How To Learn Any New Programming Language Quickly

An essential checklist of fundamentals

Bob Roebling

Nov 3, 2019 · 5 min read ★

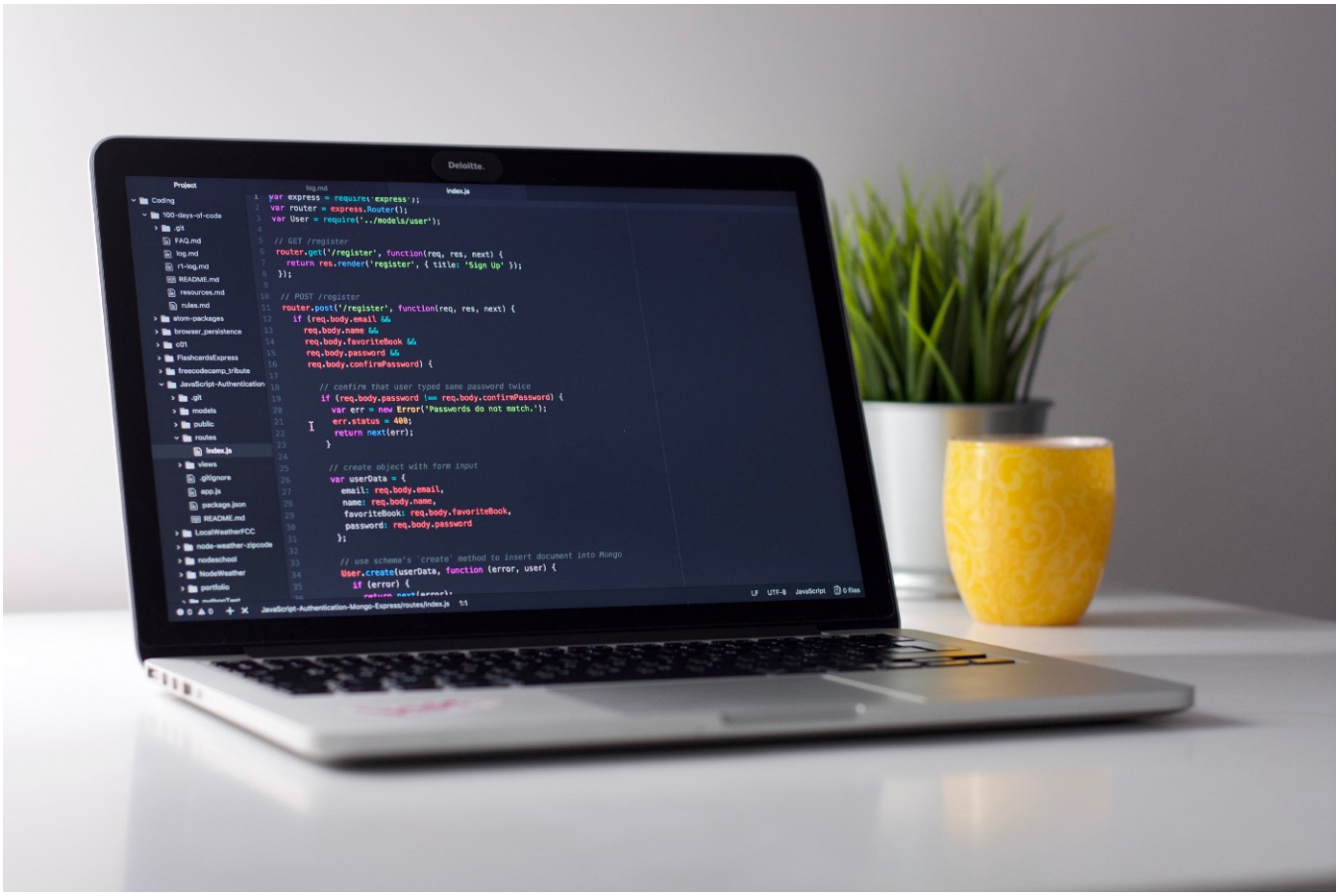


Photo by Clément H on Unsplash

This article assumes you already know at least one programming language; however, the concepts here will help you get started with programming.

When I was in school, a teacher told me something I share with new developers: **The hardest programming language you'll ever learn will be your second.**

Don't let this be disheartening — it means that when you first learn how to program, you have all of these preconceived ideas about programming. You end up making more syntax connections and assumptions than you should. Because of this, you have to "unlearn" these assumptions when you learn your second language. Try to keep this in mind if you're attempting to learn your second language or even your third.

. . .

Anatomy of Programming

There are a lot of programming languages, easily over 5,000, but the TIOBE index lists the top 250. All of the top 20 languages that aren't unique cases will have similar standard libraries.

I think the best way to think about programming is to strip out all of the extra "stuff" so you're left with only the necessities.

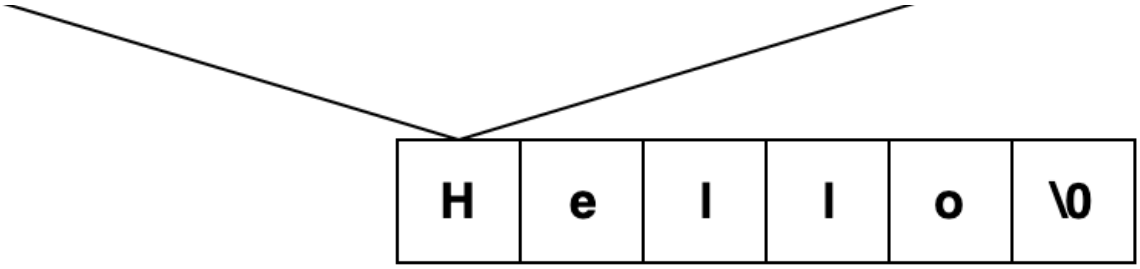
. . .

The Atoms

Every aspect of every language can be reduced down to **true** and **false**. Why? Because electricity works this way — you either have a charge, or you don't. Memory stores values in the form of **0**s and **1**s, and either this *bit* is charged, or it's not.

Eight bits is equivalent to one *byte*, which is enough to list any character on the ASCII table. The bits are flipped in such an order that it provides the **DEC**imal representation of the character. The computer understands how to translate this representation into a letter.

Position	8	7	6	5	4	3	2	1
Value	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
Bit	0	**1**	0	0	**1**	0	0	0



Basic binary representation showing how the word Hello is created.

Having an understanding of this one concept, “the why,” will make the rest of this a lot easier on you.

. . .

The Tools

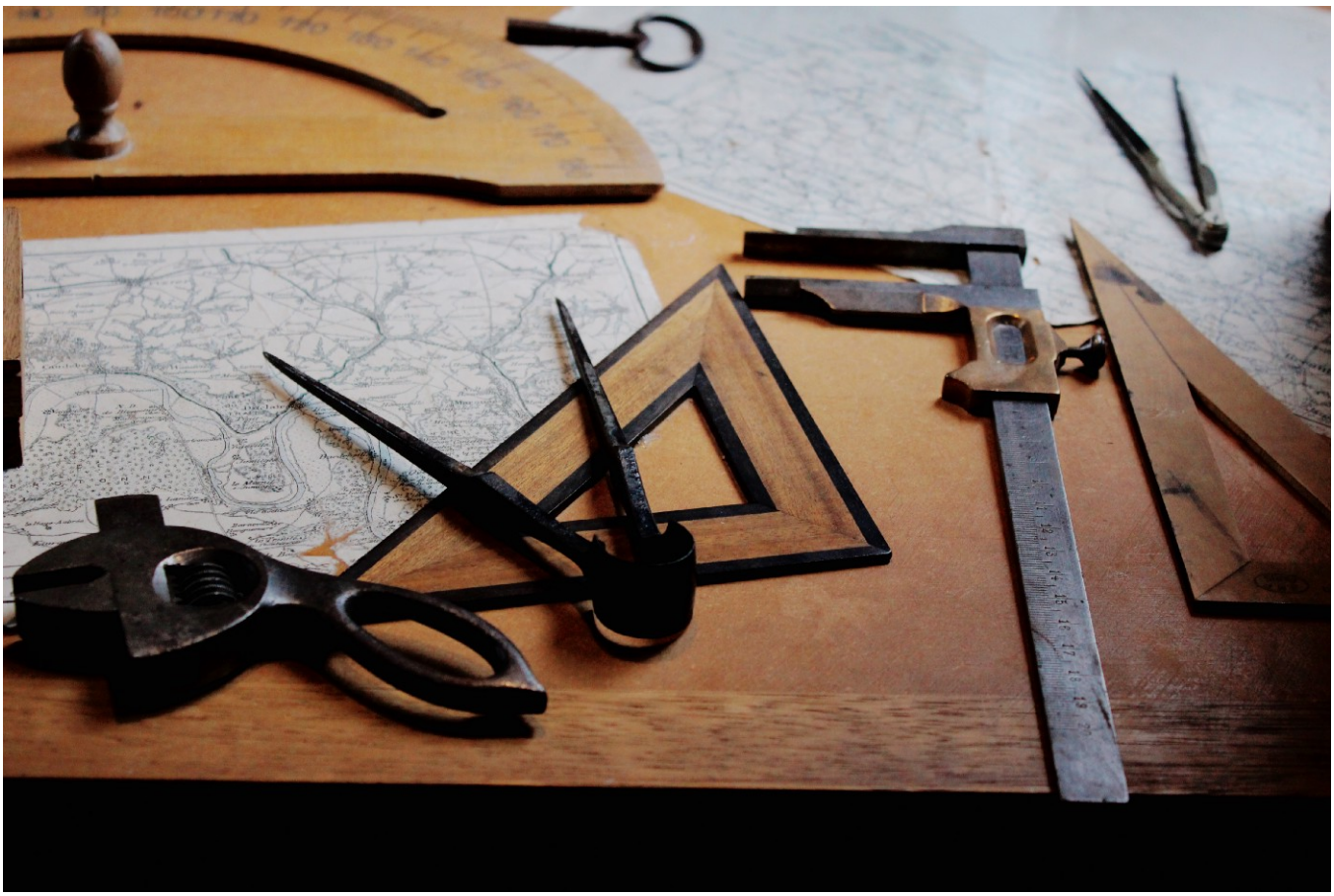


Photo by Fleur on Unsplash

The tools are all the same, and while they can be learned in any order, this is the order I usually take.

Variables

This seems simple enough, but seriously, how do you create a variable?

Operators

What are the operators, and how are they used? You can assume you have basic math operators, but what about logical operators? Is an “AND” operator spelled out as “and” or “AND,” or does it use symbols such as “&&?”

Conditionals

Surprisingly, my most read articles for both Swift and Python have to do with decision making. The next thing you need to know is how you can make decisions in your program. Does the language you are trying to learn use the traditional “if/else if/else” or something more Pythonic such as “if/elif/else?” Does your language have a “switch” or “guard” statement?

Loops

How can you loop through repetitive tasks? Does the language contain for-loops, while loops, do-while loops, or for-each statements?

Functions

Is it possible to create functions? If so, how do you do it? How do you include parameters in these functions? Knowing how to properly use functions will save you time and make your life so much easier.

Classes and structs

Does this language understand the concept of classes or structs? It sounds like a dumb question, but some languages don’t have either, or they have only one. If it does, how do you create a class or struct? Does the class require a constructor or init method?

Error handling

Errors are inevitable. When they occur, does this language have a robust error handling solution and how do you use it? Is it “try/catch,” “try/except,” or something else? Are there other clauses such as “else” or “finally” that allow other options for errors?

Testing

How do you test your code? Is there a built-in library for testing or do you have to download a separate tool?

All of these tools should be in most modern programming languages. Even the older languages such as COBOL have most of these, but they may be called something different, like paragraphs or copybooks.

Photo by Darius Soodmand on Unsplash

. . .

Getting Good

Once you understand these tools, the next thing you need to do is use them and write an application. You can know of a language by reading the docs, but you don't know the language until you've written a few applications with it.

By writing an application, you're forced to think like an *X* programmer. I can say that I know C++ because I took a class in C and read the docs on C++, but I really don't know it until I've written an application in C++ using features specific to the language.

A good starter project is Blackjack. Blackjack requires variables, operators, conditionals, loops (based on the number of players), functions, classes/structs, and error handling. You can include test cases for potential failures, such as running out of cards.

Other good starter projects might include Chutes and Ladders, Yahtzee, or a Slot Machine.

For something more advanced, try to recreate a game like Monopoly. Worry more about the mechanics and keep it text-based.

It's key to remember that if you short yourself on the difficulty of the task (such as skipping the double down or split features of blackjack), you're only limiting your comprehension of the language.

. . .

What Else?

I know that the list of things above isn't everything that a language has to offer. The truth is you can write just about anything with the tools listed above, but the additional functionality included in standard libraries just make it easier. Most standard libraries include the same functions, and so you can rely on similar names between languages.

The more you work with a language, the more you can discover about the standard library, but be sure to learn the tools beforehand.

As you work with a language, try to figure out what its strengths and weaknesses are. These will help you understand which language to use for a particular problem.

Need to do some data science quickly? Look at a few Python packages or R. Need to write a fast service? Look at C or Go. How about a web server? Look at Java or Python.

I didn't just know this by looking at the languages. I learned this by *using* these languages.

Since this is possibly my shortest article to date, I'm going to leave you with a challenge to yourself to learn a new language. Good luck!