6.009 Quiz 1, Practice 1 -- Spring 2017

This quiz is **open-book and closed-internet**. Feel free to use any physical materials you have brought, but you may not access resources online (except web.mit.edu/6.009). Proctors will be available to answer administrative questions and clarify specifications of coding problems, but they should not be relied on for coding help.

Each problem is worth 5 points, for a total of 25 points. Your raw test score will be scaled by 20/25 to compute the number of final points.

You must submit your modified `quiz.py` via web.mit.edu/6.009 before the deadline in order to receive credit. This quiz assumes you have Python 3.5 (or a later version) installed on your machine.

The `resources` directory contains the **Python Language Reference** and the **Python Library Reference** in PDF form. Please **do not import any Python modules** to use as part of your solutions -- quizzes with `import` statements (or their equivalent!) will be given a grade of 0.

As in the labs, `test.py` can be used to test your code in `quiz.py`. Remember that you can run specific tests by listing the test numbers, like so

```
python3 test.py 1 3 5      # run tests #1, #3, #5
```

Your score will be computed from the number of tests you pass. If you pass all the tests for a problem, you'll receive full credit. If you pass 2 of the 5 tests, you'll receive 40% of the credit for the problem. Note that to pass some of the tests in the time allotted, your solution may have to be reasonably efficient.

This practice quiz is on the long side -- the actual quiz will have at least one fewer question.

Problem 1: `hangman` (tests 1-5)

In the game of Hangman, the player is trying to guess the secret word one character at a time. The secret word is displayed after each guess, with as-yet unguessed characters shown as "_".

Please implement the function `hangman(secret_word,guessed_letters)` where both `secret_word` and `guessed_letters` are lists of characters. `hangman` should return a list of characters showing the secret word with unguessed characters displayed as underscores.

Examples:

`hangman(['h','i'],[])` should return `['_','_']`.

`hangman(['b','o','o','k','k','e','e','p','e','r'],['o','e'])` should return `['_','o','o','_','_','e','e','_','e','_']`.

`hangman(['q','u','i','n','q','u','u','x'],['i','u','q','n','x'])` should return `['q','u','i','n','q','u','u','x']`.

Problem 2: `mode` (tests 6-10)

Given a list of numbers, return the mode or the mean of modes if there are more than one mode. The mode is a number that appears most often in the list. It does not have to be unique, since several different numbers may appear the same number of times. In those cases, you need to return the mean of the modes. You may assume the list of numbers contains at least one element.

Examples:

`mode([314159, 314159, 2048, 333])` should return `314159`.

`mode([1, 1, 2, 2, 3])` should return `1.5`.

Problem 3: `most_repeated_character` (tests 11-15)

Given a list of characters, find sequences of a single repeated character. The repeats need to be consecutive. Return the length of the longest such sequence. Note that there may be several "longest" sequences.

Examples:

`most_repeated_character([])` should return `0`.

`most_repeated_character(['a','b','c','a','b','c'])` should return `1`.

`most_repeated_character(['a','b','c','a','a','a'])` should return `3`.

`most_repeated_character(['a','a','b','b','c','c'])` should return `2`.

Problem 4: `integer_right_triangles` (tests 16-20)

Let p be the perimeter of a right triangle with integral, non-zero length sides of length a, b, and

c. So we know

- p = a+b+c.
- $a^2 + b^2 = c^2$.

There are exactly three solutions for p = 120, listed below as `[a, b, c]`

```
[20,48,52], [24,45,51], [30,40,50]
```

Implement the function `integer_right_triangles(p)` which returns a sorted list of solutions with perimeter p. Please list the three lengths for each solution in increasing order, i.e., `[3,4,5]` not `[4,3,5]`.

Examples:

`integer_right_triangles(12)` returns `[[3,4,5]]`

`integer_right_triangles(60)` returns `[[10,24,26], [15,20,25]]`

`integer_right_triangles(152)` returns `[]`

Note: A triply nested for loop will time out and not receive full credit.

Problem 5: Encoding Nested Lists (tests 21-25)

Please implement the function `encode(seq)`, which, given a list (array) `seq` of nested lists of numbers, returns a flat list conveying the same information. The flat list is essentially what one gets by reading the nested list from left to right, replacing each open bracket with `'up'`, each close bracket with `'down'`, and each number with itself. Commas and spaces are not carried over in this translation, and note that your function is to be written over Python's representation of nested lists, which doesn't naturally include "commas" or "spaces," anyway.

Examples:

`encode([1])` should return `['up', 1, 'down']`.

`encode([1, [2], 1])` should return `['up', 1, 'up', 2, 'down', 1, 'down']`.

`encode([[[1, [2]]]])` should return `['up', 'up', 'up', 1, 'up', 2, 'down', 'down', 'down', 'down']`.

Note: We recommend using Python's built-in function `isinstance(x, list)` to check whether `x` is a list, in cases where `x` might also be a number.