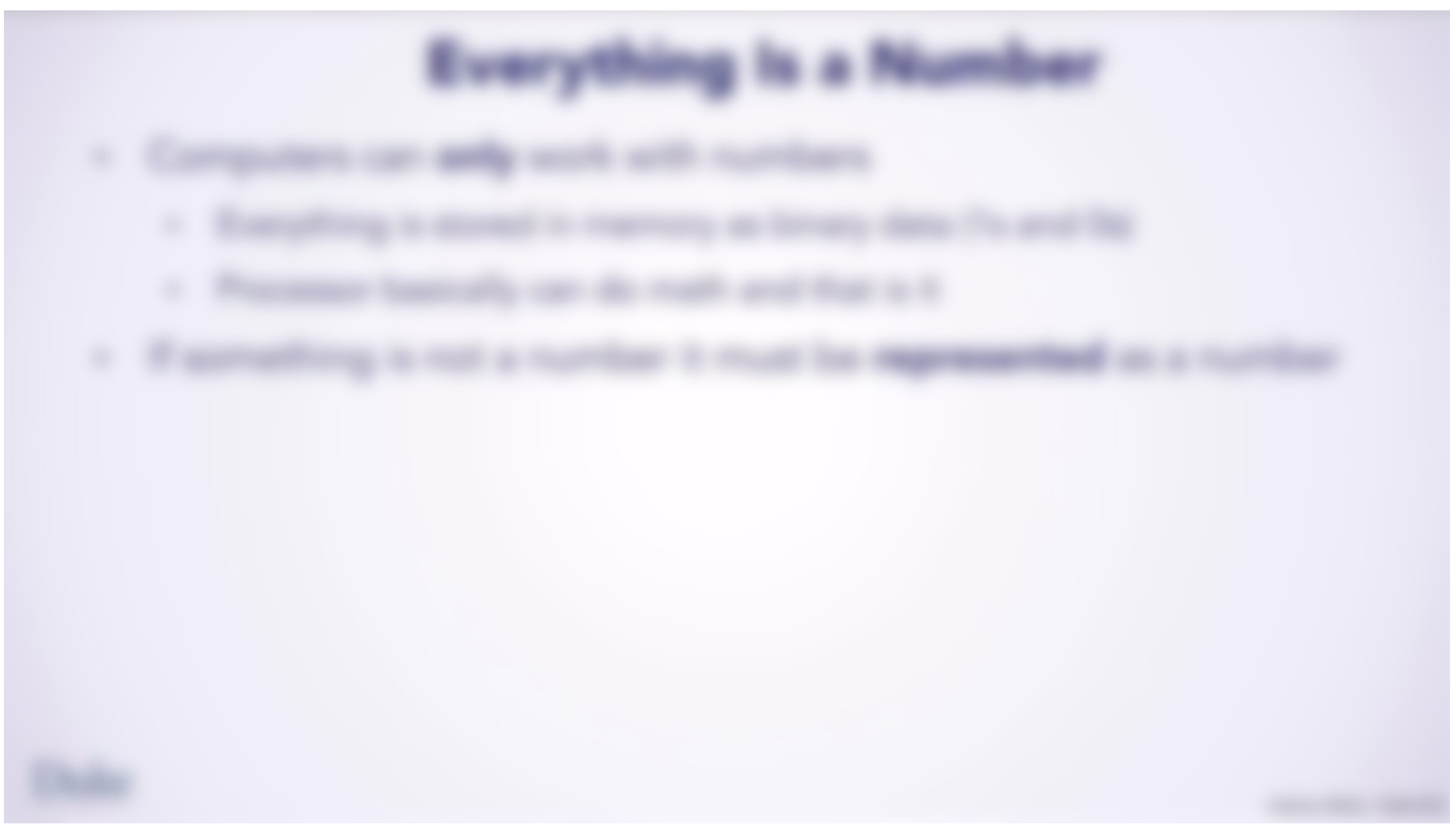
Back Next Print view Print all Index of pages

Everything is a Number and the Importance of Being Specific



Everything Is a Number

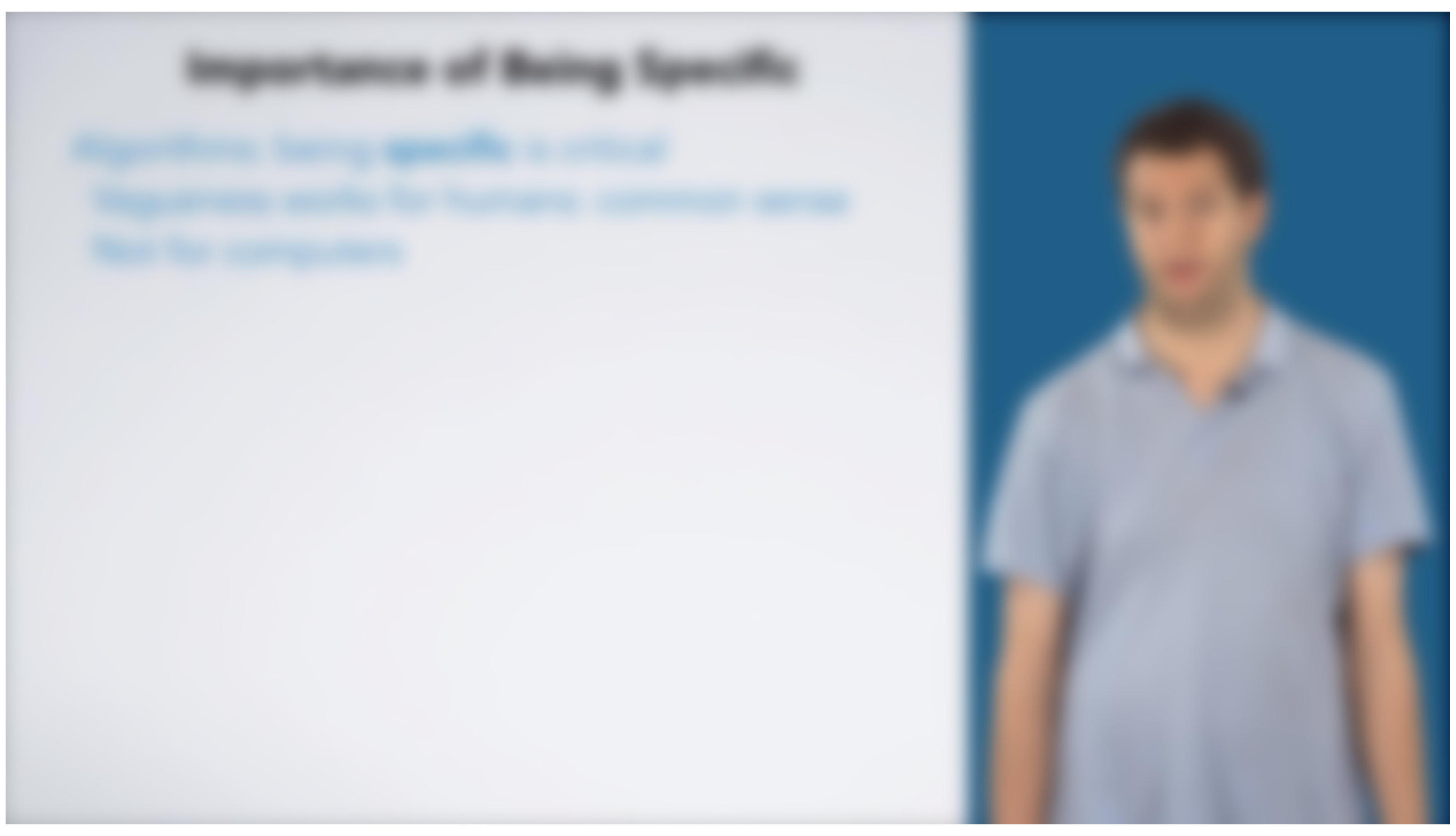
- Computers can **only** work with numbers
  - Everything is stored in memory as binary data (1s and 0s)
  - Processor basically can do math and that is it
- If something is not a number it must be **represented** as a number

A Note on the Next Video

The next video (like several others) comes from a Coursera course on programming in C. This one makes some reference to learning about C at the start, as well as things in upcoming courses. You all will learn most of these things (or their Python analogues) in the rest of this course. The important part of the video is the second half (PB&J sandwich), which is independent of language. It is also one of the more fun videos, so we hope you enjoy.

Also, this video references an assignment to write a sorting algorithm and share it with another person. That is only in the C Coursera course, not a thing you all are doing.

The Importance of Writing a Specific Algorithm



Importance of Being Specific

- Algorithms: being **specific** is critical
  - Vagueness works for humans: common sense
  - Not for computers

Next Steps

At this point, you should have a basic grasp on the idea of developing simple algorithms. This skill is one that you will practice as you go through the rest of this course, as every programming problem's key component is figuring out the correct algorithm. We cannot underscore enough the importance of working through problems in a step-by-step fashion. Many novice programmers try to skip over the first several steps and plunge right into writing code. The result is frequently a disaster, which they end up spending orders of magnitude more time trying to fix than they would have spent planning correctly in the first place.

The reasons that novice programmers give for skipping straight to step 5 vary, but a common one is "Step 3 (writing a generalized algorithm) seemed too hard." This reason is quite possibly the worst reason to skip over step 3—if making a correct plan is proving hard, how can you possibly hope to write correct code without the plan? It is better to repeat steps 1 and 2 on more examples until you can find the pattern and write down the algorithm. Another common reason that novice programmers give for skipping the first steps is "to save time"; however, they often then report spending countless hours trying to debug the resulting code. It is well worth ten or even thirty minutes of planning to avoid trying to debug a hopeless mess for multiple hours!

As you become more and more practiced at this process, you may find that steps 1–4 come naturally, and you can do them in your head without writing them down—much like what happens with basic mathematical skills. When these improvements in your programming skills happen, then there is nothing wrong with doing the easier steps in your head, as long as you are sure that you are doing them correctly. However, whenever you are programming at the boundaries of your abilities, you will need to go through these steps—so it is quite important to remember how the full process works even as you become more skilled.

Next, we will go over some tools you will need to practice programming: Linux, Emacs, and Git. You will write your programming assignments for this course on our SCI server, which uses the Linux operating system, for which you will learn some basic UNIX commands. Then you will learn how to use Emacs to read and edit code. Finally, with a primer on the revision control system Git, you will be ready to do work on our Mastery Learning Platform.

From there, we will learn a bit about reading code in Python, before we continue on to more about writing code. By reading code, we mean being able to understand exactly what a piece of code does, executing it step-by-step by hand. This skill is important for three reasons. First, it is very difficult to write when you cannot read. Reading the code will be a matter of drawing and updating diagrams which reflect the state of the program as the code executes. Writing code will be a matter of writing the syntax to effect the appropriate transformations—as spelled out in the algorithm—to the program's state. Second, being able to read your code is crucial for being able to debug your code. Third, you may end up in a variety of settings (e.g., group class projects, coding teams in industry) where you must read and understand what other people's code does so that you can work on it.

Day 1, Part 3

- ☑ Algorithms Quiz
  Make sure to complete this quiz to review all the things you've learned so far this week.

Back Next