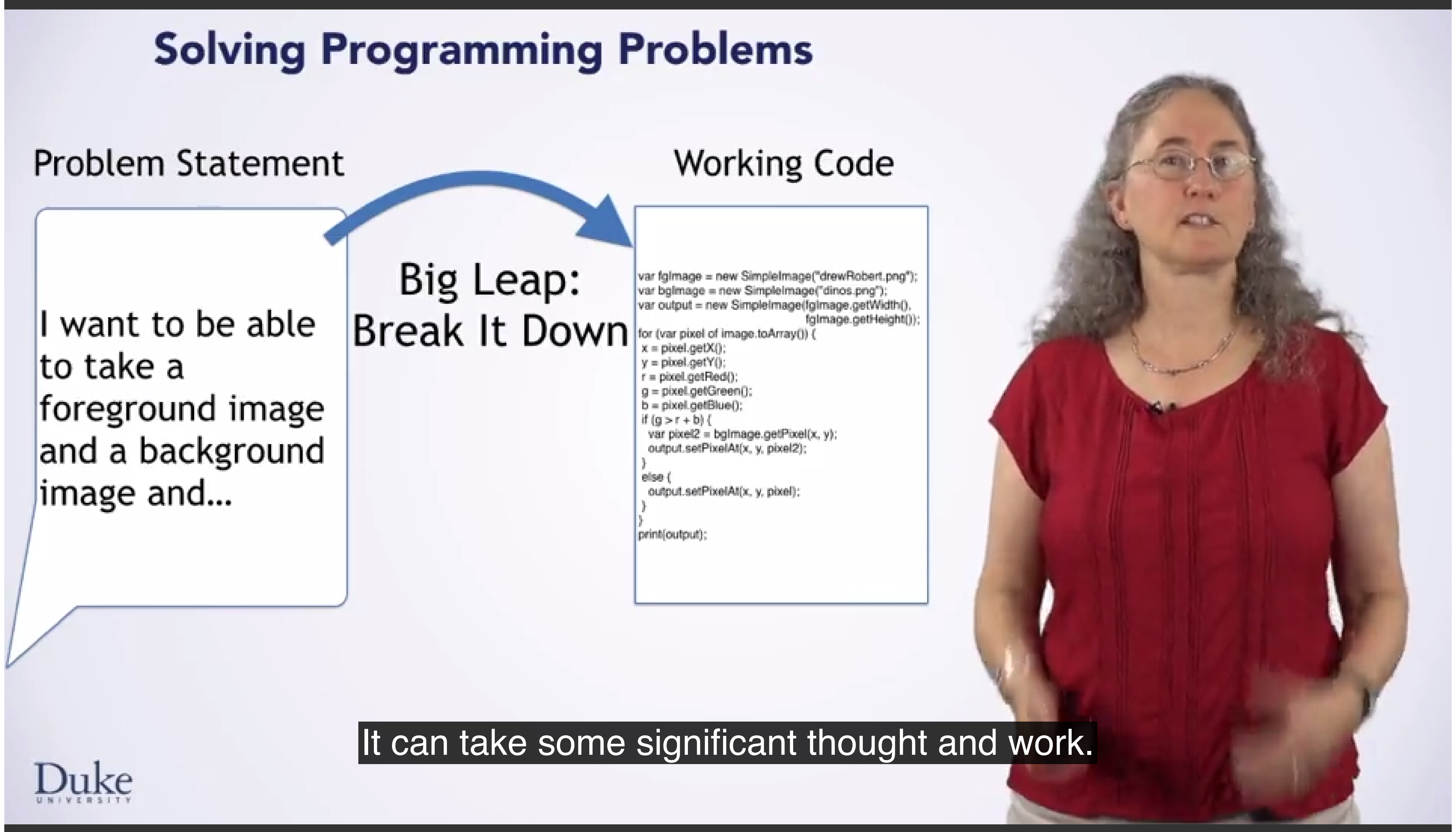

Solving Programming Problems

Problem Statement

I want to be able to take a foreground image and a background image and...

Big Leap: Break It Down

Working Code

```
var fgImage = new SimpleImage("drewRobert.png");
var bgImage = new SimpleImage("dinos.png");
var output = new SimpleImage(fgImage.getWidth(),
                             fgImage.getHeight());
for (var pixel of image.toArray()) {
 x = pixel.getX();
 y = pixel.getY();
 r = pixel.getRed();
 g = pixel.getGreen();
 b = pixel.getBlue();
 if (g > r + b) {
  var pixel2 = bgImage.getPixel(x, y);
  output.setPixelAt(x, y, pixel2);
 }
 else {
  output.setPixelAt(x, y, pixel);
 }
}
print(output);
```

Duke UNIVERSITY

It can take some significant thought and work.

Solving Programming: A Seven Step Approach

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

Interactive Transcript

Search Transcript English

0:02
Today, you're going to learn how to solve programming problems using the seven-step approach that we will use throughout the rest of this course. When you're solving a problem, you're going to start with a problem statement. You know that you want to end up with working code but going straight from a problem statement to working code is a rather large leap. It can take some significant thought and work. This is why we break it down into seven steps for you, giving you a manageable approach where you know what to do next to solve the problem. This is a seven-step process that we recommend you use whenever you are solving programming problems that are difficult. As your programming skills improve, many problems will become easy enough to just do them in your head. However, there will always be some problems that are harder, and having a step-by-step approach will be helpful to you. Let us look at each step in a bit more detail. The first step is to work an example of the problem yourself, by hand. This should be a small instance of the problem, something with about four or five pieces of data to manipulate. You don't want to try to process a million pieces of data yourself, that would take forever. If you have trouble doing the problem yourself, you cannot write a program to do it. So what should you do if you get stuck here? Well, one of two things could be wrong. Maybe the problem is just unclear. The problem statement does not give you enough information about what you're supposed to do. In a classroom setting, you could consult your teacher or TA. In a professional setting, you might work with your technical lead or customer to clarify the requirements, or you might just need to refine the problem statement yourself. The other potential problem is that you lack domain knowledge, the knowledge of the field that the problem belongs to. If you're trying to write a program to compute physical motion and you do not know the physics equations that you need, that would be a lack of domain knowledge. When you have this sort of problem, you need to find the relevant domain knowledge before you proceed. In step two, you want to write down what you just did to solve this problem. You want to be as exact as possible. Do not leave anything out, and write down how you solved it in a step-by-step fashion. At this point in the process, you are just writing down the step-by-step approach for the one particular instance you solved, not the more general problem. The tricky part in this step is that we often do things without thinking about them. If you gloss over something or you're not precise about what you did, it will make the later steps harder. In step three, you want to move from the particular instance that you solved in step one to an algorithm that works for any instance of the problem. That is, you want to devise an algorithm which can solve the problem correctly for any input that you give it. You'll do this by finding patterns in what you did and replacing specific behavior with more general behavior based on that pattern. Some important tools for finding patterns which we will delve into more deeply soon are looking for repetitive behavior, finding behavior which you do sometimes but not always, and figuring out under what conditions you do it, and figure out how specific values you use relate to the parameters you picked. So what should you do if you have trouble with this step? Go back and try steps one and two again. Use different inputs which will give you another example to work from. You can see the steps for a different instance of the problem and have more information to help you find the patterns. In step four, you want to check your algorithm before you proceed to turn it into code. If you found the patterns incorrectly or otherwise made a mistake in step three, you would like to find that out now. The way you take your algorithm is to pick at least one different input. Again, it should be a small one, and follow the steps of your algorithm for it. If your algorithm gives you the right answer, then you are ready to move on. If not, you should go back and fix it first. Now that you have devised an algorithm to solve the problem, you're ready to translate that algorithm into code. This step is where the syntax of a specific programming language comes into play. You need to write down your steps in the syntax of that language. Once you've written your code, you want to be sure that it works correctly, so you run test cases on it. Running a test case involves executing the code on a particular input and checking if it produced the right answer. The more test cases your code passes, the more confident you could be that it is correct. However, no amount of testing can guarantee that the code is right. When your program fails a test case, you know something is wrong. And when that happens, it's time to debug your program. You can watch a review video about debugging if you need more information, but at a high level, you will apply the scientific method to understand what is wrong with your program and determine how to fix it. Once you have identified the problem using the scientific method, you will need to revisit a previous step to fix it. If the problem is in the algorithm you designed, you will want to go back to step three and rethink your algorithm. If your algorithm is correct but you implemented it incorrectly in code, you will want to return to step five to fix your code. Now you have learned the seven-step approach to solving a programming problem. In the next video, we will work through an example of the process. This process can guide you through programming problems you need to solve not only throughout the rest of this course but whenever you need to solve a difficult problem. Thank you.

Downloads

Lecture Video	mp4
Subtitles (English)	WebVTT
Transcript (English)	txt

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