LEARN PYTHON

DNA Analysis

In this project, we'll use many of the concepts you've learned throughout the Python course in order to do some DNA analysis for a crime investigation.

The scenario:

A spy deleted important files from a computer. There were a few witnesses at the scene of the crime, but no one is sure who exactly the spy was. Three possible suspects were apprehended based on surveillance video. Each suspect had a sample of DNA taken from them. The computer's keyboard was also swabbed for DNA evidence and, luckily, one small DNA sample was successfully retrieved from the computer's keyboard.

Given the three suspects' DNA and the sample DNA retreived from the keyboard, it's up to you to figure out who the spy is!

The project should have methods for each of the following:

1. Given a file, read in the DNA for each suspect and save it as a string
2. Take a DNA string and split it into a list of codons
3. Iterate through a suspect's codon list to see how many of their codons match the sample codons
4. Pick the right suspect to continue the investigation on

Note: As with professional software development, you should be saving your code very often. As you code, make sure you click the "Save" button below to save your code/changes. Otherwise, you run the risk of losing all your code!

Let's begin!

DNA Analysis

- [ ] **1.** First, take a look at the list of codons on Line 1 of **dna.py**. It contains the three codons that were retrieved from the computer's keyboard. It will be up to you to match these codons to the suspects' DNA samples.

- [ ] **2.** If you're curious, check out the suspects' DNA samples by clicking on the files titled **suspect1.txt**, **suspect2.txt**, or **suspect3.txt**. It's important that you do not edit them, as that could result in a project that functions incorrectly.

- [ ] **3.** Let's start off by writing the method that will read a suspect's DNA sample.

  On Line 3, start by adding a method called `read_dna`. It should take one parameter, called `dna_file`.

- [ ] **4.** Inside of the method, create a variable called `dna_data` and set it equal to an empty string. This will be the string that will eventually contain a suspect's DNA.

- [ ] **5.** On the next line, still inside of the method, use `with` to open `dna_file` in read-only mode, as `f`. We're using `f` as short for "file." Using "file" would not be allowed since it's a keyword in Python.

- [ ] **6.** Now that the file is open, add a `for` loop inside the `with` block. The loop should iterate through each `line` in `f`.

- [ ] **7.** On the next line, inside of the `for` loop, add `line` to the empty `dna_data` using `+=`.

- [ ] **8.** Finally, to complete the method, return `dna_data` on the next line. This line of code should align with the first line of indented code in the method (which means it's not included in the `with` block).

9. Great! When used, this method will take in a file, read it, add the file's contents to an empty string, and return the updated string. This will come in handy in catching the spy.

10. Next, we'll need a method that will take a string, create a list of codons from that string, and return the list. This will make the DNA analysis much easier later.

    Add a new method called `dna_codons`. It should take one parameter called `dna`.

11. We'll need an empty list to add the codons to. On the next line, inside the method, create an empty list called `codons`.

12. If you previewed a suspect's DNA sample (for example, in **suspect1.txt**), you should have seen that the DNA sample contains a lot of letters (DNA base pairs).

    Codons are 3-letter-long units of genetic code. We'll have to iterate through a suspect's DNA string and chop the string into codons (3 letter strings).

    On the next line, add a `for` loop. The loop should have a `range` from `0` to the length of `dna`. It should also iterate in increments of `3`.

    This documentation shows to specify increments with `range`.

13. Next, we'll want to make sure that the iterator `i` doesn't exceed the length of `dna`, so let's check for that on the next line.

    Inside the `for` loop, add a line that checks if the iterator, when incremented by 3, exceeds the length of `dna`.

14. The line of code that you just wrote will make sure that you don't add a string to the codon list that isn't at least 3 letters long.

    Now, we'll want to add the codon into the `codon` list that is currently empty.

    On the next line, add to the list using the `.append()` method. Since the iterator `i` starts at 0, we'll be adding the first three characters only.

15. Great! Now that the codon list contains all the codons we need, we'll need to return the list.

    On the next line, return `codons`. This line of code will have to be written at the level of indentation that aligns with the `for` loop.

16. Perfect - you just added a method that will iterate through a string, slice it into smaller strings that are 3 letters long, and add them to a list. This functionality will help us match the sample to a suspect's DNA later.

    The next step is to create a method that will iterate through both the sample and a suspect's DNA. The method should count the number of times a codon in the sample matches a codon in the suspect's DNA.

    On the next line, add a method called `match_dna`. It should take one parameter called `dna`.

17. We'll need a way to count the number of times a codon from the sample matches a codon from a suspect's DNA.

    Inside the method, add a variable called `matches` and set it equal to zero. We will increment this variable by 1 every time there is a match.

18. The parameter that this method takes is a list, so we'll have to start iterating through the list first to find matches.

    On the next line, add a `for` loop that iterates through the `dna` list. Call the iterator `codon`.

19. As we iterate through the codons in the suspect DNA's list, we'll have to check if the codon also exists in the sample.

    Inside of the `for` loop, add an `if` statement that checks if the codon is also in the sample list.

20. If a codon in the DNA matches a codon in the sample, then we've got a single match! We'll increment the `matches` variable to reflect that.

    Inside the `if` statement, increment `matches` by `1` using the `+=` operator.

21. Since the `match_dna()` method counts matches, it would be useful if it returned the number of matches as well.

    As the last line of this method, return the `matches` variable.

22. Fantastic! You've added a method that automates a task that would normally take a long time to manually complete. Instead of having to manually match codons for three different suspects, this method does it for us and counts the matches.

    We have most of the methods we need, but let's add one more that will determine if a suspect is the criminal.

    Next, add a method called `is_criminal` that takes a parameter called `dna_sample`.

23. The `is_criminal()` method should use the other methods we've created to determine who the criminal is.

    The first thing the method will do is read in DNA samples and create a string to hold them.

    Inside the method, create a variable called `dna_data`. Set it equal to the result of calling the `read_dna()` method on the `dna_sample` parameter.

24. Now that we have the DNA data in a string, it's time to call the `dna_codons()` method to chop the string into a list of codons.

    On the next line, create a variable called `codons`. Set it equal to the result of calling the `dna_codons()` method with `dna_data` as the argument.

25. Now that we have the codon list, it's time to match the sample with the DNA.

    On the next line, create a variable called `num_matches`. Set it equal to the result of calling the `match_dna()` method with `codons` as the argument.

26. To complete the method, we will have to add in statements that check to see if the number of matches is significant.

    Next, add an `if` statement that checks if the number of matches is greater than or equal to three.

27. If the number of matches is greater than or equal to three, print out the number of matches using string formatting, as well as a message stating that the investigation should continue.

28. Otherwise, the suspect can be set free.

    Add an `else` block that prints the number of matches using string formatting, as well as a message stating the suspect can be freed.

29. That's it! This method will do all the hard work of reading a DNA sample from a suspect, comparing it to a DNA sample from the crime scene, and letting the user know whether the suspect is a criminal.

    For all of this to work, we actually have to call the method `is_criminal()` on the `.txt` files you previewed earlier.

    On the next three lines, outside of any method, call the `is_criminal()` method separately on the three `.txt` files.

30. Ready to find out who the spy is?

    In the terminal, type the following command and hit "Enter" on your keyboard:

    ```
    python dna.py
    ```

    You should see the analysis run and find out who the spy is!