6.149 Checkoff 5

`http://web.mit.edu/6.149/www/materials.html`

What to complete

Due: Thursday, January 22, 2015 at 5 p.m.

1. `checkoff5_functions.py`, which will contain your code for 5.1 - Functions
2. `checkoff5_math.py`, which will contain your code for 5.2 - Math module
3. `checkoff5_random.py`, which will contain your code for 5.3 - Random module

We may test your functions using an automated testing suite, so it is extremely important that your function names and return values are exactly what we instruct.

5.0 Review: MITx, Lectures 6-8

Complete all required exercises for Lectures 6-8 on MITx. We switched the order of lectures, so don't worry if the content seems to correspond to different lectures than how it's listed.

5.1 Functions

Download `checkoff5_functions.py`, which will contain your code for the following questions:

1. Transform your code from Checkoff 2.2 into a function called `rock_paper_scissors` that takes two parameters - `player1_move` and `player2_move` - *instead* of asking the user for input. Make sure to *return* one of the following strings, rather than printing the result:

   - Player 1 wins
   - Player 2 wins
   - Invalid input
   - Tie

   Copy and paste your commented test cases from your 2.2 code into your `checkoff5_functions.py` file. For each test case, add commented code that calls `rock_paper_scissors` and prints the result.

2. Write a function `contains_vowels` that takes a string, `word`, as a parameter and returns True if the string contains vowels. Otherwise, the function should return False.

   Create three test cases - in comments, write the input and expected output; the function call; and the actual output after testing. (Of course, if the actual output does not match the expected output, go back and fix your code.)

3. Write a function, `fahrenheit_to_celsius`, that takes one input, `temperature`, a number in degrees Fahrenheit, converts it to degrees Celsius, and returns the resulting value as a float. You may not assume anything about `temperature` except that it is a number.

   $°\text{F} = °\text{C} * \frac{9}{5} + 32$

   Create three test cases - in comments, write the input and expected output; the function call; and the actual output after testing. (Of course, if the actual output does not match the expected output, go back and fix your code.)

5.2 Caesar Cipher

A caesar cipher or shift cipher is a substitution cipher used for turning "plaintext" (the secret message we want to encode) into an encrypted message. To encrypt the message, we replace each plaintext letter and with a different letter that is some fixed number of positions down the alphabet. For more details, see http://en.wikipedia.org/wiki/Caesar_cipher. Note that in a Caesar cipher, the mapping is constant - so if a maps to e once, it will map to e for the duration of the message.

For this problem, we will assume that the cipher always shifts to the right. For example, a cipher with shift 1 would encrypt `"abcde"` to `"bcdef"`.

This problem is easier than it looks! Make sure you take time to read the specifications, then think before you code. Our solution is less than 10 lines per function!

We will write three functions:

1. `get_encoding_dictionary` will return a dictionary that maps a plaintext lowercase letter (key) to an encoded lowercase letter (value). The dictionary should contain a mapping for every lowercase letter in the alphabet. We've provided a constant variable (i.e., variable you shouldn't change) called `ALPHABET` to help you out.
2. `apply_cipher_dict` will apply a dictionary to a string; we will use it to apply an encoding dictionary to a plaintext string, resulting in an encrypted string. You may assume that all characters in the string are letters, spaces, commas, or periods. The message will not contain tab or newline characters, or other symbols. You may not assume anything else about the string.
3. `get_decoding_dictionary` will return a dictionary that maps an encoded lowercase letter (key) to a plaintext lowercase letter (value). The dictionary should contain a mapping for every lowercase letter in the alphabet.

Download `checkoff5_cipher.py` and fill out the functions to the specifications described in the docstrings (the comments enclosed in triple quotes beneath each function header). If the docstring specifies the type of an input (e.g., `num_positions` is an integer), you can assume that the function will always be called with an input that matches the definition. You must use the given function headers. You may not delete anything except lines saying `pass`.