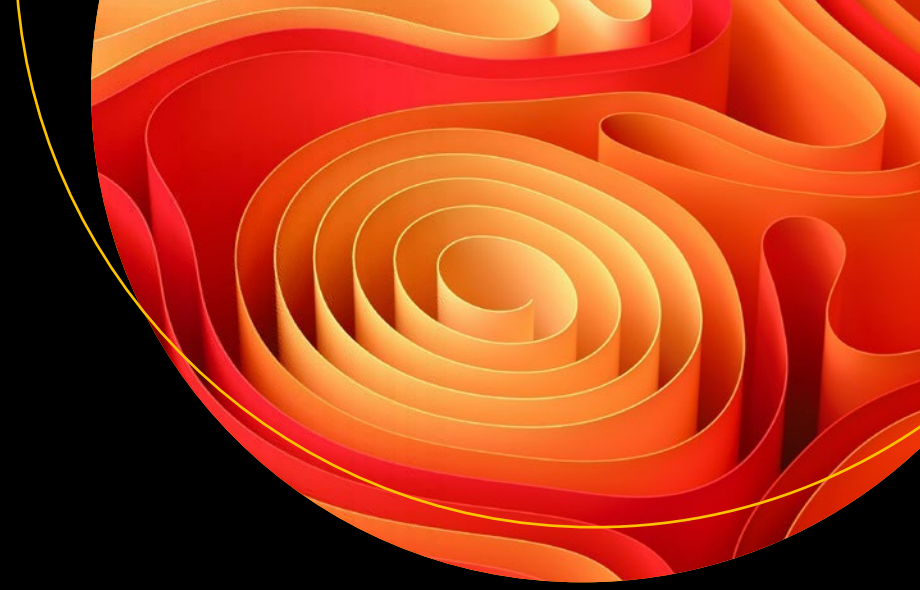

PHP 8 for Absolute Beginners

Basic Website and Web Application Development

—

Third Edition

—

Jason Lengstorf
Thomas Blom Hansen
Steve Prettyman

Apress®

PHP 8 for Absolute Beginners

Basic Website and Web Application Development

Third Edition

Jason Lengstorf
Thomas Blom Hansen
Steve Prettyman

Apress®

PHP 8 for Absolute Beginners: Basic Website and Web Application Development

Jason Lengstorf
No 392
Portland, OR, USA

Thomas Blom Hansen
Kirke Saaby, Denmark

Steve Prettyman
Palm Bay, FL, USA

ISBN-13 (pbk): 978-1-4842-8204-5
ISBN-13 (electronic): 978-1-4842-8205-2
https://doi.org/10.1007/978-1-4842-8205-2

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 Li Zhang on Unsplash (www.unsplash.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

This book is dedicated to every volunteer who provides open source code and training (videos and tutorials) for anyone who wants to improve their skills and learning. Without your dedication to help fellow programmers, we would not progress as an industry to providing the best, most reliable, and most secure programs possible.

Table of Contents

About the Authors xv

About the Technical Reviewer xvii

Introduction xix

Part I 1

Chapter 1: Getting Ready to Program 3

Objectives 3

Setting Up a Development Environment 12

What Is PHP? How Does PHP Work? 13

Apache and What It Does 15

Storing Info with MySQL/MariaDB 15

Installing PHP, Apache, and MySQL/MariaDB 16

Installing XAMPP 17

Open the XAMPP Control Panel 28

What If Apache Isn't Running? 30

Verify That Apache and PHP Are Running 31

Choosing a PHP Editor 32

Creating Your First PHP Program 34

Running Your First PHP Script 35

Summary 36

Projects 37

Chapter 2: Understanding PHP: Language Basics 39

Objectives 39

Embedding PHP Scripts 40

Program Design and Logic 42
Program Design and Logic 42
Using echo 43
What Is a Variable? 43
Program Design and Logic 45
Displaying PHP Errors 46
Secure Programming 46
Creating an HTML5 Page with PHP 49
HTML Review 49
Including a Simple Page Template 51
Including the Template 52
Secure Programming 53
Commenting Your Code 54
Avoiding Naming Conflicts 55
Page Views 59
HTML Review 60
Making a Dynamic Site Navigation 61
HTML Review 62
Passing Information with PHP 63
Accessing URL Variables 64
Using isset( ) to Test If a Variable Is Set 64
Secure Programming 65
$_GET, a Superglobal Array 66
Including Page Views Dynamically 67
Strict Naming Convention 68
Displaying a Default Page 69
Securing the Program 69
Validating Your HTML 70
Styling the Site with CSS 71
CSS Review 72
Declaring a Page_Data Class 74

Program Design and Logic 75
Classes Make Objects 76
Highlighting Current Navigation Item with a Dynamic Style Rule 77
Summary 77
Projects 77

Chapter 3: Form Management 79

Objectives 79
What Are Forms? 80
Setting Up a New PHP Project 82
Seeing for Yourself 82
Creating a Dynamic Navigation 83
Creating Page Views for the Form 84
Program Design and Logic 86
A Simple Search Form 87
The <input> Element and Some Common Types 88
Understanding the Method Attribute 88
Named PHP Functions 89
Program Design and Logic 89
The Basic Syntax for Named Functions 89
Program Design and Logic 90
Program Design and Logic 93
Using Function Arguments for Increased Flexibility 93
Creating a Form for the Quiz 95
HTML Review 96
Showing the Quiz Form 97
Secure Programming 98
Program Design and Logic 99
Secure Programming 104
Curly's Law: Do One Thing 104
Program Design and Logic 105

OOP: Using Constructors, Getters, and Setters 105
Secure Programming 109
Summary 112
Exercises 113

Chapter 4: Building a Dynamic Image Gallery 115

Objectives 115
Setting Up a Dynamic Site 116
Prerequisites: A Folder with Some Images 116
Copyright Laws 116
Creating a Navigation 117
Creating Two Dummy Page View Files 118
Creating the Index File 118
Time to Test 119
Preparing a Function for Displaying Images 120
Iteration 121
While Loop 121
For Loop 123
Using glob to Find Files in a Folder 125
For Each Loop 125
Showing All Images 125
Secure Programming 127
CSS Review 129
Creating a Form View 132
Showing a Form for Uploading Images 133
php.ini 135
$_FILES 136
Secure Programming 137
Uploading Files with PHP 140
Planning an Uploader Class 140
Using the Uploader Class 145
The Single Responsibility Principle 147

Summary 148
Projects 148

Chapter 5: Reviewing PHP 8 Basic Syntax 151

Objectives 151
From the Beginning 151
Comments 152
PHP Functions 153
Variables 154
Conditional Statements 160
Logical Operators 166
Functions 171
Arrays 177
Loops 180
Enums 183
Summary 184
Projects 184

Part II 185

Chapter 6: Databases, MVC, and Data Objects 187

Objectives 187
The Basics of MySQL/MariaDB Data Storage 188
Manipulating Data with SQL 190
Developing a Database for the Poll 192
Building a Database Using CREATE 193
Secure Programming 195
The INSERT Statement 198
The SELECT Statement 199
Secure Programming 200
The UPDATE Statement 201
Secure Programming 202

Coding a Database-Driven Site Poll 202
Separating Concerns with MVC 203
Creating the Poll Project 205
Making a Poll Controller 206
Making a Poll Model 207
Making a Poll View 209
Hooking Up Poll View with Poll Model 209
Connecting to MySQL/MariaDB from PHP 211
Sharing the Database Connection with the Poll Model 212
Retrieving Data with a PDOStatement 214
Showing a Poll Form 218
Updating a Database Table According to Form Input 219
Secure Programming 221
Summary 223
Projects 224

Chapter 7: Building the Basic Blog System 225

Objectives 225
Creating the blog_entry Database Table 226
Planning the PHP Scripts 227
Admin View: Creating the Admin Blog Site 228
Creating the Admin Entry Manager Navigation 229
Loading Admin Module Controllers 231
Creating the Admin Entry Input Form 233
Styling the Admin Editor 235
Connecting to the Database 237
Using Design Patterns 238
Writing the Entry_Table Class 239
Secure Programming 241
Processing the Admin Form Input and Saving the Entry 242
User View: Getting Data for All Blog Entries 244
Using an SQL SUBSTRING Clause 245

Using an SQL Alias 246
Preparing a User View for All Blog Entries 246
Hooking Up the User View and User Model 248
Responding to User Requests to Read More 249
Getting Entry Data 250
Secure Programming 251
Secure Programming 260
Summary 261
Projects 261

Chapter 8: Basic Blog: Entries and Comments 263

Objectives 263
Creating a Model for the Administrative Module 264
Displaying Administrative Links 265
Populating the Form with the Entry to Be Edited 267
Handling Entry Deletion 271
Deleting Entries from the Database 271
Responding to Delete Requests 272
Preparing a Model to Update Entries in the Database 274
Controller: Should I Insert or Update? 275
Secure Programming 276
Insisting on a Title 278
Secure Programming 278
User View: Building and Displaying the Comment Entry Form 281
A Combined View 283
Creating a Comment Table in the Database 285
Using a Foreign Key 286
Building a Comment_Table Class 287
Inheritance 287
Is-a Relationships 288
Using Inheritance in Our Code 288
Inserting a Comment Through the Comment Form 293

Searching for Entries 296
The Search View 297
Responding to a User Search 299
The Search Model 301
A Search Result View 302
Loading a Search Result View from the Controller 303
Exercise: Improving Search 304
Summary 305
Projects 305

Chapter 9: Basic Blog: Images and Authentication 307

Objectives 307
Deleting Entries in Related Tables 309
Understanding Foreign Key Constraints 309
Deleting Comments Before Blog Entry 310
Creating an Image Manager 311
Showing a Form for Uploading Images 312
A Quick Refresher on the $_FILES Superglobal Array 314
Uploading an Image 315
Displaying Images 320
Using an Image in a Blog Entry 324
Improving Security with Authentication 329
Creating an admin_table in the Database 330
Hashing the Password with BCRYPT 331
One-Way Hashing 331
Sufficient Security 332
Adding Administrators in the Database 332
Building an HTML Form 332
Saving New Administrators in the Database 335
Planning Login 341
Creating a Login Form 341
Hiding Controls from Unauthorized Users 342

HTTP Is Stateless 344
Superglobal: $_SESSION 344
Persisting State with a Session 345
Logging Users Out 347
Allowing Authorized Users Only 349
Summary 355
Projects 355

Chapter 10: Data Dashboard and Gaming 357

Objectives 357
Setting Up a Data Dashboard 358
Gathering Microsoft Excel, CSV, JSON, and Database Data 360
Creating the Model Data Class 365
Creating the Drop-Down and File Type Views 379
Creating the Front Door Controller and the Subcontrollers 382
Creating the Logic for a Checkers Game 397
Summary 418
Projects 418

Index 421

About the Authors

Jason Lengstorf is a turbogeek from Portland, OR. He started building websites in his late teens when his band couldn't afford to pay someone to do it, and he continued building websites after he realized his band wasn't actually very good. He's been a full-time freelance web developer since 2007 and expanded his business under the name Copter Labs, which is now a distributed freelance collective, keeping about ten freelancers worldwide busy. He is also the author of *PHP for Absolute Beginners* and *Pro PHP and jQuery*.

Thomas Blom Hansen has extensive experience teaching web programming in the Digital section of the Copenhagen School of Design and Technology. When he is not teaching, you can find Thomas fly-fishing for sea-run brown trout in the coastal waters around Denmark or possibly hiking some wilderness area in southern Scandinavia. Thomas lives in a small village with his wife, three kids, too few fly rods, and a lightweight camping hammock.

Steve Prettyman is a college instructor on PHP programming, web development, and related technologies. He is and has been a practicing web developer and is a book author. He has authored several books on PHP including *Learn PHP 7* and *PHP Arrays* for Apress.

About the Technical Reviewer

Satej Kumar Sahu works in the role of Senior Enterprise Architect at Honeywell. He is passionate about technology, people, and nature. He believes through technology and conscientious decision-making, each of us has the power to make this world a better place. In his free time, he can be found reading books, playing basketball, and having fun with friends and family.

Introduction

Modern web development relies on the successful integration of several technologies. Content is mostly formatted as HTML. With server-side technologies, you can create highly dynamic web applications. PHP is the single most used server-side scripting language for delivering browser-based web applications. PHP is the backbone of online giants such as Facebook, Flickr, and Yahoo.

There are other server-side languages available for web application development, but PHP is the workhorse of the Internet. For an absolute beginner, it should be comforting to know that PHP is a relatively easy language to learn. You can do many things with a little PHP. Also, there is a thriving, friendly community supporting PHP. It will be easy to get help with your own PHP projects.

Who Should Read This Book

This book is intended for those who know some HTML and CSS. It is for those who are ready to take their web developer skills to the next level. You will learn to generate HTML and CSS dynamically, using PHP and MySQL. You will learn the difference between client-side and server-side scripting through hands-on experience with PHP/MySQL projects. Emphasis will be on getting up and running with PHP, but you will also get to use some MySQL in your projects. By the end of the book, you will have created a number of PHP-driven projects, including the following:

- A personal portfolio site with dynamic navigation
- A dynamic image gallery where users can upload images through an HTML form
- A personal blogging system, complete with a login and an administration module

In the process, you will become acquainted with such topics as object-oriented programming, design patterns, progressive enhancement, and database design. You will not get to learn everything there is to know about PHP, but you will be off to a good start.

How to Read This Book

This book is divided into two main parts. Part I will quickly get you started writing PHP for small, dynamic projects. You will be introduced to a relatively small subset of PHP – just enough for you to develop entry-level web applications. Part I will also teach you the basic vocabulary of PHP.

Part II is a long hands-on project. You will be guided through the development of the aforementioned personal blogging system, starting from scratch. Part II will show you how to use your PHP vocabulary to design dynamic, database-driven web applications.

Source Code

All code used in this book can be downloaded from `github.com/apress/php8-for-absolute-beginners`.

PART I

CHAPTER 1

Getting Ready to Program

Objectives

After completing this chapter, you will be able to

- Understand how operating systems make programming easier
- Understand how PHP works with Apache and MySQL/MariaDB to create dynamic web pages
- Install a PHP test environment
- Determine if the test environment is working properly
- Create a simple PHP program
- Execute and test a simple PHP program

Welcome to the world of programming! Whether you have never attempted to write a program before or you have been creating programs for a while, we hope that this book will help you to understand the basics of program development. Every minute of your life has been surrounded by computers. From the monitors in the delivery room the moment you were born to the coffee maker that brewed your dark roast this morning (assuming you like coffee), computers and programs attempt to make our lives easier. Your interest in programming might have grown due to your ability to shine in the gaming world, from using social media applications, or, maybe, from watching entrepreneurs on TV pitch their inventions in an attempt to create a successful business. No matter the reason, you are here to discover how to create programs in the PHP language and, more importantly, determine if programming is something you want to do.

J. Lengstorf et al., *PHP 8 for Absolute Beginners*, https://doi.org/10.1007/978-1-4842-8205-2_1

The IT (information technology) industry provides unlimited potential for you to be creative and a true pioneer. There is no limit to what you can design and create. Your creation could help save the planet or make life easier for a disabled person. It is truly up to you to determine what you will invent. You can choose to work for a large multinational corporation, a small startup, or venture out on your own. As your career progresses, you can change your path as many times as you need. You might start with a larger company to build up experience, slide into a startup once you discover your expertise, and finally develop your own application which will pave your way to retirement. The IT industry provides you the freedom to determine your path.

So, let's begin this adventure. Since we assume you want to start programming as quickly as possible (that is why you bought this book, right?), let's briefly cover some groundwork, so we can all start from the same level of understanding.

Here we go! First and foremost, "Computers are dumb." What? But they provide us with so many amazing tools, how can they be dumb? To keep it simple, computers only know two things, 0 and 1. This is based on the idea that a circuit either has electricity (1) or it does not (0). This is the basic building block of how computer *hardware* and *software* is designed. Hardware is what we commonly think of when someone mentions a computer. The physical components, such as the keyboard, screen, circuits, memory chips, and other internal components. The software is the actual programs (applications) that communicate with the hardware. Every computer has an *operating system* which provides the ability for an application (and possibly a human) to communicate with the hardware. The operating system is like a language interpreter; it converts the information it receives (from an application or human) into the language the computer understands. Just like a human interpreter can convert English to Spanish.

A *program* is usually defined as a set of code that accomplishes a task. An *application* (app) can be many programs, types of hardware, and even people working together to accomplish the task. However, don't get hung up on these definitions, because in the real world we tend to use these two words interchangeably. As you will note, we have already done so in this book. The application software is usually not designed to directly communicate with the hardware of the computer. It talks to the operating system, which then talks to the hardware.

As an example, think about the applications you use. When you want to print something from an application, a print window appears, giving you options for your task. If you were then to open up another application on your computer and print from that application, the same print window appears. Where did that come from? The operating

system. Operating systems include coding for common tasks that applications request. These modules include blocks of code and the *interface* (graphical window) when required. An application uses an *API* (*application programming interface*) call when using one of these tools. An API is simply a line of code that tells the operating system what block of code to execute and what parameters (information) to use when executing the code.

The print API might request the operating system place the requested document into the *print queue*. The print queue is a list of documents waiting to be printed. Not only can the operating system talk to the printer, but the printer can also talk to the operating system. It actually sends a special signal (*interrupt*) back to the operating system letting it know when a print job is complete (or if there is a problem, such as out of paper). The operating system, once a print job is complete, will remove that job from the queue and send the next job to the printer.

We have greatly simplified the actual process. For example, when the requests are sent back and forth between the operating system and the printer, the instructions (language) are converted back and forth from what the operating system understands to what the printer understands. Where does this conversion happen? With the help of a little application called the *print driver*. Actually, we could dig into this even deeper, but the point is that the application, the operating system, and the hardware (including the printer) work together to accomplish tasks. Hardware is worthless without software (except maybe as a doorstop), and software can't exist without hardware (except maybe in our imagination).

Remember, we stated that computers only know 0 and 1. So how do they accomplish so much if that is all they know? As we know, computers have the ability to store information. They can store this information in the *memory*, on an *internal storage device* (*hard drive*, *chip*), or even on an *external server* (the cloud). Information that is stored by a computer is stored as a series of 0s and 1s. A *bit* is a single 0 or 1. A *byte* is a series of multiple 0s and 1s. The size (number of 0s and 1s) of a byte might vary depending on the computer (…, 32, 64, 128, …), but it accomplishes the same task of either storing information or executing a task. A *word size* is the number of bits that can be stored within a computer at one location (in memory or a storage device) or executed at one time. The larger the word size, the more information that can be saved or executed at one time. Computers with larger word sizes process information faster.

Unicode is a standard that combines bits together to represent many symbols and languages used throughout the world. Quite a task indeed! At one time, programming was mostly designed for English-speaking countries. However, today, programmers exist in all regions of the world. Let's look at an example of following a simple process of typing a character on the keyboard into a document. Assuming we have opened our favorite text processor (such as Microsoft Word), we can begin typing. When we click the letter "s" on the keyboard, it magically appears on the screen (in the document). How does that happen? When a key on the keyboard is pressed, the bit pattern (0s and 1s) for the letter is sent to the text editor via the operating system. However, most people don't understand patterns of 0s and 1s. So, when the text editor receives the information, the information that is displayed is converted on the monitor to the letter we pressed ("s"). Notice, we stated that the display is converted, not the actual data itself. When we store the data, it is still stored in *bit format*.

Again, we have simplified it, because the conversion actually involves memory, the text editor, its driver, the operating system, a graphics card, and the monitor and its driver. A lot goes on very quickly. Luckily, we can just understand that this all happens, without digging into the details. Just remember, what the computer understands, and what we understand is different.

In addition to being a language interpreter, the operating system is also a *memory manager*. In all higher-level programming languages (such as PHP), when data needs to be temporarily stored in memory or more permanently stored on a storage device (or cloud server), the operating system takes over. As a programmer, we ask the operating system to store data in memory by creating variables or constants. A *variable* is similar to the variables we used in algebra. In algebra, when A + B = C, A, B, and C are all variables. We know that they can represent any number. In programming, the same variables can also store numbers, or even characters, while, at the same time, informing the operating system to temporarily store these items in memory.

Most program languages (including PHP) reverse the algebra equation to C = A + B. Why? Glad you asked. While A, B, and C are variables, the equal sign in programming is actually an *assignment operator*. It takes whatever is on the right side (A + B) and stores it into whatever is on the left side (C). Thus, it takes the value (number) stored in A and adds it to the value stored in B and places the result into the variable C. All three of these variables reside in the memory of the computer.

The result of our addition program is stored someplace in memory, and we don't know where. What? We really don't care where, as long as we can use the variable C to retrieve it anytime we need it. We trust that the operating system has our back (it does the right thing by properly storing and protecting our information).

The operating system will either look at the values that are being stored in the variable or look at a *variable declaration statement* in the program code to help determine how to store the variable. Remember that bit patterns are used to represent all characters (including numbers) in a computer. When information is stored in memory or a storage device, it is stored in bit format. Thus, the operating system needs to know if a number is being stored or a character is being stored. There is a difference between the number 1 and the character 1. Basically, we usually don't do mathematics on a character, and we do on a number. Thus, the system needs to know if we might do mathematics. If we are going to use it in a calculation, it is stored in a format that allows us to do so. We will explore this more when we do calculations in PHP.

This also brings up another difference between us and the computer. We create numbers using the *base 10 system*. We count from 0 to 9 (ten different numbers). Then when we need to go to the next number, we add a digit to form the number 10. As you know, eventually we'll add a digit when we go to 100 (10 sets of 10). Computers, however, use the *binary system* (*base 2*) to store a 0 or a 1 (two different numbers). When the computer needs to store a number bigger than that, it adds a digit (10). You might have noticed when you are using a computer or shop for one that everything seems to actually be based on eighths (8, 16, 32, 64, …). Why? Actually, that occurs logically as we add more bits.

When converting from binary 0 to base 10 zero (or from binary 1 to base ten 1) is easy because they mean the same value (zero or one). However, a binary 10 is not the same as a base ten 10. A base ten 10 represents 10 values (ten fingers). In the last paragraph, we mentioned that when a binary number needs to go above 1, it needs to add a digit. A binary 10 is one more than 1; it is the number 2 in base ten (two fingers).

Confused? Let's look at some examples.

Table 1-1. *Comparing Base Ten to Binary*

Base Ten	Binary (Base Two)
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111

Remember, the first digit (to the right) in binary is a 0 or 1. The second digit represents a 2. Thus, 10 is 2 (2 + 0). 11 is 3 (2 + 1). Since 11 is 3, we have to add a digit to produce a 4 (100). Five is 101 (4 + 0 + 1). Six is 110 (4 + 2 + 0). Seven is 111 (4 + 2 + 1). It takes some practice to get comfortable with this. Don't get hung up on the details; if you understand the basic idea, eventually you will have an "aha" moment as your programming skills increase. Notice that when we reach three digits (111), we have actually created eight values (0–7). This logically shows that bits can be collected based on eighths. 111 111 would be two sets of 8s (8 × 2 = 16). 111 111 111 is three sets of 8s (8 × 3 = 24). 111 111 111 111 is 32 (8 × 4). It becomes logical that we look at multiples of eight when determining how the computer stores information. Let's move on; we will look at this again later.

Let's return to our operating system discussion. The operating system also will determine where in memory a variable is stored. The *algorithm* (code) the operating system uses to determine where it stores the data is well refined and very efficient. It has been tweaked for many years and provides the best solution for storage of information. The operating system looks at a lot of factors (too many for us to explore in a beginner's book) to determine where to store the data. Some of what it considers is how frequent it might be used, what program is using it, how big the data is, and what type of data is being stored.

We hope you are now seeing that the operating system makes our lives easier as programmers. We don't have to worry about details of where to store information and even where our actual program runs in memory. The operating system will allocate locations in memory for everything our program needs. It will also try to protect our program from other programs trying to cause it harm and even handle problems when our program decides to crash. Depending on what caused our program to crash, it might even let us know, via *error messages*, what happened. For example, if there is not enough memory available to run our program (or continue running our program), it will let us know.

When a program ends (either because it is done or has crashed), the operating system will clean up the memory the program used and make it available for the next program that may need it. Actually, technically, it just sets a *flag* (bit) in the memory locations which declares them as available. When another program or data is stored in these locations, the old 0s and 1s are written over. It's not necessary to empty out the location before using it again because new code or data will be placed in that location. Thus, overwriting it anyway.

Let's look at one final thing that an operating system does to help our program. The operating system is a *task manager*. It decides on when and how long our program can run before it stops or gets interrupted. Let's do a quick exercise to look at tasks running.

Exercise: Locate the task manager for your operating system. You can do a search on your computer for "task manager," or if you are using a Microsoft Windows machine, you can click the ctrl (lower left of keyboard), alt (lower left of keyboard), and delete (upper right of keyboard) keys at the same time. Then click "task manager" on the list that appears.

Your manager should be similar to Figure 1-1. The chart presented gives us a hint of many tasks the operating system is maintaining on our computer. It is managing CPU, memory, disk, and network usage. This includes any applications we are using (like Microsoft Word). Other programs, which we may or may not have directly used ourselves, are also running in the background (such as Amazon Photos). As the tabs in your manager or in Figure 1-1 indicate, the operating system also manages performance and other services.

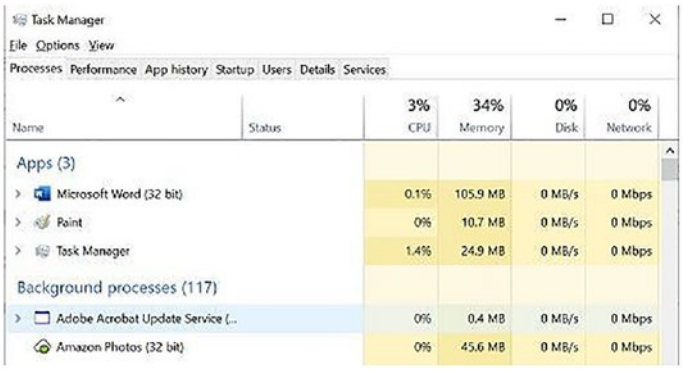

Figure 1-1. *Task manager*

When we start a program, as stated before, the system will determine memory usage and memory location for the program. It will also determine when and how long the program will run before it stops or is interrupted. The system determines the *priority* of the program (system programs like the operating system itself have the highest priority) and how much of a time slice it will allow for the program to run. The system's algorithms are very accurate in determining *runtimes* for programs. However, outside factors can slow a program down, such as emergency system problems (might be a memory shortage). The operating system can *swap* out your program when necessary to run other programs with higher priority. But, normally, we never even notice because everything executes extremely fast.

Note If you have been using computers for a while, you may have experienced using applications that never seem to work (they seem to be hung). This could be caused by a shortage of memory (or storage), which the system is trying to resolve by swapping programs in and out of memory to share what limited memory is available. However, it can cause the system to spend a lot of time swapping and allowing very little time for the application to run. It's important that you pay attention to the amount of memory (and storage) any application you install will need to run efficiently. Otherwise, not only might your application not run properly, but it could tie up your computer with constant swapping, so nothing will run properly.

We have only skimmed the surface of a very deep ocean when talking about how computers operate. There are many books, videos, and courses you can discover which can provide a much deeper understanding. Our goal, however, is to give us enough knowledge on how all of this affects any program we create. Hopefully, we have built a basic understanding of this, so let's move on to another subject.

Why start with PHP?

A very good question indeed. There are a lot of programming languages that we could choose. Why select PHP as our first experience? First, PHP is one of the easier languages to learn. You can accomplish a lot with just some basic commands and concepts. It is one of the more popular languages used because it can manage web pages and applications. It is commonly considered one of the skills needed to become a *full stack developer*.

Note A full stack developer understands both the front end of a web application (the web page displayed to the user) and the back end (code used by the web page existing on the web server). They have knowledge of front-end tools, such as HTML, CSS, and JavaScript. They have experience using languages that can support the web page on the back end (such as PHP and Java). They also may use additional tools to manage the development cycle.

With *PHP 8*, the language has become much more efficient in operation and in computations than ever before. While, as of the creation of this book, PHP is not known as a language for developing gaming platforms or big data operations due to its previous limitations, the efficiency of the newest versions could soon make this a reality. There are some groups beginning development of smartphone applications using PHP. But even if it never becomes a gaming platform, there is plenty of work for PHP to accomplish by just hosting web pages and creating applications.

Exercise: Is PHP alive and well? Don't believe us. Go to your favorite search engine and ask the following question: "Who uses PHP?" or "Is PHP alive and well?" What did you find? The answer should tell you, "Yes, PHP is alive and well" and "Lots of organizations use PHP." What organizations are currently using PHP? What are they using it for? What does the future hold for PHP? You will discover that there is current development in creating big data dashboards, smartphone applications, and gaming applications, along with web applications. Lookout Python, PHP might be after your job!

Setting Up a Development Environment

Getting a working *development environment* put together might initially be intimidating, especially for the absolute beginner. However, developers have created many types of software packages, which can install everything we need, with default settings. No longer do we need to go into the setup files to connect our environment together (unless we choose to do so). The environment is automatically linked together for us. To follow along with the projects in this book, we will need to have access to a working development (test) environment which contains *Apache* (*web server*), PHP, and *MySQL/MariaDB* (*database*). It's always desirable to test locally (on a single machine, not a server), both for speed and security. Doing this both shelters your work in progress from the hackers on the open Internet and decreases the amount of time spent uploading files to a *web server*. It allows you to completely test and secure your programs before placing them in a live environment.

PHP is a powerful *scripting language* that can be run by itself. However, PHP alone isn't sufficient for building dynamic website applications. To use PHP for a website, we need a web server that knows how to process PHP scripts. Apache is a free web server that, once installed on a computer, allows developers to test PHP scripts locally; this makes it an invaluable piece for a local development environment. Apache is the most popular web server used in conjunction with PHP.

Additionally, web applications need to store information. PHP code can be developed to store this information in a database, so it can be modified quickly and easily. This is the significant difference between a PHP application and an *HTML* site. Strictly HTML-only sites cannot store information. This is where a *relational database management system* such as MySQL can come into play. Many of the book's examples store information using the MySQL/MariaDB database systems.

Note Without going into too much detail, MySQL and MariaDB are very similar systems. When Oracle purchased MySQL, some developers wanted to ensure that a free version of MySQL would still exist (although Oracle does still provide a free version as of the release of this book). MariaDB was the result of this collaborative effort. They ensured that no coding changes would be needed when creating programs using either database system. You will even discover some of the software packages still use the term MySQL when they are really referring to a MariaDB database.

What Is PHP? How Does PHP Work?

PHP is a general-purpose scripting language that was originally conceived by Rasmus Lerdorf in 1995. Lerdorf created PHP to satisfy the need for an easy way to process data when creating pages for the *World Wide Web*.

Note PHP was born out of Rasmus Lerdorf's desire to create a script that would keep track of how many visits his online résumé received. Due to the wild popularity of the script he created, Lerdorf continued developing the language. Over time, other developers joined him in creating the software. Today, PHP is one of the most popular scripting languages.

PHP originally stood for *Personal Home Page* and was released as a free, *open source* project. Over time, the language was reworked to meet the needs of its users. In 1997, PHP was renamed PHP: *Hypertext Preprocessor*, as it is known currently. At the time we are writing this, PHP 8.1.1 is the current stable version. Older versions of PHP are still in use on many servers. However, they might not be as secure. PHP 8 has provided many powerful changes to the language which has increased usability, speed, and security. All code provided in this book works with PHP 8 (or later). Some of the code will also work with previous versions. We suggest you install and use the most current version available.

HTML is parsed by a *browser* on the user's computer after the page downloads. The browsers determine how to display the information provided from the code provided by the HTML and CSS. Since, unlike HTML, PHP code is retained on a remote web server, browsers cannot process PHP code. PHP is processed by a *PHP interpreter* connected to a web server (such as Apache). The results of the execution of the PHP code (not the actual PHP code) are included in the document (web page) before it is sent to the user's browser to be interpreted. Because PHP is processed on a server, it is a *server-side scripting language*.

Exercise: Search for a company that uses PHP. Go to their website. View the source code of the main page of their site. You can view the code by selecting "View Source" within your browser. Did you find some PHP code? The answer is no. Why? Because your browser shows the results after the PHP code has been processed by the web server. You can see HTML, CSS, and JavaScript code, but you will not see any PHP code. Where can you see the PHP code? If you had access, you could view it on the server itself.

With PHP, you can create dynamic web pages (web pages that can change according to conditions). For example: When you log in to your Facebook account, you can see your content. When you log in to another Facebook account, you see different content. We are loading the same resource (`www.facebook.com`) and code, but we are served different content dynamically. This would be impossible with strictly HTML web pages because they are *static*, meaning they can't change. Every user would see exactly the same HTML page. We will soon explore many more examples of dynamic web pages in this book.

PHP is an interpreted language, which is another great advantage for PHP programmers. Many programming languages require that you *compile* files into *machine code* before they can be run, which can be a time-consuming process. Bypassing the need to compile every time you make a code change means you're able to edit and test code much more quickly. However, this also can cause PHP to be slower than compiled programs in a live environment. Compiled programs are already machine-level programs that the server can directly run. Script programs (like PHP) have to first be interpreted before they can run. With PHP 8, this disadvantage has been removed. PHP 8 includes a *JIT (just-in-time) compiler* which can be used to compile PHP code for live sites. This removes the delay.

Note While doing initial testing of our code, we first are concerned with removing all syntax and logical errors. During this phase, we can test PHP using the original interpreted mode. After errors are removed, for heavily used web applications, we can use the JIT compiler to speed up the PHP application in the live environment. Since this is a beginner's book, most of our testing will be using the interpretive mode of PHP, to save debugging time and effort.

Because PHP is a server-side language, running PHP scripts requires a server. To develop PHP projects in a local development environment means we need to install a server on our local machine. The examples in this book rely on the Apache web server to deliver our web pages, since it is the most popular server used with PHP.