Detecting "cag" Repeats in DNA

Real-time access to and analysis of DNA sequences

Due: Wednesday Oct. 19th

Scenario: It is the year 2020. medPing©, the medical chip that "pings" your cell depending on your vital signs, has really caught on such that all citizens *must* be "enrolled" and use an embedded medPing chip (under their skin) in order to receive universal health care coverage. Our company, medPing, has recently acquired the company GeneSeq, the innovators of real-time gene sequencing. Your task is to implement a prototype that combines the GeneSeq technology with our company's software. Specifically, our targeted version of GeneSeq technology performs real-time DNA sequencing of the gene associated with Huntington's disease[1]. Mutant forms of this gene can produce too many copies of the repeating DNA sequence "**CAG**". Real-time detection of an increase in **CAG**-repeats is considered essential in early intervention to fight the disease. Your task is to write a program that (i) reads in files of DNA sequence, (ii) reports **GC**% of that sequence, and/or (iii) reports the number of (repeating) **CAG** sequences.

Huntington's disease is caused by a dominant gene that codes for the protein "huntingtin" (spelled that way deliberately). The disease has a late onset after sexual maturity and manifests itself with progressive deterioration of the nervous system. Mutant (dominant) "huntingtin" has an extra long region of **repeated glutamines**, the amino acid equivalent of the DNA sequence "**CAG**". This is called a "polyglutamine region" and the DNA that codes for it is called a "trinucleotide repeat sequence". "Repeat" is the important phrase. In normal recessive huntingtin there may be 11-35 repeated glutamines or "**CAG**". In the dominant mutant (bad) form there may be 36-121 and the number of sequences may multiply during the lifetime of an individual (see Table 1 below for a complete summary). The extra glutamines may interfere with gene regulation in neurons although the exact function of huntingtin is not yet known.

Repeat count	Classification	Disease status
<= 26	Normal	Unaffected
[27 – 35]	Intermediate	Unaffected
[36 – 39]	Reduced Penetrance	+/- Affected
> 39	Full Penetrance	Affected

Table 1. The range of the number of successive **CAG**-repeats in the huntintin gene sequence and the associated classification and disease status.

In addition to Huntington's disease, there are many other trinucleotide repeat diseases, some of which are glutamine repeat diseases. If you are interested you may want to do a literature search using "trinucleotide repeat" or "glutamine repeat" to read about other associated problems with this phenomenon.

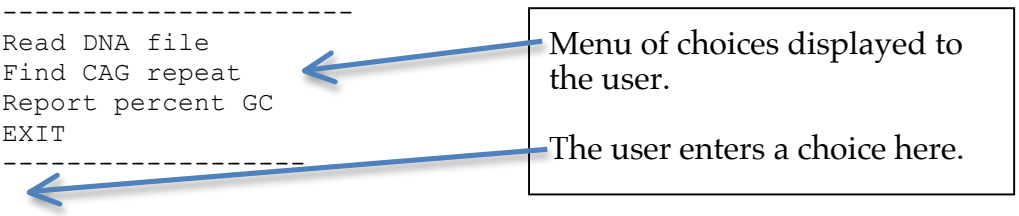
METHOD

Your program, a prototype of a larger version that will be implemented depending on the results of this effort, will be controlled by the user's responses to a menu of choices. In the future, the software will first display a menu on the user's cell phone screen (here simulated on the

[1] See wikipedia: `http://en.wikipedia.org/wiki/Huntington%27s_disease`

console), the user will then text (simulated by the keyboard) a choice back to your software, and your software will perform the desired action, displaying any results and/or error messages back to the user's cell. The user, via the cell (keyboard), can continue operation of your software *until* texting a request to EXIT (choice "4" on the menu). The menu is shown below:

```
-----------------------------
  1  -  Read DNA file
  2  -  Find CAG repeat
  3  -  Report percent GC
  4  -  EXIT
-----------------------------
ENTER:
```



A choice of **1** will prompt the user to enter a filename of DNA to open and read, for example:

gene0.fna

Once a file of DNA has been successfully opened and read by your software, the program will return control to the menu again and the user may then select choice **2** to find and report the sequence, location, and number of CAG repeats. For example, a sequence of: "ATATCA**CAG**CAG**CAG**TTA" has three (3) CAG repeats while the sequence of: "ATT**CAG**TTA" has only one (1) CAG repeat. A choice of **3** reports the percentage of the total of (**C**'s and **G**'s) combined over the entire length of DNA.

INPUT

Input comes from two places. First, the user is in constant contact with your software via the cell. Text messages from the cell (in this program simulated by the keyboard) are either (i) integer **choices** from the menu or (ii) **filenames** (string) of DNA sequence files. Second, the software reads DNA input files. *Note*: all files of DNA (e.g., "gene0.fna") *must* be placed in the same directory as your source code (e.g., yourA4.py). (Note: If you decide to make up your own test file, I recommend that you first make a copy of one of the existing test files and then rename your copy, e.g., gene5.fna, but of course keep all the input files together in the same folder with your source code).

Files of DNA sequence may contain DNA sequence in lowercase *or* uppercase, thus, you must force all DNA sequence to lowercase and thus, you should search for "**cag**", *not* "CAG"). Files of DNA are in FASTA format, which means (i) they start with a header (initial) line (which is not DNA sequence and should not be part of the CAG analysis) and (ii) FASTA files have a **.fna** file extension. The files ("gene0.fna" to "gene3.fna") represent different possibilities. For example, gene0.fna has one set of cag-repeats and no others. However, the file gene1.fna has a single cag (but that cag is not immediately followed by another cag, thus this sole cag is *not* a cag–repeat so you should keep searching); further down the string in this same file gene1.fna you will find a series of cag's; **your program should report the first set of repeating cag's** (where "repeating" means more than one, directly repeating cag. An additional file gene2.fna does not have any cag's. Obviously, you should first get your program working on file gene0.fna before trying others. *(Notice how I have created a "**test suite**" of files to test. You must become proficient in creating test suites to exhaustively test all your programs; what other types of test files could you create for this assignment?)*

OUTPUT

Sample outputs are shown on the last page of this specification and as demonstrated in class.

PROGRAM DETAILS

You *must* obtain the "Starter Kit" from the onCourse (moodle) site. Download this and un-zip!

You should play with the software as given in **`a4_yourName.py`**. I have left you comments to suggest where you should continue. Work on the project in small steps; each time printing out what you have solved so far. For example, to test if you can find your first **`cag`** triple, you might want to print out where you find it (if in fact you did find a "cag"):

```
// start looking in the string DNA at start(0) for the substring "cag"
startCAG_bp = DNA.find( "cag", 0 )

print("CAG found at %d (bp)\n\n" % startCAG_bp)
```

You will use string functions extensively. Hint: you *might* want to read up and practice with examples of: `find()`, `len()`, indexing into a string (`DNA[3]`), and the use of slices, e.g., `DNA[start:end]`.

The string `find` methods return the character (or in DNA-speak, the "base pair" or "nucleotide") locations of where a pattern was found. *Note*: all strings start counting at zero (0), thus the initial character in any string is character (base pair) #0, the next character is #1, #2, etc.

`attcagcagt`
`0123456789`

In the example above, the length of the string is ten (10) characters (or in DNA-speak, "10 base pairs" or "10 nucleotides". Note that one bp = one character). The first **`cag`** substring is found at location #3. The second **`cag`** is found at location #6. These two **`cag`** substrings form two (2) repeating **`cag`**s starting at base pair location 3 and ending at 8 bp for a total of length 6 bp.

If a string's `find` method (e.g., in the variable called `DNA`, that is, `DNA.find()`) *cannot* find the pattern, the `find()` method returns a `-1`. For example, here is part of my solution. Note: you do not have to use this code; this is but one (potential) partial solution …

```
REPEAT = "cag"    # use a constant for the cag repeat

# assume this is the initial time we've looked for "cag"
startLookingHere = 0
 :
 :
startCAG_bp = DNA.find( REPEAT, startLookingHere )
while ((startCAG_bp != -1) ):
      print ("--------------------")
      print ("Found STARTING CAG at: ", startCAG_bp)
        :
        :
```

Your program should report the first set of repeating `cag`'s (where "repeating" means two (2) or more). So if you were searching the following string of DNA, note that you are to report the first set of **repeating `cag`**'s (12-17 bp). Thus, note that we "skip" the isolated **`cag`** at 5 bp (not a repeating `cag`), but once we find a set of repeating **`cag`**'s (12-17), we can report this (and we need *not* continue to find the other repeating set of **`cag`**'s, even if the later set of repeats was longer).

"cag" doesn't repeat, so continue

Once you find two or more, you can report *all of this group of repeats* and stop;

so no need to continue to find these

```
agccccagcattcagcagtttttttcagcagcagtt
012345678901234567890123456789012345

Length(bp): 36
     :
     :
Time to find CAG repeats ...

2 Repeating CAGs found from 12 - 17 (bp)
```

In order to establish a solid algorithm ("recipe"), you *must* play with various strings ***on paper***. Caution: if you try to "hack" in Python on the fly, you will spend *much more time* debugging than is necessary. Read this section again, unless of course you have already reread it.

You can access exact locations of a string by indexing into the string in the following way:

```
DNA = "attcagcagt"

# print out the initial letter (nucleotide) and the fourth nucleotide
print ("Initial nucleotide: ", DNA[0])
print ("Fourth nucleotide:  ", DNA[3])
```

Your program's documentation must have an initial section listing your name, date, and program number. A second section must include the purpose (summary) of the program (that is, what does it do, *not* how it does it; this should be "the big picture". I expect a well-written opening sentence!); plus, a description of the INPUT that the program uses (indicating *from where* each kind of data comes, data types, etc) and finally a description of the program's OUTPUT (showing part of a sample output in the documentation is nice).

• You MUST comment EACH variable. Variables MUST have good names.

• You MUST use CONSTANTS where appropriate (I think you have quite a few to use).

```
-----------------------------
  1  -  Read DNA file
  2  -  Find cag repeat
  3  -  Report percent GC
  4  -  EXIT
--------------------------
ENTER: 1
Enter DNA filename: gene1.fna
DNA file in one string is:
 attgcagcttttc cagcag atttttttaaaa
---------------------------------------------------------------
  1  -  Read DNA file
  2  -  Find cag repeat
  3  -  Report percent GC
  4  -  EXIT
--------------------------
ENTER: 2
-------------------
Found STARTING CAG at:  4
-------------------
Found STARTING CAG at:  13
       ANOTHER CAG at:  16
++++++++++++++++++++++++++++++++++++
cag Repeats found at bp location 13
cag Repeat string is:   cagcag
Length of cag string:   6 bp
Number of cag Repeats:  2
Classification:STATUS:
   Normal:Unaffected
++++++++++++++++++++++++++++++++++++
:
:
ENTER: 3
GC percentage is: 30.00
---------------------------------------------------------------
  1  -  Read DNA file
  2  -  Find cag repeat
  3  -  Report percent GC
  4  -  EXIT
--------------------------
ENTER: 1
Enter DNA filename: gene3.fna
DNA file in one string is:
 attgcatcttttccaccacatttttttaaaacagcagcagcagcagcag……
-----------------------------
  1  -  Read DNA file
  2  -  Find cag repeat
  3  -  Report percent GC
  4  -  EXIT
--------------------------
ENTER: 2
-------------------
Found STARTING CAG at:  30
       ANOTHER CAG at:  33
         :
       ANOTHER CAG at:  144
++++++++++++++++++++++++++++++++++++
cag Repeats found at bp location 30
cag Repeat string is:   cagcagcagcag …(not all shown in this spec since it is so long)
Length of cag string:   117 bp
Number of cag Repeats:  39
Classification:STATUS:
   Reduced Penetrance:+/- Affected
```