Practical #1: Python Refresher

Exercises taken from (mostly from) http://rosalind.info/problems/topics/

1. **String Algorithms: Counting DNA Nucleotides**

A string is an ordered collection of symbols selected from some alphabet and formed into a word. The length of a string is the number of symbols that it contains.

Strings and Biology: For example, a DNA strand is represented by a string over the alphabet of nucleotides containing the symbols 'A', 'C', 'G', 'T'.

"ATGCTTCAGAAAGGTCTTACG" is an example of a DNA string of length 21.

Write a Python function that takes as input a DNA string *s* of length at most 100 nucleotides, and returns 4 integers (separated by spaces), counting the respective number of times that the symbols 'A', 'C', 'G', 'T' occur in *s.*

Sample input:

```
AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGC
```

Sample output:

```
20 12 17 21
```

2. **String Algorithms: Calculating AT content**

Write a Python function that calculates the AT content of a given DNA string (in percentage).

Sample input:

```
ACTGATCGATTACGTATAGTAGAATTCTATCATACATATATATCGATGCGTTCAT
```

Sample output:

```
50%
```

3. **String algorithms: Complementing a strand of DNA**

In DNA strings, symbols 'A' and ’T' are complements of each other, as are 'C' and 'G'. The *reverse complement* of a DNA string *s* is the string denoted by s^c formed by reversing the symbols of *s,* then taking the complement of each symbol (*e.g.,* the reverse complement of "GTCA" is "TGAC").

Write a Python function that takes as input a DNA string *s* of length at most 1000 basepairs, and returns s^c , its reverse complement.

Sample input:

AAAACCCGGT

Sample output:

ACCGGGTTTT

4. **String algorithms: Counting point mutations**

Preamble: Evolution as a sequence of Mistakes

A mutation is simply a mistake that occurs during the creation or copying of a nucleic acid, in particular DNA. Because nucleic acids are vital to cellular functions, mutations tend to cause a ripple effect throughout the cell. Although mutations are technically mistakes, a very rare mutation may equip the cell with a beneficial attribute. In fact, the macro effects of evolution are attributable by the accumulated result of beneficial microscopic mutations over many generations.

The simplest and most common type of nucleic acid mutation is a point mutation, which replaces one base with another at a single nucleotide. In the case of DNA, a point mutation must change the complementary base accordingly (*e.g., a C-G pair is changed into an A-T pair*)

Two DNA strands taken from different organism or species genomes are homologous if they share a recent ancestor; thus, counting the number of bases at which homologous strands differ provides us with the minimum number of point mutations that could have occurred on the evolutionary path between the two strands.

We are interested in minimizing the number of (point) mutations separating two species because of the biological principle of parsimony, which demands that evolutionary histories should be as simply explained as possible.

Problem: Given two strings *s* and *t* of equal length, the Hamming distance between *s* and *t* denoted

$d_H(s,t)$, represents the minimum number of symbol substitutions needed to change *s* into *t,* and is equal to the number of corresponding symbols that differ in *s* and *t.*

Write a Python function that takes as input two strings *s* and *t* of equal length, not exceeding 1000 basepairs, and returns their Hamming distance.

Sample input:

```
GAGCCTACTAACGGGAT
CATCGTAATGACGGCCT
```

Sample output:

```
7
```

5. **Reading and writing files: Writing a FASTA file**

The FASTA file format is a commonly-used DNA and protein sequence file format. A single sequence in FASTA format looks like this:

```
>sequence_name
ATCGACTGATCGATCGTACGAT
```

We note that `sequence_name` is a header that describes the sequence (the greater-than symbol indicates the start of the header line).

Often, the header contains an accession number that relates to the record for the sequence in a public sequence database. A single FASTA file can contain multiple sequences, like this:

```
>sequence_one
ATCGATCGATCGATCGAT
>sequence_two
ACTAGCTAGCTAGCATCG
>sequence_three
ACTGCATCGATCGTACCT
```

Write a Python program that will create a FASTA file for the following three sequences – make sure that all sequences are in upper case and only contain the bases A, T, G and C.

Sequence header	DNA sequence
ABC123	ATCGTACGATCGATCGATCGCTAGACGTATCG
DEF456	actgatcgacgatcgatcgatcacgact
HIJ789	ACTGAC-ACTGT--ACTGTA----CATGTG

6. **Reading and writing files: Writing multiple FASTA files**

Use the data from the previous exercise, but instead of creating a single FASTA file, create three new FASTA files – one per sequence. The names of the FASTA files should be the same as the sequence header names, with the extension *.fasta*.

7. **Reading and writing files: Conditional tests**

Download the file *data.csv,* containing some made-up data for a number of genes. Each line contains the following fields for a single gene in this order: species name, sequence, gene name, expression level.

The fields are separated by commas (hence the name of the file – *csv* stands for Comma Separated Values). Think of it as a representation of a table in a spreadsheet – each line is a row, and each field in a line is a column. All the questions below use the data read from this file.

Q1. Print out the gene names for all genes belonging to *Drosophila melanogaster* or *Drosophila simulans*.

Q2. Print out the gene names for all genes between 90 and 110 bases long.

Q3. Print out the gene names for all genes whose *AT* content is less than 0.5 and whose expression level is greater than 200.

Q4. Print out the gene names for all genes whose name begins with "k" or "h" except those belonging to *Drosophila melanogaster.*

Q5. For each gene, print out a message giving the gene name and saying whether its *AT* content is high (greater than 0.65), low (less than 0.45) or medium (between 0.45 and 0.65).

8. **String algorithms: Locating Restriction Sites [BONUS QUESTION]**

Preamble: The war between viruses and bacteria has been waged for over a billion years. Viruses called bacteriophages (or simply phages) require a bacterial host to propagate, and so they must somehow infiltrate the bacterium; such deception can only be achieved if the phage understands the genetic framework underlying the bacterium's cellular functions. The phage's goal is to insert DNA that will be replicated within the bacterium and lead to the reproduction of as many copies of the phage as possible, which sometimes also involves the bacterium's demise.

To defend itself, the bacterium must either obfuscate its cellular functions so that the phage cannot infiltrate it, or better yet, go on the counterattack by calling in the air force. Specifically, the bacterium employs aerial scouts called restriction enzymes, which operate by cutting through viral DNA to cripple the phage. But what kind of DNA are restriction enzymes looking for?

The restriction enzyme is a homodimer, which means that it is composed of two identical substructures. Each of these structures separates from the restriction enzyme in order to bind to and cut one strand of the phage DNA molecule; both substructures are pre-programmed with the same target string containing 4 to 12 nucleotides to search for within the phage DNA. The chance that both strands of phage DNA will be cut (thus crippling the phage) is greater if the target is located on both strands of phage DNA, as close to each other as possible.

By extension, the best chance of disarming the phage occurs when the two target copies appear directly across from each other along the phage DNA, a phenomenon that occurs precisely when the target is equal to its own reverse complement. Eons of evolution have made sure that most restriction enzyme targets now have this form.

Problem:

A DNA string is a reverse palindrome if it is equal to its reverse complement. For instance, “GCATGC” is a reverse palindrome because its reverse complement is “GCATGC”.

Write a Python program that takes as input a DNA string of at most 1000 basepairs and prints the position and length of every reverse palindrome in the string of length between 4 and 12. You may return these pairs in any order.

Sample input:

```
TCAATGCATGCGGGTCTATATGCAT
```

Sample output:

```
4 6

5 4

6 6
```

```
7 4

17 4

18 4

20 6

21 4
```

9. **Lists and Regular Expressions [BONUS QUESTION]**

Here's a list of made-up gene accession names:

```
xkn59438, yhdck2, eihd39d9, chdsye847, hedle3455, xjhd53e, 45da,
de37dp
```

Write a program that will print only the accession names that satisfy the following criteria – treat each criterion separately:

- contain the number 5
- contain the letter *d* or *e*
- contain the letters *d* and *e* in that order
- contain the letters *d* and *e* in that order with a single letter between them
- contain both the letters *d* and *e* in any order
- start with *x* or *y*
- start with *x* or *y* and end with *e*
- contain three or more numbers in a row
- end with *d* followed by either *a*, *r* or *p*

10. **Dictionaries: DNA translation [BONUS CRYPTIC QUESTION]**

Write a program that will translate a DNA sequence into protein. Your program should use the standard genetic code which can be found at this URL .