Programming Basics

Getting Started with Java, C#, and Python

Robert Ciesla

Apress®

PROGRAMMING BASICS

GETTING STARTED WITH JAVA, C#, AND PYTHON

Robert Ciesla

Apress®

Programming Basics: Getting Started with Java, C#, and Python

Robert Ciesla
HELSINKI, Finland

ISBN-13 (pbk): 978-1-4842-7285-5 ISBN-13 (electronic): 978-1-4842-7286-2
https://doi.org/10.1007/978-1-4842-7286-2

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Shiva Ramachandran
Development Editor: James Markham
Coordinating Editor: Jessica Vakili

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, New York, NY 100043. 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 via the book's product page, located at www.apress.com/978-1-4842-7285-5. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Dedication

Thank you to the Association of Finnish Non-fiction Writers for their support in the production of this book.

Contents

About the Author

Robert Ciesla is an author and filmmaker from Helsinki, Finland. He is also a freelance-programmer working mostly in the indie game scene. Robert is the author of *Encryption for Organizations and Individuals* (2020), *Game Development with Ren'Py* (2019), and *Mostly Codeless Game Development* (2017).

Visit www.robertciesla.com for more information. (image © by A.C. in 2021)

About the Technical Reviewer

Apoorv Gupta is a Software Engineer in New York. He has worked on several subscription products at Youtube and Google Workspace. He enjoys hiking, snowboarding and advising startups.

CHAPTER

1

Wet Toes: The Very Basics of Programming

What do video games, social networks, and your activity bracelet have in common? They run on software a group of (more or less) programmers wrote somewhere far, far away. Gadgets and hardware are only one, more visible side of the coin of our technology-driven societies. In this chapter, we'll discuss the very basics of programming. We'll also take a gander at the visible parts of digital systems: the hardware.

What Is Programming Anyway?

Basically, programming is the act of telling digital devices, such as your personal computer, what to do. We type in listings of commands, as defined by a programming language, in order to have useful or entertaining events occur. Properly programmed computers run much of the communications and online services in the world. You can mention things like ATMs, ticket readers, and smart phones as gadgets that run on software that somebody created in some programming language.

R. Ciesla, *Programming Basics*, https://doi.org/10.1007/978-1-4842-7286-2_1

Basic Hardware Rundown

As a budding programmer, you'll benefit from understanding the kind of universally found electronics you're working with. It's a good idea to have at least a basic understanding of the most commonly found components inside a computer.

These hardware components in a computer represent your work force. As a programmer, you run this show. Think of the act of programming as telling the factory workers what to build. You manufacture applications, whether they be big, complicated software projects or tutorials from some awesome book on coding.

For the purposes of this book, any relatively modern desktop or laptop computer works fine. We won't be needing any expensive hardware while getting our feet wet in programming.

---

1. Central Processing Unit (CPU)

Naturally, a digital device can't run on software alone; a **central processing unit (CPU)** is the hardware "brain" which executes code and makes things actually happen (see Figure 1-1). Even in a less complicated piece of electronics, all instructions flow toward and through a CPU (or a bunch of them). Being very small in size, these microchips have increasingly been a part of our lives since the 1970s. Every digital device has a CPU in it, probably even your stationary bicycle/clothes rack.

Figure 1-1. A top-down view of an older Intel "Pentium 4" CPU used in millions of PCs back in 2005. Image by Eric Gaba. CC BY-SA 3.0

2. Hard Drives (a.k.a. Hard Disks)

This component is there to store data just about permanently. Within a hard drive, you'll find tens of thousands of files, whether they be pictures, text files, or databases. Your operating system (e.g., Windows or macOS), too, rests within the confines of a hard drive. These devices come in two varieties: *mechanical hard drives* (see Figure 1-2) and *solid state disks (SSDs)*.

Figure 1-2. A top-down view of a Western Digital mechanical hard drive. Image by "Darkone." Licensed under CC BY-SA 2.5 (creativecommons.org/licenses/by-sa/2.5/deed.en)

Mechanical drives are more affordable, but since they have moving parts inside, they are somewhat more easily damaged than SSDs by excessive vibration and extreme weather. In addition, solid state disks usually operate much faster.

3. Video Card

Video cards are responsible for displaying a system's visuals, whether they be plain text or dazzling 3D graphics in a modern video game. These devices come in a variety of configurations and prices, ranging from $30 word processor fiends to $1000 gaming monsters (see Figure 1-3). Computer monitors are typically connected directly to a video card.

Figure 1-3. An Nvidia 7900GS video card from 2006

The video card business has basically been a duopoly between *Nvidia* and *AMD,* two multibillion tech giants, ever since the early 2000s. However, Intel is making gains in this sector as well.

---

4. Random Access Memory (RAM)

Random access memory, commonly called RAM, is used as a computer's temporary storage. Physically it most often comes in the form of stick-like add-ons (see Figure 1-4). When running any type of software, your computer uses RAM to execute it from. Switching off your device will clear out your RAM. By contrast, data written on hard drives isn't erased when powering off a computer. Save your documents on a regular basis.

Figure 1-4. A typical stick of RAM. Image by Henry Kellner. CC BY-SA 4.0. Source: upload.wikimedia.org/wikipedia/commons/3/32/DDR3_RAM_53051.jpg

---

As of 2021, 4 GB (i.e., *four gigabytes*) is an adequate amount of RAM to have for most uses. Power users, such as video editors, will benefit from having 16 GB of RAM or more.

---

5. Motherboard

All of the aforementioned four hardware components (i.e., the CPU, the video card, the hard disks, and RAM) come together at the motherboard to create a working computer unit. The motherboard also has connectors for keyboard, mice, and other control devices (see Figure 1-5).

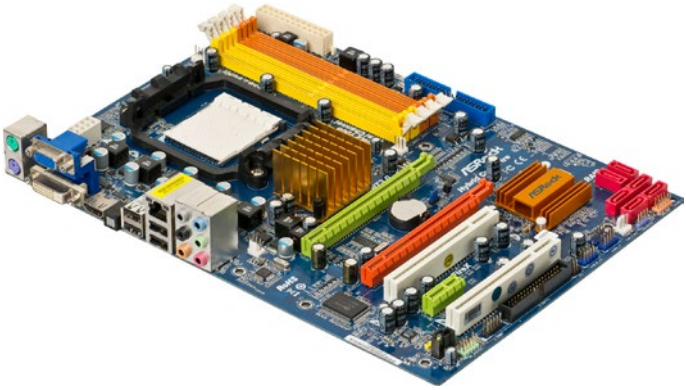

Figure 1-5. A modern PC motherboard. Image by Evan-Amos. CC BY-SA 3.0. Source: upload.wikimedia.org/wikipedia/commons/0/0c/A790GXH-128M-Motherboard.jpg

The Three Requirements of Becoming a Decent Programmer

Let's next discuss some personal priorities all programmers should have in order to advance in their craft, whatever their starting level might be:

1. **Self-confidence**: Ask yourself this, why do you want to learn to code? Some perfectly valid answers include "For professional development," "To maintain my faculties," and "I want to be a part of something great." Now, programming is sometimes considered a frightening activity by laypeople. It does take some guts to sit down, tune out, and enter the world of bit manipulation. Just remember that you, too, can achieve competence in this field, even if you're a complete beginner. Confidence comes from experience. Line by line you will obtain more good vibes and gain independence from programming books and online tutorials.

2. **The right language**: Not all of us benefit from becoming fluent in Esperanto or Classical Latin. When learning a new language, we tend to go for something useful, such as Spanish or French. Similarly, choosing a programming language which best suits your intentions is of paramount importance. If you want to eventually code recipe apps for mobile users, becoming proficient in, say, FORTRAN from 1957 only gets you so far. For this reason, this book introduces three of the most popular programming languages of our times: Java, C#, and Python.
3. **Patience**: After choosing which programming language you want to specialize in, you quite simply just have to stick to it. It takes anything between six months and a year of hands-on experience to become proficient in a new language. This is actually good news. Coding is great for insomnia and boredom. It may also ward off dementia, as it does fire those brain synapses to quite an extent.

A Novice Programmer's Glossary

We'll now delve into some essential terminology related to the hallowed hobby of coding. There are hundreds of terms and concepts related to the various programming techniques and languages available. However, we'll only be focusing on the most relevant associated keywords, and in no particular order.

Input/Output

Input in the context of programming refers to us entering data for a piece of software running on a computer to process. This comes in the form of typed text, mouse commands, or various types of files. For example, a word processing program (e.g., Microsoft Office) most often takes its input mostly as alphanumerical data provided by keystrokes. *Output* refers to data that has been processed by software. In a word processor this usually refers to a file document saved with the program. Such output can also be directed at printers or other devices. The output from programmers (carbon dioxide and other things notwithstanding) is typically a working application, whether it's a completed tutorial file or a bigger project.

Algorithm

A working program listing basically constitutes an *algorithm,* which refers to a set of steps created to solve problems. Most software consists of numerous sub-algorithms. In the case of, say, a video game, there are algorithms for displaying graphics, saving and loading the game state, and playing audio files to name just a few.

Flowchart

Programming projects and their algorithms are often visualized using *flowcharts*, especially in a team environment. These are a great way to demonstrate basic program flow in most instances.

Flowcharts consist of only a few universal elements (see Figure 1-6). In their most fundamental form, they use four symbols. These are the *terminal* (rounded rectangle), the *process* (rectangle), the *decision* (diamond/rhombus), and the *flowline* (arrowhead). The terminal symbol is used to denote the beginning and the end of a program flow. Any operations and general data manipulation are represented by process rectangles.

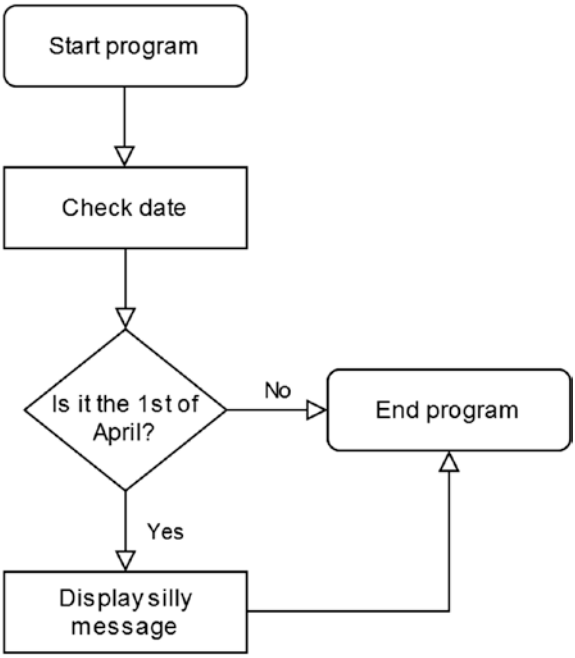

Figure 1-6. A very simple flowchart describing a program for April fools

Flowcharts are interpreted from top to bottom and left to right in most cases. *The American National Standards Institute (ANSI)* created the standards for flowcharts and their symbols back in the 1960s. This set of symbols has been expanded on during the 1970s and 1980s by the *International Organization for Standardization (ISO)*. For the purposes of this book, we'll stick to the originals.

Source Code

This term refers to the collection of the more or less typed-in programming listings each software project is made of. As a programmer, you are a creator of *source code*. Simple programs come in the form of a single piece of source code, whereas complicated software, such as operating systems (e.g., Windows), potentially consists of tens of thousands of listings all constituting a single product.

Syntax

A *syntax* is a set of rules and principles that govern the structure of sentences in a given language, including in the context of programming. Different programming languages use different keywords for specific actions. Now, behold actual lines of programming which display a string of text in two programming languages:

Table 1-1. A demonstration of the syntactical differences between two programming languages

Java	FORTRAN
System.out.print("Hello! I like Cake!");	1 print *, "Hello! I like Cake!"

Java, like you may have gathered already, is one of the main languages featured in this book. The other programming language used in Table 1-1 is called *FORTRAN*. Devised mainly for scientific computation, this language was created way back in the 1950s by IBM. A lot of industrial hardware runs on FORTRAN. Even some geeks still use it for the tech chic (and to a small extent, so did we).

You may notice one of our examples in Table 1-1 started with a number (1). This is known as a *line number* in coding parlance, and the practice has been pretty much abandoned a while ago. As a rule, current-generation programming languages don't need line numbering.

Routine

A *routine* in the context of programming is a term for code which does a specific task and is intended to be summoned repeatedly at will by the coder. For example, a program may contain a simple routine for playing a sound effect. Instead of writing and rewriting the code each time said sound effect is needed, the programmer will trigger the same code (i.e., the routine) ad hoc.

Depending on the context and the programming language in use, a routine is sometimes also referred to as *a sub-routine, a function, a procedure,* or a *method.* We'll address the nomenclature in more detail later in this book.

File Formats

A *file format* is a method of encoding data. By 2021, you've encountered many a file format already in your daily life. Digital photographs, love letters typed in OpenOffice, and those sassy Excel spreadsheets all represent different file formats. An image file (e.g., *apress_is_great.jpg*) resting on one's hard drive can only be used with software that deciphers it the way it was intended to, as an image. Similarly, opening *love-letter.doc* in a photo-editing suite would not provide you with optimal results, displaying gibberish at best. Most operating systems associate different available file formats with the right software, so you can safely double-click files and expect them to load up just fine.

ASCII

American Standard Code for Information Interchange (ASCII) is a character-encoding standard that assigns letters, numbers, and other characters for use in computing and other digital equipment. In essence, what you are reading now is ASCII code. As a programmer, you'll come across the term rather frequently. An "ASCII file" is often used as a shorthand for "human-readable text file." The system dates back to 1963.

On the Internet of today, the most commonly used character-encoding standard is the UTF-8, which includes all of the ASCII alphanumericals as well as many other symbols.

Boilerplate Code

The term *boilerplate* refers to programming code which is more or less automatically inserted into a program, needing little to no editing. When you start a new project in, say, a C++ environment, current-generation development tools will usually set you up with the necessary boilerplate code needed to run the program. If not, you can relatively safely copy-paste boilerplate code from your older working projects right into your new one to get started.

Software Framework

A *software framework* **is a set** of generic functionalities which typically spare the coder a lot of time. There's no reason to reinvent the wheel, especially in software projects. A framework includes various software libraries of various focus, including file manipulation routines, audio playback, and 3D graphics routines (in the case of 3D video game development and other highly visual applications).

For the purposes of this book, we won't be delving deep in any complicated software frameworks, but it's important you understand the concept.

Full Stack

A *full stack* is the software that makes up a fully working web application, such as an online database. A web application is often divided into two areas: a *front end* and a *back end.* The front end is made for the user; it contains all the user interface elements needed to use the app. The back end consists of web servers, frameworks, and databases. *A full stack developer* is therefore someone who knows their way across both the front end and the back end of online application coding.

In Closing

Finishing this chapter you'll have hopefully gained some understanding of the following:

- The basic five hardware components in a computer
- The three main requirements for becoming a programmer
- Some essential programming concepts, including source code, syntax, and boilerplate code
- What flowcharts refer to and what their basic building blocks are