6.S189 Homework 2

`http://web.mit.edu/6.s189/www/materials.html`

What to turn in

Checkoffs 3, 4 and 5 are due by **5 PM** on **Monday, January 15th**. Checkoff 3 is over Exercises 3.1 - 3.2, Checkoff 4 is over Exercises 4.1 - 4.3, and Checkoff 5 is over Exercises 5.1 and 5.2.

Exercise 3.1 – Defining A Function

Recall how we define a function using `def`, and how we pass in parameters. In the homework template `homework_2a.py` (available on the course website), transform your code from exercise 2.2 (the rock, paper, scissors game) into a function that takes parameters, *instead* of asking the user for input. Make sure to *return* your answer, rather than printing it.

For this, and all future exercises, include *at least* 3 test cases below your code.

Exercise 3.2 – Math Module

In this exercise, we will play with some of the functions provided in the `math` module. A module is a Python file with a collection of related functions. To use the module, you need to add the following line at the top of your program, right underneath the comments with your name:

```
import math
```

Example: if you want to find out what is sin(90°), we first need to convert from degrees to radians and then use the sin function in the `math` module:

```
radians = (90.0 / 360.0) * 2 * math.pi
print math.sin(radians)
```

You can do this work in the interpreter by typing `import math` and then these lines.

For mathematical functions, you can generally call `math.func`, where `func` is whatever function you want to call. For example, if you want the sine of an angle `a` (where `a` is in radians), you can call `math.sin(a)`. For logarithms, the function `math.log(n)` calculates the natural logarithm of `n`. You can calculate the log of any base b (as in

$\log_b(n)$) using `math.log(n, b)`. The math module even includes constants such as e (`math.e`) and π (`math.pi`). Documentation for the math module is available at `http://docs.python.org/2/library/math.html`

Many computations can be expressed concisely using the "multadd" operation, which takes three operands and computes $a * b + c$. One of the purposes of this exercise is to practice pattern-matching: the ability to recognize a specific problem as an instance of a general category of problems.

In the last part, you get a chance to write a method that invokes a method you wrote. Whenever you do that, it is a good idea to test the first method carefully before you start working on the second. Otherwise, you might find yourself debugging two methods at the same time, which can be very difficult.

1. Write a function `multadd` that takes three parameters, `a, b` and `c`. Test your function well before moving on.
2. Underneath your function definition, compute the following values by making a call to the function `multadd` and print out the result:
   - `angle_test` $= \sin\left(\frac{\pi}{4}\right) + \frac{\cos(\frac{\pi}{4})}{2}$
   - `ceiling_test` $= \left\lceil \frac{276}{19} \right\rceil + 2\log_7(12)$

     Hint: If you are unfamiliar with the notation $\lceil\rceil$, this represents the *ceiling* of a number. The *ceiling* of some float x means that we always "round up" x. For example, $\lceil 2.1 \rceil = \lceil 2.9 \rceil = 3.0$. Look at the math module documentation for a way to do this!

   If everything is working correctly, your output should look like:

   ```
   sin(pi/4) + cos(pi/4)/2 is:
   1.06066017178
   ceiling(276/19) + 2 log_7(12) is:
   17.5539788165
   ```

3. Write a new function called `yikes` that has one argument and uses the `multadd` function to calculate the following:
   $$xe^{-x} + \sqrt{(1 - e^{-x})}$$
   There are two different ways to raise e to a power- check out the math module documentation. Be sure to return the result! Try `x=5` as a test; your answer should look like:
   `yikes(5) is 1.0303150673`.

This is the last problem covering Wednesday's material.

Exercise 4.1 – For & While Loops

Be sure to test your code for each part before moving on to the next part. Remember the difference between `input` and `raw_input`? If not, look at Exercise 2.1 again. Write your solutions to this problem in the homework template `homework_2b.py`.

1. Using a for loop, write a program that prints out the decimal equivalents of $1/2, 1/3, 1/4, \ldots, 1/10$.
2. Write a program using a while loop that asks the user for a number, and prints a countdown from that number to zero. What should your program do if the user inputs a negative number? As a programmer, you should always consider "edge conditions" like these when you program! (Another way to put it- always assume the users of your program will be trying to find a way to break it! If you don't include a condition that catches negative numbers, what will your program do?)
3. Write a program using a for loop that calculates exponentials. Your program should ask the user for a base `base` and an exponent `exp`, and calculate $\texttt{base}^{\texttt{exp}}$. Yes, this could be implemented using the built in exponentiation operator (`base ** exp`), but that isn't terribly educational now is it?
4. Write a program using a while loop that asks the user to enter a number that is divisible by 2. Give the user a witty message if they enter something that is not divisible by 2- *and make them enter a new number*. Don't let them stop until they enter an even number! Print a congratulatory message when they *finally* get it right.

Exercise 4.2 – Random Module

1. Write a method `rand_divis_3` that takes no parameters, generates and prints a random number, and finally returns `True` if the randomly generated number is divisible by 3, and `False` otherwise. For this method we'll use a new module, the `random` module. At the top of your code, underneath `import math`, add the line `import random`. We'll use this module to generate a random integer using the function `randint`, which works as follows:

   ```
   random.randint(lo, hi)
   ```

   where `lo` and `hi` are integers that tell the code the range in which to generate a random integer (this range is inclusive). 0 to 100 is probably a decent range.
2. Write a method `roll_dice` that takes in 2 parameters - the number of sides of the die, and the number of dice to roll - and generates random roll values for each die rolled. Print out each roll and then return the string "That's all!" An example output:

   ```
   >>> roll_dice(6, 3)
   4
   1
   6
   That's all!
   ```

Exercise 4.3 – The game of Nims/Stones

In this game, two players sit in front of a pile of 100 stones. They take turns, each removing between 1 and 5 stones (assuming there are at least 5 stones left in the pile). The person who removes the last stone(s) wins.

Download `nims.py` from the website and open it up. Check out the lines of text in between the sets of `'''`, underneath the definition of `play_nims`. This is called a docstring, and is handy to use to tell users of your program what parameters to pass in, and what your program does.

In this problem, you'll write a function to play this game; we've outlined it for you. It may seem tricky, so break it down into parts. Like many programs, we have to use nested loops (one loop inside another). In the outermost loop, we want to keep playing until we are out of stones. Inside that, we want to keep alternating players. You have the option of either writing two blocks of code, or keeping a variable that tracks the current player. The second way could be slightly trickier, but it's definitely do-able!

Finally, we might want to have an innermost loop that checks if the user's input is valid. Is it a number? Is it a valid number (e.g. between 1 and 5)? Are there enough stones in the pile to take off this many? If any of these answers are no, we should tell the user and re-ask them the question.

If you choose to write two blocks of code, the basic outline of the program should be something like this:

```
while [pile is not empty]:
   while [player 1's answer is not valid]:
      [ask player 1]
   [execute player 1's move]

   [same as above for player 2]
```

Be careful with the validity checks. Specifically, we want to keep asking player 1 for their choice as long as their answer is not valid, BUT we want to make sure we ask them at least ONCE. So, for example, we will want to keep a variable that tracks whether their answer is valid, and set it to False initially.

When you're finished, test each other's programs by playing them!

This is the last problem covering Thursday's material.

Exercise 5.1 – Working With Lists

Download `strings_and_lists.py` from the course website. Study the function `sum_all`.

`sum_all` takes a list of numbers as a parameter (note how we specify, with a comment, what the type of the parameter must be), and returns the sum of all the numbers in the list.

Now make a new function `cumulative_sum` that modifies `sum_all` so that instead of returning the sum of all the elements, it returns the cumulative sum; that is a new list where the i^{th} element is the sum of the first $i+1$ elements from the original list. For example, the cumulative sum of `[4, 3, 6]` is `[4, 7, 13]`.

Exercise 5.2 – Pig Latin

In the file `strings_and_lists.py`, write a function `pig_latin` that takes in a single word, then converts the word to Pig Latin. To review, Pig Latin takes the first letter of a word, puts it at the end, and appends "ay". The only exception is if the first letter is a vowel, in which case we keep it as it is and append "hay" to the end.

E.g. "boot" → "ootbay", and "image" → "imagehay".

It will be useful to define a list at the top of your code file called `VOWELS`. This way, you can check if a letter `x` is a vowel with the expression `x in VOWELS`. Remember - to get a word except for the first letter, you can use `word[1:]`.

Be sure to look at the first optional problem for ways to improve on your Pig Latin converter.

This is the last problem covering Friday's material.