Object-Oriented Programming
50:198:113 (Spring 2016)

Homework:	**1**	**Professor:**	**Suneeta Ramaswami**
Due Date:	**2/8/16**	**E-mail:**	`rsuneeta@camden.rutgers.edu`
Office:	**321 BSB**	**URL:**	`http://crab.rutgers.edu/~rsuneeta`
		Phone:	**(856)-225-6439**

Homework Assignment 1

This assignment will review material covered in the introductory course on Python. It requires you to write functions, use loops and lists, and process strings. In this and all future assignments, you are graded not only on the correctness of the code, but also on clarity and readability. 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.

The assignment is due by 11:59PM of the due date (given above). The point value is indicated in square braces next to each problem. Each solution must be the student's own work. Assistance should be sought or accepted only from the course instructor or the TA. Any violation of this rule will be dealt with harshly.

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. Points will be deducted if your program produces errors because the functions do not satisfy the stated prototype. In particular,

1. After importing the `problem1` module, I will call the `magic_list` function to test `problem1.py`.
2. After importing the `problem2` module, I will type `craps()` in the `Python` shell to test `problem2.py`.
3. After importing the `problem3` module, I will type `pig_latin_translator()` in the `Python` shell to test `problem3.py`.

For each problem, put all your function definitions in one file. *Do not create a separate file for each function.* The files `problem1.py`, `problem2.py`, and `problem3.py` should contain *all* the function definitions for Problems #1, #2, and #3, respectively. **Please read the submission guidelines at the end of this document before you start your work.**

Problem 1 [10 points] Magic numbers. A positive integer m greater than 1 is said to be a *magic number* if the sum of all its divisors (other than m) equals m. For example, 6 is a magic number because $1 + 2 + 3 = 6$, and 28 is a magic number because $1 + 2 + 4 + 7 + 14 = 28$, whereas 15 is *not* a magic number because $1 + 3 + 5 = 9 \neq 15$, and 24 is *not* a magic number because $1 + 2 + 3 + 4 + 6 + 8 + 12 = 36 \neq 24$.

In this program, you will write two functions, as described below. *Important note:* Do not repeat code unnecessarily. If you have a function to carry out a specific task, call that function rather than repeating those lines of code.

- *(5 points)* A function called `magic` with a single parameter n that returns `true` if n is a magic number and `false` otherwise.
- *(5 points)* A function called `magic_list` with a single parameter `num`. It **returns a list** of all magic numbers between 2 and `num` (inclusive). *Note:* There are not that many magic numbers out there!! To see a magic number other than 6 and 28, import your module in the Python shell and call `magic_list` with parameter 500. To see the next magic number, you will need to go even higher (over 8000).

Here is a sample output:

```
>>> magic_list(100)
[6, 28]
```

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.[1] 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
```

[1]The `input` function reads in a value as a string. Use the `int(numstring)` function which returns the integer corresponding to the digit string `numstring`. For example, `int("123")` returns the integer `123`.

```
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 [20 points] Pig Latin. Pig Latin is a form of coded English often used for amusement. There are many variations on the method used to form pig Latin sentences. Here, we use the following simple algorithm to translate an English word into pig Latin:

- If the English word begins with a consonant, place the initial cluster of consonant letters at the end of the word, followed by the letters "`ay`" to it. For example, "`road`" translates to "`oadray`", "`scream`" translates to "`eamscray`", and "`fly`" translates to "`flyay`".
- If the English word begins with a vowel (a vowel is one of 'a','e','i','o', or 'u'), simply add the letters "`way`" at the end of the word. For example, "`apple`" translates to "`appleway`" and "`I`" translates to "`Iway`".

Write a program that translates English language sentences into pig Latin. We will assume that the sentence is made up of words separated by blanks. A word is a string of letters without blanks and may end in a punctuation mark, which we assume to be one of . (period) ! (exclamation) ? (question mark) or , (comma). We assume there are no other punctuation marks in the sentence. The punctuation marks and blanks stay as they are in the pig Latin translation. Implement the following functions in your program.

- *(8 points)* Write a function called `pig_latin_word` with a single parameter `word`. Assume that `word` is a string of characters without any spaces or punctuation marks in it. The function returns a string that is the pig Latin translation of `word`.

- *(7 points)* Write a function called `pig_latin_sentence` with a single parameter `eng_sentence`, which is an English language sentence. The function returns a string which is the pig Latin translation of `eng_sentence`. *Hint:* Construct the pig Latin translation by breaking up `eng_sentence` into words, each of which can be translated using `pig_latin_word`. Make sure you handle punctuation marks properly.
- *(5 points)* Write a function called `pig_latin_translator` without any parameters. The function repeatedly reads in an English sentence from the user and then uses the function `pig_latin_sentence` to print out the pig Latin translation of that sentence.

Here is a sample run:

```
                 ------------------------------
                 English to Pig Latin Translator
                 ------------------------------

Enter the English sentence: Look! His tie is ugly.

In Pig Latin: ookLay! isHay ietay isway uglyway.

Do another? [y/n] y

Enter the English sentence: Are you speaking Pig Latin?

In Pig Latin: Areway ouyay eakingspay igPay atinLay?

Do another? [y/n] n

Goodbye!
```

Submission Guidelines

Implement the first problem in a file called `problem1.py`, the second one in a file called `problem2.py`, and the third one in a file 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. Use your web browser to go to the website `https:sakai.rutgers.edu`.
2. Log in by using your Rutgers login id and password, and click on the `OBJECT-ORIENTED PROG S16` tab.
3. Click on the 'Assignments' link on the left and go to 'Homework Assignment #1' to find the homework file (`hw1.pdf`).
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 8, 2016.