeric codes that represent the English letters, various punctuation marks, and other characters. For example, the ASCII code for the uppercase letter A is 65. When you type an uppercase A on your computer keyboard, the number 65 is stored in memory (as a binary number, of course). This is shown in **Figure 1-12**.

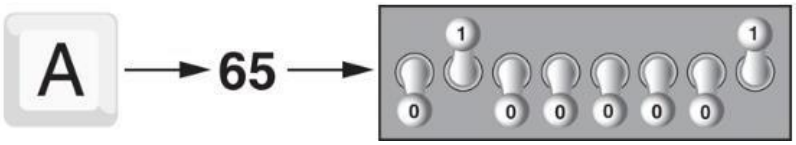


Figure 1-12 The letter A is stored in memory as the number 65

In case you are curious, the ASCII code for uppercase B is 66, for uppercase C is 67, and so forth. The ASCII character set was developed in the early 1960s and was eventually adopted by almost all computer manufacturers. ASCII is limited, however, because it defines codes for only 128 characters. To remedy this, the Unicode character set was developed in the early 1990s. *Unicode* is an extensive encoding scheme that is compatible with ASCII, but can also represent characters for many of the languages in the world. Today, Unicode is quickly becoming the standard character set used in the computer industry.

Advanced Number Storage

Earlier, you read about numbers and how they are stored in memory. While reading that section, perhaps it occurred to you that the binary numbering system can be used to represent only integer numbers, beginning with 0. Negative numbers and real numbers (such as 3.14159) cannot be represented using the simple binary numbering technique we discussed.

Computers are able to store negative numbers and real numbers in memory, but to do so they use encoding schemes along with the binary numbering system. Negative numbers are encoded using a technique known as *two's complement,* and real numbers are encoded in *floating-point notation*. You don't need to know how these encoding schemes work, only that they are used to convert negative numbers and real numbers to binary format.

Other Types of Data

Computers are often referred to as digital devices. The term *digital* can be used to describe anything that uses binary numbers. *Digital data* is data that is stored in binary format, and a *digital device* is any device that works with binary data. In this section, we have discussed how numbers and characters are stored in binary, but computers also work with many other types of digital data.

For example, consider the pictures that you take with your digital camera. These images are composed of tiny dots of color known as *pixels*. (The term pixel stands for *picture element*.) As shown in **Figure 1-13**, each pixel in an image is converted to a numeric code that represents the pixel's color. The numeric code is stored in memory as a binary number.

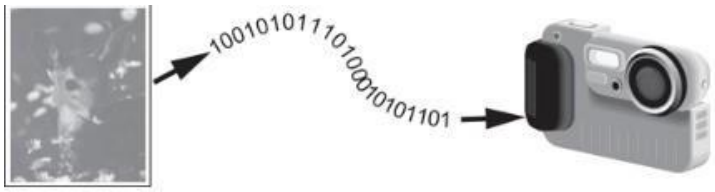


Figure 1.13 A digital image is stored in binary format

The music that you play on your CD player, iPod, or MP3 player is also digital. A digital song is broken into small pieces known as *samples*. Each sample is converted to a binary number, which can be stored in memory. The more samples that a song is divided into, the more it sounds like the original music when it is played back. A CD quality song is divided into more than 44,000 samples per second!

1.3 How a Program Works

Earlier, we stated that the CPU is the most important component in a computer because it is the part of the computer that runs programs. Sometimes the CPU is called the "computer's brain" and is described as being "smart." Although these are common metaphors, you should understand that the CPU is not a brain, and it is not smart. The CPU is an electronic device that is designed to do specific things. In particular, the CPU is designed to perform operations such as the following:

- Reading a piece of data from main memory
- Adding two numbers
- Subtracting one number from another number
- Multiplying two numbers
- Dividing one number by another number
- Moving a piece of data from one memory location to another
- Determining whether one value is equal to another value

As you can see from this list, the CPU performs simple operations on pieces of data. The CPU does nothing on its own, however. It has to be told what to do, and that's the purpose of a program. A program is nothing more than a list of instructions that cause the CPU to perform operations.

Each instruction in a program is a command that tells the CPU to perform a specific operation. Here's an example of an instruction that might appear in a program:

10110000

To you and me, this is only a series of 0s and 1s. To a CPU, however, this is an instruction to perform an operation. **1** It is written in 0s and 1s because CPUs only understand instructions that are written in *machine language*, and machine language instructions always have an underlying binary structure.

A machine language instruction exists for each operation that a CPU is capable of performing. For example, there is an instruction for adding numbers, there is an instruction for subtracting one number from another, and so forth. The entire set of instructions that a CPU can execute is known as the CPU's *instruction set.*

The machine language instruction that was previously shown is an example of only one instruction. It takes a lot more than one instruction, however, for the computer to do anything meaningful. Because the operations that a CPU knows how to perform are so basic in nature, a meaningful task can be accomplished only if the CPU performs many operations. For example, if you want your computer to calculate the amount of interest that you will earn from your savings account this year, the CPU will have to perform a large number of instructions, carried out in the

proper sequence. It is not unusual for a program to contain thousands or even millions of machine language instructions.

Programs are usually stored on a secondary storage device such as a disk drive. When you install a program on your computer, the program is typically copied to your computer's disk drive from a CD-ROM, or downloaded from a website.

Although a program can be stored on a secondary storage device such as a disk drive, it has to be copied into main memory, or RAM, each time the CPU executes it. For example, suppose you have a word processing program on your computer's disk. To execute the program, you use the mouse to double-click the program's icon. This causes the program to be copied from the disk into main memory. Then, the computer's CPU executes the copy of the program that is in main memory. This process is illustrated in **Figure 1-14**.

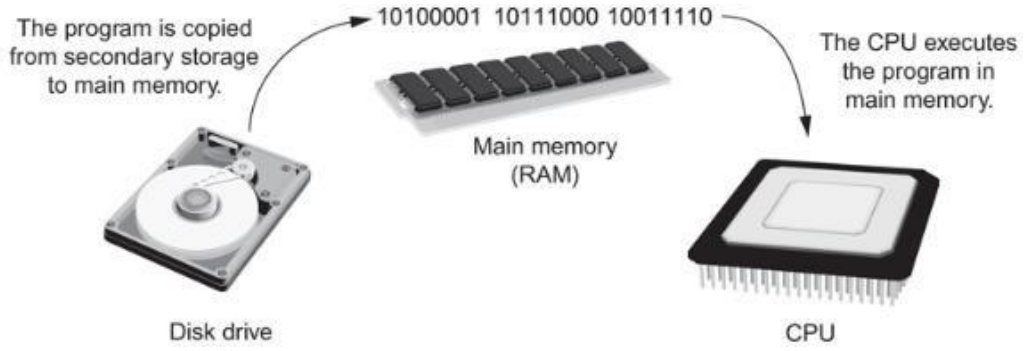


Figure 1-14 A program is copied into main memory and then executed

When a CPU executes the instructions in a program, it is engaged in a process that is known as the *fetch-decode-execute cycle*. This cycle, which consists of three steps, is repeated for each instruction in the program. The steps are:

Fetch. A program is a long sequence of machine language instructions. The first step of the cycle is to fetch, or read, the next instruction from memory into the CPU.

Decode. A machine language instruction is a binary number that represents a command that tells the CPU to perform an operation. In this step, the CPU decodes the instruction that was just fetched from memory, to determine which operation it should perform.

Execute. The last step in the cycle is to execute, or perform, the operation.

Figure 1-15 illustrates these steps.

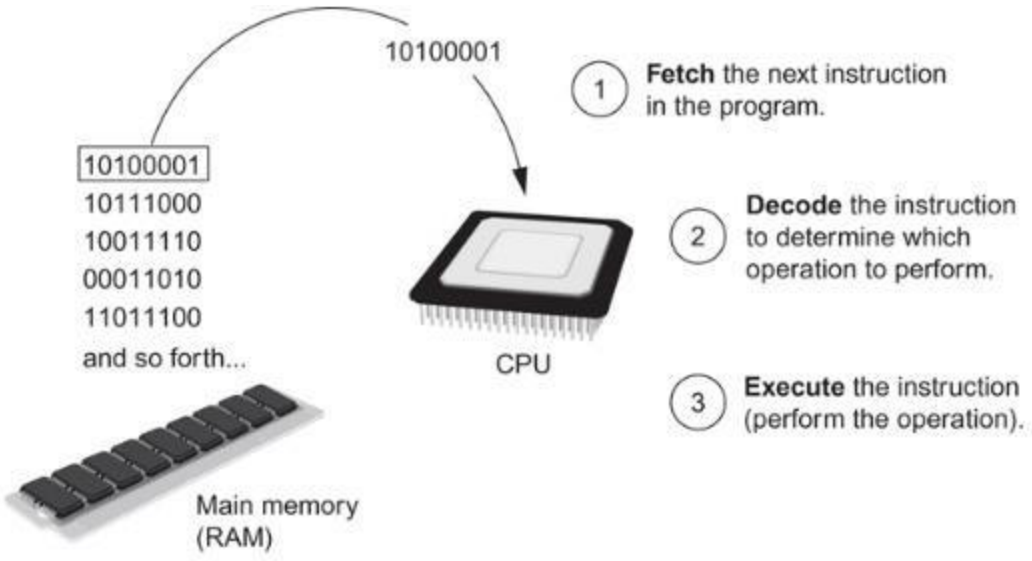


Figure 1-15 The fetch-decode-execute cycle

From Machine Language to Assembly Language

Computers can only execute programs that are written in machine language. As previously mentioned, a program can have thousands or even millions of binary instructions, and writing such a program would be very tedious and time consuming. Programming in machine language would also be very difficult, because putting a 0 or a 1 in the wrong place will cause an error.

Although a computer's CPU only understands machine language, it is impractical for people to write programs in machine language. For this reason, *assembly language* was created in the early days of computing as an alternative to machine language. Instead of using binary numbers for instructions, assembly language uses short words that are known as *mnemonics*. For example, in assembly language, the mnemonic **add** typically means to add numbers, **mul** typically means to multiply numbers, and **mov** typically means to move a value to a location in memory. When a programmer uses assembly language to write a program, he or she can write short mnemonics instead of binary numbers.

Assembly language programs cannot be executed by the CPU, however. The CPU only understands machine language, so a special program known as an *assembler* is used to translate an assembly language program to a machine language program. This process is shown in **Figure 1-16**. The machine language program that is created by the assembler can then be executed by the CPU.

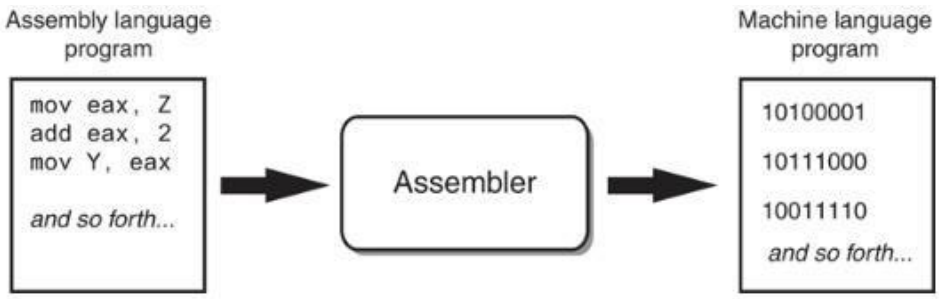


Figure 1-16 An assembler translates an assembly language program to a machine language program Assembly language

1.4. High-Level Languages

Although assembly language makes it unnecessary to write binary machine language instructions, it is not without difficulties. Assembly language is primarily a direct substitute for machine language, and like machine language, it requires that you know a lot about the CPU. Assembly language also requires that you write a large number of instructions for even the simplest program. Because assembly language is so close in nature to machine language, it is referred to as a *low-level language*.

In the 1950s, a new generation of programming languages known as *high-level languages* began to appear. A high-level language allows you to create powerful and complex programs without knowing how the CPU works and without writing large numbers of low-level instructions. In addition, most high-level languages use words that are easy to understand. For example, if a programmer were using COBOL (which was one of the early high-level languages created in the 1950s), he or she would write the following instruction to display the message *Hello world* on the computer screen:

```
DISPLAY "Hello world"
```

Python is a modern, high-level programming language that we will use in this book. In Python you would display the message *Hello world* with the following instruction:

```
print('Hello world')
```

Doing the same thing in assembly language would require several instructions and an intimate knowledge of how the CPU interacts with the computer's output device. As you can see from this example, high-level languages allow programmers to concentrate on the tasks they want to perform with their programs, rather than the details of how the CPU will execute those programs.

Since the 1950s, thousands of high-level languages have been created. **Table 1-1** lists several of the more well-known languages.

Table 1-1 Programming languages

Language	Description
Ada	Ada was created in the 1970s, primarily for applications used by the U.S. Department of Defense. The language is named in honor of Countess Ada Lovelace, an influential and historic figure in the field of computing.
BASIC	Beginners All-purpose Symbolic Instruction Code is a general-purpose language that was originally designed in the early 1960s to be simple enough for beginners to learn. Today, there are many different versions of BASIC.
FORTRAN	FORmula TRANslator was the first high-level programming language. It was designed in the 1950s for performing complex mathematical calculations.
COBOL	Common Business-Oriented Language was created in the 1950s and was designed for business applications.
Pascal	Pascal was created in 1970 and was originally designed for teaching programming. The language was named in honor of the mathematician, physicist, and philosopher Blaise Pascal.
C and C++	C and C++ (pronounced “c plus plus”) are powerful, general-purpose languages developed at Bell Laboratories. The C language was created in 1972, and the C++ language was created in 1983.
C#	Pronounced “c sharp.” This language was created by Microsoft around the year 2000 for developing applications based on the Microsoft .NET platform.
Java	Java was created by Sun Microsystems in the early 1990s. It can be used to develop programs that run on a single computer or over the Internet from a web server.
JavaScript	JavaScript, created in the 1990s, can be used in Web pages. Despite its name, JavaScript is not related to Java.
Python	Python is a general-purpose language created in the early 1990s. It has become popular in business and academic applications.

Language	Description
Ruby	Ruby is a general-purpose language that was created in the 1990s. It is increasingly becoming a popular language for programs that run on Web servers.
Visual Basic	Visual Basic (commonly known as VB) is a Microsoft programming language and software development environment that allows programmers to create Windows-based applications quickly. VB was originally created in the early 1990s.

Each high-level language has its own set of predefined words that the programmer must use to write a program. The words that make up a high-level programming language are known as *key words* or *reserved words*. Each key word has a specific meaning, and cannot be used for any other purpose. **Table 1-2** shows all of the Python key words.

Table 1-2 The Python key words

and	del	from	None	True
as	elif	global	nonlocal	try
assert	else	if	not	while
break	except	import	or	with
class	false	in	pass	yield
continue	finally	is	raise	
def	for	lambda	return	

In addition to key words, programming languages have *operators* that perform various operations on data. For example, all programming languages have math operators that perform arithmetic. In Python, as well as most other languages, the + sign is an operator that adds two numbers. The following adds 12 and 75:

```
12 + 75
```

There are numerous other operators in the Python language, many of which you will learn about as you progress through this course.

In addition to key words and operators, each language also has its own *syntax*, which is a set of rules that must be strictly followed when writing a program. The syntax rules dictate how key words, operators, and various punctuation characters must be used in a program. When you are learning a programming language, you must learn the syntax rules for that particular language.

The individual instructions that you use to write a program in a high-level programming language are called *statements*. A programming statement can consist of key words, operators, punctuation, and other allowable programming elements, arranged in the proper sequence to perform an operation.

Compilers and Interpreters

Because the CPU understands only machine language instructions, programs that are written in a high-level language must be translated into machine language. Depending on the language in which a program has been written, the programmer will use either a compiler or an interpreter to make the translation.

A *compiler* is a program that translates a high-level language program into a separate machine language program. The machine language program can then be executed any time it is needed. This is shown in **Figure 1-17**. As shown in the figure, compiling and executing are two different processes.

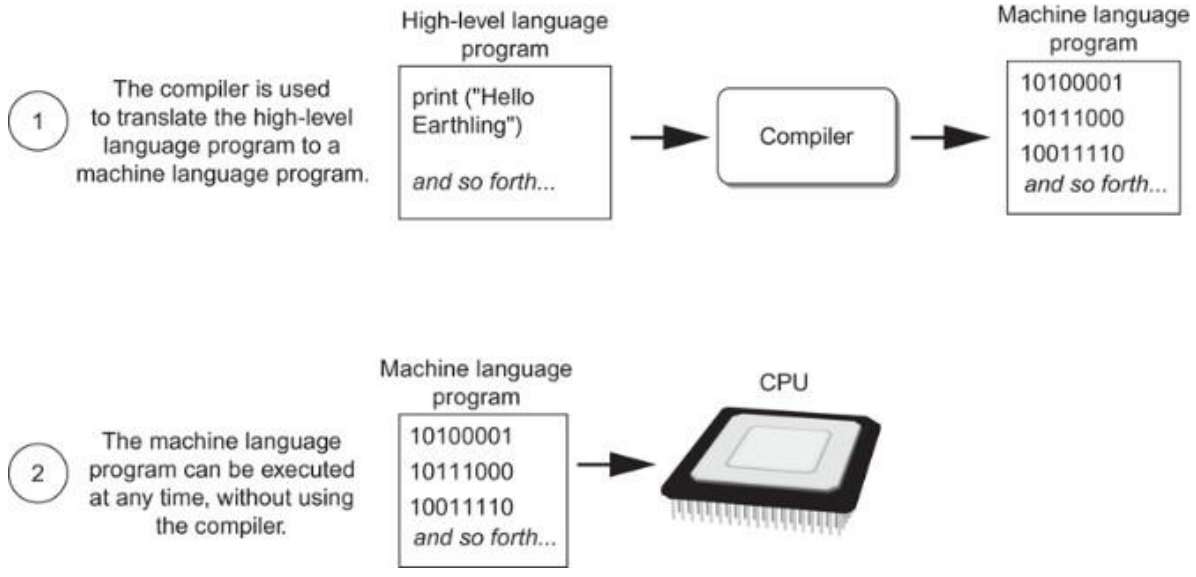


Figure 1-17 Compiling a high-level program and executing it

The Python language uses an *interpreter*, which is a program that both translates and executes the instructions in a high-level language program. As the interpreter reads each individual instruction in the program, it converts it to machine language instructions then immediately executes them. This process repeats for every instruction in the program. This process is illustrated in **Figure 1-18**. Because interpreters combine translation and execution, they typically do not create separate machine language programs.

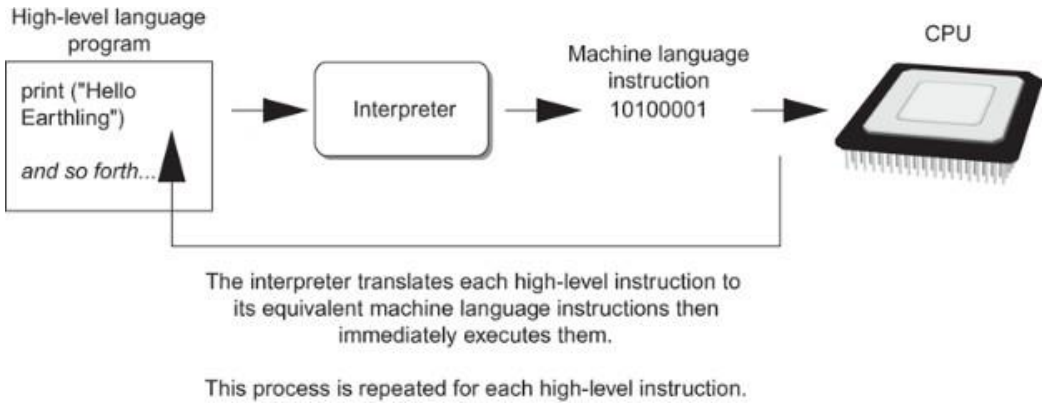


Figure 1-18 Executing a high-level program with an interpreter

The statements that a programmer writes in a high-level language are called *source code*, or simply *code*. Typically, the programmer types a program's code into a text editor then saves the code in a file on the computer's disk. Next, the programmer uses a compiler to translate the code into a machine language program, or an interpreter to translate and execute the code. If the code contains a syntax error, however, it cannot be translated. A *syntax error* is a mistake such as a misspelled key word, a missing punctuation character, or the incorrect use of an operator. When this happens, the compiler or interpreter displays an error message indicating that the program contains a syntax error. The programmer corrects the error then attempts once again to translate the program.

1.4. Web Technology

The World Wide Web -www

The World Wide Web (WWW) is a network of online content that is formatted in HTML and accessed via HTTP. The term refers to all the interlinked HTML pages that can be accessed over the Internet. The World Wide Web was originally designed in 1991 by Tim Berners-Lee while he was a contractor at CERN.

The World Wide Web is most often referred to simply as "the Web."

The Internet

The **Internet**, sometimes called simply "the Net," is a worldwide system of computer networks - a network of networks in which users at any one computer can, if they have permission, get information from any other computer (and sometimes talk directly to users at other computers).

Using the Internet, computers connect and communicate with each other, primarily using the TCP/IP (Transmission Control Protocol / Internet Protocol). Think of TCP/IP as a book of rules, a step-by-step guide that each computer uses to know how to talk to another computer.

ISPs (Internet service providers)

ISPs (Internet service providers), the companies that provide Internet service and connectivity, also follow these rules. The ISP provides a bridge between your computer and all the other computers in the world, which are all a part of the Internet. The ISP uses the TCP/IP protocols to make computer-to-computer connections possible and transmit data between them. When connected to an ISP, you are assigned an IP address, which is a unique address given to your computer or network and allows it to be found while on the Internet.

URL (Uniform Resource Locator)

Abbreviated as *URL*, a Uniform Resource Locator is a way of identifying the location of a file on the internet. They're what we use to open not only websites, but also to download images, videos, software programs, and other types of files that are hosted on a server.

Opening a *local* file on your computer is as simple as double-clicking it, but to open files on *remote* computers, like web servers, we must use URLs so that our web browser knows where to look. For example, opening the HTML file that represents the web page explained below, is done by entering it into the navigation bar at the top of the browser you're using.

Uniform Resource Locators are most commonly abbreviated as *URLs* but they're also called *website addresses* when they refer to URLs that use the HTTP or HTTPS protocol.

URL is usually pronounced with each letter spoken individually (i.e. *u* - *r* - *l*, not *earl*). It used to be an abbreviation for *Universal Resource Locator* before being changed to Uniform Resource Locator.

Examples of URLs

You're probably used to entering in URL, like this one for accessing Google's website:

```
https://www.google.com
```

The entire address is called the URL. Another example is this website (first) and Microsoft's (second):

```
https://lifewire.com
https://www.microsoft.com
```

Web Server and Hosting

Server hosting refers to the outsourcing of an organization's server placement and platform to a third-party Managed Hosting Provider (MSP). A client uses the Internet to connect to data and applications on a managed server and pays a recurring fee to the hosting provider. A MSP usually operates and manages large data centers with dozens, hundreds or thousands of hosted servers for two or more clients. This model is known as colocation or colocated hosting.

The server hosting model provides a best-of-all-worlds scenario by giving organizations access to server platforms to host their applications and data without shouldering attendant costs.

HTML and CSS

ML (the Hypertext Markup Language) and CSS (Cascading Style Sheets) are two of the core technologies for building Web pages. HTML provides the *structure* of the page, CSS the (visual and aural) *layout*, for a variety of devices. Along with graphics and scripting, HTML and CSS are the basis of building Web pages and Web Applications. Learn more below about:

What is HTML?

HTML is the language for describing the structure of Web pages. HTML gives authors the means to:

- Publish online documents with headings, text, tables, lists, photos, etc.
- Retrieve online information via hypertext links, at the click of a button.
- Design forms for conducting transactions with remote services, for use in searching for information, making reservations, ordering products, etc.
- Include spread-sheets, video clips, sound clips, and other applications directly in their documents.

With HTML, authors describe the structure of pages using *markup*. The *elements* of the language label pieces of content such as “paragraph,” “list,” “table,” and so on.

What is XHTML?

XHTML is a variant of HTML that uses the syntax of XML, the Extensible Markup Language. XHTML has all the same elements (for paragraphs, etc.) as the HTML variant, but the syntax is slightly different. Because XHTML is an XML application, you can use other XML tools with it (such as XSLT, a language for transforming XML content).

What is CSS?

CSS is the language for describing the presentation of Web pages, including colors, layout, and fonts. It allows one to adapt the presentation to different types of devices, such as large screens, small screens, or printers. CSS is independent of HTML and can be used with any XML-based markup language. The separation of HTML from CSS makes it easier to maintain sites, share style sheets across pages, and tailor pages to different environments. This is referred to as the *separation of structure (or: content) from presentation.*

Text Editor

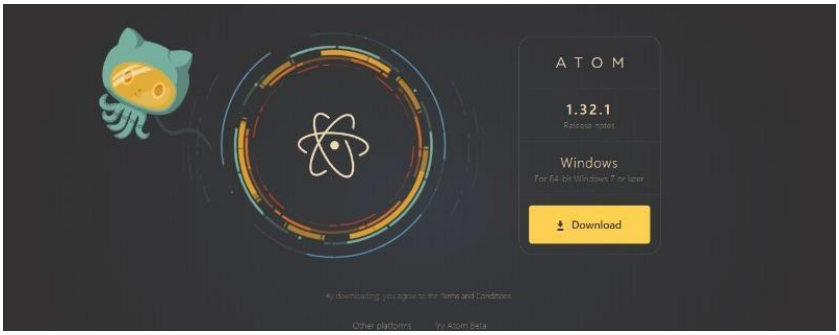
A **text editor** is program that allows you to open, view, and edit plain **text** files. Unlike word processors, **text editors** do not add formatting to **text**, instead focusing on **editing** functions for plain **text**. **Text editors** are **used by** a wide variety of people, for a wide variety of purposes.

Editing HTML and CSS code can be done without any specific tools. In fact, if you have a simple text editor, you are good to go. However, just because you can do something doesn't mean it is the best way to do it – and that applies to web development as well.

Web development IDE does all the things simple text editors do plus a number of more advanced stuff that you can't do with text editors. For instance, while an editor such as Sublime or Atom can be used as an HTML CSS JavaScript editor, they only allow you to write code.

Of course, they come with a bunch of convenient features such as syntax highlighting, customizable interfaces, and extensive navigation tools, you will need additional features to make a functional app. For example, you will need a debugger and a compiler.

Atom by GitHub



Atom by Github is the best editor for JavaScript if you are looking for something customizable and easy to use. It has a built-in package manager for installing new packages or start creating your own within this cool tool.

Atom comes pre-installed with four UI and eight syntax themes in a variety of colors. The rich and supportive community also creates cool themes for everybody to use so you might find what you're looking for there.

Here are some of the Atom's best features:

- It works across different operating systems such as OS X, Windows, or Linux
- Find, preview, and replace text as you type in a file or across all your projects.
- Easily browse and open a single file, a whole project, or multiple projects in one window.

Atom is a desktop app built with HTML, JavaScript, CSS, and Node.js integration. It runs on Electron, a framework for building cross-platform apps using web technologies. It is definitely a web development IDE worth checking out if you are looking for JavaScript development tools and best HTML IDE.

Bibliography

Duckett, J. (2016). *HTML and CSS Design and build websites.* Indianapolis: John Wiley and Sons Inc.

Duckett, J. (2016). *JavaScript and JQuery: interactive front-end developer.* Indianapolis: John Wiley and Sons Inc.

HTML, CSS, JavaScript, JQuery, and Bootstrap. (2017, June). Retrieved from w3schools: www.w3schools.com