CS 232 – Programming in Python

Assignment #4

Deadlines

This assignment is due on **Thursday, May 5, 2016 @ 5:00 PM (beginning of lab)**

How to submit your work on the assignment:

- Your assignment will be turned in as a computer file (the "module") containing your Python code. The module will contain only the code that satisfies the problems' requirements.

The File Naming Convention

- Properly name the file that contains your work. The file's name will contain the following, with dashes in between each part listed below and no spaces:

  * The course name, **`CS232`**

  * Then a dash and the assignment number as a two-digit number, in this case **`04`**

  * Then a dash and your HSU account username with your initials and a number – in this example, I'll use **`jqs123`** for a typical student named John Q. Smith

  * Windows should automatically add a **`.py`** at the end of the file. This may be invisible, depending on how Windows is configured on the computer you're using. If you're using a Mac, you'll likely need to add a **`.py`** to the end of the filename.

  Put all the rules together, and John Q. Smith's Assignment 1 file in CS 232 would be named

  `CS232-04-jqs123.py`

  Your file name will be different from this because your HSU username is different, but that's the **only** difference!

Submitting your file electronically

For this assignment, you will email your file to the instructor as a file attachment. Future assignments may change the way you submit your work, but email will do for now.

Send your email to: **`david.tuttle@humboldt.edu`**

Type the Subject line of the email as: **`CS 232 – Assignment #4`**

When I receive your file in the correct way, I will send you an acknowledgment by reply email. If there's a problem, I will let you know by reply email that you need to submit the file again.

If you do NOT get an email reply within 12 hours, try sending the file again. If you wait until too close to the deadline, you run the risk that an improper submission may result in a late penalty!

Documentation of your Python code

For this assignment, please place all functions within a single Python module (file).

Begin your **Python module** (file) that you submit with at least the following opening comments:

* a comment containing the name of the file,
* a comment containing your name, and
* a comment containing the date that your module was last modified

For example:

```
# CS232-04-jqs123.py
# John Q. Smith
# Last modified: February 30, 2099
```

In this homework, the documentation at the beginning of each function should be written:

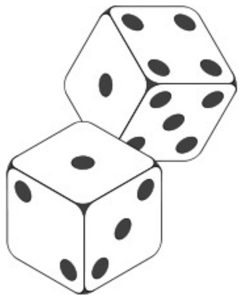
For example:

```
# add_them: float float → float
# purpose:  expects two numeric (float) values
#           returns the sum of the two values
# side effects: prints a message containing the sum of the two values
```

PROJECT – CREATING A "DIE" OBJECT AND ROLLING IT

Use these libraries and global variables that should remain immutable (they're "constants") – you MUST use these constants in your code so if they're changed, everything will still work as it should. Your code should handle four 20-sided die rolled nine times per test just as easily as five 6-sided die rolled three times per test.

```
import random
import collections
SIDES = 6         # Number of sides on each die
NUM_DICE = 5      # Number of dice in the game
NUM_ROLLS = 3     # Number of rolls for each test
NUM_TESTS = 1000  # Number of tests to perform (extra credit)
```



Part I

Define a class called **`Die`** (singular for "dice") that will have the following methods:

`__init__(self, number_of_sides)` creates a **`Die`** object with the given number of sides (for example, **`a_die = Die(6)`** a "normal" 6-sided die, or **`my_die = Die(20)`** for a gamer's 20-sided die. It should even be able to handle an odd number, such as a 17-sided die! We are not limited by the need for physical symmetry here.

roll(self) will roll the **Die** based on the number of sides and assigns the value rolled to a "hidden" variable – a positive integer from 1 to the number of sides. For example, a 6-sided **Die** will generate and return a value from 1 to 6. Hint: **random.randint(1,100)** will generate a random integer whose value is equally likely to be any integer from 1 through 100, including both 1 and 100.

value(self) will return an integer, the current value of the die as it was last rolled. If the die has never been rolled, it should return a value of **0**.

Also, in your "main" code at the bottom of the Python module, write Python code to create a list called **my_dice** whose items are **Die** objects. This **my_dice** list will represent the dice in your main executable code, and must match the requirements for the **the_dice** parameter for the pre-written functions.

Part II

Using **Die** objects as defined above, you will define the function called **play_yahtzee(the_dice)** (much like **play_hangman()** in the previous project), that will perform the simulation as specified below.

The function will return **True** if the outcome is a Yahtzee, and **False** if it is not.

In the game of Yahtzee, a player rolls dice. In your function, the player will try to get a Yahtzee by following these rules:

(1) For the first roll, roll all the dice
(2) Determine which value is on the most dice, and set those dice aside so they won't be re-rolled
(3) Re-roll all the dice that weren't set aside
(4) Repeat steps (2) and (3) until you're out of rolls (in Yahtzee, it's a maximum of 3 rolls)
(5) If, after the rolls, all the dice have the same value, you have a Yahtzee! If not, then you don't.

Consider the following code to print the values of all your dice to the screen. The parameter **the_dice** is a list of **Die** objects – notice how each die is represented by **the_dice[i]** :

```
# print_dice: list of Die -> nothing
# purpose: expects a list of Die objects
#    returns nothing
# side effect: prints to the screen the values on the dice
def print_dice(the_dice):
    print("The dice values are:", end=' ')
    for i in range(0, NUM_DICE):
        print(the_dice[i].value(), end=' ')
    print()
```

You will be supplied the following helper functions (which are <u>**NOT TO BE MODIFIED**</u>):

- A function **is_yahtzee(the_dice)** that returns **True** if the dice all have the same value, and **False** if they don't.

- A function **best_value_to_keep(the_dice)** that looks at the values on the dice and returns which value is found most often on the dice, so you can determine which dice to NOT roll.

- A function **`print_dice(the_dice)`** that you'll use to print the dice after each roll.
- For full credit, your code should run a single Yahtzee test and print "Yahtzee!" to the screen if, after **`NUM_ROLLS`** rolls, all of the **`NUM_DICE`** dice have the same value, or "No Yahtzee" if they don't. Your printed output of the **`play_yahtzee(the_dice)`** function MUST look as follows, with calls to **`print_dice(the_dice)`** and a single-line message of success or failure:

```
The dice values are: [1, 6, 2, 5, 5]
The dice values are: [5, 5, 4, 5, 5]
The dice values are: [5, 5, 5, 5, 5]
*** Yahtzee! ***
```

or

```
The dice values are: [6, 5, 1, 4, 4]
The dice values are: [6, 1, 6, 4, 4]
The dice values are: [6, 3, 6, 4, 6]
    No Yahtzee
```

For 10% extra credit: Write code to run multiple tests (use the **`NUM_TESTS`** variable) and, after all the tests are run, prints to the screen a brief report on how many tests were conducted, and how many Yahtzees were recorded.

For the curious: according to an article on **`statistics.about.com`**, the expected success rate of a Yahtzee using five 6-sided dice based on the method outlined in this project is 4.74%.

If you change the global variables, you can see how the chances of a Yahtzee change when you change the sides on each die, or change the number of dice, or change the number of rolls!

Special Note

We all know that the game of Yahtzee is more than just trying to get a Yahtzee. But this project is NOT about doing anything else beyond its specifications.

STICK TO THE SPECIFICATIONS FOR THIS PROJECT – attempts to add more features that were not requested will be met with **DEDUCTIONS** in your score!

A good programmer meets the specifications of the tasks assigned and does not modify or add to specifications on a whim. Provide good, solid code, without errors, that doesn't do too little or too much, and you will be rewarded.