Individual Programming Project Option 1: Search pattern repetition in DNA

A DNA string is a sequence of the bases `a`, `c`, `g`, and `t` in any order, whose length is usually a multiple of three. In reality, it is not necessarily a multiple of three, but we will simplify it as such for discussion. For example,

`aacgtttgtaaccagaactgt`

is a DNA string with a length 21 bases. Recall that a sequence of three consecutive letters is called a codon. Assuming the first codon starts at position 1, read *left-to-right*, the codons are

`aac`, `gtt`, `tgt`, `aac`, `cag`, `aac`, and `tgt`

Those of you who know a little about genomics know that the open reading frame can be shifted to get a different set of codons. I want any of you who know this much to assume for discussion simplicity that there is only one open reading frame – the one starting at position 1.

Instructions

Write a Python program, which will be given three command line arguments:

- a pattern string; that is, any permutation of the letters `a`, `c`, `g`, or `t` of any length,
- a positive integer ***N***, as well as
- the pathname of a dna file.

Generally, the dna file should be a text file containing one or more DNA strings, each on a single line with no space characters of any kind except for the terminating newline character at the end of the line. Each line is just a sequence of the bases `a`, `c`, `g`, and `t` in any order.

The program should do the following tasks:

- print each line in which the pattern string is repeated ***N***-times later in the line at starting positions that are multiples of the pattern string's length apart.
- report the total number of times the pattern is repeated in this fashion.

For example, if the program is named `searchrepetition.py`, and an end-user types in the command line

```
$ ./searchrepetition.py ccc 4 dna_example_1
```

The program searches for the codon `ccc` as every fourth three-letter sequence. Suppose `dna_example_1` contains the DNA string below

```
acgtttgggcccagctctccgccctcacacacaccccggggt
```

which is broken up into groups of three letters here for ease or reading,

```
acg ttt ggg ccc agc tct ccg ccc tca cac aca ccc cgg ggt
```

it satisfies the example requirement specified by the user, since `ccc` is the fourth three-letter sequence and it even reoccurs twice at positions that are multiples of three apart,

```
$ ./searchrepetition.py ccc 4 dna_example_1
Searching for pattern ccc every 4 positions in file.

String: acgtttgggcccagctctccgccctcacacacaccccggggt
Pattern is repeated: 2
```

whereas this one in a different dna file does not:

```
acg ttt ggg ccc agc tct ccg gcc cca cac aca ccc cgg ggt
```

where although `ccc` occurs twice later in the line, the first reoccurrence is not a multiple of three positions from the first one. Therefore if the file does not find any repetition of this pattern at all in the dna file, it must print a message to the user instead stating that no sequence is found.

```
$./searchrepetition.py ccc 4 dna_file_example2

Repetition of codon ccc every 4th position not found.
```

Error checking:
The script should check that the dna file has only the letters `a`, `c`, `g`, and `t` and no other letter characters. If it does not satisfy this constraint, the script should output an appropriate user error message and then exit.

The script does not have to check that the file contains a number of characters equal to a multiple of three nor if the file ends with a newline character, where the number of characters is equal to a multiple of three plus 1. It does not have to satisfy this constraint, but for some extra credit for the project, the script could output an appropriate user error message and then exit.

Testing:
Use the DNA text files as test files for your program, located in the Linux Lab network in the `cs132` course directory:

```
/data/biocs/b/student.accounts/cs132/data/dna_textfiles
```

The program should be thoroughly tested; you can create your own sample dna files based on the ones provided.