Knowing how to use a computer program without knowing how a computer works is an example of:

- Annotation
- Automation
- Abstraction

Correct

- Authentication

Continue

Everything Is a Number

Have a question? Discuss this lecture in the week forums.

Interactive Transcript

Search Transcript English

0:03
Welcome back. In this lesson, you're going to learn an important principle of computer science, everything is a number, or put a different way, computers only work with numbers. If we were to delve into the hardware, we would find that it only deals with bits, 0's and 1's. We're not going to look into the particulars of the hardware, but the important thing to know is that computers can only work with numbers, and really, can only do math. In fact, there is nothing that a computer can do that you cannot do. Computers can just do math really fast, so they can operate on large sets of data billions of times more quickly than you or I could by hand. Fortunately, you do not typically need to think about the details of the bits, due to a wonderful principle called abstraction. Before we talk more about how everything is a number, we'll take a minute to talk about abstraction.

0:54
Abstraction is the separation of the interface, what a thing does, or how you use it from the implementation, how it does that, or how it works. You can understand abstraction and its usefulness with a non-computer related example. Think for a moment about driving a car. Abstraction lets someone drive a car without knowing how it works. I know that if I press down on the gas pedal, my car will go faster. However, I do not know about the complicated inner workings of my car that make that happen. Abstraction often comes in layers, and what layer you need to work at depends on what you need to do. For someone driving a car, the level of abstraction appropriate to think about is what the controls of the car do. The inner workings under the hood are hidden and not important. However, for a mechanic, the details under the hood are important and are what she works with day to day. But even there, there's a different level of abstraction. The mechanic is concerned with how the parts of the engine fit together and work together, but maybe not with the physics that go into designing the engine. The engineers who designed the car would have worked at that level of abstraction, applying physics to make a car that works properly. Of course, there is an even lower level of abstraction and more theoretical physics that they did not concern themselves with. We could keep going down until we reach the most theoretical physics at the boundaries of human knowledge, all things that you do not need to know to drive a car.

2:19
These same concepts apply in programming. The level of abstraction that you need depends on what you are doing. Before this class, you have probably used a computer without knowing anything about how programs work. Now you will learn how programs work, but there will be deeper levels of abstraction that you generally will not need to know about.

2:39
Now that you know about abstraction, let us return to the principle that everything is a number, giving you a bit of a peek under the hood.

2:46
You may be a bit surprised by this idea, since you're used to using your computer working with things that do not seem like numbers, such as letters. However, these are relatively easy to encode as numbers. We could go with a = 1, b = 2, and so forth. What computers actually do is represent characters, symbols that can be letters, digits, punctuations, and so on. One way to encode characters is ASCII, in which, A is 65, a is 97,! is 33, and various other characters have other numerical values. If you needed to look up the numerical value of a character, you could find them in an ASCII table. However, you generally do not need to worry about the particular numerical values, due to the joys of abstraction.

3:32
Once we have characters, we can also have strings, sequences of characters such as "Hello!".

3:39
Strings com up quite often in computer science, as programmers frequently want to manipulate text. In fact, you've seen strings in HTML.

3:48
Strings are another great example of abstraction. You can write "Hello!", and it gets turned into numbers that your computer can work with. You generally do not have to think about the numeric representation, but you can manipulate it if your programming task calls for it. So why is it so important to know that everything is a number? Well, sometimes you want to do math with things that do not seem like numbers. Cryptography, the science of keeping information secure, relies on the ability to do math on strings. When you visit a website using HTTPS, the information sent back and forth between your computer and the web server is encrypted so that nobody else can read it. That process involves doing math on that data.

4:29
Another reason that it is important to understand that everything is a number is because this is why programming languages have types, which tell it how to operate on these numbers. Even though everything is a number, programmers want to interpret those values in different ways. Do the numbers mean letters? Do they mean pictures? Do they actually just mean plain numbers? The type of the data tells what the numbers mean, and thus, how to operate on them. If we add two strings together, we might want to concatenate them, put one after the other to form a longer string. In that case, the string "1" + the string "1" would be the string "11". If we have just the number 1 and we add it to 1, we would get 2.

5:13
A third reason why the everything is a number principle is important is that whenever you have data you want to work with, you need to represent it as numbers. You may be able to use existing types, such as strings, which already represent information as numbers, to help you out. Such is the joy of abstraction.

5:30
One other thing that may surprise you is that programs themselves are numbers. Then again, it may not surprise you, since you just learned that everything is a number.

5:39
As you have already seen, you express your algorithms to the computer by writing text. The code that you type in is a string.

5:47
Another program takes this string and figures out how to turn it into numerical instructions that the computer can execute. The fact that programs are numbers is actually quite powerful. It means that you can download and run new programs on your computers. They're just data, like everything else. The fact that programs are also just like data is at the heart of many security issues. Hackers give a program input that have the numeric encodings of instructions they want to execute, and then trick the program into running it. Of course, as you write programs, you do not need to worry about the numerical encoding details, all because of the wonderful ideas of abstraction.

6:27
So in this lesson, you learned that everything is a number, an important principle since computers can only do math. You learned about abstraction, the separation of interface from implementation, and how that means that you may not always need to think about how things are numbers to compute with them.

Downloads

Lecture Video mp4

Subtitles (English) WebVTT

Transcript (English) txt

Lecture Slides pdf

Would you like to help us translate the transcript and subtitles into additional languages?