HS 608 Project 1: Predicting Huntington's Disease
Upload to Canvas by Fri, Feb 26th, 2016, 11:59 pm:
hunt.py and a copy/paste of your doctests and end-to-end testing

Huntington's disease is a genetic disorder that leads to degeneration of brain cells. Initial symptoms are uncontrolled movements. Irritable moods are also experienced. In the advanced stages of the disease there is cognitive decline and dementia.

Huntington's disease is caused by a dominant mutation of either of an individual's two copies of the gene that codes for the protein huntingtin (HTT). In the mutant form of HTT that causes Huntington's disease there are too many repeats of the codon CAG, resulting in a buildup of amyloids in the brain, causing mental deterioration.

Project Scenario

You work in the medical field analyzing the DNA of patients who want to know whether they will come down with Huntington's disease. Couples want to know what their combined genetic outlook is on passing on Huntington's disease to children. Patients get their DNA sequenced in the HTT region of the genome and give you the nucleotide sequence. You analyze the DNA, counting repeats of CAG, and let them know their classification and disease status according to this chart:

Repeat count	Classification	Disease status
<28	Normal	Unaffected
28–36	Intermediate	Unaffected
37–42	Reduced Penetrance	Somewhat Affected
>42	Full Penetrance	Affected

If the news is bad then you advise the patients to make an appointment with a counselor who is equipped to help them deal with this difficult news.

You realize that you could write a simple program that you could give to the counselor. Then patients could make one office visit for both DNA analysis and counseling. Patients would not have to wait and this would be a cost saving measure.

The Assignment:

Write a python program that will take user input of the patient's first name, last name, and DNA sequence (for CAG repeat at the end of their HTT gene). Name your script hunt.py. Your program will calculate the number of CAG repeats in the DNA, as well as the classification and disease status of the patient (from the above chart). **Output should**

include first and last names of the patient, the DNA sequence, the number of CAG repeats, the classification and disease status. Your code should include at least 3 functions: a function to get the name and DNA input from the user, a function to count CAG repeats in the DNA and a function to calculate the classification and disease status depending on the number of CAG repeats.

The challenging task will be going through the DNA input to count repeats of "CAG". (**You may NOT use built-in functions like string count, code this yourself! *** see below.**) For simplicity **assume that the DNA input begins with the CAG repeats, and we will stop counting when we encounter either the end of the DNA string or the end of the succession of CAGs.** You may assume that the DNA sequence is accurate and you do not have to test if it has characters that are not A, C, T or G.

You will need to break the DNA string into the slices so you can check for GAG repeats. Start with a simple example, like dna="CAGCTG". How many repeats should you count? Put this and a few other test cases (ex: CAGCAGCA) into a docstring and then code so as to pass the tests.

Grading:

1. Correctness: 30 pts of your grade: Your program should compute the correct CAG repeat count for all possible DNA inputs (including input of empty string) and also produce the correct classification and disease status for each count. Your program is required to prompt the user for and read keyboard input of first and last name, as well as DNA sequence. **Of course all projects must be able to run** (ie – not have a syntax error).

2. Function use: 10 pts: You must **call all 3 functions and pass arguments from the main program.** Values obtained in functions must be **returned to main and stored in global variables, and output from main**. Be sure to define the three required functions so they fulfill the specifications in section ***The Assignment*** above. You are **required to use the function names and headers provided below. Functions *prediction* and *countCAGs* must pass the doctests provided** below the headers. There are further specifications for function ***get_input*** below its header, but no doctests required for it.

3. Documentation: 10 pts: Provide **header documentation including the author's name, a brief description of the program and how to use it**. Use **meaningful names for your variables**. In general, do not place comments inside a function body unless there is a complicated calculation that needs to be explained or referenced. However be sure to **document each function briefly with 1) an accurate explanation of what it accomplishes or returns, 2) how its parameters are used, and 3) preconditions**, if there are any. You may follow the convention of placing your function description within the same multiline string as your doctests, as in the docstrings below.

4. Testing: 20 pts: Testing serves as proof that **the program you turned in works correctly or that you have detected all errors.** Test thoroughly with **both doctests and end-to-end testing.** For each function test all its paths of execution. Test all the different types of DNA strings that may be encountered. Your function must give correct results for every legal input, including **boundary input** (ex: 27 CAG repeats result in a classification that is different from 28 repeats). It is hoped that there are no errors in the hunt.py you upload to Canvas. However, if there are any, testing is expected to reveal them. You get full testing credit for testing an input that doesn't work correctly, as long as you note what error your testing reveals. Use doctests for unit testing. You must pass the doctests provided for functions countCAG and prediction, but **your testing grade will be based on your own test cases**. For doctests, copy/paste into a text file the results of running your doctests in verbose mode from the command line. (See testing notes below). Then turn off the doctests and run your program several times, entering information as the user to **demonstrate that your program works as a whole, passing results from one function, to main, then from main to another function, etc**. Copy/paste **these end-to-end runs** also into your file for test cases. Upload the testcases file to Canvas separately from your python script.

5. Quality of solution: 10 pts: Do not make your solution unnecessarily complex. Also, do not condense your solution to the point where it is incomprehensible. Program with proper style: **function definitions at the top of the program and main program at the bottom**; do not intersperse __main__ throughout your program. **Do not use global variables inside functions**. The code that runs your doctests should appear as part of your main program.

Collaboration: Students are to do their own work on all projects. You may talk with others about general approaches or to understand the problem statement, but each student is to develop his or her own solution. In addition, using solutions from any other source is forbidden. All projects are to be individual and original efforts.

Testing notes:
You must pass my doctests below, but your testing grade will be based on your own test cases. Doctests are tricky for functions that call the built-in input or raw_input functions, so your get_input function can be tested as part of your end-to-end runs.

To output the results of all your doctests on hunt.py so you can copy/paste them into your testcases file, type this at the command line:
python hunt.py –v

Function Names:
You are required to use the following function names and headers: You may download this file and copy-paste the function headers and doctests to your hunt.py.

```
def countCAG(dna):
    """
    Place your function description here.
    Examples:
    >>> countCAG("C")
    0
    >>> countCAG("CAGCA")
    1
    >>> countCAG("CAGCATCAGCAGCAG")
    1
    >>>
countCAG("CAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAG
CAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCA
GCAGCAGCAGCAGCA")
    41
    """


def prediction(numCAG):
    """
    Place your function description here.
    Examples:
    >>> prediction(27)
    ('Normal', 'Unaffected')
    >>> prediction(35)
    ('Intermediate', 'Unaffected')
    >>> prediction(42)
    ('Reduced Penetrance', 'Somewhat Affected')
    >>> prediction(45)
    ('Full Penetrance',  'Affected')
    """



def get_input():
```

Your get_input function should return a tuple containing whatever the user input for first name, last name and DNA, such as:
('Patricia', 'Francis-Lyon', 'CAGCAG')

*** The code you **write** **may use the bracket operator([]), the concatenation operator (+), the len function, and slices**. You may use **no other built-in functions or operations.** NB: this means No min function, NO **in** operator, etc. The exception is that you may use the **in** operator within the for loop header (but not the body).