So, we've talked about how algorithms are simply clear descriptions of the steps in the process for solving a problem. We can use whatever language we want for the description as long as the language is clear in meaning among the parties involved. It doesn't even have to be text based, we can use pictures.

In fact, many humans seem to have a preference for describing concepts using pictures. There's that old saying, a picture is worth 1000 words. So we can use pictures to diagram our algorithm and we call that a **flowchart**. Now, it's called that because, well, first it is a chart. It's a big sheet on which we draw a set of graphical items.

Each item that we draw is a step we want to execute in the algorithm. Second, a flowchart indicates the flow of the steps, how they're related to each other in order of execution. In other words, which step happens first, which step happens after that, which step or steps might want to repeat, and which steps might want to skip in certain cases.

The flow. Now humans like to make rules for others to follow. So even with something like picture-based descriptions of algorithms, some people have to come up with standards. Now standards are usually a good thing. In Lewis Carroll's children's book, Through the Looking Glass, the character Humpty Dumpty the Egg says he pays words to mean whatever he wants them to mean.

For instance, when he says glory, what it actually means is, there's a nice knock down argument for you. Which isn't what it means at all. So suffice it to say Humpty's non-standard use of English doesn't work out so well for his communications with Alice, which is why we need standards.

Now, in the standardized flowcharting language, we draw interconnected shapes on a piece of paper. The shapes are standardized and each shape means something different, so that we can tell immediately what the intention is. For program flow charts, it got to a point where there's actually a template you can use to get the shapes exactly picture perfect.

Here is an image of one that my father used here when he did business computing back in the 70s. It's a piece of graph outlined plastic where you use a number two pencil to draw inside the shapes so you can get it exactly right. And as if that weren't enough, it even comes with instructions.

It's like the coloring book coming with instructions, it tells you what the shapes do and what you use them for. Anyway, the shapes are connected by arrows which tell us what order to do the things in. And this lets us arrange things as we like on the chart.

Bottom to top, right to left, zigzag, even a spiral if you were to get really bored, as long as the arrows connect them in the correct execution order. In general though, we tend to be boring and organize the steps in a top down orderly fashion. So, let's warm up our markers and draw our

first flow chart.

The first two shapes we are going to learn are that a circle or oval marks the start and end point for the process, and a rectangle is used for a simple step. So, let's try one out. Here is a flowchart, for our hair shampooing problem. So we start our shampooing flowchart with, what-else, start.

Our next step if you remember correctly, was to wet our hair. Step after that is apply shampoo. Step after that is to lather. Next step, rinse and last step is repeat. And there you have it, how to shampoo your hair in five easy steps. Now, let's get a little more technical and computery.

We don't shampoo our hair on the computer, we do math. So let's draw a flow chart to compute the circumference of a circle of radius five. So for that, we need a new flow chart. Again, we start at the beginning with a start. Our next step consists of computing that the circumference is equal to 2 (brush off those old high school math cells in your brain) 2 times pi times radius which is 5.

And that's the entire algorithm. Very dull, and furthermore, it always calculates the same answer, and after all that, it doesn't even tell us what that answer is. So let's improve upon it a little bit by inserting a person into the process.

What if we have a user who wants to calculate the circumference of a circle of any size? For that, we need a different algorithm and therefore a different flow chart, this time with input and output. Flow charts use a slanted parallel ground for this purpose. So again, start, but now, were going to ask the user to input the radius that they want to compute the circumference for it.

Then the next step, well, we've already seen that. That's that the circumference is equal to 2 times pi but this time it's times the radius which the user gave us. And because we're so upset about the computer keeping the answer a secret, we're going to once again have a box where we tell the computer to output the answer that it computed.

And what do we go to next? Of course, the end. So that's a little more interesting. But what if we were worried that the user might enter a negative number? And what if we were feeling rather forgiving, and we want to work around that. If we use the standard formula as we did it before, our circumference would just come out to be a negative number.

And we don't like that. So conditionals to the rescue. In flow charting, we use a diamond to indicate a conditional. It tells us what our decision is based upon. And the arrows coming out of the diamond tell us which set of steps we are going to execute for each possible outcome.

So continuing with the flow chart, let's just start drawing over here. We draw a diamond. And inside the diamond, we phrase our question which is, is the radius that the user entered, in fact, less than zero, which would be bad. We have two arrows that come out of this conditional.

Each leading to a possible step. And we labeled arrows to indicate what the result was of the question that we asked. So here we say, was the radius less than zero? Yes, this branch is for where the answer is no. What do we do, if the answer was yes, in other words, the user entered an invalid radius, while we say that we're going to compute a new variable, which is A the absolute value, which is the negative radius.

And on the other side, where it turns out, the user didn't enter something that was less than zero, we'll just say that a is in fact the radius that the user entered. What happens after this? Well, the two join back up again. Now the question is, this little mini flow try and decide how to incorporate that into our program.

And the answer is, it is in the arrows. So if you look at our main flow chart, we asked the user to input an r and we did calculations based upon it. Now we're going to splice ourselves in there and say after the user has entered value. We then go to the steps that determine whether the value is good or not.

And after we've computed a cleaned up value, we go back and insert ourselves there. Hopefully that makes sense. Now, there's no reason why we actually have to have any steps on one or the other side. We can have nothing there. For example, what if we were also concerned that the user might enter radius of 0?

And for that there's really no fix. So we will ask the question, is the radius equal to 0? And if the answer is no, now you notice I put the no on the other side this time it's completely up to me. Nothing special happens. But if the answer is Yes, in other words, the user was silly enough to enter a number equal to zero.

What we will do is I'll put a message to the user saying, and once again, he's placed back in. And I think you can figure out how we're going to insert that into the rest of our flow chart. This is not the way I would present the final flowchart to someone who was trying to understand what I was trying to do, I would clean it up.

But the fact is that the arrows tell us exactly what it is that I'm asking the other side to do. And lastly, and this will really blow your mind, we can make the arrow go anywhere we want, including backwards. For example, if we look at our original hair shampooing flowchart, remember the repeat step?

Now, you might think of that as that box representing repetition of the whole thing, but we choose to represent Repeat as to go back and do some of the steps over again. So for example, we might go back to here. Or, we could go back to one of the other steps.

Notice something about this. What I have there is one literal translation of the shampoo algorithm. You should notice that the flowchart is much more explicit than our verbal description

we used in the first module. We know exactly what sub-set of steps we are asking the computer to repeat.

However, being allowed to be more explicit means we're responsible for being more explicit and this example specifically says you should go back to the Lather step and we already know that's wrong. We could change this to include more steps. For example, just by redrawing the arrow, we could go there, or we could go there, one or the other.

We know that going all the way back to wet our hair is too much. So instead, we'll have it go back there, and what that tells us is that what we should repeat is the application of shampoo, the lathering and the rinsing, which is exactly right. One other thing to point out, when we go back to a step we've already executed with that backwards arrow, that doesn't mean do that one step.

It means to pick up the entire execution flow from that point onwards again. So if you look at the way you've drawn it here, the algorithm says that we should spend our entire lives in the shampoo. Doomed to repeat, apply shampoo, lather, rinse, steps over and over again.

And we call this an infinite loop. We're going to learn a lot more about this in much greater detail later in the course. Now, there are a whole bunch more details to flowcharting, but I think we've given you a sufficient taste. You might think, really? We're supposed to use this stuff.

Well, yes. When you're just jotting down ideas, whether it's for how you're planning out activities for the day. Or you are going to tell your friends to get to your house, or drawing out grand plans for conquering the world, it's often useful to diagram out your thoughts. Another use of flow charts is to convert actual programs back into a flow chart form, because the pictorial representation often makes flaws in your reasoning stand out.

I've programmed professionally for several decades. And I still routinely diagram my algorithms with flow charts, scouts honor. You should too. That is it for now. Thanks for watching Align online.