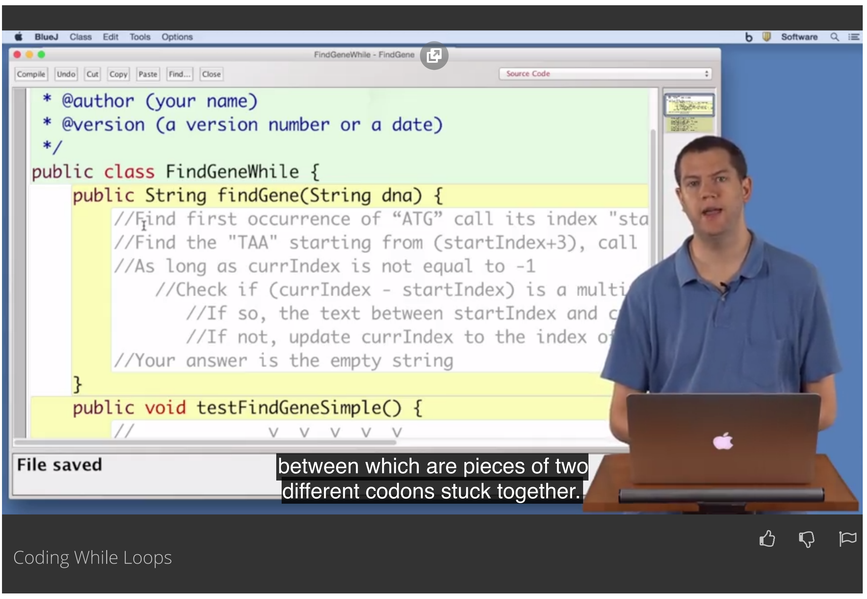

Have a question? Discuss this lecture in the week forums.

Interactive Transcript

Search Transcript

English

0:03
Okay, now you've learned about while loops, and are ready to turn this algorithm into code, which will search for a TAA, which is a multiple of three away from the ATG, even if there are other TAAs in between which are pieces of two different codons stuck together. So here's the algorithm that we came up with, and now we're ready to turn it into code. Our first step, it says, is to find the first occurrence of ATG and call its index, startIndex. So as you're hopefully becoming familiar with by now, we're going to say, int startIndex equals dna.indexOf ATG, because that's going to find us the first index of ATG.

0:48
Our second step says find the TAA starting from startIndex plus 3, and call this result currIndex. So I'm going to say, int currIndex equals dna.indexOf, and we're looking for TAA and we want the start from startIndex plus 3. This should also be becoming hopefully familiar to you with indexOf starting from a position. Now we're going to say, as long as currIndex is not equal to negative 1. This is what you have just learned is a y loop. I want it to repeat some steps as long as some condition is true. So while currIndex is not equal to negative 1, I want to do these steps here. And I'm going to go ahead and put, I've indented my steps to match the grouping that was in our algorithm in the slides. And I'm going to just go ahead and put a curly brace here so I won't get confused later.

1:49
All right. And so what do we want to do as long as currIndex is negative 1, well, or is not negative 1? We want to check if currIndex minus startIndex is a multiple of 3. You've seen how to do this previously where we can subtract currIndex minus startIndex, take that to mod 3 and see if it's equal to 0. And we said if so, we want to do one thing, and if not, we want to do another thing. And as you've probably gotten familiar with by now, if not is going to be else. We want to do some other thing in this case.

2:32
So if so, the text between startIndex and currIndex plus 3 is your answer. So anytime a method knows its answer right away and is done, we're going to return that answer back to whoever called this, which finishes the method and gives back the answer. So we want return and we want the text between startIndex and currIndex plus 3. What method would you use for this? Hopefully, you've become familiar enough with string methods by now that substring is what comes to mind as what you'd want. So dna.substring startIndex, currIndex plus 3. And then we said if not, update currIndex to the index of the next TAA starting from currIndex plus 1. So any time we want to update a variable we're going to use an assignment statement. And we can find the next one after that again using indexOf to search from a specific position, indexOf TAA, from currIndex plus 1, all right? So that just wrote that step down as it was. My brace is not indented right, so I'm just going to fix that. And then if we get to the end of this loop, if we keep going in currIndex

3:49
is negative 1, meaning we couldn't find anymore TAAs, then our answer is going to be the empty string.

3:59
All right, so I'm going to save that.

4:02
going to go over here and I'm going to hit Compile.

4:07
And it says, reached end of file while parsing.

4:10
And oops, I messed up in here somewhere. I had written some test cases before we started. You'll notice that's highlighting this curly brace, and if I scroll up, it's highlighting this curly brace. When I wrote the test cases, I forgot to put the curly brace that ends the method. Fix that really quickly, Compile it, and findGeneSimple. Yeah, I copied and pasted this testing code and changed the method name and didn't change that.

4:40
So we'll fix a couple careless mistakes there.

4:44
This is the danger of copying and pasting. You should never do it.

4:53
Now really it will compile. All right, so I'm going to make one of these. And you'll notice here that I've written a little test method like we've had in the previous ones. Since things are getting a little more complicated with our test cases, I put these comments here just to mark where the codons are. So this ATG is one codon, and then I've just marked here that this CTG is another codon. And so we can see that this TAA here, for example, is a whole codon. And this TAA here is not a whole code on. So we would expect this method to find this gene right here, and similarly for these other ones. So I'm just going to run this method really quickly. Here we go. And for this first one, what did we expect? We expected ATGCGTAATTAA. ATGCGTAATTAA. So we got exactly what we wanted. And notice, this is where that while loop we wrote just came into use, because this looks like a TAA. But we would have found that it wasn't a multiple of three, and we would have kept going and found there actually is one later on. And then similarly, we've done this other string, which is a little bit longer. It has some more things that look like maybe they're TAAs but they're not. But it does have a TAA at the end. So we get that really long gene there. And then at the end here if you look, we don't actually have any TAAs, and we get the empty string, which is what we'd want.

6:29
So there we go, we've turned this algorithm into code using a while loop. And now it will look for any TAA which is a multiple of three, even if there's one that's earlier that's not a multiple of three away, so that we can get the right answer.

Downloads

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

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