Assignment 3: due 8PM on Sunday, March 17, 2019

Summary of Instructions

Note	**Read the instructions carefully and follow them exactly**
Assignment Weight	**6% of your course grade**
Due Date and time	**8PM on Sunday, Match 17, 2019**
Important	**As outlined in the syllabus, late submissions will not be accepted**
	Any files with syntax errors will automatically be excluded from grading. Be sure to test your code before you submit it
	For all functions, both in Part 1,2 and 3, make sure you've written good docstrings that include type contract, function description and the preconditions if any.

–

This is an individual assignment. Please review the Plagiarism and Academic Integrity policy presented in the first class, i.e. read in detail pages 11-19 of course outline (course-outline2019.pdf). You can find that file on Brightspace under Course Info. While at it, also review Course Policies on pages 12- 14.

The goal of this assignment is to learn and practice the concepts covered thus far, in particular: strings/ list (including indexing, slicing and string/list methods), control structures (if statements and for-loops), use of range function, function design and function calls.
The only collection you can use are **list, tuple and 2D list**. You may not use any other collection (such as a set, or dictionary) in this assignment. Using any of these in a solution to a question constitues changing that question. Consequently, that question will not be graded.

You can make multiple submissions, but only the last submission before the deadline will be graded. What needs to be submitted is explained next.

The assignment has two parts. Each part explains what needs to be submitted. Put all those required documents into a folder called a3_xxxxxx where you changed xxxxxx to your student number, zip that folder and submit it as explained in Lab 1. In particular, the folder should have the following files:

Part 1: a3_part1_xxxxxx.py, a3_part1_xxxxxx.txt
Part 2: a3_part2_xxxxxx.py and a3_part2_xxxxxx.txt
Part 3: a3_part3_xxxxxx.py and a3_part3_xxxxxx.txt

Both of your programs must run without syntax errors. In particular, when grading your assignment, TAs will first open your file a2_part1_xxxxxx.py with IDLE and press Run Module. If pressing Run Module causes any syntax error, the grade for Part 1 becomes zero. The same applies to Part 2 and 3.

Note: You must import the Random library to solve this assignment.

Part 1: Function Library (70 points)

For this part of the assignment, you are required to write and test several functions (as you did in Assignment 1 and 2). You need to save all functions in part1_xxxxxx.py where you replace xxxxxx by your student number. You need to test your functions (like you did in Assignment 1) and copy/paste your tests in part1_xxxxxx.txt. Thus, for this part you need to submit two files: a3_part1_xxxxxx.py and a3_part1_xxxxxx.txt

[1] Implement a function `coprime` described below. The function takes as input two positive integers `x` and `y` and returns True if `x` and `y` are coprimes and False otherwise. Two positive integers `x` and `y` are said to be coprime if the only positive integer that divides both of them is 1. That is, the only common positive factor of the two numbers is 1.

```
def  coprime(x,y):
    '''(int,int)->bool
    Precondition:  x  and  y  are  both  positive  integers
    >>>  coprime(1,7)
    True
    >>>  coprime(21,14)
    False
    >>>  coprime(14,15)
    True
    >>>  coprime(7,7)
    False'''
```

[2] Write the body of the function `all_coprime_pairs` described below. The function takes as input a list, `L`, of positive distinct integers (that is, no two integers in `L` are the same). The function should return a list (of tuples) containing all pairs of numbers in `L` that are coprimes. If no pair of numbers in `L` is coprime, the function should return an empty list. Your solution must include a function call to the function `coprime` designed in the previous question. The order of the tuples or the order of the two numbers within the tuples is not important.

```
def all_coprime_pairs(L):
    '''(list)->list_of_tuples
    Precondtion: L is a list of positive distinct integers and
        len(L)>=2

>>> all_coprime_pairs([21,1, 7,14, 15])
[(21, 1), (1, 7), (1, 14), (1, 15), (7, 15), (14, 15)]
>>> all_coprime_pairs([18,6,9])
[]
```

[3] Write a function named `zero_out` that accepts two lists of integers a1 and a2 as parameters and replaces any occurrences of a2 in a1 with zeroes. The sequence of elements in a2 may appear anywhere in a1 but must appear consecutively and in the same order. For example, if variables called a1 and a2 store the following values:

```
a1 = [1, 2, 3, 4, 1, 2, 3, 4, 5]
a2 = [2, 3, 4]
```

The call of `zero_out(a1, a2)` should modify `a1`'s contents to be `[1, 0, 0, 0, 1, 0, 0, 0, 5]`.

Note that the pattern can occur many times, even consecutively. For the following two lists `a3` and `a4`:

```
a3 = [5, 5, 5, 18, 5, 42, 5, 5, 5, 5]
a4 = [5, 5]
```

The call of `zero_out(a3, a4)` should modify `a3`'s contents to be `[0, 0, 5, 18, 5, 42, 0, 0, 0, 0]`.

You may assume that both lists passed to your function will have lengths of at least 1. If *a2* is not found in *a1*, or if *a1*'s length is shorter than *a2*'s, then *a1* is not modified by the call to your function. Please note that *a1*'s contents are being modified in place; you are not supposed to return a new list. Do not modify the contents of *a2*.

[4] Write a function `coin_flip` that accepts as its parameter an input file name. Assume that the input file data represents results of sets of coin flips that are either heads (H) or tails (T) in either upper or lower case, separated by at least one space. Your function should consider each line to be a separate set of coin flips and should output to the console the number of heads and the percentage of heads in that line, rounded to the nearest tenth. If this percentage is more than 50%, you should print a "You win" message. For example, consider the following input file:

```
H T H H T
T t   t T h   H
   h
```

For the input above, your function should produce the following output:

```
3 heads (60.0%)
You win!

2 heads (33.3%)

1 heads (100.0%)
You win!
```

The format of your output must exactly match that shown above. You may assume that input file contains at least 1 line of input, that each line contains at least one token, and that no tokens other than h, H, t, or T will be in the lines.

[5] A *run* is a sequence of adjacent repeated values. Write a body of function `Run()` that generates a sequence of **20 random die tosses** in a list and that prints the die values, marking the runs by including them in parentheses, like this:

```
1 2 (5 5) 3 1 2 4 3 (2 2 2 2) 3 6 (5 5) 6 3 1
```

[6] Craps is a dice-based game played in many casinos. Like blackjack, a player plays against the house. The game starts with the player throwing a pair of standard, six-sided dice. If the player rolls a total of 7 or 11, the player wins. If the player rolls a total of 2, 3, or 12, the player loses. For all other roll values, the player will repeatedly roll the pair of dice until either she rolls the initial value again (in which case she wins) or 7 (in which case she loses)

a) Implement function `craps()` that takes no argument, simulates one game of craps, and returns 1 if the player won and 0 if the player lost.

```
>>> craps()
0
>>> craps()
1
>>> craps()
1
```

b) Implement function `testCraps()` that takes a positive integer n as input, simulates *n* games of craps, and returns the fraction of games the player won.

```
>>> testCraps(10000)
0.4844
>>> testCraps(10000)
0.492
```

[7] Write a function called `is_all_even` that takes a list of lists as a parameter and returns True if all of the integer elements of the lists are even. For example, if passed in the following list of lists:

```
lis = [[2, 4, 4], [2, 8, 88, 14], [30, 60, 92]]
```

a call of is_all_even(lis) would return True. If an empty list is passed to your function it should return True.

[8] Write a function `range` that accepts a list of lists as a parameter and that returns the range of values contained in the list of lists, which is defined as 1 more than the difference between the largest and smallest elements. For example if a variable called `lis` stores the following values: `[[18, 14, 29], [12, 7, 25], [2, 22, 5]]` The call of `range(lis)` should return 28, because this is one more than the largest difference between any pair of values (29 - 2 + 1 = 28). An empty list is defined to have a range of 0. You may not make any assumptions about the range of numbers in the list of lists. You may not alter the list. You may not create any other data structures.

Part 2: (Simulation: coupon collector's problem) (10 points)

Coupon Collector is a classic statistics problem with many practical applications. The problem is to pick objects from a set of objects repeatedly and find out how many picks are needed for all the objects to be picked at least once. A variation of the problem is to pick cards from **a shuffled deck of 52 cards** repeatedly and find out how many picks are needed before you see one of each suit. Assume a picked card is placed back in the deck before picking another. **Write a program to simulate the number of picks needed to get four cards, one from each suit and display the four cards picked (it is possible a card may be picked twice)**. Here is a sample run of the program:

```
4 of Diamonds
8 of Spades
Queen of Clubs
8 of Hearts
Number of picks: 9
```

Part 3: (Game: hangman) (20 points)

Write a hangman game that randomly generates a word and prompts the user to guess one letter at a time, as shown in the sample run. Each letter in the word is displayed as an **asterisk**. When the user makes **a correct guess**, **the actual letter** is then **displayed**. When the user finishes a word, display the number of misses and ask the user whether to continue playing. Create a list to store the words, as follows:

```
words = ["write", "program", "that", "receive", "positive",
"change", "study", "excellent", "nice"]
```

```
(Guess) Enter a letter in word ******* > p Enter
(Guess) Enter a letter in word p****** > r Enter
(Guess) Enter a letter in word pr**r** > p Enter
   p is already in the word
(Guess) Enter a letter in word pr**r** > o Enter
(Guess) Enter a letter in word pro*r** > g Enter
(Guess) Enter a letter in word progr** > n Enter
   n is not in the word
(Guess) Enter a letter in word progr** > m Enter
(Guess) Enter a letter in word progr*m > a Enter
The word is program. You missed 1 time

Do you want to guess another word? Enter y or n>
```