

Finding a Gene in DNA

Finding All Genes in DNA

	Introduction	48 sec
	Conceptual Understanding	4 min
	While Loops	9 min
	While Loop Syntax and Semantics	3 min
	Coding While Loops	6 min
	Three Stop Codons	5 min
	Coding Three Stop Codons - Part I	7 min
	Coding Three Stop Codons - Part II	4 min
	Logical And / Or	8 min
	Coding And / Or	6 min
	Finding Multiple Genes	5 min
	Translating to Code	8 min
	Programming Exercise: Finding Many Genes	10 min
	Practice Quiz: Finding All Genes in DNA	4 questions

Debugging Code

Using the StorageResource Class

Review

Consider the following code:

```
1 ~ if ( (x%2 == 0 || y%2 == 0) && x>3 ){
2     System.out.println("condition was true")
3 }
```

If x has the value 5 and y has the value 6, will the print statement be executed?

(Hint: This expression may look complicated, but don't worry, you have learned everything you need to evaluate it! Work on each part of the expression separately, then put the parts together to figure out whether the whole condition will evaluate to true or false.)

☐ No

☒ Yes

Correct

When x is 5 and y is 6:

Continue

5:33 / 8:02

Logical And / Or

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

Downloads

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

Would you like to [help us translate](#) the transcript and subtitles into additional languages?

Interactive Transcript

Search Transcript

English

0:03

Okay, now you have an algorithm which works with any of the three stopCodons. However, let's explore how you can make this code work if we decided to have find stopCodon return at -1 when there's no valid stopCodon, rather than the length of the string. Note that this is a perfectly valid design choice and mirrors what the .indexOf method does. But we need to learn a new concept to make it work with our code. You'd want to change line six of the gene finding algorithm to reflect this choice. You cannot just take the minimum since -1 is smaller than any valid index. So what you want to do is pick the smallest number that's not -1. Let's look at a few examples. Here, taalIndex is -1, and tgalIndex is 3, and tagIndex is 6. What index should you pick for the stopCodon?

0:58

Well, we want 3 because it's the smallest number which is not -1. Notice, we can't just pick the minimum one because that would give us -1 which is not a valid value here. What if we had 5, -1, and 8? We'd want 5.

1:14

Likewise, with 10, 4, and -1 we'd want 4. And with -1, -1 and 11 we'd want 11. Let's think about how we can express this decision process algorithmically. In the first example we compared these two numbers first. Even if you think you're comparing all three at once, you're really comparing two at a time very quickly and perhaps without thinking about what you're doing. Of these two values we prefer 3 since the alternative is -1. And then we compare 3 against 6. We choose 3 since it's smaller than 6. In the second example we compare 5 to -1.

2:02

Then we choose 5 and compare this against 8. In this choice you'd pick the smaller, which is 5. In the third example you'd compare 10 against 4. You'd choose 4 because it's smaller.

2:19

And then you'd compare 4 against -1 and choose 4. In the final example, you're comparing -1 against -1. It doesn't really matter which you pick, they're the same. You then compare -1 against 11 and you choose 11. Note, you should think about what you would do if all three values were negative. What would that mean? It would mean that no valid stopCodon exists. We'll have to use that in our code. Now how do we think through how we made these choices? We're only going to look at the logic involved in making the choice between one pair. That same logic will work with the second choice. In the first example we picked tgalIndex because taalIndex was -1. In the second example we picked taalIndex because tgalIndex was -1. In the third example where neither is -1 we picked tgalIndex because it's smaller than taalIndex.

3:26

And in the final example it didn't matter which we picked, since both were -1.

3:32

Let's write down what our logic is for making that selection.

3:36

If taalIndex is -1, we'd want to pick tgalIndex.

3:43

But that's not all there is to it. We'd also choose tgalIndex if both tgalIndex is different than -1 and tgalIndex is less than taalIndex. Notice how we've expressed this logic with or and and. Logical connectives which let us make complex conditionals out of simple ones. Now let's put that logic into the algorithm we're working on.

4:11

We'll use that logic to pick between taalIndex, and tgalIndex. And we'll store the best choice there is in a variable called minIndex. Then we'll choose the same logic to choose between minIndex and tagIndex.

4:29

If minIndex is -1, there were no valid stopCodons so we'll give back the empty string.

4:38

Otherwise, we've found a gene.

4:41

Now let's see how we express these ORs and ANDs in Java.

4:46

You can express AND with two ampersands. This is a Shift+7 on most U.S. keyboards. We can see in this example if (x < y && y < z).

5:03

You can express OR with two vertical bars, also called pipes. On most U.S. keyboards this symbol is Shift+backslash, and we can see an example here. if (a > b || c < d).

5:19

In the particular case the algorithm we're working on, you could express the step as shown here. Notice how the OR corresponds to the two vertical bars and [the AND corresponds to the two ampersands](#).

5:34

AND and OR have special rules called short circuit evaluation. Which basically means that if Java can figure out the result of an entire expression involving AND and OR by evaluating only the first operand, then Java will skip evaluating the second operand. Let's look at an example.

5:56

Suppose x is 8 and y is 1.

6:02

x is less than y is not true because 8 is not less than 1. So x < y evaluates to false. There's no reason to evaluate y < z, because whether it's true or false the whole AND expression will be false. Because false AND anything is false.

6:25

For OR, consider this example and suppose a is 3, and b is 1. Here, a is greater than b is true, because 3 is greater than 1. So the whole OR expression will be true, regardless of whether c is less than d or not. So there's no point in evaluating the expression c < d because true OR anything is true.

6:53

Why is short circuit evaluation important? If you skip evaluating y < z in this example, it doesn't really matter so much. But it's much more useful when the second operand could crash your program if it's evaluated. For example, here we're checking if x is less than the length of a string AND that the character at the xth index of the string is the letter a.

7:22

If the first condition is false then x is not a valid index, it's beyond the end of the string. So trying to get the x character with the .charAt method would crash the program with a StringIndexOutOfBoundsException. But fortunately this line of code is safe because of short circuit evaluation. The .charAt method will never be called when x is greater than or equal to the length of the string. Relying on short circuit evaluation is a great example of defensive programming, and one of the tools in your Java programming toolkit. Have fun!