Program Languages and Computers

by Devmountain Tutorials

WHAT'S COVERED

This section will explore the basic concepts of programming languages and computers by discussing:

1. COMPUTERS
2. PROGRAMMING LANGUAGES
3. ALGORITHMS AND AUTOMATION
4. DAY IN THE LIFE OF A DEVELOPER

1. COMPUTERS

A graphical representation of the simplest computer. It consists of a long piece of paper tape with a 1-by-1 grid printed on it. Each cell stores the number 1 or 0. The computer is also able to keep track of its current location within the grid.

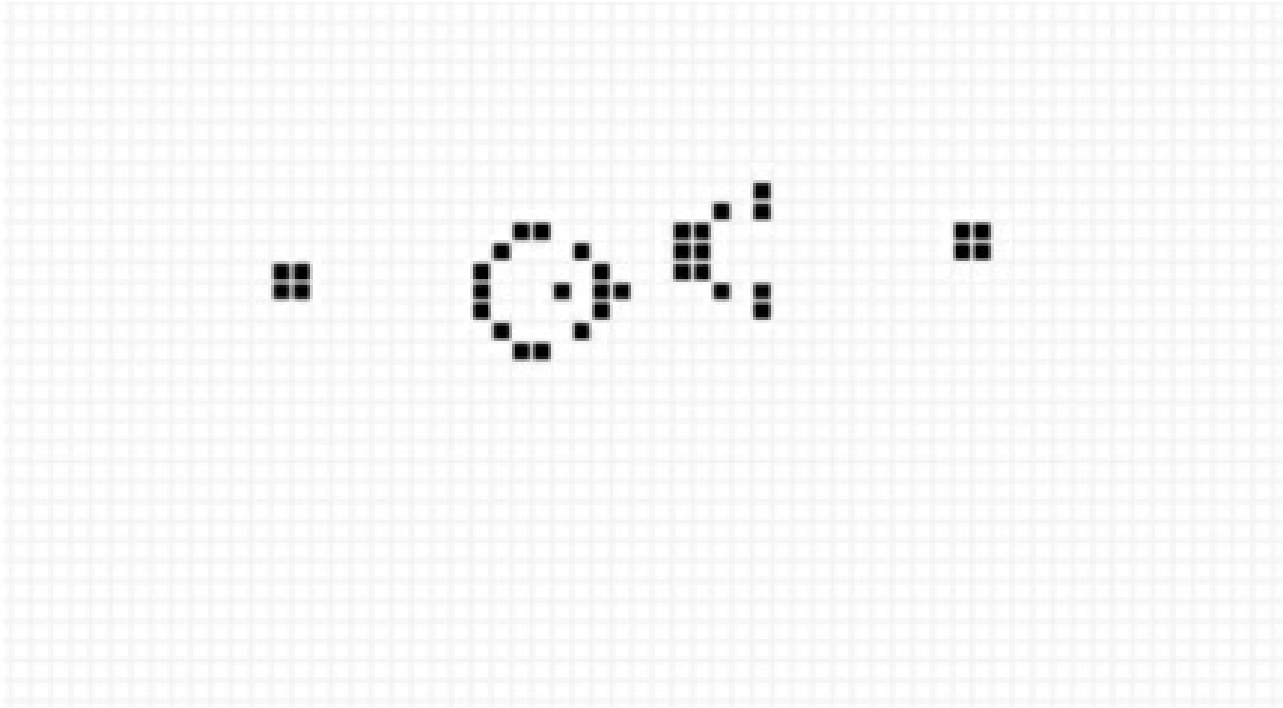
A computer doesn't need to have a wide screen or keyboard or even a latest processor to be considered a **computer**. In fact, most primitive machine that can still be considered a computer doesn't even need electricity! You can build one right now — all you need is a pencil and a long roll of paper tape with a grid printed on it.

In software engineering, a computer is anything that can do different things based on changing conditions ("if the sky is gray, grab an umbrella before you go outside") and the ability to store data ("Ada's phone number is 555-1235").

BIG IDEA

The idea that a machine must possess these qualities to be considered a computer is known as **Turing completeness**.

For example, Conway's Game of Life, a cellular automaton, is a computer.



A screenshot of Conway's Game of Life

Thankfully the computers that software engineers work with are slightly more complex. Still, even *those* computers aren't very smart. If you went to the grocery store with a friend and told them to "grab a box of pasta," they'd likely understand exactly what you want them to do. Computers don't understand English (at least, not the way humans are able to understand English) so if you gave the same command to a computer, it won't have any idea what you're talking about. That's why **programming languages** were invented. Developers use these languages to express their ideas in a format that computers can understand and execute.

TERM TO KNOW

Computer

A computer is anything that can perform different functions based on changing conditions and has the ability to store data.

Programming Language

A formal language designed to tell computers what to do.

2. PROGRAMMING LANGUAGES

Programming languages are **formal languages** — languages designed by people for specific, often technical, applications. For example, mathematicians use infix notation to express mathematical equations (like 2 + 2). Although there are other ways to notate math equations, infix notation is popular because it helps us visualize the relationship between numbers (2) and operators (+). In contrast, **natural languages**, like English, are languages that weren't designed by people. Instead, they evolved naturally over time.

Both types of languages have rules about valid syntax. These rules define legal **tokens** and dictate how those tokens should be arranged. Tokens are the basic building blocks of a language. The tokens of the English

language are words like red, are, and roses. The tokens used in math equations are numbers (2), symbols (+), and sometimes letters (n).

Tokens are meaningless until they're arranged in a meaningful way. For example, red are roses is a nonsensical phrase and 2 2+=4 is not a valid equation. Although both phrases are made of valid tokens, the tokens aren't arranged in a structure that makes sense. Once we rearrange the tokens roses are red and 2+2=4, you're finally able to figure out the structure of each phrase.

Although natural languages and formal languages are both used to express ideas, there are differences between the two that often make it hard for natural language speakers to adjust to formal languages.

The rules of natural languages are relatively loose compared to formal languages. Humans are able to handle ambiguous messages using contextual clues, body language, and other information. If someone writes the message, "I'll see you at jome" we have no problem assuming that the message is supposed to read, "I'll see you at home." We even communicate with words that aren't to be taken literally whenever we use sarcasm, irony, idioms, and metaphors.

Formal languages like programming languages are extremely strict. Unlike humans, computers can't handle ambiguity and only understand literal definitions. As a result, you can't go shopping with a computer and tell it to "grab a box of pasta." Instead, you'd have to give it very specific instructions that would look something like this:

- Go to aisle #3.
- Go to the shelf where they keep boxes of spaghetti.
- Take the first box in row 2.
- Bring the box to me.

The fact that a computer needs such explicit, unambiguous commands to execute a comparatively simple task might make you wonder why we rely on computers to do anything at all!

TERM TO KNOW

Formal Language

A language designed by people for specific, often technical, applications.

Natural Language

A language that evolved naturally over time.

Token

The basic building block of a language.

---

3. ALGORITHMS AND AUTOMATION

Computers can't make intelligent decisions like humans do but they can perform thousands of calculations in the blink of an eye. Also, computers don't need food or sleep so as long as there's electricity, they can continue chugging along almost indefinitely. That's why we use computers to automate laborious operations that would be exhausting (and boring) for a human to complete.

The solutions that engineers come up with are referred to as **algorithms**. An algorithm is a set of step-by-step instructions that describe how to arrive at the solution. Engineers must turn algorithms into **programs** by

translating their algorithm into code using a programming language. The programs are given to computers, who execute them exactly as they're written.

Later in the course, we'll talk about algorithms in more detail and explain how some famous algorithms work. For now, it's more important to remember that, although computers are capable of doing very complex tasks, they're actually simple machines. Without engineers to tell them what to do and how to do it, computers would be completely useless.

TERM TO KNOW

Algorithm

A set of step-by-step instructions that describe how to solve a problem.

Program

Programs are algorithms that computers understand. Engineers write programs using programming languages.

---

4. DAY IN THE LIFE OF A DEVELOPER

In theory, all the interesting things that web engineers do sounds great! But how does this look in practice? What does a web engineer's average day in the workplace look like?

Imagine you are a web engineer at a large company in Silicon Valley. You're a member of the team that works on one of the company's many products — a mobile application that's a personal finance tool. Your teammates are project managers, graphic designers, user experience researchers, and a handful of additional engineers like you.

You start your day by attending a **stand-up meeting** or **scrum meeting** — a daily meeting with your teammates. During this meeting, you listen as each member of the team talks about what they accomplished the day before and what they're planning on doing today. Your team also uses this meeting to collaborate with one another, coordinate task assignments, and update estimations on when a task should be completed. When one of your teammates describes getting stuck on a problem you're familiar with, you let her know that you've encountered a similar problem. She schedules a meeting with you to do this later in the day.

After the stand-up meeting, you check your calendar. The next thing you have to do is get on a video conference call with a customer to watch them use the new feature you've been working on and get feedback. The customer is mostly happy with how the feature works, but it doesn't work *quite right* with her overall workflow. You agree — her feedback makes sense. You tell the customer to stay tuned for updates and sign off. Now, the only thing bothering you is how you can pivot based on her feedback without starting over from scratch.

Luckily, you're a web engineer so you're used to having requirements that can change at the last minute. As such, you designed your code to be modular and flexible. All you need to do to improve your feature is move a couple modules around! You spend some time making your code presentable and readable before publishing it online for a code review from your team.

While you're waiting for other engineers to review your code, you head over to the meeting you had arranged with your coworker during the stand-up meeting. Both of you decide to tackle this problem by **pair programming**. It is a collaborative technique in software engineering when two engineers program on one computer. One person types while the other person describes what the code should be doing at a high level.

Then, they switch roles. The pair programming session was fruitful and you solved a major problem together.

Meanwhile, your fellow engineers have reviewed your code and some of them have posted feedback. Some of these comments are clarification questions. Other comments point out areas where you can make improvements to your code. You answer the clarification questions and agree to make the improvements.

There are only a couple hours left before the end of the workday. This is the time you've set aside as **whitespace time**. Your company requires that engineers allocate 20% of their time to whitespace time — time that can be spent learning a new technology or working on a personal projects. You use that time to volunteer and mentor a woman who attends a software engineering bootcamp.

At the end of the day, you head home to get some rest before doing it all again the next day.

TERM TO KNOW

Stand-up Meeting or Scrum Meeting
A daily meeting where colleagues update their teammates on the status of their tasks and coordinate the rest of the day.

Pair Programming
A collaborative technique where two engineers write code on one computer. Usually, one engineer types while the other engineer describes what the code should be doing at a high level.

Whitespace Time
Whitespace time is a common benefit that tech companies will provide their employees. It's the name for an amount of time that an engineer should devote to personal projects or volunteer work.

TERMS TO KNOW

Algorithm
A set of step-by-step instructions that describe how to solve a problem.

Computer
A computer is anything that can perform different functions based on changing conditions and has the ability to store data

Formal Language
A language designed by people for specific, often technical, applications.

Natural Language
A language that evolved naturally over time.

Pair Programming
A collaborative technique where two engineers write code on one computer. Usually, one engineer types while the other engineer describes what the code should be doing at a high level.

Program
Programs are algorithms that computers understand. Engineers write programs using programming languages.

Programming Language
A formal language designed to tell computers what to do.

Stand-up Meeting or Scrum Meeting

A daily meeting where colleagues update their teammates on the status of their tasks and coordinate the rest of the day.

Token

The basic building block of a language.

Whitespace Time

Whitespace time is a common benefit that tech companies will provide their employees. It's the name for an amount of time that an engineer should devote to personal projects or volunteer work.