Chapter 1

Web Development Basics

LINUX PROFESSIONAL INSTITUTE, WEB DEVELOPMENT ESSENTIALS EXAM 030-100 OBJECTIVES COVERED IN THIS CHAPTER:

✓ **031 Software Development and Web Technologies**

- 031.1 Software Development Basics: The candidate should be familiar with the most essential concepts of software development and be aware of important programming languages.
  - Key Knowledge Areas:
    - Understand what source code is
    - Understand the principles of compilers and interpreters
    - Understand the concept of libraries
    - Understand the concepts of functional, procedural and object-oriented programming
    - Awareness of common features of source code editors and integrated development environment (IDE)
    - Awareness of version control systems
    - Awareness of software testing
    - Awareness of important programming languages (C, C++, C#, Java, JavaScript, Python, PHP)
- 031.2 Web Application Architecture
  - Key Knowledge Areas:
    - Understand the principles of APIs

The main focus of this chapter is the first subobjective of the Linux Professional Institute (LPI) Web Development Essentials Exam, software development. Along the way, you'll learn about the different types of web development and get a bird's-eye view of how programs are created. We'll compare writing environments and compiled versus interpreted languages, then look at different programming paradigms and what happens after a program is written. Grab your favorite note-taking tool, and let's get started!

Developer Types

Most likely, you already know that web development is the process of creating and maintaining web pages. Web pages can be found on servers across the Internet or on your own corporate server used to disseminate information internally. Regardless of where the site exists, there are different developer roles at work creating and maintaining the site.

Let's look first at the *front-end developer*. Front-end developers work on the parts of a website with which the users interact. It's more than just how the website looks. There are actions to consider, such as what happens when you click a button or mouse over a particular spot. These developers are creating the user experience. You can think of this as an artistic endeavor, but front-end developers also need to be able to program using software such as JavaScript, Hypertext Markup Language (HTML), and Cascading Style Sheets (CSS)—which are all covered in later chapters. Front-end development is a combination of programming skill and artistic design to create a positive experience for whoever visits the website, whether they are on a PC, a Mac, or their smartphone. The front-end of a website is also sometimes called the *client-side*.

Back-end developers are responsible for what's happening behind the scenes in a website; they're responsible for the parts that the user doesn't see but that are extremely important. Consider your favorite website for purchasing gifts. The background of that website most certainly has databases containing information about you, your prior orders, and their available inventory. The back-end is also called the *server-side*.

Finally, there are *full-stack developers*. As you might have guessed, full-stack developers are responsible for the entire process of designing a website, which includes both what the end users experience and the information and functionality that resides with the servers supporting the website.

Creating Software

Software, in a very broad sense, consists of lines of instructions for a processor to follow. That processor can be located in a computer, a smartphone, or even the microcontroller of your favorite Internet of Things (IoT) device. Software allows users to interact with hardware and data. Several different methods for developing software exist, most of which will have several iterations of the process before a program is complete. You may have heard of the Agile, Waterfall, Shell, or Sprint methods, but full-on software development and project management isn't within the scope of this certification, so we'll discuss creating software in more general terms.

Software often starts with *pseudocode*. Pseudocode is in human language, describing what we would like the program to do, for example:

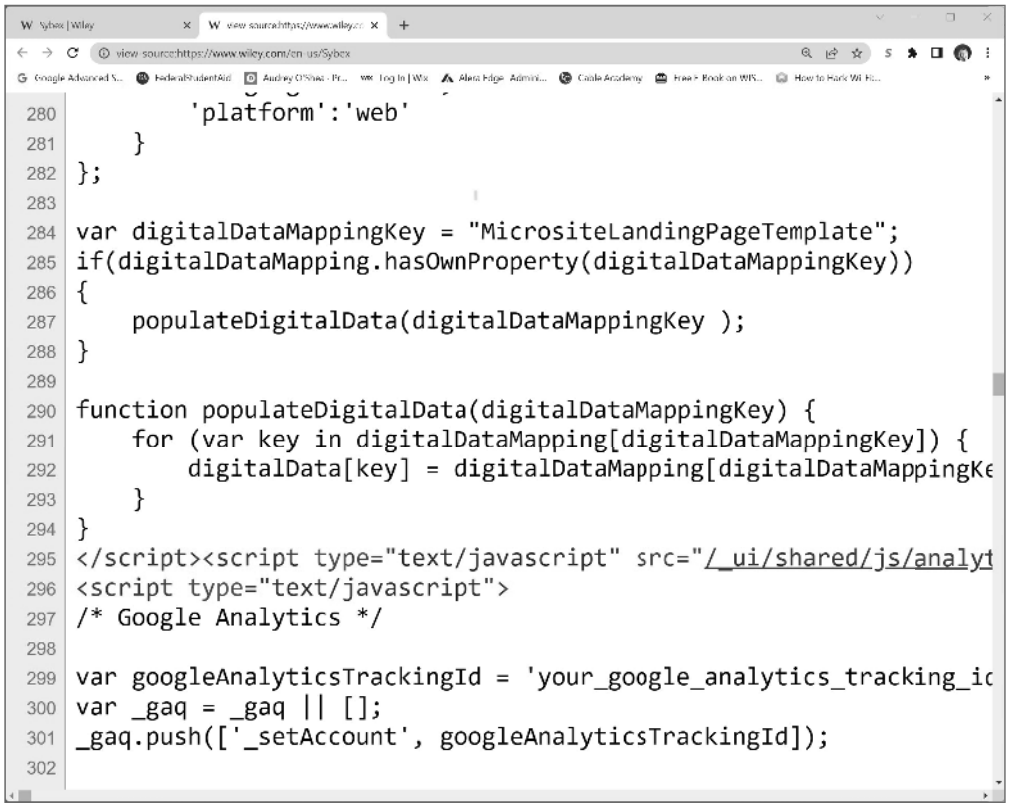
- Ask the user to input their first name.
- Ask the user to input their date of birth.
- Calculate the user's age as of today.
- Print out "Hello %firstname%. I see that you are %age% years old." (% signs indicate something that changes.)

From there, software is turned into *source code*. Source code is still written in a format that humans can understand, if those humans are programmers. Source code is the result when a programmer takes those lines of pseudocode and turns them into instructions that are written in a particular programming language. In the Windows environment, most web browsers will allow you to see the source code of a web page, such as in Figure 1.1, by pressing Ctrl+U.

Just as humans speak different languages, there are different *programming languages*. Programming languages consist of a specific set of rules and words that are used to tell software or hardware to perform tasks. Each language has its own words and rules, similar to human grammar, that distinguish one language from another. Examples of these languages are C, C++, C# (the # is read as sharp,) Java, JavaScript, Python, Ruby, Perl, and PHP, to name a few. Each of these languages will have its advantages and disadvantages, and one may be preferred over another, depending on the project at hand. Some are used more on the front-end (for example, HTML, CSS, JavaScript, jQuery), whereas others tend to be used more on the back-end (for example, C#, Java, Python, PHP, Ruby, SQL).

Each of these languages will have their own syntax. *Syntax* refers to the exact order and presentation of the commands, including spaces, punctuation, capitalization, and so on. For example, in JavaScript, I would declare a variable to store the user's input for their first name with a line that says `var firstName`. The same variable in C++ would be declared as `string firstName`.

FIGURE 1.1 Source code example



Data can be presented in different types such as characters, strings, Boolean, integers, and floats. (These types of data will be discussed more in depth in Chapter 14, "JavaScript Data.") A first name, as in the previous example, would be a `string` of characters, and `var` refers to the term *variable*, which is data that may change.

Different programs also use different keywords. *Keywords* are words reserved by a program because they have a specific meaning in that particular program. Some JavaScript keywords are `goto`, `float`, `short`, `finally`, and `char`, in addition to many more. C++ also uses the keywords `goto`, `float`, `short`, and `char`, but has many others that are not the same as JavaScript, such as `auto` and `union`. These are just a few of the keywords that are available in various languages. A single language can have dozens to hundreds of keywords.

Keywords can be described as different types depending on what they do. The following are just a few keywords that are used in Python. Keywords can be operators (`and`, `or`, `not`, `is`), or they can control program flow (`if`, `else`), control how many times (iterations) a program does something (`for`, `while`), or import data (`import`, `from`, `as`). They can even handle problems (`try`, `except`, `finally`, `else`). These are just some of the functions

available using keywords. Remember, some keywords are used in many programs, whereas others are specific to a particular program. Like learning any human language, once you master one, the others are easier to learn.

Just as different programming languages may share the same keywords, programmers may share their work in *libraries*. At its most basic level, a library is a collection of code that is shared to be reused by others. Programming languages have a standard library that is distributed with the program to provide functionality and solve common problems and needs. Python, for example, has an extensive standard library and thousands of third-party packages (libraries or modules) that can be added to the program. The common assumption is that a library is a collection of packages, and a package is a collection of modules, although all three terms are sometimes used interchangeably.

Libraries may solve a particular problem that many programmers encounter, or an uncommon problem that is annoying for just a few programmers. An example of a common problem is how to work with a particular LCD screen. A programmer will create code to interface with the LCD screen and share it with other programmers as a library. Libraries are essential to programmers and save time because the programmers don't need to re-create the solution. Every programming language will have libraries available for it, and some libraries will work with more than one language.

Regardless of the programming language, libraries need to be imported before they can be used. The exact command to import a library will vary by program. For Python, it's `import libraryName`, where you would replace `libraryName` with the actual name of the library. C++ uses `#include <packageName>` where `packageName` is replaced with the actual name of the package. Figure 1.2 shows a line of code that includes a library in the Arduino integrated development environment (IDE) for working with an LCD screen.

FIGURE 1.2 Adding a library

To further complicate the process, source code is not always written in just one programming language. Because each language has different strengths, one language may embed, or call, something written in another language. This happens often with websites that have JavaScript code embedded in an HTML document. Programmers may also use an *application programming interface (API),* which provides a set of rules and protocols that let two dissimilar systems share information, or in this case, two different programming languages. APIs make a programmer's work simpler, and many APIs have been created for various purposes. An API may be private, for use within a single entity, or it may be shared with specific partners or be public and available to anyone to use.

Text Editors and IDEs

Digital devices, such as computers, tablets, and smartphones, are only able to discern if a particular circuit is on or off, so everything in a computer system is represented as either a 1 (the circuit is turned on) or a 0 (the circuit is turned off.) The *binary number system* represents all values using only two digits; 1 and 0. Machines understand binary notation, and although some of us can easily convert short binary numbers to decimal numbers, a document written entirely in binary wouldn't be easily understood by humans. It's amazing to think that a detailed, colorful image or even a symphony playing is just a bunch of 1s and 0s to a computer!

Machine language is binary code that can be read by computing devices but is not readable by most humans. Source code must be converted to machine language *before* the computing device can follow the source code to carry out the programmer's wishes.

You may have wondered what tool you can use to write your own source code. Source code can be written using almost any simple text editor. A *text editor* is a program that allows the user to enter text in a human language or in a programming language, then save it with an appropriate file extension for the language in which the document is written. Examples of text editors are Notepad in Microsoft Windows, or Notepad ++ that works on Microsoft Windows, TextEdit that works with Macs, or Vim that is included in many Linux distributions. In Exercise 1.1, "Hello World," we will use Vim to write your first program. Atom is an open source and cross-platform text editor.

Programs are written in plain text. Some text editors, such as Microsoft WordPad, add formatting characters to the text and are not a suitable choice for writing source code. Other text editors, like Notepad++, use different colors to alert the programmer when something is amiss, such as an error in syntax. Some even provide autosuggestions for completing lines of code.

If you would like to try it, the latest edition of Notepad++ can be downloaded from `https://notepad-plus-plus.org/downloads`. Download the installer onto a Windows-based machine, taking note of where you saved it. Double-click the installer file and follow the onscreen prompts to complete the installation. Notepad++ is a favorite of mine because the colors quickly show me the error of my ways, enabling me to avoid hours of troubleshooting code.

Each program will have a specific file extension to identify the language that the program is written in. Regardless of which tool you use to create your source code, be sure to save it using the proper extension. A *file extension* is the part of a filename that follows a period, such as `figure1.jpg`. The `jpg` after the period tells the operating system that this is a JPEG image. The following are some popular programming extensions and the programs they identify:

- `.c`: C and C++
- `.cpp`: C++

- `.cs`: Visual C#
- `.css`: Cascading Style Sheet
- `.htm` or `.html`: HTML files
- `.java`: Java
- `.js`: JavaScript
- `.php`: PHP
- `.py`: Python
- `.sql`: Structured Query Language

Although Structured Query Language (SQL) is the language specifically used to communicate with a database, it makes the list. SQL meets the definition of a programming language because it requires very specific keywords and syntax to communicate between humans manipulating data and a database.

Some source code, such as HTML, can be used as soon as you've saved it from your text editor, and we will do just that in Chapter 5, "HTML Introduction." Other languages require an additional step called compiling. *Compiling* is a process that converts source code to machine language. For those languages that need to be compiled, you need to download additional tools, unless you use an *integrated development environment (IDE)*. An IDE provides the text editor, a compiler, and other tools necessary to convert source code to machine language.

An IDE must support the language that the programmer is writing in. Code::Blocks is a free IDE that supports several languages. Visual Studio Code is cross platform and is very popular and free to try, but continued use requires a subscription. PyCharm is, as you likely guessed, an IDE for the Python language, but it also supports JavaScript and SQL. It has a free version and a full (paid) version as well. Many IDEs have the features of better text editors and may have debugging and testing features as well.

Compiled Languages

A *compiled language* is one that requires the source code to be converted into machine language *before* it can be used. Source code must be compiled into machine code for each platform that it will run on, such as Windows, macOS, or Linux. Remember that machine language is simply 1s and 0s, but they must be arranged in a manner that a processor can use. The GNU Complier Collection (GCC) and Tiny C Compiler (TCC) are both free compilers, although their licensing is somewhat different. The GCC can be used to compile many different languages, including C and C++. Intel also has a C++ compiler, as does IBM, and there are other compilers for use with ARM processors and different programming languages.

Compiled programs take a little longer to create because of the extra compiling step, and they need to be compiled multiple times to be available for multiple platforms. For example, to have the program available to run on Windows, macOS and Linux, it would need to be compiled three times, once for each OS. The advantage of compiled programs is that they

tend to run more quickly when executed than programs that are not compiled. C++ and C are typically compiled languages.

Some compilers can also output bytecode. *Bytecode* is the output of a compiler that is intended to be interpreted by a runtime or virtual machine (VM) instead of running directly on a processor. Bytecode is somewhere between source code and machine code. It's a bit slower than running a compiled language, but the advantage of bytecode is that it can be used by multiple platforms—it isn't necessarily compiled for each specific one. A *runtime* provides the environment that bytecode needs to run. Similar to a mini operating system, the runtime provides access to hardware, software, and the user. A Java Virtual Machine (JVM) provides a runtime environment for Java programs. Each platform will have its own JVM that is often installed with the operating system. Python automatically compiles its code into bytecode, so it will run on any machine that has a Python interpreter on it. Bytenode is a bytecode compiler for node.js.

Interpreted Languages

You've likely seen real-life language interpreters before. When foreign dignitaries meet, they will need an interpreter to convert the conversation from one language to another, if they speak different languages, one phrase at a time. Interpreted languages are like that. Interpreted languages don't need to be compiled to be used. Python and PHP are examples of languages that are typically interpreted, although like most programs, they could also be compiled. A web browser is an interpreter for HTML pages. A software *interpreter* executes instructions in the source code as it reads them, communicating those instructions to the processor, which understands machine language.

The advantage of interpreted code is that it can be used cross platform, meaning that the same HTML code from your website can be read by a Mac, Windows PC, or a smartphone. The disadvantage of interpreting code is that it tends to run more slowly than compiled code because of the extra interpreting step necessary when the code is run.

Source code is the input to either a compiler or an interpreter. The output from a compiler is either bytecode or a program, while the source code that is read by an interpreter is often called a script, rather than a program. *Scripts*, therefore, are source code that can be executed without being compiled.

Programming Paradigms

You may have heard someone mention a paradigm shift, meaning that they had a sudden change of understanding in the way something exists or is done. *Programming paradigms* are approaches to how programming is done. A program needed to compute a certain value can be written in different paradigms, and the way programming is done will shift, depending on which programming paradigm is being used.

Any paradigm exists in multiple programs and multiple programming paradigms can exist in a single program. Different paradigms are appropriate for different types of

programs, and a paradigm may only be used if the language the program is written in supports that paradigm. Let's look at three general paradigms (approaches) to programming.

Procedural Programming

Procedural programming is likely what most people think of when they think of a computer program. A *procedural program* follows a list of instructions that are executed in order, starting with the first line, then moving on to the second line, third line, and so on. The focus of procedural programming is the process that is followed.

Parts of a program that need to be repeated are identified as *procedures*. Procedures, also known as functions, or subroutines, can be predefined and named. When the main program is run, functions and subroutines are called as needed. Although very similar, functions and subroutines are not the same. A *function* is a chunk of code that runs when called and returns a value to the main program, whereas a *subroutine* is a chunk of code that runs but does not return a value to the main program. Because of these named and reused chunks of code, procedural programming has a somewhat modular approach.

Even though procedural programming may use functions and subroutines, those will still be called by a main script that will be followed in sequential order.

Procedural programming can make the task of coding simpler because procedures can be used more than once by simply calling the procedure, instead of entering the code multiple times. This approach reduces the size of the program, and therefore, the memory overhead needed to run it. Procedural programming also mitigates the chance of human error when program changes are needed, because rather than making the same change in many places, it only needs to be changed one time, in the procedure, rather than every place that the procedure is needed.

Figure 1.3 depicts a simple procedural program that calls a function. Under `void setup()`, the line that says `Find_Current();` calls the function. The function is written in the lower section under `void loop()`. In the lower section, the variables `Vin`, `voltage`, `current`, and the constant value `r` are declared. *Variables* are used to store information in a program. Some variables may change as the program runs. *Constants* are variables that will not change while the program is running.

The sample program uses Ohm's law to calculate the current, measured in amps, where the resistance is 1000 and the voltage is read on an analog input pin. However, the output from the pin is a binary value that must be converted to voltage (`voltage=(Vin*5)/1024;`). The resistance is known to be 1000 (`r=1000;`) and will not change. Ohm's law states that current equals voltage divided by resistance calculated in the line `current=voltage/r;`. The last line of the function, `Serial.println(current);`, prints the current as a number. `Serial.println("Amps");` prints the word Amps.

Returning to look at the beginning of the program, `Serial.println("Current is ")` will print the words `Current is` and when the function is called it will read the pin, calculate the current (measured in amps), and print the current as a numeric value followed by the word `Amps`.

FIGURE 1.3 Simple function call

```
void setup() {
  //print current
  Serial.begin(9600);
  Serial.println("Current is ");
  Find_Current();  //this line calls the function
 }
void loop() {
}  //calculate and return current
    float Find_Current()
    {
      float Vin;
      Vin=analogRead(A1);
      float voltage;
      voltage=(Vin*5)/1024;
      float current;
      int r;
      r=1000;
      current=voltage/r;
      Serial.println(current);
      Serial.println("Amps");
```

Object-Oriented Programming

Procedural programming is all about the process, but *object-oriented programming (OOP)* is a programming paradigm centered around data objects and the class to which they are assigned. A *class* in OOP includes the attributes and methods that relate to each data object of that class. *Attributes* are the OOP term meaning a variable, and *methods* are subroutines attached to an OOP class.

The programming shown in Figure 1.1 could also be created using OOP. We could first define a class called `current` to which we assign objects such as `circuit 1`, `circuit 2`, and so on. The class would have attributes called `Vin` and `r`, and methods called `voltage` and `current`.

An example that's often used to explain object-oriented programming is a class called `car`. A car will have attributes such as make, model, color, mileage, gas-in-tank, and functions such as drive, brake, fill-up, and so on. We could create an object called `Car1`, where `make=Mercedes`, `model=C300`, `color = blue`, `mileage=50,000`, and `gas-in-tank=10`. If we run the function `fill-up`, the function may increase the `gas-in-tank` to 14. If we run the function `drive`, the `mileage` attribute would increase by the miles driven. In this way, the methods directly affect the state of data outside of the method. In programming, a *state* refers to the values stored in a program, similar to a snapshot of the program. We could have other cars, each with attributes of `make`, `model`, `color`, `mileage`, and `gas-in-tank` describing that car, and the functions defined in the class `car` would be used to change the attributes of that car.

Classes can also contain other classes. For example, we could have a class called `vehicles`, and within that class are the classes `cars`, `SUVs`, and `trucks`.

Functional Programming

Functional programming is a programming paradigm that is based solely on functions. In functional programming, a function can be treated as data and passed from one function to another. Unlike procedural programming, which follows a string of commands and where the function can change the state of a program, in functional programming the only effect that the function has is to return a value based on the equations it executes. It will always produce the same output if given the same inputs. By contrast, object-oriented programming interacts with the state of the data, changing the existing data to something new.

Maintaining Software

Once a program has been written, managing and maintaining that program becomes the primary focus. Simple programs might have the entire program contained in a single file, but many programs are very complex and have multiple parts stored in multiple folders and files. For example, open the folder that holds the source code for your operating system and list the files contained there. Storing parts of the program in an organized way, with different logical folders for different parts of the program, will make maintaining the program easier. For example, an accounting program might have a general ledger folder that holds all the files for that section, an accounts payable folder, an accounts receivable folder, and so on. A program could also be sorted by files affecting the front-end (i.e., the user interface) and files affecting the back-end of the program.

Complicated programs may also have several or hundreds of people who adjust the code to add features or fix problems. Those changes need to be managed, and the way to do that is through a version control system.

Version Control Systems

Version control systems (VCSs) are specialized software used to keep track of changes made to a computer program, when they are made, and who makes them. They can also be used to manage those changes by holding them in a queue and not updating the main source code until others have had a chance to review that change. Version control systems can be centralized or distributed, and can facilitate communication among teams who may be working in different offices or even different parts of the world.

Version control can be used to avoid problems caused by two programmers working on the same section of source code or lock code that is being edited so that only one person can change it at a time. They can also identify code changes that caused a problem. Software teams can look at the original code and what was changed and more easily revert to prior coding if needed. The features of version control can help programming teams finish a project more quickly and with fewer negative outcomes.

Git is a widely used, distributed version control software. Apache Subversion is a popular centralized VCS, and Mercurial is a popular distributed VCS, but Git is by far the most popular. All three are open source and free.

GitHub and GitLab are VCSs that may sound like Git, but they are owned by different people. Git is open source and maintained by Linux. However, GitHub is owned by Microsoft and is not open source. GitLab is owned by GitLab Inc., and only some of GitLab is open source.

All VCSs need a repository. A *repository* is a place that holds the files for version control. Repositories can be hosted on a local server or online, but the online service must support them. Some online services are free, and some are not.

Software Testing

Whenever software changes are made, it's necessary to test those changes to ensure that they work properly and don't cause any problems with the other parts of the program. If you're using an IDE to create or modify software, the IDE will do some of the testing for you. IDEs can detect errors in syntax, keywords, and functions, but they can't check the logic for you.

Summary

This chapter provided an introduction into the world of web development by discussing developer roles. If you plan to build your career in this field, you need to determine where to start. Will you be a front-end developer or a back-end developer? If you're like me, you just want to know all of it immediately like a full-stack developer, but you still need to pick a place to start.

We examined the process of creating software, going from pseudocode to source code, to bytecode or machine code. Then we examined the different programming languages, each with its own syntax, keywords, and grammar, just like human languages. There are also libraries that house the work of others that we can utilize. Front-end languages have to do with making the user experience pleasant and functional. Examples of these include HTML, CSS, JavaScript, and jQuery, while others are used more on the back-end, on the server to make the whole thing functional. Common languages used there include C#, Java, Python, PHP, Ruby, and SQL. As a practical matter, you'll need to know the file extensions that are used with each of those programming languages.

In order for any programming language to be understood by a computing device, it needs to be either interpreted or compiled. Interpreting is performed by browsers or, in the case of bytecode, by a runtime or virtual machine such as a JVM or a Python interpreter. Code that is interpreted is often called a script rather than a program. When it comes to speed, compiled programs normally run the quickest with the least overhead, then bytecode with scripts running more slowly but being the most versatile in terms of the platforms that a single script can be run on.

Programming paradigms, which is a fancy way of saying the approach that the programmer takes, include functional, object-oriented, and procedural approaches. Each has situations that make it the best or the worst way to program. We dipped our toes into the language of programming by discussing variables, constants, methods, attributes, and classes. And now you know the state of a program is simply a snapshot of how it exists at a given moment in time.

Finally, we looked at the importance of version control systems (VCSs), keeping files organized into logically separate folders (because there will be numerous files), and testing changes before they go live.

Exam Essentials

Know the languages. Be able to identify languages typically used for the front-end and back-end, including whether those languages are typically compiled or interpreted. Also know the file extensions for those languages. Being able to immediately identify the language of a file by its extension will save you time when you're out there working on a project designed by someone else.

Understand libraries and APIs. Know what a library is and that they can be a part of the language or a package that someone else has developed. APIs are similar because they are prewritten and save a programmer time, but the difference is that libraries are used for a specific program, whereas APIs facilitate communication between two different systems. It would be worth your time to explore what libraries and APIs are available for a project. Knowing where to look, and looking before you re-create the wheel, will save you countless hours when writing a program.

Compare/contrast compilers and interpreters. Know the advantages and disadvantages of scripts, bytecode, and programs, and the methods of creating them (interpreting and compiling).

Identify programming paradigms. Be able to look at a section of code and determine the paradigm used to create it. Know how to discern if it's an OOP, procedural, or functional paradigm.

Explain VCS, file organization, and software testing. These three concepts are tenets of web development. They'll save you time and frustration, and maybe even your job.

EXERCISE 1.1

Hello World

Hello World is the first program that most people learn to write. It's a tradition in the programming world, so we will do the same. If you're new to programming, we recommend you use a lab computer that's set aside for this purpose, not the computer you use for your daily work.

We'll use Ubuntu 22.04.1 throughout the book. If you have a different Linux distribution, it might work slightly differently.

First, let's use Vim to write a simple program in C++. If Vim is not installed, the following directions will guide you through that process.

1. Open Terminal.
2. Type **`vim hello.cpp`** and press Enter. We use `.cpp` because that is the extension for a C++ file.
   - If you get an error message that states `Command 'vim' not found...`, then you will need to install it. In that case, type **`apt install vim`** and press Enter.
   - The package will begin to install and tell you how much space will be used. Type **`Y`** and press Enter to continue, assuming you have the available space on your drive. After a minute or two, you should be back at the root.
   - Once Vim is installed, type **`vim hello.cpp`** and press Enter.
3. You should now be in the Vim text editor. Alternately, you can launch the Vim editor by clicking Show Applications at the lower-left corner of the desktop (it's the grid of 9 blocks) and clicking the Vim icon. Pressing F1 will show you help in Vim. Typing **`:q`** and then pressing Enter will take you back to the Vim Terminal from help. (You'll need to press i to enter Insert mode, where `--INSERT--` shows in the lower-left corner.)
4. In the Vim terminal, enter the following lines of code, pressing Enter after each line. Extra line returns are added for ease in reading. Pay attention to capitalization and the different styles of brackets and where they are used.

```
#include <iostream>
using namespace std;

int main() {

        string myvar[]="Hello World!";

        cout << myvar;

        cout <<endl;

        cout <<"This is my first C++ program.";

        return 0;
        }
```

- The first line loads the `iostream` (input/output) library.
- The second line tells the program to use the identifiers from the standard library (as opposed to identifiers from other libraries).
- `int main ()` will be looking for an integer at the end of the program. The last line of the program, `return 0;`, is that integer. Using 0 is the accepted way to say the program has executed successfully. This is the program's exit code.
- The next line creates a `string` variable named `myvar` whose content is "Hello World!"
- `cout << myvar;` tells the program to output that variable to the output device (the monitor), and `cout <<endl;` tells the program to insert a line break.
- `cout <<"This is my first C++ program.";` outputs that line to the monitor.

5. To save the file and exit the Vim editor, first press the Escape key to exit Insert mode. Then type **`:wq hello.cpp`** and press Enter. This will save your file with the specified filename and exit Vim. You should be back at a regular terminal.
6. The file should be stored in your user's Home folder. If you need to reopen it, click the folder icon on your desktop and double-click the filename. Remember that if you saved it under the root user, you will only be able to save an edited version if you're in the terminal as root.

EXERCISE 1.1 *(continued)*

7. Now we need to install the compiler. Open Terminal as superuser and enter the following lines of code, pressing Enter after each line:

```
apt install build-essential
g++ --version
gcc (Ubuntu 9.2.1-17ubuntu1) 9.2.1 20191102
```

8. When the compiler installation is done, it's time to compile the program you wrote. Enter the following line of code:

```
g++ -o hello hello.cpp
```

   The first `hello` in the line is the executable name. The second is the name of the C++ file that we're compiling.

9. To run the program, type **`./hello`** and press Enter.

Review Questions

1. What is another name for the front-end of a website?
   - **A.** HTML-side
   - **B.** Client-side
   - **C.** Server-side
   - **D.** JavaScript-side

2. What can be described as lines of instruction for a processor to follow?
   - **A.** Software
   - **B.** Data structures
   - **C.** Pseudocode
   - **D.** Back-end development

3. A new programmer writes the following. What is it an example of?

   Ask the user to input their first name.
   When they click on accept, say hello "firstname."
   Show button for continue and return to first page.

   - **A.** Source code
   - **B.** Programming language
   - **C.** Software
   - **D.** Pseudocode

4. Which of the following are instructions written in a specific programming language?
   - **A.** Source code
   - **B.** Programming language
   - **C.** Software
   - **D.** Pseudocode

5. What is the following an example of?

```
Float Vin:
Vin=analogRead(A1);
Float voltage;
Voltage=(Vin*5)/1024;
```

   - **A.** Source code
   - **B.** Programming language
   - **C.** Software
   - **D.** Pseudocode

6. Which of the following are languages that are typically used in programming the front-end of a website? (Choose two.)
   - **A.** PHP
   - **B.** CSS
   - **C.** Ruby
   - **D.** JavaScript

7. What term is used to describe the placement of commands, options, special characters, and so on used in a line of code for a particular language?
   - **A.** Keyword
   - **B.** Program
   - **C.** Syntax
   - **D.** Front-end

8. A programmer wants to control how many times a section of code is run. Which of the following keywords might they use? (Choose two.)
   - **A.** `and`
   - **B.** `for`
   - **C.** `while`
   - **D.** `times`

9. Your friend is a beginning programmer who needs to write some code to use with a sensor. What will you suggest they do?
   - **A.** Look for a keyword to use.
   - **B.** Look for pseudocode to use.
   - **C.** Look for an existing library.
   - **D.** Look for an existing script.

10. You are working on a project, fixing source code someone else wrote that is not working as expected. You're working in HTML, and suddenly come across some code in JavaScript. What will you do?
    - **A.** Delete the code.
    - **B.** Determine if the code is correct.
    - **C.** Replace the code with HTML.
    - **D.** Replace the code with a library.

11. Which of the following refers to code that is written in a language which a computing device can directly understand? (Choose two.)
    - **A.** Machine language
    - **B.** Binary
    - **C.** Source code
    - **D.** Pseudocode

12. What are Notepad, TextEdit, and Vim examples of?

A. Interpreters

B. Keywords

C. Compilers

D. Text editors

13. A friend has asked you to take a look at some code they wrote and provide any suggestions to make it better. The filename is `funcode.cpp`. What programming language are they using?

A. C

B. C++

C. Python

D. Perl

14. Which of the following keywords are operators? (Choose two.)

A. `and`

B. `for`

C. `not`

D. `if`

15. Which of the following can be the output of a compiler? (Choose two.)

A. Source code

B. Pseudocode

C. Bytecode

D. Machine language

16. You need to write a program that will execute instructions in order but at times call procedures that will be used repeatedly to calculate a value, then return to the string of commands and continue following them in order. What programming paradigm will you be using?

A. Functional

B. Object-oriented

C. Procedural

D. Reactive

17. What is true of the procedural programming paradigm?

A. Coding can take less time because you don't need to rewrite code that is repeated.

B. The focus of the paradigm is the state of the data.

C. Data is grouped into classes that have attributes and methods.

D. The purpose of a function is only to return a value, not modify the state of the program.

18. What type of software is used to keep track of program changes so mistakes can be rolled back if necessary?

A. AMD

B. Git

C. OOP

D. VCS

19. Where does a VCS hold files of current and prior program code?

A. Code bank

B. Repository

C. Folder

D. Paradigm

20. What type of programming paradigm is being used in the following example where `Void printcount ()` declares a procedure and the code under `Int main()` causes the code under `printcount` to run?

```
Void printcount()
{ for (int x=0; x<5; x++)
  Cout <<"Hi";
Cout << endl;
}
Int main()
{ printcount();
}
```

A. Functional

B. Object-oriented

C. Procedural

D. Reactive