Homework 2

Intro to Programming (Term I/2017–18)

built on 2017/09/21 at 23:14:12 **due:** *thu sep 28 @ 11:59pm*

This assignment will give you more practice on Python expressions and acquaint you with the concepts of conditional execution as well as function declaration and usage. You will write code and hand it in electronically.

New to this assignment: We're providing a starter package, which can be downloaded from the course website. In it, you will find function stubs, together with simple test inputs. Read below to see how to use these sample inputs.

Overview:

Problem	File Name
1.	`posneg.py`
2.	`aplusb.py`
3.	`aggregate.py`
4.	`snakeoil.py`

Problem	File Name
5.	`nyctime.py`
6.	`digit.py`
7.	`alphanum.py`
8. (extra)	`roman.py`

Collaboration

We interpret collaboration very liberally. You may work with other students. However, each student ***must*** write up and hand in his or her assignment separately. Let us repeat: You need to write your own code. You must not look at or copy someone else's code. You need to write up answers to written problems individually. The fact that you can recreate the solution from memory will be taken as proof that you actually understood it, and you may actually be interviewed about your answers.

Be sure to indicate who you have worked with (refer to the hand-in instructions).

Logistics

We're using a script to grade your submission before any human being looks at it. Sadly, the script is not as forgiving as we are. *So, make sure you follow the instructions strictly.* It's a bad omen when the course staff has to manually recover your file because the script doesn't like it. Hence:

- Save your work in a file as described in the task description. This will be different for each task. **Do not save your file(s) with names other than specified.**
- You'll zip these files into a single file called `a2.zip` and you will upload this one zip file to Canvas before the due date.
- Before handing anything in, you should thoroughly test everything you write.
- At the beginning of each of your solution files, write down the number of hours (roughly) you spent on that particular task, and the names of the people you collaborated with as comments. As an example, each of your files should look like this:

```
# Assignment XX, Task YY
# Name: Eye Loveprograming
# Collaborators: John Nonexistent
# Time Spent: 4:00 hrs

... your real program continues here ...
```

- The course staff is here to help. We'll steer you toward solutions. Catch us in real-life or online on Canvas discussion.

Preliminary Testing

We expect you to test your programs thoroughly before handing them in. To aid initial testing, we're providing you a suite of test inputs for every task. This comes in the form of a `docstring` at the beginning of each function. For example, here is how one of the functions looks in the starter package:

```
def my_min(p, q, r):
    """Return the minimum of p,q, and r, without using min

    >>> my_min(3.0,1,9)
    1
    """

    # delete this comment line and add your code
    return ___
```

Below the **def** header, there is a wall of text wrapped between a pair of triple quotes (`"""`). This is known as a `docstring`. Inside such a `docstring` contains some descriptions, as well as sample tests. These tests are run by the code residing at the bottom of the file, which probably makes no sense to you at the moment.

To take advantage of this, once you're done fleshing out the function, just run it. If your code passes all these tests, the program will be silent. If your code fails some tests, it will yell at you. *Continue to test your function thoroughly even after it passes our preliminary tests.*

Task 1: Positive, Negative (5 points)

For this task, save your work in `posneg.py`

Write a function `pos_neg(a, b, negative)`, which takes as input two integer values `a` and `b`, as well as a Boolean `negative`. The function returns `True` if one is negative and one is positive. Except if the parameter `negative` is True, then return `True` only if both are negative. NOTE: the number 0 is neither negative nor positive.

For example:

- `pos_neg(1, -1, False)== True`
- `pos_neg(-1, 1, False)== True`
- `pos_neg(-4, -5, True)== True`

Task 2: A Plus Absolute B (5 points)

For this task, save your work in `aplusb.py`

You may recall that **abs**`(x)` returns the absolute value of `x`. That is, if `x` is positive, it **abs**`(x)` is the same as `x`. Otherwise, **abs**`(x)` is equal to `-x`. Notice that `add(x,y)` computes `x + y`, and `sub(a,b)` computes `x - y`.

Your Task: Fill in the blanks in the following function definition for adding `a` to the absolute value of `b`, without calling **abs**. Consult the starter package for more details.

```
from operator import add, sub

def a_plus_abs_b(a, b):
    """Return a+abs(b), but without calling abs.

    >>> a_plus_abs_b(2, 3)
    5
    >>> a_plus_abs_b(2, -3)
    5
    """
    if b < 0:
        f = _____
    else:
        f = _____
    return f(a, b)
```

Task 3: Aggregate: Min, Mean, Median (10 points)

For this task, save your work in `aggregate.py`

In this problem, you're writing functions to compute certain aggregations of three numbers that you take as input parameters. These three numbers can be *any combination of floats and ints*, and you can assume that they are *distinct*. Your output must be a number. It can be either an int or a float as long as the value is correct.

Write one function for each of the following. You'll save them in the same file.

- a function `my_min(p, q, r)` that takes 3 numbers and returns (not print) the minimum of the three numbers. For example, `my_min(3.0,1,9)` should return 1.
- a function `my_mean(p, q, r)` that takes 3 numbers and returns (not print) the average of the three numbers. You should recall that the average of p, q, and r is simply $\frac{1}{3}(p+q+r)$. Hence, as an example, `my_mean(3, 7, 4)` should return $4.6666\cdots$.
- a function `my_med(p, q, r)` that takes 3 numbers and returns (not print) the median of the three numbers. Remember that the median of three numbers is the number where one other number is smaller than it and one other number is larger than it. For instance, `my_med(4,1,5)` should return 4. Also, `my_med(13,5.0,12)` should return 12.

REMARKS: For this task, use only Boolean/math expressions and conditional statements (if-statements). Do *not* use built-in functions for sorting or finding **min**, **max**, etc.

Task 4: Snake Oil (10 points)

For this task, save your work in `snakeoil.py`

Selling snake oil is a lucrative business. In a radical move, Company S now sells snake oil at 17 baht per liter. However, shipping is complicated:

- For an order less than (<) 10 liters, shipping is 20 baht/liter.
- For an order between 10 and 100 liters (inclusive), shipping is 500 baht flat.
- Then, for an order over (>) 100 liters, shipping is free; additionally, you get a 3% discount.

This means if you order 5 liters, you'll pay a total of 185.0 baht. If you order 20 liters, you'll pay 840.0 baht. If you order 200 liters, you'll pay 3298.0 baht.

For this problem, write a function `price(vol)` that takes in an order's volume in liters (float or int) and returns a float, the total amount this customer will have to pay.

Task 5: What Time Is It? (10 points)

For this task, save your work in `nyctime.py`

New York is 5 hours behind London. You will write a function `nycHour(londonHour)` that takes an integer (between 0 and 23, inclusive) representing the (current) hour in London and returns the corresponding hour in New York City (NYC).

London time, however, is given in 24-hour time, but NYC time must be returned as a string in 12-hour time (so the result would be between 1 and 12, inclusive, followed by either `am` or `pm` *without any space between them*).

TIPS: The function is to return a string—printing is not the same as returning.

Here are some examples:

- `nycHour(0)` should return `"7pm"`
- `nycHour(11)` should return `"6am"`
- `nycHour(23)` should return `"6pm"`
- `nycHour(17)` should return `"12pm"`
- `nycHour(5)` should return `"12am"`

Task 6: k-th Digit (10 points)

For this task, save your work in `digit.py`

For this task, you will implement a function `kthDigit(x, b, k)` that takes as input three integers, x ($x \geq 0$), b ($b \geq 2$), and k, and returns the k-th digit of x counting from the right when represented in base b. The return value must be an **int**.

To refresh your memory of number bases, you may find the following websites useful: `http://en.wikipedia.org/wiki/Radix` and `http://www.purplemath.com/modules/numbbase.htm`.

As a basic example, consider the number 789 (in base 10), so

- `kthDigit(789, 10, 0)` returns `9`.
- `kthDigit(789, 10, 1)` returns `8`.
- `kthDigit(789, 10, 2)` returns `7`.
- `kthDigit(789, 10, 3)` returns `0`.

As another example, consider the number (987). This is $(3\texttt{db})_{16}$ in base 16, where `b` represents 11 and `d` represents 13. So then:

- `kthDigit(987, 16, 0)` returns `11`.
- `kthDigit(987, 16, 1)` returns `13`.
- `kthDigit(987, 16, 2)` returns `3`.

REMARKS: Use only Boolean/math expressions and conditional statements (if-statements). Do *not* use built-in functions for converting integers into a string representation.

Task 7: Call Me Maybe? (10 points)

For this task, save your work in `alphanum.py`

To help increase memorability, business owners sometimes encode their phone numbers as mnemonic phrases, known as *phonewords*. These are alphanumeric equivalents of a phone number, which can be derived by using the mapping of letters on the digits of a telephone keypad. Many popular phone keypads have letters and numbers that look similar to the figure on the right. With this mapping, we can make the following conversion, for example:

Image from code golf

- `FLOWERS` translates into `3569377`;
- `PrOGrAM` translates into `7764726`; and
- `Battery` translates into `2288379`.

Your Task: You will implement a function `phoneWord2Num(word)` that takes as input a string of length exactly 7, where each position is an upper- or lower- case letter in the English alphabet. The function must return an integer (of type `int`) representing the phone number that is the equivalent of the input string. As an example, we would call `phoneWord2Num("PrOGrAM")` and expect the integer 7764726 as the return value.

TIPS: You may wish to write a new function (generally known as a helper function because its role is to help complete the main task at hand) that takes in one letter and returns a number corresponding to that letter.

To implement such a helper function, you may find comfort in knowing that if `letter` is a single character and `pattern` is a string, the expression `letter in pattern` returns True or False indicating whether the `letter` belongs in that `pattern`. This means:

```
q1 = 'a' in 'pxrt' # q1 will be False
q2 = 'x' in 'pxrt' # q2 will be True
```

RESTRICTIONS: You can only used Python features we have covered in this class so far. Remember that we haven't done iterations (for-, while- loops) or lists.

Task 8: EXTRA: Roman Numerals (0 points)

For this task, save your work in `roman.py`

READ THIS BEFORE YOU ATTEMPT IT: This is an extra problem for those who want a more challenging task. It is worth 0 points on the assignment; however, you'll earn

- bragging rights;
- a place on the course's wall of fame;
- brownie points that may be exchanged for real points if needed at the end; and
- above all, an opportunity to practice programming.

You surely have encountered Roman numerals: I, II, III, XXII, MCMXLVI, etc. They are everywhere. Roman numerals are represented by repeating and combing the following seven characters: `I` = 1, `V` = 5, `X` = 10, `L` = 50, `C` = 100, `D` = 500, `M` = 1000. To understand how they must be composed, we borrow the following excerpt from *Dive Into Python*:

- Characters are additive. `I` is 1, `II` is 2, and `III` is 3. `VI` is 6 ("5 and 1"), `VII` is 7, and `VIII` is 8.

- The tens characters (`I`, `X`, `C`, and `M`) can be repeated up to three times. At 4, you need to subtract from the next highest fives character. You can't represent 4 as `IIII`; instead, it is represented as `IV` ("1 less than 5"). The number 40 is written as `XL` (10 less than 50), 41 as `XLI`, 42 as `XLII`, 43 as `XLIII`, and then 44 as `XLIV` (10 less than 50, then 1 less than 5).
- Similarly, at 9, you need to subtract from the next highest tens character: 8 is `VIII`, but 9 is `IX` (1 less than 10), not `VIIII` (since the `I` character cannot be repeated four times). The number 90 is `XC`, 900 is `CM`.
- The fives characters cannot be repeated. The number 10 is always represented as X, never as `VV`. The number 100 is always `C`, never `LL`.
- Roman numerals are always written highest to lowest, and read left to right, so the order the of characters matters very much. `DC` is 600; `CD` is a completely different number (400, 100 less than 500). `CI` is 101; `IC` is *not* a valid Roman numeral (because you can't subtract 1 directly from 100; you would need to write it as `XCIX`, for 10 less than 100, then 1 less than 10).

YOUR TASK: Implement a function `toRoman(n)` that takes an integer `n` ($1 \leq n < 4,000$) and returns a string that represents the number `n` in Roman numerals.

Despite the complexity this problem may seem at first, there are nice (and not tedious) ways to implement the logic to convert an integer into a Roman numeral using only features we have learned so far. You should not write more than 30 lines of code.