First python program

The hello.py program is one of the first python programs that people write.

Documenting code is an important and often overlooked process in software engineering. Don’t fall into the trap of not adding comments to your code! Note, you will not receive full credit on your assignment if your code is not clearly documented. To add comments you use the ’#’ character for single line comments, or you can use triple-quotes (“””) to begin a block of comments and then add another set of triple-quotes to end that block comment.

To get an idea of what documenting and comments look like, here is an old-school version of the ’hello.py’ program, documented:

In [ ]:
```
#!/usr/bin/env python3
# Name: David Bernick (dbernick)
# Group Members: John Cleese (jcleese) and Eric Idle (eidle)

"""
Print out the string Hello World when run.
"""
# remember that strings in python use either single (') or double (")
quotes.
print ("Hello World")
```

Add your own similar-looking documentation and comments to the the following code, customizing it for yourself, and lets make it print: Hello **your name** instead. In my case, it would print "Hello David"

helloMe

In [ ]:

```
#!/usr/bin/env python3
# Name: David Bernick(dbernick)
# Group Members: none

"""
Print out the most common statement in computer programming, Hello Wor
ld.
"""
class Announcer (str): # Announcer is now defined as a kind of python
string.
    def printMe (self):
        print (self)

student = Announcer ('Hello David')
student.printMe()
```

There are a few things to mention about this style of coding. Classes are templates for *objects*. The *announcer* class is a kind of *string* class, and we tell python that by including (str) after the name of the class. We have added a method (printme) to the announcer class, and that method allows objects of class announcer to *print* themselves. In python, we access data that is saved in an object using the word *self*. Since objects of this class are strings, *self* refers to the data ('Hello David' in this case). You will notice that python methods are defined using the keyword: def *name* (self, parameters)

The line: *student = announcer ('Hello David')* creates a new name (student), of class *announcer* and it is given a parameter '*Hello David*'. That parameter is given to announcer as its data. Objects of class announcer refer to that data using the name *self*.

The line: *student.printme()* sends a printme() message to the student object, and the student object then prints out its data.

Return of “Hello World”

We will now take a look at a more sophisticated, second person version of *helloMe*. As currently written, it should ask you to enter a name; you type in the name and then press the *enter* key (maybe labelled return).

The output you should see is:

Hi there, I am *name*

Look at *helloAgain* below. You’ll notice that it’s more complex and longer than *helloMe*. Now modify *helloAgain* so that, in addition to asking for that person’s name, the program also prompts the user for their favorite pet like this:

“What is your favorite pet? :”

To make this clearer, here are a few lines that belong in your revised *helloAgain*.

In [ ]:
```
name = input( "What is your name? : " )
pet = input( "What is your favorite kind of pet? : ")
newPerson = person (name, pet)
newPerson.introduce()
```

helloAgain

In [ ]:
```
#!/usr/bin/env python3
# Name: David Bernick(dbernick)
# Group Members: none

"""
Build a Person object and have it introduce itself.

input: a string of arbitrary length, which is used to name the new per
son object
output: greeting printed to screen
"""

class Person:
    def __init__(self,name,pet):
        self.myName = name
        self.myPet  = pet

    def introduce (self):
        print ("Hi there, I am {0}, and i like {1}s".format(self.myNam
e,self.myPet))


# put your new code here.
```

Your new version of *helloAgain* should now print something that looks like this:

“Hi there, I am *name* and I like *pets* !”

replacing the italicized text with what the user entered at the command line. Note the statement uses a plural form by appending the letter s, your program should do this too.

So, if you decided that your new person is named Fred, and Fred is fond of his pet aardvark, then your program would output:

Hi there, I am Fred and I like aardvarks!

Remember to save and add documentation/comments to *helloAgain* .

Brief Biography Program

Now that you've gotten a little experience programming and seen a few python programs, next you'll write a program that will do the following:

- Print your full name (first and last)
- Print your SFSU email
- Print whether your are an undergraduate or graduate student
- Print your major
- Print why you are taking this class and what you hope to get out of it
- What problem in molecular biology most interests you.
- Print your prior computer programming experience

This program should NOT ask the user for this data. This data should be constants in your program. The constants should NOT be part of the person class -- use parameters that are given to your person object when it is instantiated. The name of the program is "myBio" and a sample execution of the program should look something like this:

My name is ***David Bernick***.
My username is ***dbernick***.
I am a ***professor***.
My major is ***Bioinformatics and Biomolecular Engineering***.
I'm not taking this class, but I teach it.
I am a synthetic bioengineer and I'm interested in extreme life, especially hypertherthermophiles and hypersalinophiles, with a focus on small RNA and novel proteins from extreme viruses.
I have prior programming experience using ***Python, Perl, Java, Fortran, Pascal, PL/1, ADA, Basic,TPL, APL, Assembly, and C++ (Im not very fond of this though)***.

Each piece of info should occupy only a single line.

In order to get full credit, the program must use attributes of the person class for each of the ***emphasized strings*** above and contain appropriate documentation/comments.

Note that in Python, *names* that are part of a class are called *attributes*. Functions that are part of a class are called *methods*. In other languages, *names* are sometimes called *variables*.

myBio

In [ ]:

```
#!/usr/bin/env python3
# Name: David Bernick(dbernick)
# Group Members: none

"""
Build a Person object and have it introduce itself.

input: a string of arbitrary length, which is used to name the new per
son object
output: greeting printed to screen
"""

class Person:
    def __init__(self,name):
        self.myName = name

    def introduce (self):
        print ("Hi there, I am {0}".format(self.myName))


name = 'David Bernick'
username = 'dbernick'
studentType = 'professor'
major = ' '
...
```

Debug buggy code

Practice makes us better; usually debugging makes code closer to perfect. This program is supposed to ask the user to enter a DNA sequence and then output the AT content of the sequence— total number of As and Ts divided by the sequence length—as a fraction of the total sequence length (a number between 0 and 1).

For example, given the string:

AAAATGAATGGCTAACTTTTGAA

the program should output: (make sure that you pay attention to the number of significant digits in the output!)

AT content = 0.739

Unfortunately there are a few bugs in the program. Your task is to identify and correct those bugs. This part of the assignment is to help familiarize you with the debugging process. It is possible to identify the bugs without using any of the following techniques just by looking over the program, but this will be good practice and will be essential when your code gets more complex.

To debug countAT-bugs.py, run the program, which will produce an error. Figure out what the error is and correct the code. It's likely that after you correct the first error you will notice another error. Repeat this process until you fix all the bugs and the program runs properly.

Save a corrected copy of the program called countAT-bugsSoln.py that includes:

- Documentation and comments
- A program overview that includes how many bugs you found and what lines from the original file they were found at.

countAT-bugs

In [ ]:
```
#!/usr/bin/env python3
# Name: David Bernick(dbernick)
# Group Members: none

class dnaString (str):
    def length (self):
        return (len(self))

    def getAT (self):
        num_A = self.count(A)
        num_T = self.count("T")
        return ((num_A + num_T)/ self.len() )

dna = input("Enter a dna sequence: ")
upperDNA = dna.upper()
coolString = dnaString(upperDNA)

print ("AT content = {0:0.1f}".format(coolString.getAT()) )
```

Counting DNA Letters

Now that you are inspired by your ability to debug the countAT-bugs code, let's create a new program that will ask the user to enter a DNA sequence and then output the total sequence length and the number of each base {total As, Cs, Gs, Ts} found in the sequence. For example, given the string:

AAAATGAATGGCTAACTTTTGAA

the program should output:

Your sequence is 23 nucleotides long with the following breakdown of bases:

number As = 10 number Cs = 2 number Gs = 4 number Ts = 7

Write a program called countNucleotides (put it below) that will take in a DNA sequence entered by a user and produce the outputs specified above. Note that "specified" means ... "required"

To get full credit on this assignment, your code needs to:

- Run properly
- Contain documentation/comments
- Include an overview about what your program is designed to do with expected inputs and outputs. Use a MarkDown Cell to write your overview.

countNucleotides

In [ ]:
```
#!/usr/bin/env python3
# Name: David Bernick(dbernick)
# Group Members: none

...
```