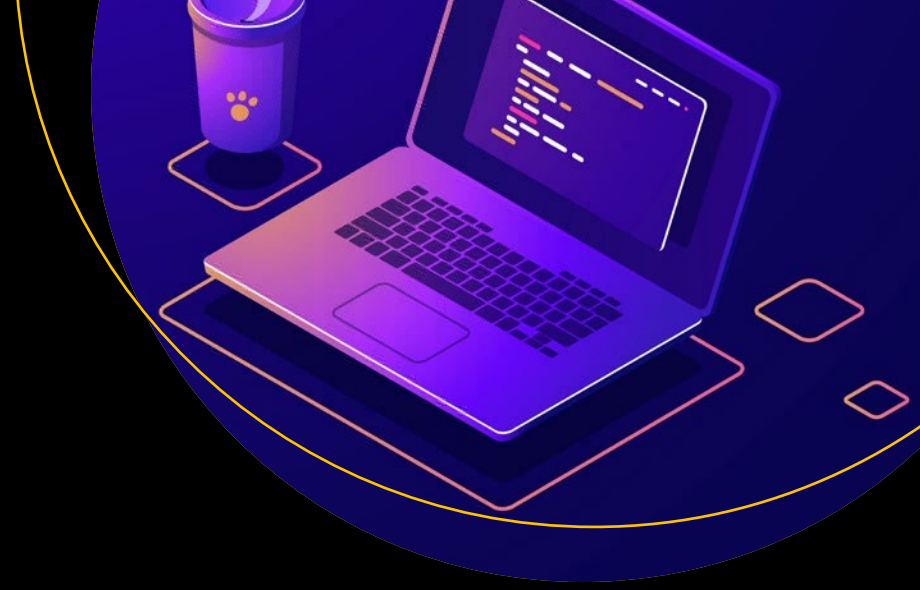

Programming for Absolute Beginners

Using the JavaScript Programming Language

Jonathan Bartlett

Apress®

Programming for Absolute Beginners

Using the JavaScript Programming Language

Jonathan Bartlett

Apress®

Programming for Absolute Beginners: Using the JavaScript Programming Language

Jonathan Bartlett
Tulsa, OK, USA

ISBN-13 (pbk): 978-1-4842-8750-7 ISBN-13 (electronic): 978-1-4842-8751-4
https://doi.org/10.1007/978-1-4842-8751-4

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Steve Anglin
Development Editor: James Markham
Coordinating Editor: Mark Powers

Cover designed by eStudioCalamar

Cover image by Fullvector on Freepik (www.freepik.com)

Distributed to the book trade worldwide by Apress Media, LLC, 1 New York Plaza, New York, NY 10004, U.S.A. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub (https://github.com/Apress). For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Most good programmers do programming not because they expect to get paid or get adulation by the public, but because it is fun to program.

—Linus Torvalds

Table of Contents

About the Author

Jonathan Bartlett is a software developer, researcher, and writer. His first book, *Programming from the Ground Up*, has been required reading in computer science programs from DeVry to Princeton. He has been the sole or lead author for eight books on topics ranging from computer programming to calculus. He is a senior software developer for McElroy Manufacturing, spearheading projects in web, mobile, and embedded software. He is now the author of several Apress books including *Electronics for Beginners* and more.

About the Technical Reviewer

Germán González-Morris is a polyglot software architect/engineer with more than 20 years in the field, with knowledge in Java(EE), Spring, Haskell, C, Python, and JavaScript, among others. He works with web-distributed applications. Germán loves math puzzles (including reading Knuth) and swimming. He has tech-reviewed several books, including an application container book (Weblogic), as well as titles covering various programming languages (Haskell, Typescript, WebAssembly, Math for coders, and regexp). You can find more details at his blog site (`https://devwebcl.blogspot.com/`) or Twitter account (`@devwebcl`).

Acknowledgments

I want to take a moment and thank everyone who helped me write this book. First, I want to thank those who read and appreciated my first programming book, *Programming from the Ground Up*. The encouragement I received from that book has given me the encouragement to continue writing and educating throughout the years.

Next, I want to thank my homeschool summer co-op class for being guinea pigs for this material. Your questions, your successes, and your difficulties all informed the writing of this book. You were both my motivation to write in the first place and the first proving ground for the material.

I would also like to thank my family, my friends, and my church, all of whom are essential parts of my life. Thanks especially to my wife who puts up with me when I am too focused on my writing to notice what the kids have been up to or to put a stop to whatever trouble they have found themselves in!

CHAPTER 1

Introduction

The modern world is filled with computers. Computers run our phones, our cars, and even our refrigerators. Computers manage our businesses, our calendars, and our social lives. With the world relying on computers for so many functions, it is important to know how these devices work. Even if you never need to program a computer yourself, chances are that, at some point in your life, you will be involved with software development. You may be an accountant who needs to tell a computer programmer how you want your purchasing system set up. You may be an engineer who needs to describe your engineering process so that a programmer can automate it. In all such tasks as these, it is important to know something *about* how computers are programmed, even if you are not personally writing the software.

1.1 What You Will Learn

When programming computers, a programmer uses a **programming language** to tell the computer how to do something. Because computers are not intelligent beings, they can't understand ordinary human languages. Computers understand a type of language called **machine language**, which will be discussed further in Chapter 5. Machine languages are very different from the kind of languages ordinary people use. Therefore, programming languages were developed to meet programmers halfway—they are more humanlike than machine language and more machinelike than human language.

Numerous programming languages have been developed over the years. Some that you may have heard of include Java, JavaScript, Ruby, Python, C#, Go, Rust, and Swift. Although each language looks different, they are all trying to do the same task of helping you to interface with the machine in a way that is friendlier and easier to manage than machine language. In fact, most programming languages are geared around very similar concepts, and some of them even look similar. Therefore, learning any programming language will help you more easily learn any other programming language. I have rarely

J. Bartlett, *Programming for Absolute Beginners*, https://doi.org/10.1007/978-1-4842-8751-4_1

hired people for my development team who already knew the programming language that my team uses. If someone learns one programming language and practices until they are good at it, then the effort to learn a new language is fairly minimal.

You may wonder why, if the languages are so similar, there are so many programming languages to choose from. The fact is, when engineering anything, trade-offs have to be made. Sometimes in order to make one type of task easier, another type of task has to be made harder. In my kitchen I have both a mixer and a blender. Both of them operate on the same basic principles—you put food into the main container area, an electric motor turns, and some attachment combines the food together. While these tasks are very similar and operate on the same principles, there are many types of food in the world and many ways that they need to be mixed, such that the mixer works better for some tasks and the blender for others. Similarly, with programming languages, some of them are better suited to different tasks. Also, the choice of programming language is dependent on the programmer. Just as different types of cars suit the preferences and tendencies of different types of drivers, so do different programming languages suit the preferences and tendencies of different types of programmers. Because of these reasons, there are numerous programming languages available for nearly any task you might want to perform.

The programming language covered in this book is called JavaScript. I like to teach JavaScript as a first language for several reasons. First of all, JavaScript was developed to be a first language. One of the goals of the language was to make it easy for new programmers to get started quickly. Even though JavaScript was designed to make programming easier for new programmers, it is not any less powerful as a language. Second, JavaScript has become the *de facto* programming language for website interfaces. If you use a website that does anything besides link to other web pages, JavaScript is probably involved. Therefore, learning JavaScript will have immediate practical benefits in learning how the Web operates. Third, the tools for programming JavaScript are available on every computer. You don't need to download any special tools to program JavaScript. If you have a computer with a web browser, you can program JavaScript! Finally, JavaScript is very similar to other popular programming languages such as C#, Java, and Swift. Therefore, knowing JavaScript will not only be immediately beneficial for programming websites, it is also a language that makes it easy to transition to other popular systems.

This book is for the first-time programmer. No prior programming experience is assumed. This book does assume that you have a basic understanding of how to use your computer and browse the Internet. That is all that you need!

You will learn not only the basics of computer programming but also a more general knowledge of how computers and data work. You will learn where computers came from, how they work, how computers work with data, how data is transmitted, and how web pages work. This book will not go in-depth in all of these subjects, but it will give you a basic working framework that will help you better understand ideas that you may encounter elsewhere.

1.2 How to Use This Book

This book follows several conventions to help you along your programming journey. First, this book will introduce you to new terminology. In order to highlight the important words, terms will be printed in **bold print** the first time that they are used. You can find a complete list of terms in Appendix A. These terms are important, and you should memorize their meanings.

When this book lists out computer programs, parts of computer programs, or anything that should be typed in directly (and precisely), it will be offset from the text and written in a special font to help you see that it is a computer program. Computer programs will look like this:

```
window.alert("This is an example of a computer program.");
```

When discussing smaller pieces of code within a paragraph, code that is under discussion will look like this.

Now, there are many different types of computers, each with different operating systems and software loaded on them, with each of those having different versions. There are also numerous different web browsers, each with different features available and slightly different ways of working. This book attempts to walk you through setting everything up on each operating system. If there is anything in this book that depends on the specific operating system or browser that you are using, Appendix B has the steps for several different systems, including Windows and Mac operating systems. This book will refer you to the appropriate section of the Appendix when needed. Though this book works with any modern web browser (basically anything released after 2008), I recommend that you use Google Chrome. As of the time of this writing, Google Chrome is the easiest browser to work with as a programmer. That being said, you should be just fine with any web browser, including Brave, Firefox, Safari, Chrome, Opera, or Edge.

This book contains several practice questions and practice activities. The goal of these questions and activities is to provide you with a hands-on way of understanding the material. By doing the questions and activities, the text will become much more meaningful and understandable. More importantly, they might show you the places where you did not fully understand the text. Many people have a tendency to skip over things if they don't understand them well. Practice questions and activities give you a chance to slow down and make sure you know which parts you understood and which parts you need to read again and spend time thinking about. Practice questions build on each other, so by doing them all in the order given, you can see exactly where you are having problems.

At the end of every chapter is a review section which covers the most important concepts of each chapter. After that is a section to help you practice applying your knowledge to problems. These questions require you to further engage your brain and really think about what you learned in that chapter and what it means.

Appendix A contains an extended glossary of terms used in this book, plus others you are likely to encounter when reading about programming. This chapter will help you find your bearings as you read and talk with other people about programming. I would suggest that, concurrent with your readings, you also take the time to look through the glossary for words that you may have heard but did not understand at the time.

Also, if you run into problems when writing code, Section B.6 has several suggestions for getting you back on the right track.

1.3 For Younger Programmers

This book is primarily geared for people who are coming to computer programming as a new career, college students, or even high school students. However, it can also be used for middle school students with some modification. Middle school students, generally, are not cognitively ready for all of the material after Part 3. This doesn't mean it can't be covered or read, but it might be good to pick and choose material that is appropriate to student interests and abilities. If parts are difficult to understand, they can be returned to at a later time.

All right, are you ready? Let's get started!

PART I

Computers, Data, and Communication

The purpose of this part is to help you understand how computers work on the inside. While it is possible to learn programming without learning how your computer works, in the long term, knowing what is going on inside the computer will help you write better programs. This part covers the basics of how computers communicate, how they store data, and how computer programming works at the lowest level.

CHAPTER 2

A Short History of Computers

The history of computers is weird and wonderful. What started as an abstract philosophical quest ended up setting the course for society for over a century and continues to be one of the most profound parts of modern life. The goal of this chapter is to trace an outline of where computing started, where it has been, and where it is now.

2.1 The Prehistory of Computers

Humans have always had tools. Humans have built fires, made spears, and built houses from the beginning. At first, however, technology was limited to standing structures or tools that were extensions of yourself—like knives or bows and arrows. Very little early technology was powered and free-functioning. It was manually powered by human effort. Therefore, since the power of a machine was limited to what humans could drive, only small machines could be devised.

The ability to power a machine led to huge advances in technology. The earliest power source was probably water, where water could turn a wheel to grind wheat or operate a sawmill. These water-based power sources, however, were fairly limited in the types of devices they could drive. Such technology was mostly limited to standing wheel-based inventions.

This was essentially the state of technology from about 300 BC to the early 1700s AD. At this point in history, technology had two main limiting factors. The first was limitations of power availability, and the second was the need for customized parts. The industrial revolution solved both of these problems. The steam engine allowed the creation of powered machines anywhere. Powered machines were no longer tied

J. Bartlett, *Programming for Absolute Beginners*, https://doi.org/10.1007/978-1-4842-8751-4_2

to being near streams but could now go anywhere, since the power could be generated from fire and stored water. Eventually this even allowed the creation of trains, since the power could move with the vehicle.

The other invention of the industrial revolution was interchangeable parts. This allowed a standardization and maintenance of equipment that was previously unattainable. Instead of having each part be a unique piece, the parts became standardized which allowed for the machines to become more specialized. It is one of the more curious paradoxes of technology that as the *pieces* of technology become less unique, the more advanced and unique the systems created from those parts can become. Standardization allows for users of technology to stop having to think about all of the low-level decisions and focus on the larger, more meaningful decisions. This also allows for better communication *about* systems, because the parts can be more readily described. If I can give you a schematic that lists premade parts, it is much easier to design and communicate that design than if I also had to describe how each individual part was supposed to be made.

So the introduction of available powered machinery and standardized parts in the industrial revolution led to an explosion of specialized machines. We then had machines to perform any number of tasks that a person could want to do. The next step was the introduction of machines which were directed not by people directly controlling the machine but by coded instructions. The earliest of these machines was the Jacquard Loom, which used punched cards to signify a pattern woven into a fabric. The cards had punched holes to signify to the machine the raising or lowering of the particular thread causing it to be visible or hidden in the pattern. Thus, the loom could be programmed to make a pattern by specifying at each point whether each thread should be raised or lowered.

Later inventions applied this concept to mathematics. Calculating machines had been around for a long time, with Blaise Pascal's mechanical calculator having been invented in the mid-1600s. However, this required the power of physical manipulation to actually accomplish the addition. Most mathematical tasks are not single-step like addition but require a process of several steps, sometimes repeating steps, before finding an answer. Charles Babbage invented a more advanced machine to perform navigational calculations. In this machine, the user entered the input, and then the machine used that input to run a series of steps which eventually yielded results. Babbage eventually designed a machine that could take a list of arbitrary instructions much like a modern computer, but he was never able to build that design.

Once humans had the ability to power a machine, create a machine that operated on external instructions, and use those instructions to perform mathematical functions, they had all of the pieces in place to create a computer. However, the revolution that brought about computing took place not from an invention, but from a problem in philosophy.

2.2 The Idea of a Computer

What separates modern computers from the calculating machines of the past is that modern computers are **general-purpose computers**. That is, they are not limited to a specific set of predesigned features. I can load new features onto a computer by inputting the right program. How did we get the idea of creating such a general-purpose machine?

It turns out that a question in philosophy led to the creation of general-purpose machines. The question was this—was there a way to create an unambiguous procedure for checking mathematical proofs? This seems like an odd question, but it was a big question in the nineteenth century. There had been many "proofs" where it was unclear if the proof actually proved its subject. Thus, philosophers of mathematics tried to find out if there was a way to devise what was then called an "effective procedure" for checking the validity of a mathematical proof. But that leads to another question—what counts as an "effective procedure" anyway? If I list out the steps of a procedure, how do I know that I've given you enough details that you can accomplish this procedure exactly as I have described it? How can I tell that my instructions are clear enough to know that the procedure that I have listed can be unambiguously accomplished?

Alan Turing and Alonzo Church both tackled this problem in the 1930s. The results showed that one could define unambiguous procedures with the help of machines. By describing a machine that could perform the operation, one can be certain that the operation of the procedure would be unambiguous. In addition, Turing described a set of operations which could be used to mimic any other set of operations given the right input. That is, Turing defined the minimum set of features needed for a computing system to become truly programmable—where the programmer had an open-ended ability to write whatever software he or she wanted. Machines and programming languages that are at least as powerful as Turing's set of features are known as **Turing-complete** or **Universal** programming languages. Nearly every modern programming language in common usage is Turing-complete.

It is interesting to note that the creation of computing came from a question in philosophy. Many are eager to dismiss the role of philosophy in academics as being impractical or unimportant. But, as we see here, like all truths, philosophical truths have a way of leading to things of deep practical importance.

And what happened to the original question—can you develop an effective procedure for checking proofs? The answer is, strangely, no. It turns out that there are true facts that cannot be proven via mechanical means. But to learn that answer, we had to develop computers first. Of course, that leads to another interesting intersection between computers and philosophy. If there are true facts that cannot be mechanically proven, how could we know that? The only way must be because our minds cannot be represented mechanically. This puts a limit on the potential capabilities of artificial intelligence and shows that even though computer programmers have developed some very clever means of pretending to be human, the human mind is simply outside the realm of mechanism or mechanistic simulations.[1]

2.3 The Age of the Computer

Shortly after Turing described the necessary feature set for computers, people began to build them. Probably the first Turing-complete machine was Konrad Zuse's Z3 computer, built in 1941. Although the Z3's operating principles were somewhat similar to modern computers, the Z3 was still largely a mechanical device. The first general-purpose, digital electronic computer was the ENIAC in 1946 and was about a thousand times faster than its mechanical predecessors. It should be noted that the ENIAC was the size of a very large room, but it had roughly the same processing power as a scientific calculator. Its main jobs included performing calculations for the production of the hydrogen bomb and calculating tables for firing artillery.

The next generation of computers introduced what is normally termed the **von Neumann architecture**, which means that the computer had a single memory area which held both programs and data. This is based on the fact that both a program and the values that the program generates can both be represented by numbers. Therefore, the same memory can be used both for the program that tells the computer what to do

[1] If you want to dive deeper into this subject, you can see my article in MindMatters, "Why I Doubt That AI Can Match the Human Mind," available at https://bit.ly/3FOtOsS. You may also want to check out my YouTube video, "How to Build an Artificial Intelligence Using the Doctrine of Man," available at https://youtu.be/FzXW7p3AG1Y.

and for the data that the program generates and operates on. This makes the computers much easier to program and use, which led to the ability to sell computers commercially. The first commercially available computer to implement this idea was the Manchester Mark 1. The first mass-produced computer was the UNIVAC I, followed shortly after by IBM's 650. These computers were still massive in size but contained less memory storage space than a single graphic on a modern computer. The UNIVAC I was the first computer to have an external tape storage, and external disk storage (similar to modern hard drives) followed soon after.

The next move for computer hardware was toward miniaturization. The original computers used large devices called vacuum tubes to perform data processing (see Figure 2-1, left column). These vacuum tubes would allow or not allow current to flow based on whether other wires had current flowing through them or not. Combinations of many of these tubes could allow for data to be stored as current flow, for mathematical operations to be performed on such data, and for the data to be moved around.

After the vacuum tube came the invention of the transistor (see Figure 2-1, middle column). Transistors generally have three wires, where the middle wire controls whether the electricity can flow between the other two wires. As with vacuum tubes, transistors can be wired together to create digital computer memory, digital computer logic operations, and digital information pathways. Transistors, while they performed the same basic functions as the vacuum tube, were able to do so in a much smaller package and operating on a lot less power. Transistors allowed much smaller devices to be built which also required almost 1,000 times less power. The Metrovick 950, released in 1956, was the first commercial computer that operated on this principle.

Miniaturization continued with the advent of **integrated circuits** or what are often called **microchips** or just **chips** (see Figure 2-1, right column). An integrated circuit basically allows for miniaturized transistors to be stored on a small, single plate of silicon. When integrated circuits were first introduced, they only had a few transistors. Today, integrated circuits come in a variety of sizes, and the ones used for desktop computing can hold billions of transistor equivalents on a 2-inch square chip. Integrated circuits basically brought computers as we know them into the world. However, when they were first introduced, they were primarily used by very large businesses.

In the 1960s, Douglas Engelbart led a research team to look at the future of computing. In 1968, Engelbart presented what has been termed "the mother of all demos," which predicted and demonstrated nearly all aspects of modern personal computing, including graphical interfaces, networking, email, video conferencing,

collaborative document editing, and the Web. This served as an inspiration for a number of companies to start pushing to make this vision of computing a reality. Engelbart had accomplished it in a lab, but others were needed to make it a commercial reality.

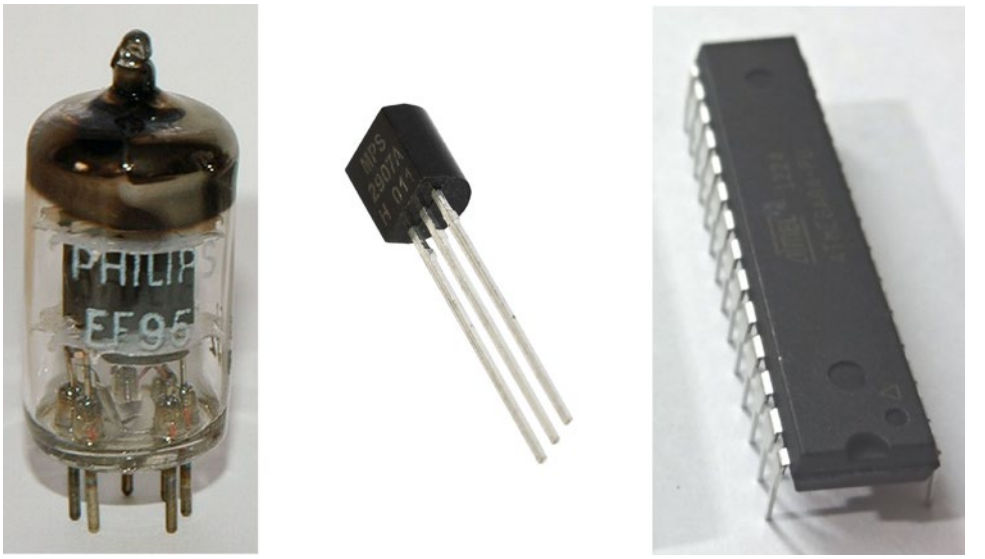

Figure 2-1. *Advancements in Computer Hardware Miniaturization*

The picture on the left is of a vacuum tube (photo courtesy of Tvezymer on Wikimedia). Vacuum tubes are still around today, primarily for audio applications. The picture in the middle is of a transistor. Transistors were much smaller, required fewer materials to produce, and used much less power but still did largely the same job as vacuum tubes. The picture on the right is a modern microchip used in appliances (photo courtesy of Vahid alpha on Wiki-media). Such a microchip contains the equivalent of a few hundred thousand transistors.

The first recreational personal computer was the Altair, and the first commercial personal computer was the Apple I which came out in 1976, after which a flood of personal computers entered the market. IBM eventually entered the market, with Microsoft providing the software for the computer. The interfaces for these computers were usually text-only. However, eventually Apple released the Macintosh, which inaugurated the age of graphical user interfaces. Shortly after, Microsoft released Windows, which brought the graphical interface to the IBM side of the personal computer world.

2.4 Computers in the Age of Networks

Thus far, computers had been largely isolated machines. You could share files through disks, but, by and large, computers operated alone. When you link together two or more computers, it is called a **network**. Though networking technology had been around for quite a while, it had not been cheap enough or popular enough to make an impact for most personal computer users.

For most users, networking started with office file sharing systems, usually using a type of local networking called **Ethernet** which runs networking services over specialized networking cables. People would use applications that were installed on their own computers but store the files on a **server** so that the other members of the office could access it. A server is a computer on the network that provides one or more services to other computers and users on a network. A software program that accesses a server is often called a **client**. Many office networks eventually added **groupware** services to their networks—local email and calendar sharing systems that allowed the office to work together more efficiently. While smaller organizations were focused on local services such as file sharing and groupware, larger institutions were also at work linking networks together. This allowed organizations to share information and data between each other more easily.

At the same time, a few home computer users started reaching out to each other through the phone system. A device called a **modem** allowed a computer to access another computer over standard telephone lines. Services called **bulletin-board systems** (known as a BBS) started popping up which allowed people to use their computers to access a remote computer and leave messages and files for other users.

These developments laid the groundwork for the idea of the **Internet**. At the time it was developed, there were many different, incompatible networking technologies. Organizations wanted to connect their networks to other organizations' networks but were finding it problematic since everyone used different types of networks. In the 1970s and 1980s, DARPA, the Defense Advanced Research Projects Agency, developed a way to unify different types of networks from different organizations under a single system so that they could all communicate. This network, known as ARPANET, became very popular. Other large, multi-organizational groups started using the design of ARPANET to create their own network. Since these networks all used the same basic design, they were eventually joined together to become the Internet in the late 1980s.

The 1990s witnessed the rise of Internet Service Providers, or ISPs, which provided a way for computer users to use the modems that they used to use for bulletin-board systems to connect their computers to the Internet. Instead of using a modem to connect to a single computer, like they did with bulletin-board systems, the ISP allowed a user to use their modem to connect to a whole network. This began the mass public adoption of the Internet by both individuals and organizations of all stripes.

In the early days of the Internet, the speed of the network was very slow, and only text could be transmitted quickly. Eventually, modems were replaced with more advanced (and faster) ways of connecting to the Internet, such as DSL, cable, and fiber. This allowed more and more complex content to be transmitted over the Internet. Also, because these technologies do not tie up a phone line, they can be used continuously, rather than intermittently. In addition, wireless technologies, such as **WiFi** and cellular-based networking, allowed users to connect to the Internet without being tied down by cables. These developments together led to the near-ubiquitous availability of the Internet that we have today.

So, today, nearly all computer software is built with the network in mind. In fact, much of the software that people use on a daily basis operates not on an individual computer, but over a network. This allows for users to access software programs no matter where they are or what computer they are using. It has also changed software development so that the focus of computer software is no longer on individuals and individual tasks but on organizing groups of people.

2.4.1 Review

In this chapter, we covered the basic history of computers. We have learned the following:

- Humans have used tools to accomplish tasks from the beginning.
- Early tools were limited by available power options.
- Advances in power technology allowed for the improvements and industrialization of tools.
- Standardization of parts allows for more complex machines to be built and serviced.
- Electricity allowed for the movement of power to any needed location.

- The ability to control a machine via instructions, such as the Jacquard Loom, allowed for the creation of more general-purpose tools which could be specialized by providing the right sets of instructions.
- Alan Turing and Alonzo Church identified the logical requirements for making general-purpose computations.
- Several early computers were built around the idea of a general-purpose calculating machine.
- Advances in electronics allowed for storage of millions of transistors onto a single microchip.
- The availability of microchips led to the era of personal computing.
- The increased usage of computers in organizations eventually led to the need to have better means of communication between computers.
- Networks were invented to allow computers to be hooked together to share file and messages.
- The isolated networks around the world were eventually unified into a single Internet-work, known as the Internet.
- The growth of the Internet combined with the ability to access the Internet wirelessly has made the Internet a primary factor in computer usage.
- The ubiquity of the Internet has led programmers to start designing applications with the network in mind first, rather than as an afterthought.

2.4.2 Apply What You Have Learned

1. Take some time to think about the history of technology and the Internet. What do you think is next on the horizon for technology?
2. The pace of technology appears to have been accelerating over the past century. What do you think has caused this acceleration?

3. Pick your favorite piece of technology mentioned in this short history and research it. What inspired the person who developed it? What other inventions came after it? Was it successful? Write a few paragraphs describing the technology you have chosen, how it functioned, and how it impacted the future of technology.

CHAPTER 3

How Computers Communicate

Before we start our study of computer programming, we are going to begin by studying the way that computers communicate. The Internet is basically a giant communication system. Communication systems operate using **protocols**. A protocol is a predefined sequence of steps used to ensure proper communication.

We actually use protocols every day. Think about what happens when you answer the phone. What do you say and why do you say it? Think about what happens when you answer the phone (see Figure 3-1). This signals to the person calling us that we have picked up the phone and we are ready to start talking. If we didn't say "hello," the person might think that we accidentally accepted the call without knowing it or that we are not quite ready to talk yet. Then, at the end of the call, we usually say something like, "Thanks for calling! Goodbye!" This signals to the other person that we are done with the conversation. If we didn't tell them goodbye, they might think that we are still on the line and continue talking. If they heard silence, they may presume that either we were not speaking because we were upset or that there was a technical problem. Therefore, we end our conversations with a "goodbye" to let the person we are talking to know that the conversation is over.

This is the essence of a protocol. A communication protocol is a sequence of steps or possible steps that enable two parties to communicate or interact and know the status of the communication or interaction. Because computers cannot think or feel, computers rely on very rigid and exact protocols to allow them to communicate with each other. In fact, computers use hundreds of different protocols to communicate different types of data in different ways. Most of the time, there are actually multiple protocols happening at once.

J. Bartlett, *Programming for Absolute Beginners*, https://doi.org/10.1007/978-1-4842-8751-4_3

Figure 3-1. *Even Answering the Phone Has a Protocol*

Think about writing a letter. When you write a letter, there is a basic protocol that governs the form of a letter—at minimum it should have a date, a greeting, and a closing. However, if you decide to mail the letter, you have to send it through the mail service, which has its own protocol. To send the letter through the mail, you need to take the letter, fold it up, and put it in an envelope. What you write on the envelope is governed by another protocol designed by the US Postal Service. Their protocol requires a return address on the top left corner of the envelope, a destination address in the middle of the envelope, and a stamp in the top right corner. Now you have two protocols happening simultaneously—the letter-writing protocol and the envelope-addressing protocol. These protocols are *layered*, which means that one of the protocols runs fully inside of the other protocol. In computer jargon, we would say that the envelope protocol *encapsulates* the letter protocol. The envelope protocol takes the results of the letter protocol, packages it up, and puts its own protocol on top.