Lab for Bioinformatics Tools – Spring 2023
Exercise 1: regex and file parsing

Given: 3.4.23, Due: 17.4.23

Submit a single zip file with your code file. The name format of the submitted file should be "123456789_ex1.zip" – replace "123456789" with your ID number.

Python guidelines:

- Use the **provided skeleton** to submit your code.
- Test your code using python 3.7 and up.
- Code should be readable and well organized. Add comments where needed.
- Code should be efficient.
- It is advised to break code into functions wherever possible.
- **Use REGEX module and not RE! https://pypi.org/project/regex/**

A restriction enzyme is an enzyme that cleaves DNA at specific recognition sites known as restriction sites. Many restriction enzymes exist, and they vary in their **recognition site** and the **pattern of cleavage**. Some enzymes leave "blunt ends" – in which the enzyme cuts both strands at the same position in both strands. Others leave "sticky ends" in which a stretch of unpaired nucleotides remains in the end of each cleaved strand (overhangs).
Examples (from Wikipedia; the patterns correspond to the patterns `CCC|GGG` and `G|AATTC`, respectively, the green lines represent the exact cleavage site):

Blunt ends Sticky ends

1) The restriction enzyme's pattern is mainly composed of the four nucleotides ACGT but also of the following:
   a. The restriction (cleavage) site is denoted by a pipe – "|" (shift+"\" in your keyboard!)
   b. Ambiguous sites, i.e., the letter N or W which represent one of several possible nucleotides: N=C/G/T/A and W=A/T.
      For example, the recognition pattern of the restriction enzyme 'EcoRII' is `CCWGG` (so it may recognize either `CCAGG` or `CCTGG` in the DNA strand.
   c. Repetitive characters. The number of repetitions will appear right after that character. For example, in the pattern `GGCCN3|C`, N appears 3 times exactly. Note that the repetitive character may appear a decimal number of times and <u>we do not want to be restricted to a single digit</u>!
   d. Hint – use re.sub

Overall, the *restriction_pattern* is over the alphabet "ACGTWN|" and numbers. Complete the function *REpattern2regex* that receives a pattern *restriction_pattern* and returns its regex formulation as a list of two strings. The two strings represent the regex of the patterns before and after the pipe. In case that an invalid character is included in the pattern, raise a ValueError exception (with a message to your choice).

2) "Sticky ends" make it possible for two cleaved DNA sequences (that were cut with the same restriction enzyme, or at least have complementary overhangs) to bind. In the following example, since AATT and TTAA are complementary, the blue and orange overhangs can bind, resulting in the deletion of the black region:

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

Write a function *chop_by_restriction_enzymes* that receives a nucleotide sequence, *seq*, and a pattern, *restriction_pattern,* and returns a list of all non-nested chops of *seq* between every two restriction enzyme's pattern.
For example, the input and output should be:

- Inputs: "`TAGAATTCAGCAGCATGCAGCTCAGAATTCA`", "`G|AATT`"
  Output: `['AATTCAGCAGCATGCAGCTCAG']`
- Inputs: "`TAGAATTCAGCAGAATTCATGCAGCTCAGAATTCA`", "`A|ATT`"
  Output: `['ATTCAGCAGA', 'ATTCATGCAGCTCAGA']`

Mandatory: use regex patterns to find the sequences between every two restriction sites. If there are zero or one restriction site, return an empty list. If more than two appear, the function should return the non-nested sequences.

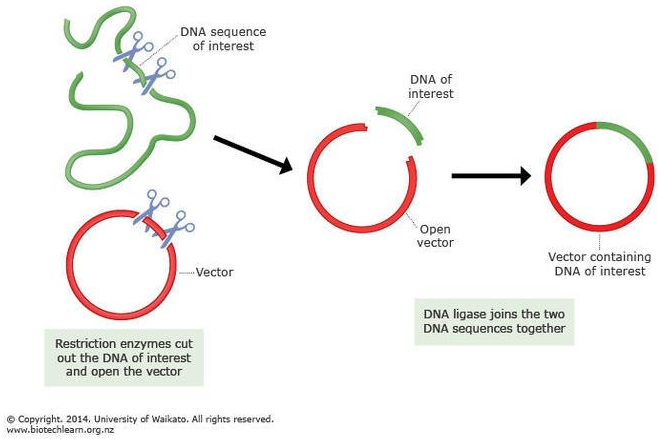
Notes:

(1) *seq* is a valid DNA sequence (i.e., over the alphabet **ACGT**, **uppercase,** and **non-empty**).
(2) Note that if the pattern appears in the opposite strand, it will be reversed in the forward orientation. For simplicity, search for restriction sites in the forward strand only.
(3) The most efficient way to return all non-overlapping strings is to use re.findall and .*? that restricts the search to the shortest string that matches. To match but not return prefix or suffix, use the "(?<)" and "(?=)" patterns. See lecture slides (And google).
(4) You may assume that matches are **consecutive** (appear one after the other) **and not contain one another**. For such cases we would require special treatments that are case-specific! That means that you may rely on the result of re.findall for shortest strings.
(5) You can check your function with examples from the Wikipedia page (you may find many others online!).

3) Change the function *chop_by_restriction_enzymes*: add a default parameter *additional_pattern*, set to *None*, and update the function such that if a value is passed, the function would search for non-overlapping strings between *restriction_pattern* and *additional_pattern* (in that order). Make sure that if the value is None- the previous functionality is maintained! Again, disregard matches in the opposite strand and assume matches are consecutive and not self-containing.
4) In cloning, the goal is to insert a certain gene (a DNA insert) into an expression vector. To be able to clone a DNA insert into an expression vector, both have to be treated with restriction enzymes that create compatible sticky ends. However, if the same enzyme is used to cleave both ends of both the insert and the vector, the insert might ligate in an opposite orientation and will lose directionality of the expression (in the example above, the black region may be chopped and

ligated in the opposite direction because the loose ends are compatible). To this end, two enzymes that induce different overhangs are required.

Write a function *can_be_cloned* that receives two sequences, *vector* and *insert*, and two patterns of restriction enzymes, *pattern1* and *pattern2* (as described in question 1). The function should return *true* if the insert can be ligated to the vector in the correct orientation with no confusion (i.e., that the two overhangs are different) and else *false*.



- **The order of *pattern1* and *pattern2* doesn't matter, consider both possibilities**
- **The vector is circular (meaning that the end is followed by the beginning of the sequence), but you may assume that it's not (because researchers usually have a specific part where they want the insert to be cloned)**

5) A fasta file is a format that contains a list of nucleotide sequences. Every record in the file contains a header (the name of the sequence) and a sequence. The header of a new record always begins with a '>' at the beginning of the row, **which is not a part of the name**. The sequence of the record begins in the next line, and may spread over several rows until a new record, beginning with a '>' appears or until the end of the file (if this is the last record). Note that there is one sequence per header, no matter how many lines it spreads over – that is, the sequence is the concatenation over these rows. For example:

```
>Human HBB-1
CCGGGAAAAGGGTCCCACCTGACGGGAAATTGGAG
TGGAGGGCGACAAATCATCTAGTTTAAGTTCGGCCT
GTCACACATTGTACATGAGATAAAAGGCA
>Mouse pro-Insulin
ACCCTGGTCTGAACTAGTGAATTTCCTTCTACGTACAGAGTCGACAACGAGC
>Monkey HBB-2
TGCGGGACATGTCACAGATAACAGAGATCGAAGCTGCATCCGTATGTTCGTTCGGGCACATTGTGAA
AGACATCAACGTACTTCGCCCTTTGGCAGGCGATGCGGGACATGTCACAGATAACAGAGATCGAAGC
TGCATCCGTATGTTCGTTCGGGCACATTGTGAAAGACATCAACGTACTTCGCCCTTTGGCAGGCGA
```

Write a function *which_inserts_can_be_cloned* that receives the path to a fasta file and two restriction enzymes' patterns. The first record in the fasta would be the vector. The other records would be of potential inserts. The function should return a list with the names of records which can be inserted into the vector by the denoted enzymes.

An example fasta file cloning_data.fa is attached to this exercise.