

Introduction to Professional Web Development in JavaScript

- [Summary](#)
- [Expanded](#)

Chapters

1. [1. Introduction](#)
2. [2. How Programs Work](#)
3. [3. How To Write Code](#)
4. [4. Data and Variables](#)
5. [5. Making Decisions With Conditionals](#)
6. [6. Errors and Debugging](#)
7. [7. Stringing Characters Together](#)
8. [8. Arrays Keep Things in Order](#)
9. [9. Repeating With Loops](#)
10. [10. Functions Are at Your Beck and Call](#)
11. [11. More on Functions](#)
12. [12. Objects and the Math Object](#)
13. [13. Modules](#)
14. [14. Unit Testing](#)
15. [15. Scope](#)
16. [16. More on Types](#)
17. [17. Exceptions](#)
18. [18. Classes](#)
19. [19. Terminal](#)
20. [20. We Built the Internet on HTML](#)
21. [21. Styling the Web With CSS](#)
22. [22. Git More Collaboration](#)
23. [23. The DOM and Events](#)
24. [24. HTTP: The Postal Service of the Internet](#)
25. [25. User Input with Forms](#)
26. [26. JSON](#)

27. [27. Fetch](#)
28. [28. TypeScript](#)
29. [29. Angular, Part 1](#)
30. [30. Angular, Part 2](#)
31. [31. Angular, Part 3](#)
32. [32. Booster Rockets](#)

1. [Index](#)

Studios¶

1. [4.10. Studio: Data and Variables](#)
2. [5.7. Studio: Goal Setting and Getting into the Right Mindset](#)
3. [8.6. Studio: Strings and Arrays](#)
4. [9.10. Studio: Loops](#)
5. [10.11. Studio: Functions](#)
6. [11.11. Studio: More Functions](#)
7. [12.8. Studio: Objects & Math](#)
8. [13.7. Studio: Boosting Confidence](#)
9. [14.7. Studio: Unit Testing](#)
10. [17.5. Studio: Strategic Debugging](#)
11. [18.6. Studio: Classes](#)
12. [20.5. Studio: Making Headlines](#)
13. [22.7. Studio: Communication Log](#)
14. [23.8. Studio: The DOM and Events](#)
15. [25.11. Studio: HTTP and Forms](#)
16. [27.4. Studio: Fetch & JSON](#)
17. [28.8. Studio: TypeScript](#)
18. [29.8. Studio: Angular, Part 1](#)
19. [30.8. Studio: Angular, Part 2](#)
20. [31.7. Studio: Angular, Part 3](#)

Assignments¶

1. [Assignment #1: Candidate Testing](#)
2. [Assignment #2: Scrabble Scorer](#)
3. [Assignment #3: Mars Rover](#)
4. [Assignment #4: HTML Me Something](#)
5. [Assignment #5: Launch Checklist Form](#)
6. [Assignment #6: Orbit Report](#)

Appendices¶

1. [About This Book](#)
2. [Style Guide](#)
3. [Git Workflows](#)
4. [Git Stash](#)
5. [Organizing Your Repositories](#)
6. [Tested Code](#)
7. [Array Method Examples](#)

8. [DOM Method Examples](#)
9. [String Method Examples](#)
10. [Math Method Examples](#)
11. [Terminal Commands](#)
12. [Setting up Software for the Class](#)
13. [Exercise Solutions](#)
14. [Feedback](#)



1. [Contents](#)
2. 1. Introduction

1. Introduction¶

1. [1.1. Why Learn To Code?](#)
2. [1.2. Why Learn JavaScript?](#)
3. [1.3. About LaunchCode Programs](#)
 1. [1.3.1. Goals](#)
 2. [1.3.2. Course Activities](#)
 1. [1.3.2.1. Textbook Reading](#)
 2. [1.3.2.2. Exercises](#)
 3. [1.3.2.3. Graded Assignments](#)
4. [1.4. Blended Learning](#)
 1. [1.4.1. In-Class Time](#)
 1. [1.4.1.1. Large Group Time](#)
 2. [1.4.1.2. Small Group Time](#)
5. [1.5. Class Platforms](#)
 1. [1.5.1. Canvas](#)
 1. [1.5.1.1. Login to Canvas](#)
 2. [1.5.1.2. Canvas Dashboard](#)
 3. [1.5.1.3. Syllabus Page](#)
 4. [1.5.1.4. Assignments Page](#)
 2. [1.5.2. Repl.it](#)
 1. [1.5.2.1. Repl.it Account Creation](#)
 2. [1.5.2.2. Online Code Editor](#)
 3. [1.5.3. GitHub Classroom](#)

6. [1.6. Using This Book](#)
 1. [1.6.1. Concept Checks](#)
 2. [1.6.2. Examples](#)
 1. [1.6.2.1. Repl.it](#)
 3. [1.6.3. Supplemental Content](#)
 4. [1.6.4. JavaScript in Context](#)
- [← Chapters](#)
- [1.1. Why Learn To Code? →](#)



1. [Contents](#)
2. [1. Introduction](#)
3. 1.1. Why Learn To Code?

1.1. Why Learn To Code? ¶

How many times do you use a computer in a day? What do you use it for? Maybe you use one to check email and social media, to watch TV, and to even set an alarm for the next day. Computers and technology are *everywhere* in our society.

With the rise of technology and computers, coding has risen as well. At its most basic level, coding is how humans communicate with computers. With code, humans tell computers to complete specific tasks and store specific information. Many would argue that due to the prevalence of computers, learning to code is vital to living in the 21st century. Writing and reading code is becoming a new form of literacy for today's world.

As our needs change, technology changes to meet them. When we needed a way to talk to each other over long distances, we got phones. As our needs to communicate changed, our phones became portable. Then, we gained the ability to use our phones to send quick written messages to each other.

The technical skills required to make the phones of 20 years ago are not the same skills required to make a phone today. A career as a technologist, specifically as a programmer, is one of lifelong learning.

Learning to code is not only valuable and challenging, it is also fun. Every *EUREKA!* moment inspires us to keep going forward and to learn new things. You may find some concepts difficult to understand at first, but these will also be the skills you take the most pride in mastering. While the journey to learning to code is long and winding, it is also rewarding.

From the moment that you write your first line of code, you are a programmer. We hope you enjoy the flight!

- [← 1. Introduction](#)
- [1.2. Why Learn JavaScript? →](#)



- 1. [Contents](#)
- 2. [1. Introduction](#)
- 3. 1.2. Why Learn JavaScript?

1.2. Why Learn JavaScript? ¶

With all the different coding languages in the world, it can be difficult to choose which one to learn first. **JavaScript** is a programming language that has many different applications. Programmers can use JavaScript to make websites, visualize data, and even create art! This class will start with JavaScript for several reasons.

The main reason is that JavaScript has become a key language for web development. Understanding how the internet works and being able to create web applications are important skills in today's landscape. With JavaScript, you can work on your own web applications and support current applications for a company.

Another reason is that JavaScript is very much like other programming languages. Throughout the course and your career, you will hear that once you learn one programming language, it is easier to learn another. Programmers have found this to be true, especially if the new language closely matches the first.

- [← 1.1. Why Learn To Code?](#)

- [1.3. About LaunchCode Programs →](#)



- 1. [Contents](#)
- 2. [1. Introduction](#)
- 3. 1.3. About LaunchCode Programs

1.3. About LaunchCode Programs¶

1.3.1. Goals¶

We want our programs to help you build your problem solving skills and encourage you to learn how to learn. Whether you use the coding skills you gain in this program to get a job as a developer is up to you. However, no matter the path you take after this program, learning how to learn will help you continually adapt to the changing needs of your industry.

To get you ready for a career in technology, our goal is to teach you the skills found in a wide variety of industries.

1.3.2. Course Activities¶

We have created the course activities to make the most of your time. It is important to actively engage with each activity to maximize your learning potential. Skipping the textbook reading or falling behind on assignments can quickly lead to struggling to complete the course.

1.3.2.1. Textbook Reading¶

Think of this textbook as your first destination in your learning journey. In addition to reading, this text includes small questions that can help you reinforce your understanding of the new material. Reviewing your notes from a given chapter before moving on to the next is another great way to make the most of your learning potential.

1.3.2.2. Exercises¶

At the end of most of the textbook chapters, you will find a page of *Exercises*. These are small coding problems and are a chance for you to implement what you have just learned. While exercises do not count towards your final grade in the class, it is essential to practice in order to reinforce your understanding of the new concepts.

1.3.2.3. Graded Assignments¶

Graded assignments are larger projects where you demonstrate what you have learned and challenge yourself. Assignments oftentimes cover multiple lessons.

- [← 1.2. Why Learn JavaScript?](#)
- [1.4. Blended Learning →](#)



-

1. [Contents](#)
2. [1. Introduction](#)
3. 1.4. Blended Learning

1.4. Blended Learning¶

This page covers what to expect during in-class time for students enrolled in a LaunchCode blended learning course. Students taking an independent learning course may skip this page.

We only have a short amount of time in class to learn a lot, so using a **blended learning model** helps us make the most of our time in this course. A blended learning model incorporates in-class learning with online materials like this textbook.

1.4.1. In-Class Time¶

In class, you join fellow students on the same learning journey as you. We encourage students to engage, interact, and encourage each other throughout the class.

In-class time is run by an instructor and teaching assistants. This dedicated staff facilitates the activities and provides support to the students.

1.4.1.1. Large Group Time¶

During the large group time, the whole class participates in the lesson, led by the instructor. The lesson is not a substitute for doing the prep work before class. It's a time for us to review examples as a group and shore up concepts from the reading.

1.4.1.2. Small Group Time¶

After the large group time, we break up into small groups, each led by a teaching assistant. During small group time, we do in-class coding activities called studios. This is a time to ask for individual support if you need it. It is meant to be a place where you can feel comfortable talking openly about concepts you are struggling with.

- [← 1.3. About LaunchCode Programs](#)
- [1.5. Class Platforms →](#)



-

1. [Contents](#)
2. [1. Introduction](#)
3. 1.5. Class Platforms

1.5. Class Platforms¶

Besides this book, this class uses additional platforms for enrollment, assignments, and grading.

1.5.1. Canvas

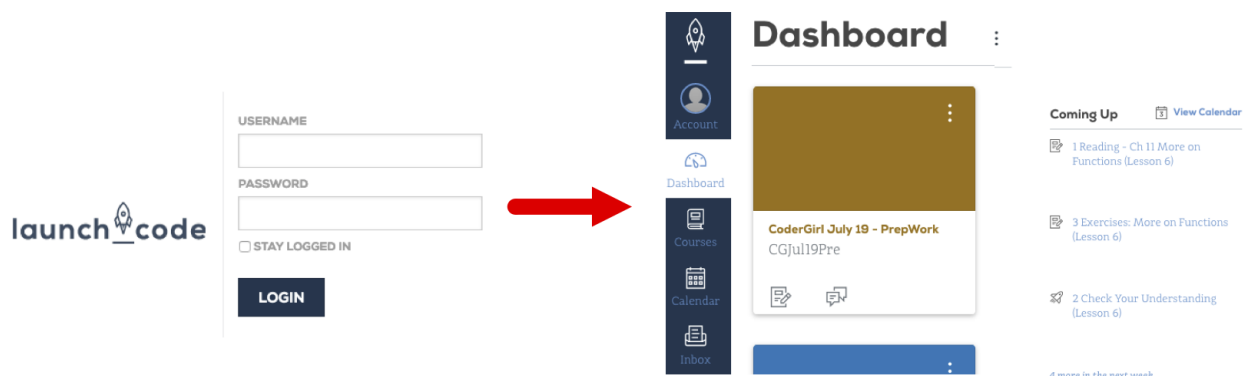
LaunchCode monitors your progress in this class through a management system called **Canvas**. It provides a central location to manage the flow of information, but it does not hold the actual course content. Instead, it links to the lessons you need, and it keeps a record of your completed assignments and scores.

1.5.1.1. Login to Canvas

Access Canvas and the course materials at <https://launchcode.instructure.com/>. To login, use your launchcode.org username and password, which are the same ones you used to apply for this class.

1.5.1.2. Canvas Dashboard

After logging in, you will arrive at your **dashboard**. Your Canvas dashboard displays the LaunchCode courses you can access, upcoming due dates, and several menu items.



Clicking on a course title takes you to that class' homepage. This page shows upcoming due dates, announcements, general information, and menu options. You will probably use the *Syllabus* and *Assignments* options the most often.

Home

Assignments



Discussions

Grades

People

Pages

Files

Syllabus



Quizzes






Modules

1.5.1.3. Syllabus Page

The syllabus page provides general information, such as a description of the class, the timeline for the course, a calendar, and a todo list. Scrolling down on the page shows the *Course Summary*, which holds links to individual tasks (reading, quizzes, assignments, etc.).

This page is a good place to answer the questions "What do I need to do next?" and "How can I quickly find and review an old topic?".

Course Summary:

Date	Details
Sat Jun 15, 2019	 Prep Work Quiz due by 11:59pm
	 Prep Work Reading due by 11:59pm
Mon Jun 17, 2019	 1 Reading: Data and Variables (Lesson 1) due by 1pm
	 2 Exercises: Data and Variables (Lesson 1) due by 1pm
	 3 Studio: Data and Variables (Lesson 1) due by 3pm

1.5.1.4. Assignments Page

This page sorts required tasks by date or type. We regularly add new tasks to this list so check back here often. Old content remains active, allowing you to use the links for reference and review.


SHOW BY DATE


SHOW BY TYPE

▸ Assignments

▸ Prep Work

▾ Lesson 1

 **1 Reading: Data and Variables (Lesson 1)**
Due Jun 17 at 1pm

 **2 Exercises: Data and Variables (Lesson 1)**
Due Jun 17 at 1pm

Clicking on a specific title brings up information about that task, including the due date, points possible, instructions, and links.

1 Reading - Ch 7 Stringing Characters Together (Lesson 3)

Due Jun 24 by 12pm Points None

Please read [Chapter 7 - Stringing Characters Together](#)

2 Check Your Understanding - Ch 7 (Lesson 3)

Due Jun 24 at 12pm Points 7 Questions 7
Time Limit None Allowed Attempts Unlimited

Instructions

Please answer these questions after completing: [1 Reading - Ch 7 Stringing Characters Together \(Lesson 3\)](#)

Take the Quiz

Even though much of the course content can be accessed without logging in, the best choice is to begin your course work from within Canvas. That way your progress gets recorded and your scores will update smoothly as you complete quizzes. Also, submitting files for the larger assignments should only be done through Canvas.

1.5.2. Repl.it

[Repl.it](#) is a free online code editor, and it provides a practice space to boost your programming skills.

For this class, Repl.it provides opportunities to respond to prompts, questions, "Try It" exercises, and studios embedded within the reading. These tasks are neither tracked nor scored.

1.5.2.1. Repl.it Account Creation¶

Creating a Repl.it account is covered in [Chapter 2](#). You'll need to create an account to use Repl.it.

1.5.2.2. Online Code Editor¶

Repl.it is an online code editor for various languages. Coders collaborate by sharing Repl.it URLs.

Repl.it is used for:

1. Publicly sharing code examples and starter code
2. A place to practice new concepts by writing and running code

Tip

You never have to click save when using Repl.it. Repl.it automatically saves your code on their servers.

1.5.3. GitHub Classroom¶

GitHub Classroom provides online code storage. For this class, it also allows for graded assignment submission and evaluation.

GitHub is a web application that uses *version control*. We'll learn more about GitHub and what version control is in a future lesson.

Note

Results from work submitted in GitHub classroom, appear in Canvas after being verified.

Remember, Canvas holds student grades and quizzes but NOT the course content. Instead, it provides *links* to the reading and other assignments.

- [← 1.4. Blended Learning](#)
- [1.6. Using This Book →](#)

1. [Contents](#)
2. [1. Introduction](#)
3. 1.6. Using This Book

1.6. Using This Book¶

Throughout this book, you will find a variety of different sections and practice exercises. We are writing this guide to help you make the most of the book.

1.6.1. Concept Checks¶

Many pages end with a "Check Your Understanding" header. This section is full of questions for you to double check that you understand the concepts in the reading. Although your score does not count towards your final grade in the class, you should use it to help evaluate your understanding of the main concepts.

1.6.2. Examples¶

Examples are times when we tie a concept we have just learned to a potential real world application.

The label "Try It" signals an example that includes code you can modify and augment to quickly reinforce what you have just read. Play around with these!

1.6.2.1. Repl.it¶

As we mention on the previous page, we expect you to use your **Repl.it** account to practice your programming skills.

As you explore the prepared examples in this book, feel free to make changes to the code. If you want to save your edits, click the *Fork* button at

the top of the workspace, and Repl.it will store a copy of the code in your personal account.

1.6.3. Supplemental Content¶

Occasionally, you will find a link to "Booster Rockets". While not required reading, "Booster Rockets" can boost your learning.

1.6.4. JavaScript in Context¶

Our approach is different from other ways you can learn JavaScript. The book focuses on programming fundamentals. These fundamentals are problem-solving and transferable concepts. While we will cover the exact way to perform certain tasks in JavaScript, we want to remind you that these tasks are relatively common and many programming languages have ways to carry them out.

- [← 1.5. Class Platforms](#)
- [2. How Programs Work →](#)



- 1. [Contents](#)
- 2. 2. How Programs Work

2. How Programs Work¶

- 1. [2.1. Introduction](#)
 - 1. [2.1.1. Algorithms](#)
 - 2. [2.1.2. Check Your Understanding](#)
- 2. [2.2. Programming Languages](#)
 - 1. [2.2.1. Languages](#)
 - 1. [2.2.1.1. How Computers Run Programs](#)
 - 2. [2.2.2. How Many Programming Languages Are There?](#)
- 3. [2.3. The JavaScript Language](#)
 - 1. [2.3.1. Front End vs. Back End Changes](#)

4. [2.4. Your First Program](#)
 1. [2.4.1. Create a Replit Account](#)
 1. [2.4.1.1. Creating a New Repl](#)
 2. [2.4.1.2. The Replit Workspace](#)
 2. [2.4.2. Begin Your Coding Journey](#)
 1. [2.4.2.1. Working with a Prepared Repl](#)
 2. [2.4.2.2. Now Play](#)
 3. [2.4.3. Check Your Understanding](#)
- [← 1.6. Using This Book](#)
- [2.1. Introduction →](#)



1. [Contents](#)
2. [2. How Programs Work](#)
3. 2.1. Introduction

2.1. Introduction¶

"It'll take a few moments to get the coordinates from the navicomputer." - Han Solo

Given a set of inputs, Han's computer analyzes the data and returns information about safely navigating a hyperspace jump. The computer does this by running a **program**.

At the most basic level, a *program* is a set of instructions that tell a computer or other machine what to do. These instructions consist of a set of commands, calculations, and manipulations that achieve a specific result. However, the computer cannot solve the problem on its own. Someone---a programmer---had to figure out a series of steps for the computer to follow. Also, the programmer had to write these steps in a way the computer can understand.

2.1.1. Algorithms¶

Imagine following a recipe for baking a batch of cookies. After the list of ingredients comes a series of step-by-step instructions for producing the treats. If you want to make something else, like a cake or a roast, you follow a different set of steps using a different set of ingredients.

An **algorithm** is like a recipe. It is a systematic series of steps that, when followed, produce a specific result to help solve a problem. Programmers design algorithms to solve these small steps in a carefully planned way. The results then get combined to produce a final answer or action.

Let's take a look at an example of an algorithm---alphabetizing a list of words:

apple, pear, zebra, box, rutabaga, fox, banana, socks, foot

One possible set of steps for solving the task could be:

1. Arrange the words from a - z based only on the first letter:

apple, box, banana, fox, foot, pear, rutabaga, socks, zebra

2. If more than one word starts with 'a', rearrange those words based on the second letter. Repeat for the words that start with 'b', then 'c', etc.:

apple, banana, box, fox, foot, pear, rutabaga, socks, zebra

3. If multiple words start with 'a' and have the same second letter, rearrange those words based on the third letter. Repeat for the 'b' words, then the 'c' words, etc.:

apple, banana, box, foot, fox, pear, rutabaga, socks, zebra

4. If other repeats exist, continue sorting the list by comparing the 4th, 5th, 6th letters (etc.) until all the words are properly arranged.

This is not the ONLY way to solve the task, but it provides a series of steps that can be used in many different situations to organize different lists of words.

Alphabetizing is a process we can teach a computer to do, and the algorithm will complete the process much more rapidly than a human. However, unlike the alphabet song that many of us still sing in our heads when arranging a list of words, programmers must use a different method to train the computer.

2.1.2. Check Your Understanding¶

Question

Select ALL of the following that can be solved by using an algorithm:

1. Answering a math problem.
2. Sorting numbers in decreasing order.
3. Making a peanut butter and jelly sandwich.
4. Assigning guests to tables at a wedding reception.
5. Creating a grocery list.
6. Suggesting new music for a playlist.
7. Making cars self-driving.

- [← 2. How Programs Work](#)
- [2.2. Programming Languages →](#)



1. [Contents](#)
2. [2. How Programs Work](#)
3. 2.2. Programming Languages

2.2. Programming Languages

"Computer, scan the surface for lifeforms."

"Hey Siri, what movies are playing nearby?"

Even though today's tech makes it seem like computers understand spoken language, the devices do not use English, Chinese, Spanish, etc. to carry out their jobs. Instead, programmers must write their instructions in a form that computers understand.

Computers operate using **binary code**, which consists only of 0s and 1s. For example, here is the binary version of the text Hello World:

```
01001000 01100101 01101100 01101100 01101111 00100000 01010111
01101111 01110010 01101100 01100100
```

Each set of 8 digits represents one character in the text.

To make things a little easier, binary data may also be represented as **hexadecimal** values. Here is Hello World expressed in *hex*:

48 65 6c 6c 6f 20 57 6f 72 6c 64

To run an algorithm, all of the steps must be written in binary or hex so the computer can understand the instructions.

Note

Fortunately, we do not need to worry about binary or hexadecimal code to make our programs work!

2.2.1. Languages¶

Writing code using only 0s and 1s would be impractical, so many clever individuals designed ways to convert between the text readable by humans and the binary or hexadecimal forms needed by machines.

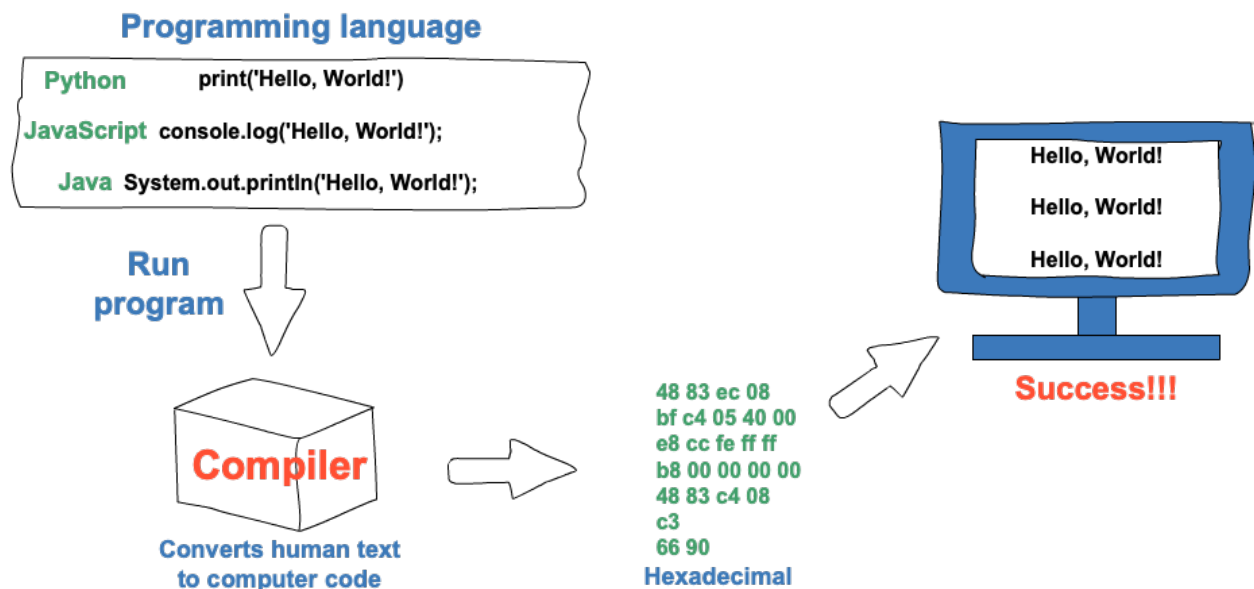
A **programming language** is a set of specific words and rules for teaching a computer how to perform a desired task. Examples of programming languages include Python, JavaScript, Basic, COBOL, C++, C#, Java, and many others.

These *high-level languages* can be written and understood by humans, and each one has its own characteristic vocabulary, style, and syntax.

2.2.1.1. How Computers Run Programs¶

Since computers only understand binary code, every programming language includes a **compiler**, which is a special tool that translates a programmer's work into the 0s and 1s that the machines need.

If we want to print Hello, World! on the screen, we would write the instructions in our chosen programming language, then select "Run". Our code gets sent to the compiler, which converts our typed commands into something the computer can use. The instructions are then executed by the machine, and we observe the results.



In the example above, the *syntax* for printing Hello, World! varies between the Python, JavaScript, and Java languages, but the end result is the same.

2.2.2. How Many Programming Languages Are There?¶

Ask Google, "How many programming languages are there?" and many results get returned. Even with all these options, there is no specific answer to the question.

There are hundreds, if not thousands, of programming languages available. However, most are either obsolete, impractical, or too specialized to be widely used.

Arguments occur whenever someone makes a top 10 list for programming languages, but regardless of the opinions, one fact remains. Once you learn one language, learning the next is much, much easier. Adding a third becomes child's play.

The reason for this is that thinking like a coder does not change with the language. Your logic, reasoning, and problem solving skills apply just as well for JavaScript as they do for Python, Swift and C#. To display text on the screen in Python, we use `print()`, for JavaScript we use `console.log()`, for C# the command is `Console.WriteLine()`. The *syntax* for each language varies, but the results are identical.

- [← 2.1. Introduction](#)
- [2.3. The JavaScript Language →](#)