What is an Algorithm ?

An algorithm is a process or a set of rules required to perform calculations or some other problem-solving operations, especially by a computer. The formal definition of an algorithm is that it contains a finite set of instructions which are carried out in a specific order to perform a specific task. Algorithms are not a complete program or code; rather, they are just a solution (logic) to a problem, which can be represented in multiple ways. A few ways that programmers write algorithms, besides code, are as informal descriptions, flowcharts, or pseudocode.

Sometimes, when studying up on a new topic, it's a good idea to just watch a quick YouTube video about it or read a few articles. Part of this bootcamp is preparing you to self-study, so that you can learn on the job. Let's practice a bit of that now.

Some tips before you watch these videos:

- You can watch YouTube videos at an increased speed. When getting a feel for a topic, content creators can feel a bit slow, so crank up the speed to 1.5 or 2x if you need them to get to the point faster. Click the little gear icon in the lower right of the video to see the options.
- Use the pause button and take notes if you need to. Sometimes, they're not too slow at all, or they throw a few vocabulary words at you that you know you need to understand before progressing. Pause the video, look up what you need to, then dive back in.
- Turn on the closed-captioning/subtitles. Many videos on YouTube, especially the well-produced ones, have closed-captioning subtitles that were written specifically for that video, but even the ones that don't will have some auto-generated words that represent what the creator is saying. As long as Google's algorithms can cope with the creator's accent, these tools can be really useful, so turn them on by clicking the CC button.
- Have an IDE running to test some snippets of what you're seeing. Once you're feeling more confident with Java, you may want to play around with the code that a video presents to you. Do it. It's always a good idea to have a file running somewhere called "test" or the like, just for the sake of experimentation. (Note:

this may not apply to the following videos, as they're more verbal explanations, but it's a good tip to have in your back pocket, now that we've set up an IDE!)

Here's a Khan Academy video introducing algorithms, what they are, and some of the things we use them for:

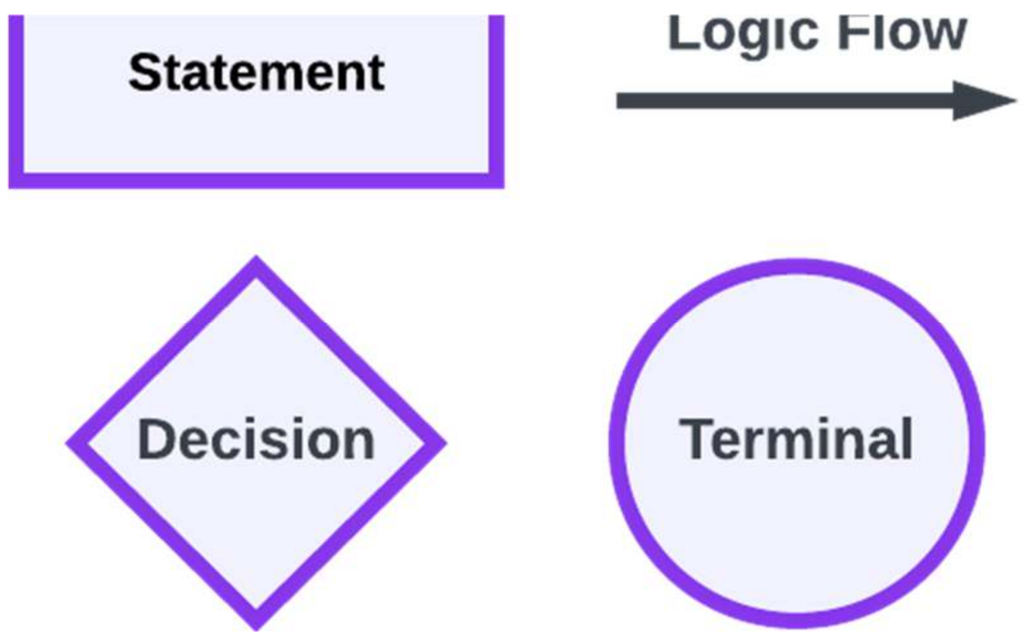
https://youtu.be/CvSOaYi89B4
Here's a TedEd video that better explains how to start working on your own algorithms. We'll continue this conversation below, but please do watch it in order to get a good head start:

https://youtu.be/6hfOvs8pY1k
InfoWarningTip
Remember: there's no shame in looking up outside resources, whether you're on the job or working your way through Academy. Do your best to ensure that you make it through the self-led curriculum before we join up for instructor-led lessons, but feel free to branch out and take a look wherever you need to in order to clarify your understanding. Maybe you could even share your best discoveries with us this afternoon, if something works well for you.

#Flowcharts and Pseudocode

How do you represent an algorithm? Attempting to code before fully understanding its underlying algorithm can lead to bugs, so what's a better alternative? Two options are **flowcharts** and **pseudocode**:

#Using Flowcharts to Represent Algorithms
A flowchart is a visual representation of an algorithm's control flow. This representation illustrates statements that need to be executed, decisions that need to be made, logic flow (for iteration and other purposes), and terminals that indicate start and end points. Figure 1 reveals the various symbols that flowcharts use to visualize algorithms.

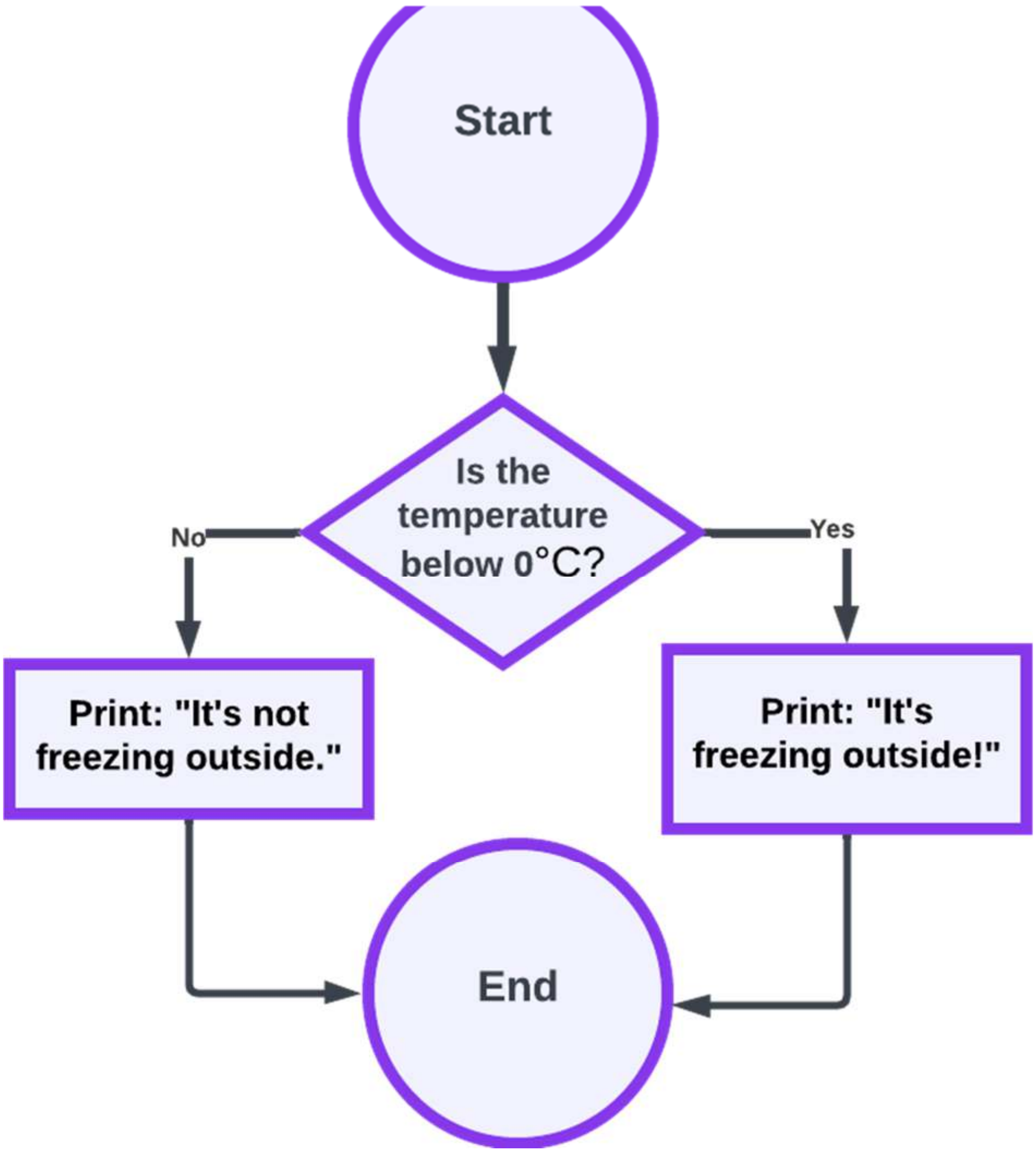


The basic components of a flowchart for an algorithm

The four basic parts of an algorithm flowchart are these:

- **Statement:** Statements are represented by rectangles and contain information about variables and actions required for the algorithm.
- **Logic Flow:** Logic flow is represented by an arrow which could be straight or curved. These arrows are used to show how the algorithm progresses or loops depending upon the inputs provided.
- **Decision:** Decisions are represented by diamond shapes and contain break points at which the algorithm will make decisions. Think of these like the `if` statements we recently learned about.
- **Terminal:** Terminals show where our algorithm begins and ends and are represented as circles. Keep in mind that many algorithms will have multiple endpoints.

Here's a super simple example:


Start
Is the temperature below 0°C?
No
Yes
Print: "It's not freezing outside."
Print: "It's freezing outside!"
End

A flowchart showing the logic for a program that would print a statement based on whether or not it's below freezing outside.

In sum: a flowchart's simplicity and its ability to present an algorithm's control flow visually (so that it's is easy to follow) are its major advantages.

Flowcharts have several disadvantages, though:

- It's easy to introduce errors or inaccuracies into highly-detailed flowcharts because of the tedium associated with drawing them. This tedium can be made better with tools like draw.io or Lucidcharts, but flowcharts can still get out of hand quite quickly.

- It takes time to position, label, and connect a flowchart's symbols, even using tools to speed up this process. This delay might slow your understanding of an algorithm.

InfoWarningTip
Flowcharts operate similarly to your average **whiteboarding** session. Whiteboarding is what it's called when developers draw their ideas out on a whiteboard to explain their thinking, and you'll often see shapes and arrows like the ones above in these whiteboard diagrams. We tell you this to explain that thinking in these terms is important beyond simply writing code: it's a style of understanding that can flesh out difficult problems and ensure that you're on the same page as your peers.

#Using pseudocode to represent algorithms

An alternative to flowcharts is **pseudocode**, which is a textual representation of an algorithm that approximates the final source code. Pseudocode is useful for quickly writing down an algorithm's representation. Because syntax is not a concern, there are no hard-and-fast rules for writing pseudocode. The key is making sure that anyone with whom you plan to share your pseudocode can understand it–including yourself at a later date.

Essentially, you should strive for consistency when writing pseudocode. Being consistent will make it much easier to translate your pseudocode into actual source code. For example, consider the following pseudocode representation of the previous flowchart:

CopyCC#C++ClojureCSSDartGoHaskellHTMLJavaJavaScriptJSONJSXKotlinMarkdown PascalPerlPHPPlain TextPythonRRubyRustSchemeShellSQLSwiftTypescriptVB.NETVBScriptXMLYAML

```
// Assign temperature to temp variable
// If temp is > 0, print warm message.
// If temp is < 0, print freezing message.
```

The idea behind using pseudocode is to give yourself a starting place for writing your code. Often, this means breaking your problem down into smaller steps and thinking about what kinds of variables, statements, and checks you'll need to accomplish your goal. Pseudocode can be a one-and-done process, or be used as a form of drafting, from the simplest statements possible to the most granular details of what you need to write. It's up to you how deep you want to go, but it's a great way to practice making the kinds of decisions that will lead you to writing good code.

InfoWarningTip
In coding interviews, your ability to write pseudocode is often tested. Although it's tempting to jump right into writing code once you've gotten good at the basics, make sure you take some time to practice this important skill so that your possible-future-employers can see how well you can sort out your ideas on-the-fly.

Which of the two ways of showing your thought process do you like better: pseudocode or flowcharts? Why?
Pseudocode and flowcharts are both valuable tools for planning and visualizing the thought process before writing actual code.
Pseudocode: Textual Representation: Pseudocode is essentially a textual representation of your algorithm or program logic. It uses a mixture of natural language and programming-like constructs to outline the steps and decisions in your code.
Flowcharts ; Visual Representation: Flowcharts use graphical symbols and arrows to represent different steps, decisions, and paths in your algorithm. They provide a visual representation of the logical flow of your program.

#Summary

Algorithms help us solve difficult problems in systematic and efficient ways. Often times, writing algorithms starts with writing pseudocode or a flowchart that represents the steps that we need to take to achieve our goal. Flowcharts have some details that most people can agree on, but pseudocode can be whatever makes sense to you and the team you're working with. Get used to using both, as they're a good way to get your ideas down on the page and get you ready to write more complex code.

#Designing Your First Algorithm

Imagine you are a developer at a robotics firm; the first task that you have been assigned is to develop instructions for a robot to make a peanut butter and jelly sandwich. Here's the twist: the robot has **not** been programmed to do tasks as a human being would, such as intuiting smaller instructions from larger ones. You must provide **precise instructions**, so that the robot will make a proper sandwich without any undue mess. For instance: telling a robot to scoop peanut butter out of a closed jar would probably result in disaster.

#Design an Algorithm to Make a Peanut Butter and Jelly Sandwich

Think about the steps it would take to get the robot to successfully make a peanut butter and jelly sandwich, and write them in the box below. No need to worry about actual code–just think about the steps and get the job done.

Write the steps below:

1.get two slices of bread,the peanut butter and jelly

2.get the spoon.

3.Scoop the peanut butter and put it on the bread.

4.Wash the spoon.

5.Scoop thejelly and put it on the bread.

6.Spread the peanut butter and jelly on the bread.

7.Put the breadtogether.

8.Save it on a plate.