CS204
Assignment No. 2

Purpose: Gain experience in using and manipulating python lists.

Background on DNA, Restriction Enzymes, and PCR:
This background is interesting, but not really needed to do the assignment. There are some good stories here, but if you want to get to the assignment, you can skip this stuff.

In this assignment you'll simulate cleavage (cutting) of DNA by *restriction enzymes*. DNA is made up of sequences of molecules known as nucleotides that are linked together to form a DNA strand. Biochemists use the letters 'A', 'C', 'T', and 'G' (standing for adenine, cytosine, thymine, and guanine) to represent a linked sequence of nucleotides within a DNA strand. Restriction enzymes are molecules that recognize and cut very specific "target" nucleotide sequences. These target sequences are commonly referred to as restriction sites.

Three scientists shared the Nobel Prize in 1978 for the discovery of restriction enzymes. They're also an essential part of the process called PCR *polymerase chain reaction* which is one of the most significant discoveries/inventions in chemistry and for which Kary Mullis won the Nobel Prize in 1993.

You can see animations and explanations of both restriction enzymes and PCR at DnaTube and Cold Spring Harbor Dolan DNA Learning Center.

Restriction Enzymes

(source: http://www.astbury.leeds.ac.uk/gallery/leedspix.html)

PCR: Polymerase Chain Reaction

(source: http://www.roche.com/pages/facets/1/pcr1.jpg)

Kary Mullis, the inventor of PCR, is an interesting character. To see more about him see this archived copy of a 1992 interview in Omni Magazine, this 1994 interview as part of *virus myth*, his personal website which includes information about his autobiography *Dancing Naked in the Mind Field*, though you can read this free Nobel autobiography as well.

The simulation is a simplification of the chemical process, but provides us a nice starting point for our exploration of python and also provides us with a basis for future assignments.

The Assignment:

In project #1 we represented DNA strands using Python strings. Due to Python's ability to take slices of strings, along with a large set of string methods, using strings to represent DNA strands made our job as programmers very easy. But nothing is free. That ease of programming came at a cost. In particular, whenever we modified a string, Python actually creates an entire new string. For example, say you had a DNA strand that contained a million characters and you simply wanted to append an additional character to the end, or maybe change a single character in the middle. In any case, Python would create an entirely new string – this can be expensive when our strings are long (as they can be when we represent DNA).

Thus for this assignment, we are going to replace our underlying representation. Rather than using a string, we will change to using a Python list. Python lists are mutable, this allows us to perform certain operations and change the list rather than creating new lists. One must be careful, as some List operations still create new lists (e.g., list concatenation). Our goal in this project is to use mutation operation whenever possible. For example, if we want to change a character in the middle of a strand, we can use indexing notation and assignment, rather than slicing and concatenation; the former will mutate an existing list while the latter will create a new list.

As with project 1, this assignment is a simplification of the chemical process, but provides an example of the DNA manipulation using simple Python lists. We will use a python string containing the single letters 'A', 'C', 'T', and 'G' to represent the sequence of nucleotides within a DNA strand.

Sample representation of a DNA molecule "ACTTGATTGGGTTGCTTGCC":

A	C	T	T	G	A	T	T	G	G	G	T	T	G	C	T	T	G	C	C

Note that the list should the same size as the strand that we are representing, and thus there will be no unused list elements after the nucleotides within the strand.

Cleavage by a restriction enzyme will simply mean removing a specified sequence from our DNA strand. Cleaving requires a target nucleotide pattern. The sequence to be removed will be the nucleotides that reside between matching pairs of the target pattern. A cleave will remove all nucleotides beginning after the first occurrence of the target pattern and continue through the end of the second (matching) target pattern. We will write 2 functions cleave and cleaveAll, which will implement cleaving in 2 different forms. Cleave will cleave only the first such sequence while cleaveAll will remove all such sequences between matching pairs within the DNA strand.

Thus, for e.g. if we invoke cleave("TTG") on the above DNA strand, we would remove the first set of highlighted characters and the resultant DNA strand will be :

A	C	T	T	G	G	G	T	T	G	C	T	T	G	C	C

Notice how the data was shifted down to fill the hole created by the deleted sequence.

In comparison, if we invoke cleaveAll("TTG") on the original DNA strand above, we would remove both sets of highlighted characters and the resultant DNA strand will be :

A	C	T	T	G	G	G	T	T	G	C	C

The nucleotides in our DNA are character values, so we will need to break input strings into individual characters in each method.

dna_strand.py describes all the functions to be implemented.

Functional Specifications:
You will be supplied the module definition file: **dna_strand.py.** The file contains the declaration of a set of functions to manipulate lists representing DNA. Your task will be to implement all the functions within the class definition that the file contains. You have also been provided with an initial test program: **project2.py**. You should add code to the project2.py file to fully test your DNAStrand class.

Implementation details:
Here are a few notes that might be helpful:

1. You are to download the files that are provided with this specification. The files contains starter code for this project. Once you download the files, you should create a new PyCharm project and place the provided source files in the **src** directory of the new project. Then refresh the project in PyCharm. The code as provided interprets cleanly, though it will report many errors when it runs since you still need to add correct code for all the class methods.
2. You will be performing some operations with python strings. Python string operations are described at http://docs.python.org/library/stdtypes.html#string-methods.
3. You will be performing many operations with python lists. Python list operations are described at http://docs.python.org/tutorial/datastructures.html and http://docs.python.org/library/stdtypes.html#sequence-types-str-unicode-list-tuple-bytearray-buffer-xrange and http://docs.python.org/library/stdtypes.html#mutable-sequence-types. Note that when you write your dna_strand methods, you are expected to manipulate the list of characters that you have created. You are not allowed to convert your list of characters into a string object and then operate on that

string object. In particular, when you are asked to compare two DNAStrands or find nucleotides sequences, you must operate directly on your list. For the most part, your usage of strings should be limited to getting their length (with the len() method) or accessing individual characters (using array indexing with [ ] or a for loop).

4. The dna_strand.py specifies that you are suppose to throw an exception if someone attempts to access a part of your DNAStrand that is not within the bounds of the DNA strand that you are currently representing. To throw an exception in python, you can use the follow statement: `raise IndexError("index out of range");` Note that you do not need to specify that the method may throw an exception; you can simply throw the exception if the error condition occurs.
5. Since we are using a python lists in the DNAStrand class, the python interpreter takes care of creating the list for you whenever a DNAStrand object is created. Your constructors should not have any "**new**" operations (I state this for the benefit of the Java programmers in the class).
6. At the top of the dna_strand.py file, you must identify yourself as a graduate or undergraduate student by setting the class variable appropriately. All graduate students must implement a set of additional class methods, as per the .py file.
7. Even though we plan to use this class to represent DNA strands, we will not restrict the user to only using the characters 'A', 'C', 'G', and 'T'; rather we will allow any characters to be stored in the DNA strand. It is okay for your code to be case sensitive, but if you like the extra challenge you can make your code case insensitive.

Submission for grading:

When you have completed your work on this assignment, please submit the two source files for grading: **dna_strand.py,** and **project2.py.** Submit ONLY the source files – please do not submit a zip file containing your entire project. You submit the files by visiting the assignment page in Oak and attaching the files (one at a time) by clicking on the "Browse my computer" button and finding the file to attach.

Grading:

This project is worth 50 points. Your grade on this project will be based on the following:

1. The correctness of your methods implemented in the dna_strand.py file.
2. The use of good programming style.
3. The thoroughness of your testing performed in the project2.py file. Note: the thoroughness of your testing often has a great impact on the correctness of your code (see item #1 above).

You should also review the syllabus regarding the penalties for late programming assignments.

Acknowledgements:

This assignment is loosely based on a project by Owen Astrachan at Duke University which is in the collection of ACM SIGCSE Nifty Assignments.