CS 232 – Programming in Python

Assignment #1

Deadlines

This assignment is due on **Thursday, February 11, 2015 @ 5:00 PM.**

How to submit your work on the assignment:

- Your assignment will be turned in as a computer file (the "module") containing your Python code. The module will contain only the code that satisfies the problems' requirements. **There is NO NEED to submit a testing module! I will run your code with my own testing module.**

The File Naming Convention

- Properly name the file that contains your work. The file's name will contain the following, with dashes in between each part listed below and no spaces:

  * The course name, **`CS232`**

  * Then a dash and the assignment number as a two-digit number, in this case **`01`**

  * Then a dash and your HSU account username with your initials and a number – in this example, I'll use **`jqs123`** for a typical student named John Q. Smith

  * Windows should automatically add a **`.py`** at the end of the file. This may be invisible, depending on how Windows is configured on the computer you're using. If you're using a Mac, you'll likely need to add a **`.py`** to the end of the filename.

  Put all the rules together, and John Q. Smith's Assignment 1 file in CS 232 would be named

  `CS232-01-jqs123.py`

  Your file name will be different from this because your HSU username is different, but that's the **only** difference!

Submitting your file electronically

For this assignment, you will email your file to the instructor as a file attachment. Future assignments may change the way you submit your work, but email will do for now.

Send your email to: **`david.tuttle@humboldt.edu`**

Type the Subject line of the email as: **`CS 232 - Assignment #1`**

When I receive your file in the correct way, I will send you an acknowledgment by reply email. If there's a problem, I will let you know by reply email that you need to submit the file again.

If you do NOT get an email reply within 12 hours, try sending the file again. If you wait until too close to the deadline, you run the risk that an improper submission may result in a late penalty!

Documentation of your Python code

For this assignment, please place all functions within a single Python module (file).

Begin **each Python module** (file) that you submit with at least the following opening comments:

* a comment containing the name of the file,
* a comment containing your name, and
* a comment containing the date that your module was last modified

For example:

```
# CS232-01-jqs123.py
# John Q. Smith
# Last modified: February 30, 2099
```

Before each individual function that you've designed, there should be a set of comments that:

* list the data types for the input arguments and the output (use "object" is no specific type is needed)
* give a brief explanation of the purpose of the function, describing what the function expects as input and what it returns as output
* any "side effects" (getting data from the user, printing to the screen) that occurs when the function runs

For example:

```
# Problem 1
# add_them: float float → float
# purpose:  expects two numeric (float) values
#           returns the sum of the two values
# side effects: prints a message containing the sum of the two values
```

PROBLEM 1 – 20 POINTS

Let's start with an (admittedly bizarre) function **multiples_in_range** that takes a numeric value, a beginning value, and and ending value as arguments. It prints, one per line, each of the values of the numeric value that fall strictly between the beginning value and the ending value. You can assume --- without adding code to check --- that all three arguments are positive numbers.

For example, the function call **multiples_in_range(2, 7, 13)** would print the following:

```
8
10
12
```

Test this function within the Python interactive window until you are confident that it works. Do NOT include testing code in the Python module when submitting!

PROBLEM 2 – 20 POINTS

Here's another classic beginner's function to code, to test your branching chops in Python. Write a function **grade** which takes a numeric grade as argument, and returns (NOT prints, but returns) the corresponding letter grade **A**, **B**, **C**, **D**, or **F** as its result. You are **required** to use a **single `if`** statement to determine the letter grade --- do **not** use 5 separate **`if`** statements.

You are not required to check grade's argument for "reasonable" numeric values, though you may do so if you wish. (Use the classic grading scale – for a numeric grade N, N >= 90 results in an A, 80 <= N < 90 is a B, 70 <= N < 80 is a C, 60 <= N < 70 is a D, and N < 60 is an F.

Test this function within the Python interactive window until you are confident that it works. Do NOT include testing code in the Python module when submitting!

FOR A 5 POINT BONUS: within your **grade** function, **after** you've determined the base letter (NOT as part of that determination), **concatenate** a + or – to the base letter as appropriate, where the 1's digit being a 7, 8, or 9 results in a + appended to the grade, and a 1's digit of 0, 1, or 2 results in a – appended to the grade. You will need to account for the fact that separate **`print()`** calls are printed on different lines – there is a workaround, so search for that in the Python documentation!

PROBLEM 3 – 20 POINTS

Write a function that takes better advantage of Python's dynamic typing, as follows:

There are more elegant ways of testing that we'll learn about soon, but for now note that both of the following allow a user to test if a value is an integer:

```
val = 3
if type(val) == type(0):
    print("val is an integer!")
 else:
     print("val is not an integer!")

val = 7
if str(type(val)) == "<class 'int'>":
    print("val is an integer!")
else:
    print("val is not an integer!")
```

Write a function **bump_it** that, if given an integer, returns a value that is one more than that integer; if given a floating point number, returns a value that is **`0.1`** more than that floating point number; and if given a string, returns a string that is the same except that it has an exclamation point concatenated to the end of it. For an argument of any other type (boolean, or a molecular data type) passed to it, it should simply return the argument passed to it.

Test this function within the Python interactive window until you are confident that it works. Do NOT include testing code in the Python module when submitting!

PROBLEM 4 – 20 POINTS

Write a function **nickname** that expects one string argument in the form of a one-word first name It will generate and return a goofy nickname, meeting the following requirements:

* If the name isn't capitalized, then capitalize it (there's a string method for this!)
* If the length of the name passed is less than 5 letters, it will simply return the original name as the nickname;
* If the length of the name passed is 5 or more, it will grab the first 5 letters to begin the nickname, and:
    * If the 5th letter of the name is a **`y`**, strip off the **`y`** and return the result as the nickname;
    * Otherwise, add **`y`** to the end of the 5 letters and return the result as the nickname;

For example:

```
>>> print(nickname('Sally'))
Sall
>>> print(nickname('ed'))
Ed
>>> print(nickname('Horatio'))
Horaty
>>> print(nickname(('elizabeth'))
Elizay
```

Test this function within the Python interactive window until you are confident that it works. Do NOT include testing code in the Python module when submitting!

PROBLEM 5 – 20 POINTS

Note that string method **`isupper()`** returns **`True`** if all of the characters in the calling string are uppercase, and returns `False` otherwise. For example:

```
>>> 'HELLO'.isupper()
True
>>> greeting = 'Hi'
>>> greeting.isupper()
False
>>> 'hi'.isupper()
False
```

`islower()` works analogously for returning if a string is all-lowercase. **`upper()`** returns a version of the calling string in all-uppercase characters, while **`lower()`** returns a version of the calling string in all-lowercase characters.

Write a function **`flip_it`** that expects a string argument:

* If the argument is in all-uppercase, it should return a version of the argument in all-lowercase.
* If the argument is in all-lowercase, it should return a version of the argument in all-uppercase.
* If the argument is neither all-uppercase nor all-lowercase, look at the **first** character's case.
    * If the first character is uppercase, return a version of the argument where only the FIRST letter is changed to lowercase, and the rest are returned unchanged.
    * If the first character is lowercase, return a version of the argument where only the FIRST letter is changed to uppercase, and the rest are returned unchanged.

Test this function within the Python interactive window until you are confident that it works. Do NOT include testing code in the Python module when submitting!