Python Project: Creating a DNA Sequence Class

Classes

A class is a template for a particular kind of object. So far in this course we have met the string, list, tuple, numpy ndarray and several other classes. Each has its “attributes”, or defining variables, and “methods”, or built-in functions that operate on that class. For example, the numpy ndarray class has the attribute “shape” which stores the dimensions of the array, and the method “max()” for finding the maximum value in the array. To work with a class we have to make a specific example or “instance” of the class template. So we could write a=np.ndarray([1, 2, 3]) to make a 1D array with 3 entries. Or we could write b=np.ndarray([]) to make an empty array.

When we create an instance, what actually happens “behind the scenes” is that the class’s __init__ method is called. __init__ sets up all the attributes of the class, ready for use by the methods. All class methods have to have the “self” as an input argument. self simply means “this instance”.

Two python scripts are provided for this project.

- dnasequences.py contains an (incomplete) class definition for the “dnasequence” class. There is an __init__ method, several attributes, an “isdna” method for checking if a sequence is valid (incomplete), and an “isequal” method for comparing 2 sequences.
- project.py imports the dnasequence module and shows how to create instances of the dnasequence class. It also contains some handy sequences (crick_str, crick2_str, watson_str) for testing your code.

The dnasequence class is designed like many other python classes: you can create an empty or non-empty class instance. For example, myseq=dnasequence(“ACGT”) or myemptyseq=dnasequence() are both valid. The optional string argument is put into the “seq” attribute, which is a numpy 1D array.

Task

In Assignment 1 you learned how to read data files. In Lab 2X you devised simple methods for modifying and comparing lists of characters representing DNA sequences. In Lab 4 you were introduced to making your own classes. All of these skills are required for the project. The project is to create a DNA sequence class complete with a set of methods for creating, testing, comparing and cutting them up. The idea is that you can start to analyse real genomic data and, in the process, learn how to design and write more complex, larger-scale software projects.

The tasks are written below. The first 5 tasks are for a Grade 3 (pass) and the last 2 are for higher grades. For each task, your report must include a brief algorithm description in words and pseudo-code.

Program structure

Any class methods you create should go in dnasequences.py. Anything else goes in project.py or, if you like, in a new file. One suggestion is to offload file reading (Task 6) to a separate file. There are 7 tasks (including those for extra credit) requiring a fairly large number of lines of code. Organise it into functions to make it readable and flexible. Write comments for important bits of code. Good coding style is like a well-organised desk: it makes your work more efficient and reduces mistakes.

Tasks

Look at the methods that have already been written in dnasequence.py for help in writing your own methods. The solutions to the Lab exercises can also help you with most of these tasks. It is OK to copy your own code or that in the reference solutions but be careful to ensure that it is fit for purpose. Feel free to work together on how to solve the problems but write your own codes.

Grade 3

1. In the dnasequence class, check if the "seq" attribute contains only valid bases A,C,G or T. You can do this by completing the isdna method:

```
def isdna(self):

  code...

  return True or False
```

The isdna method is already partly written and is called from inside __init__. All you have to do is fill in the code for checking if seq contains valid bases.

2. Create a method that returns the complement of a dnasequence. For example, if a particular instance has seq=["A","G"] then its complement has seq=["T","C"]. Complements are defined in the "complements" attribute, which is of type dict. Dict variables allow you to look up an item given its key. Read more on dict variables in the comments in dnasequences.py, in the python help or online. Your method should return a new instance of dnasequence whose "seq" attribute is complement of "seq" in the original instance.

Use your complement method and the “isequal” method to test if crick_str is the complement of watson_str. Print True (they are complements) or False (they are not complements).

3. Create a method that finds the first pair of non-matching bases in 2 sequences. The method should first check that the sequences are of equal lengths. If they are not, it should terminate the program using sys.exit(). If they are equal lengths, the method should return the index of the mismatching bases. If the method reaches the end the sequences without finding a mismatch it should return -1 (the index of the last value). Use it to find the index of the first non-matching bases in crick_str and crick2_str. Print the result.

4. Add a function to project.py (not a new class method) that reads in ASCII-format genome files. Genome files contain a header line containing a name and format information. The second line is a (very) long string of DNA bases. (If you want to view the file, use a text editor that wraps long lines around the window.) The function has to return a new instance of dnasequence from the sequence in the genome file. Read in the file "genome_01.dat" and print the total number of bases it contains.

EXTRA CREDIT TASKS

Grade 4

5. The data in the file "genome_01.dat" encodes a genome: several genes separated by a separator or "full-stop" signal: "AAAAAAAAAATTTTTTTTTT", 10 A followed by 10 T bases. The first gene does not begin with a separator and the last gene does not end with a separator. Create a method that, given the output of Task 4, returns the first gene. Your method should be able to handle sequences that do not contain a separator.

There are multiple ways to tackle this problem using the string class or the list class methods. The "find" method in the string class is probably simplest. "find" returns the index of the first character of whatever you are looking for. If it doesn't find it, the "find" method returns -1 (the "end" index).

Your method should return a new instance of dnasequence containing the gene sequence. For example, if the input dnasequence has the sequence "CCGATCGAAAAAAAAAAATTTTTTTTTT", your method should return a new dnasequence with seq="CCGATCGA". Print the length of the first gene.

Use your method to create a list of dnasequence instances from genome_01, each containing a single gene. Plot the gene lengths as a bar chart.

Grade 5

6. Read in the file "genome_02.dat". Use your dnasequence class methods to find "swap" mutations in each gene of genome_02: single bases that do not match genome_01.dat. You can assume that the genes are of matching lengths and in the same order in each genome. Make a scatter plot of the number of swap mutations per gene against gene length. Is there a trend?

Reporting

The report must contain the following sections:

1. dnasequence class description. A description of the class methods that you created. This should be a short overview, with pseudo-code algorithms, without references to the actual source code. It should be possible to understand how the program works without looking at the code. You do not need to describe the function that reads in files.
2. If attempting the Grade 4 and 5 tasks, include the plots you created.

Preferably use the student portal for reporting. Upload the source code as separate files - do not paste into the report. A compressed/zipped folder of files is fine. Comment your source code well and write docstrings for each method and function. Tasks 3-8 require you to print information to the screen. Make the print statements informative and easy to read.

Grading

The assignment is worth up to a Grade 5. The report is worth 10 points for the description of your code. Each Grade 3 task is worth 5 points: 3 for making a reasonable attempt at solving the problem, 1 for good style and clear commenting, and 1 for correctness of the result (including the plots if doing tasks 5 and 6).

For Grade 3 you need 20 out of a possible 30 points, of which at least 5 have to be from the report.

For Grade 4 you need 25 out of a possible 35 points, of which at least 3 have to be from task 5.

For Grade 5 you need 30 out of a possible 40 points, of which at least 3 have to be from task 6 and at least 3 have to be from task 5.

You will have a chance to resubmit. Note that getting the correct answer is only worth up to 6 points. Grades 4 and 5 are achievable simply by making a good attempt at solving tasks 5 and 6. The Table shows the marking scheme.

Table: Required and maximum available points for each grade

	Grade 3	Grade 4	Grade 5
Report	5/10	5/10	5/10
Tasks 1-4	/20	/20	/20
Task 5		3/5	3/5
Task 6			3/5
TOTAL (required/max)	20/30	25/35	30/40