CS116 Winter 2011 Assignment 6
Due: Tuesday, March 8 at 10:00AM

The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.

Important Information:

1. Read the course Style Guide for Python (Appendix C in the course notes) for information on assignment policies and how to organize and submit your work. In particular, your solutions should be placed in files a6qY.py, where Y is a value from 1 to 4.
2. Be sure to download the interface file to get started.
3. The interface file contains several strings, which you should use to format your output. Do not change these values, or you will lose correctness marks on your assignment.
4. Do not use any advanced Python language features, such as iteration or lists. ***Any repetition should be implemented using recursion – do not use loops.***
5. You are encouraged to use helper functions in your solutions as needed. Simply include them in the same file with your solution, but do not declare them locally. You should follow the full design recipe for your helper functions (which includes testing).
6. If you use print statements for debugging purposes, be sure to remove them (or comment them out) before you submit your solutions.
7. Do not copy the purpose directly from the assignment description – it should be in your own words.
8. You may assume that all consumed data will satisfy any stated assumptions, unless indicated otherwise.
9. You may import and use the math module if needed for any question.

Language level: Any Python in the range 2.5.0 to 2.7.1. (Do **not** use Python 3.0 or higher)

Coverage: Module 6

Provided Files: `a6interface.py`, and the module `a6.py`

Question 1: Validating SINs

The Social Insurance Number (SIN) is a nine-digit number that you need to work in Canada or to have access to government programs and benefits. Not all possible nine-digit numbers are valid SINs though. A SIN is validated, using the ninth digit as a check-digit, in the following manner:

Let $S = S_1S_2S_3S_4S_5S_6S_7S_8S_9$ be a candidate SIN, where each S_k is a single digit ($0 \le S_k \le 9$).

- multiply each of S_2, S_4, S_6 and S_8 by 2 to obtain

$$C_2 = 2 \times S_2,\ C_4 = 2 \times S_4,\ C_6 = 2 \times S_6, \text{ and } C_8 = 2 \times S_8$$

- add all of the **individual digits** of these four values together to obtain first_sum
- add all of the digits of S_1, S_3, S_5, and S_7 together to obtain second_sum
- add these together to obtain total_sum = first_sum + second_sum and let next_10 be the smallest multiple of 10 that is greater than or equal to sum.
- let check = next_10 – total_sum
- if check is equal to S_9, the ninth digit of S, then $S = S_1S_2S_3S_4S_5S_6S_7S_8S_9$ is a valid SIN. Otherwise, it is invalid.

For example, if S = 124356874

- $C_2 = 2\times2 = 4$, $C_4 = 2\times3 = 6$, $C_6 = 2\times6 = 12$, $C_8 = 2\times7 = 14$
- first_sum = 4 + 6 + (1 + 2) + (1 + 4) = 18
  (note: $C_6 = 12$ and $C_8 = 14$ both have two **digits** and we have to add all individual digits for first_sum)
- second_sum = 1 + 4 + 5 + 8 = 18
- total_sum = 18 + 18 = 36
- next_10 = 40
- check = next_10 - total_sum = 40 – 36 = 4
- since $S_9 == 4 ==$ check, this SIN is valid

For this question you will complete the Python function `is_valid_sin` that consumes a `str`, which is the string representation of the integer SIN candidate, and produces a `bool` indicating if the candidate SIN is valid (`True`) or invalid (`False`). A candidate SIN is invalid if it fails the test described above **or** if it does not contain exactly nine digits.

For example,

- `is_valid_sin("124356874") => True`
- `is_valid_sin("124356877") => False`
- `is_valid_sin("123") => False`

Note: You can assume that any input to the function will be a string, of arbitrary length, that consists of only numeric digits (0,1,2,...,9).

Hint: If `s` is a `str` then `s[i]` gives the `i`-th character in the string, where counting starts with zero. For example, the first (zeroth) character in the string s is given by `s[0]` and the last character is given by `s[len(s)-1]`.

Question 2: Rock-Paper-Scissors

Consider the hand game rock-paper-scissors. In this game, two players count out "rock", "paper", "scissors", then simultaneously give a hand gesture indicating one of rock, paper, or scissors. The winner of the game is decided by the following rules:

- rock beats scissors
- scissors beats paper
- paper beats rock

For example, suppose Alice and Bob are playing the game. If Alice plays rock and Bob plays scissors, then Alice wins. If Alice plays paper and Bob also plays paper then it is a tie.

For this question you will complete a Python function that allows you to play multiple games of rock-paper-scissors against the computer. In particular, you will complete the Python function `rock_paper_scissors`, which consumes no parameters and produces nothing.

The function has the following side effects: the function initially prompts the user for input, expecting only one of "rock", "paper", "scissors" or "end". If "end" is input the function displays the user's game statistics represented by the integer (2*wins + ties – losses), where wins is the number of games the user beat the computer, ties is the number of tied games and losses is the number of games that the computer won. If the game was played (that is, the user inputs one of "rock", "paper", or "scissors") then the function displays the computer's choice and displays who wins the current game. The process is repeated until the user inputs "end".

For example, consider the display input/output after calling `rock_paper_scissors()`, where user input is in bold:

```
Rock-Paper-Scissors, play! [rock,paper,scissors,end]: rock
Computer plays paper
Computer wins
Rock-Paper-Scissors, play! [rock,paper,scissors,end]: paper
Computer plays rock
User wins
Rock-Paper-Scissors, play! [rock,paper,scissors,end]: rock
Computer plays scissors
User wins
Rock-Paper-Scissors, play! [rock,paper,scissors,end]: paper
Computer plays paper
Tie
Rock-Paper-Scissors, play! [rock,paper,scissors,end]: end
Game stats = 4
```

The values of wins, ties and losses correspond to one calling of the function `rock_paper_scissors`. Calling the function again starts a new round of games.

For example, calling `rock_paper_scissors()` again, the input/output might be as follows, where user input is again in bold:

```
Rock-Paper-Scissors, play! [rock,paper,scissors,end]: paper
Computer plays scissors
Computer wins
Rock-Paper-Scissors, play! [rock,paper,scissors,end]: end
Game stats = -1
```

If no games are played then the game stat is equal to the `int` 0.

For this question, a function `get_move` is provided in the interface file. This function will be used to generate the computer's choice for each game. The function consumes nothing and produces a string that is either "rock", "paper" or "scissors", chosen randomly.

Note 1: **Be sure to match our output exactly!** The interface file contains the exact strings you will need for this function. Do not modify these strings. The auto-testing will fail and you will not receive any correctness marks if your implementation uses different strings.

Note 2: You are not allowed to create or use any new global (state) variables.

Hint: You might find accumulative recursion useful.

Note 3: Since `get_move` generates "random" choices for the computer's play, testing your function will be more difficult. For testing purposes there is a global variable called `all_rocks`, provided in the interface file, which when assigned the value `True` causes `get_move` to return "rock" each time it is called. For your testing, set `all_rocks = True`, and write different test cases for that situation.

Do not use the variable `all_rocks` inside your function or inside your helper functions. It is provided only to help you with your tests.

Question 3: Fibonacci again... and the Dangers of Floats

Recall that the n-th Fibonacci number, which we will denote by F(n), is given by the recursive definition

$$F(n) = F(n-1) + F(n-2)$$

where F(1) = 1 and F(0) = 0, are the two base cases. In Module 3, we saw several Scheme functions that computed F(n). (For example, see slides 3.32 and 3.38 for two efficient implementations.)

However, the n-th Fibonacci number F(n) can also be computed directly using the closed-form expression

$$F(n) = (\varphi^n - (1 - \varphi)^n) / \sqrt{5}$$

where $\varphi = (1 + \sqrt{5})/2$ is the golden ratio.

For this question, you will import and use the `math` module to complete the Python function `fib`, which consumes an `int` n and produces an `int` that is equal to F(n) using the closed-form expression from above.

As the value of n gets large, you might notice that `fib(n)` is not equal to the n-th Fibonacci number. The reason for this is that the intermediate floating point numbers being used in the computation can only store a finite amount of digits and as n gets large, the number of digits in F(n) exceeds the amount of digits that can be stored in a float.

In the module `a6`, there is another function `fib` that computes the correct Fibonacci number for any input (until Python's maximum recursion depth is exceeded.) Using this other function to compare the output of your function, you will determine a threshold integer m such that your function computes the correct Fibonacci number for all $0 \leq n \leq m$, and computes an incorrect Fibonacci number whenever $n > m$. The contract for your function should contain this restriction on the consumed parameter (to ensure correct output). You will need to **import** the module `a6` to use this other function. For your tests, you may compare the output of your function to that of the imported module's function (which will be assumed to be correct.)

Note: You must use the proper module notation to access the `fib` function in module `a6`. For example, `f12 = a6.fib(12)`.

Recap: For this question you must

1. Complete the Python function `fib` that produces the n-th Fibonacci number using the closed-form expression $F(n) = (\varphi^n - (1 - \varphi)^n) / \sqrt{5}$. (you will not receive any correctness marks if your solution computes the Fibonacci numbers in another way.)
2. Update the contract of your function to include a restriction on the valid parameter values for your function. That is, in your contract, indicate for what parameter values your function will produce the correct output.

Question 4: Tax Time

Consider the following (fictional) rules for computing your yearly taxes:

- The tax rate is determined by age:

Age:	< 18	18-29	30-64	65 and older
Tax rate:	0.05	0.1	0.5	0.025

- Taxable income is determined by several factors.
  - People can deduct some fraction of rent paid during the year
    - people under 25 can deduct all rent paid (if any) from income
    - people 25 and older can deduct ½ of rent paid (if any) from income
  - students of any age can deduct some fraction of their tuition paid from their income
    - people under 18 or older than 64 can deduct all of their tuition from their income
    - people 18-29 can deduct 75% of tuition paid from income
    - people 30-64 can deduct 50% of tuition paid from income
  - To promote people aged 35-55 to further their education, people this age that are students (regardless if they paid tuition or not) have their tax rate reduced to 0.25 instead of 0.5 as stated above.
  - However, if the income of a person of any age is 250000 or greater and that person is also a student, then they will receive no deductions at all (for rent paid, tuition paid, or tax rate reduced; as described above).
- Total taxes due is then determined as (income – any deductions) * tax rate.

You will complete the Python function `taxes`, which consumes a `float` (income), an `int` (age), a `float` (rent), a `bool` (student) and another `float` (tuition). The function produces an `int` which is the total taxes owed, based on the consumed parameters and the rules given above (simply use the `int()` function to convert the **final** intermediate floating point value of the taxes owed to convert to an integer.) If the total taxes is computed to be less than zero, however, the function will produce the `int` 0. (no tax refunds this year!)

For example,

```
taxes( 25000.00, 19,  3000.00,  True,  20000.00) => 700
taxes( 25000.00, 17,  3000.00,  True,  20000.00) => 100
taxes( 45000.50, 23, 10000.00, False,      0.00) => 3500
taxes( 15000.00, 77, 10000.00,  True,  19000.00) => 0
taxes(400000.00, 55, 25000.00,  True, 100000.00) => 200000
```