Object-Oriented Programming
50:198:113 (Spring 2022)

Homework:	**1**	**Professor:**	**Suneeta Ramaswami**
Due Date:	**2/7/22**	**E-mail:**	`suneeta.ramaswami@rutgers.edu`
Office:	**321 BSB**	**URL:**	`http://crab.rutgers.edu/~rsuneeta`
		Phone:	**(856)-225-6439**

Homework Assignment 1

The assignment is due by 11:59PM of the due date. The point value is indicated in square braces next to each problem. Each solution must be the student's own work. You may seek or accept assistance only from the course instructor. Any violation of this rule will be dealt with harshly.

This assignment is primarily a review of loops, strings, lists, functions, and writing modular programs with minimal code repetition. In this and all future assignments, you are graded not only on the correctness of the code, but also on clarity and readability. Hence, I will deduct points for poor indentation, poor choice of object names, and lack of documentation. For documentation, use a common sense approach. While I do not expect every line of code to be explained, all code blocks that carry out a significant task should be documented *briefly* in clear English.

Please read the submission guidelines at the end of this document before you start your work.

Important note: When writing each of the following programs, it is important that you name the functions exactly as described because I will assume you are doing so when testing your programs. If your program produces errors because the functions do not satisfy the stated prototype, points will be deducted. If I am unable to import your module due to syntax errors, I will automatically deduct 50% of the points for that problem. You may implement additional helper functions if you wish, but you must, at a minimum, implement the functions you are asked to implement in each of the three problems. In particular,

1. I will type `multi_column_print()` in the `Python` shell to test `problem1.py`.
2. I will type `craps()` in the `Python` shell to test `problem2.py`.
3. I will type `int_to_text()` in the `Python` shell to test `problem3.py`.

For each problem, put all your function definitions in one module. The files `problem1.py`, `problem2.py`, and `problem3.py` should contain *all* the function definitions for Problems #1, #2, and #3, respectively.

Problem 1 [15 points] Multi Column Printing. In this problem, you are asked to implement a function called `multi_column_print` with two parameters: a list `L` and an integer `numcols`. The list `L` has an arbitrary number of items (each item can be any built-in object, such as integer, real, list, string etc.). Your function should display the items in `L` equally distributed in `numcols` columns. You should ensure that the columns are neatly formatted and aligned by using string formatting with a suitable field width. Some sample runs are shown below.

```
>>> from problem1 import *
>>> L = [[1, 2, 3, "abc"], "hello", [10, 20, 30], 'a', 45, 27, 99, 4.5, 3.14159]

>>> multi_column_print(L, 3)
 [1, 2, 3, 'abc']            hello     [10, 20, 30]
                a               45               27
               99              4.5          3.14159

>>> L = [2**i for i in range(20)]

>>> multi_column_print(L, 5)
      1      2      4      8     16
     32     64    128    256    512
   1024   2048   4096   8192  16384
  32768  65536 131072 262144 524288

>>> multi_column_print(L, 7)
      1      2      4      8     16     32     64
    128    256    512   1024   2048   4096   8192
  16384  32768  65536 131072 262144 524288

>>> multi_column_print(L, 2)
      1      2
      4      8
     16     32
     64    128
    256    512
   1024   2048
   4096   8192
  16384  32768
  65536 131072
 262144 524288
```

Problem 2 [20 points] Craps, a game of chance. A popular game of chance frequently seen at casinos is a dice game known as "craps". The game is played as follows. A player rolls two dice. Each die is the familiar six-sided die with faces containing 1, 2, 3, 4, 5, and 6 dots. After the dice are rolled, the next step is determined by the sum of the dots on the two top faces.

- If the sum is 7 or 11 on the first throw, the player wins.
- If the sum is 2, 3, or 12 on the first throw, the player loses (called "craps").
- If the sum is 4, 5, 6, 8, 9, or 10 on the first throw, then that sum becomes the player's "point". Now, the only way for the player to win is by continuing to roll the dice until she "makes her point". If the player rolls a 7 before making her point, she loses. In other words, if the roll of the dice adds up to her point, she wins; if it adds up to 7, she loses; if it adds up to neither, she rolls the dice again.

Note that there are two ways for a player to win: Either by rolling a 7 or 11 in the first roll of the dice, or by rolling her point in a subsequent roll of the dice.

You are asked to write a program to play craps to allow wagering. The player starts with an initial bank balance of $1000. Each game starts with a wager (which of course must be no

bigger than the current bank balance). After one game, the bank balance is updated and the player is allowed to play again, repeatedly, until she quits, or the bank balance falls to $0. You will accomplish all this by implementing the following functions.

1. **(3 points)** Write a function called `roll_dice` with no parameters. The function rolls two dice and return the sum of the result. You can simulate the roll of a die by using the `randint(a, b)` function from the `random` module. `randint(a, b)` returns a random integer `N` such that `a` $\leq$ `N` $\leq$ `b`.
2. **(10 points)** The second function called `play_one_game` plays a single game of craps, using the rules described above. This function does not have parameters. It should print out sentences informing the player about the sequence of actions taking place in the game (see sample runs below - your code should mimic the output shown). It returns an integer value of 1 if the player wins the game, and in integer value of 0 if the player loses the game.
3. **(7 points)** The third function called `craps` does not have any parameters. It uses a `while` loop to allow the player to play as many rounds of craps as she wants, as long as the bank balance is not $0. The function should first prompt the user to enter a wager. If the wager is greater than the bank balance, repeatedly prompt the player to re-enter the wager until a valid wager is entered. Then run one game of craps (using the `play_one_game` function), inform the player if she won or lost and update the bank balance accordingly. If the new balance is $0, print a message (`"Sorry, you're broke!"`), and quit. As long as the balance is greater than $0, the player may repeatedly choose to play again. If the player quits with a bank balance greater than $0, print a congratulatory message if money was made and a sympathetic message if money was lost.

A sample run is shown below.

```
                ----------------------------
                Welcome to the Craps program
                ----------------------------

Your initial bank balance is $1000.

What is your wager? 100
Okay, let's play.

You rolled 11
You win!!

Your new bank balance is $1100

Do you want to play again? [y/n] y

What is your wager? 500
Okay, let's play.

You rolled 12
Sorry, you lose!

Your new bank balance is $600
```

```
Do you want to play again? [y/n] y

What is your wager? 700
Cannot wager more than $600. Re-enter wager: 800
Cannot wager more than $600. Re-enter wager: 400
Okay, let's play.

You rolled 4
Your point is 4

You rolled 5
You rolled 9
You rolled 10
You rolled 9
You rolled 3
You rolled 7
Sorry, you lose!

Your new bank balance is $200

Do you want to play again? [y/n] y

What is your wager? 500
Cannot wager more than $200. Re-enter wager: 200
Okay, let's play.

You rolled 10
Your point is 10

You rolled 6
You rolled 11
You rolled 6
You rolled 10
You win!!

Your new bank balance is $400

Do you want to play again? [y/n] n

Sorry you lost money. Better luck next time!
```

Problem 3 [15 points] Integers to Text. Implement a function called `int_to_text` with a single parameter `N`, which is an integer between 0 and 1000 (inclusive). The function returns a string that is the English text equivalent of that integer. For example, `int_to_text(89)` should return the string `'eighty nine'`, `int_to_text(7)` should return `'seven'`, `int_to_text(14)` should return `'fourteen'`, and `int_to_text(621)` should return `'six hundred twenty one'`. **Important:** Make good use of lists and if/elif statements to implement your function compactly. If you have a *very* large number of if/elif statements or if you have extremely long lists (length longer than 10, say) in your code, you are on the wrong track. For instance, my implementation has just five if-else clauses.

Submission Guidelines

Implement the first problem in a module called `problem1.py`, the second one in a module called `problem2.py`, and the third one in a module called `problem3.py`. **Your name and RUID should appear as a comment at the very top of each file.**

Test each of your programs thoroughly before submitting your homework. When you are ready to submit, upload your files on Sakai as follows:

1. Go to the "Assignments" tab of the Canvas site for this course.
2. Click on "Programming Assignment #1" under Homework Assignments.
3. Download the homework document (`hw1.pdf`). Carefully read the problem descriptions in the homework document before creating the module for each problem.
4. Use this same link to upload your homework files (`problem1.py`, `problem2.py`, and `problem3.py`) when you are ready to submit.

You must submit your assignment at or before 11:59PM on February 7, 2022.