Final Project
Option #1 – Restaurant Simulation

Goals

- Complete a project that demonstrates problem solving skills and logical thinking.
- Demonstrate knowledge of Python programming concepts learned throughout the semester.
- Choose only **one** option.

Overview

- You will be writing a program that simulates a restaurant using object oriented programming. You will be creating objects that represent a waiter, diners, a menu, and menu items, as well as reading in data from a csv file.
- It is recommended that you implement the code in the order listed below. After completing each part, it is also recommended that you test the functionality of your existing code before proceeding to the next part.
- To best understand the structure and flow of the simulation, please read all the directions through before beginning any code.

Requirements

- Create a new Python project named **`ITP115_project_`*`lastname_firstname`*** (replace ***`lastname`*** with your last/family name and ***`firstname`*** with your first name).
- This project will be a folder containing the required class files, the helper file, and the given csv file.
- Use proper coding styles and comments.
- Name newly created python files as directed below.
- Project should perform error-checking (on all inputs).

Files Provided

Download the following Python files and put them in your newly created project.

`RestaurantHelper.py`

- We have provided a helper file for you called **`RestaurantHelper.py`** which includes helper methods to allow diners to randomly show up to the restaurant.
- You will not need to interact directly with this file, however it is important that you add it to your directory with the other project files.
- Important Note: For this file to run, you will need to have **`BeautifulSoup`** installed. If you do not yet have it installed, follow the directions in the Week 8 Web Scraping lecture (slide 25) to do so.

`Run.py`

- We have also provided a main function for you in a file called **`Run.py`** which starts the simulation. This is the only file that you will need to run and it should also be placed in the same directory as your project files.
- You will need to update the restaurant name variable (currently an empty string) to your own restaurant name.
- Currently, a few lines are commented out because the classes detailed below are not yet implemented. Once the classes are fully implemented you should be able to uncomment those lines (the "**`TODO`**" comments will detail where you should be uncommenting) to run the full simulation. Right now, the program will only print out different times, pausing for 1 second between intervals.

Part 1 – Creating the Restaurant's Menu

To represent a menu, you will be creating two separate classes: **`MenuItem`** and **`Menu`**.

MenuItem Class

- Start by defining the **`MenuItem`** class in a file titled **`MenuItem.py`**.
- This class will represent a single item that a diner can order from the restaurant's menu.
- The class will use the following instance attributes (all should be private):
  - **`self.__name`**: a string representing the name of the **`MenuItem`**
  - **`self.__type`**: a string representing the type of item
  - **`self.__price`**: a float representing the price of the item
  - **`self.__description`**: a string containing a description of the item
- Also define the following methods:
  - **`__init__`**
    - Inputs (4): the name (string), type (string), price (float), and description (string) of the dish
    - Return value: none
    - Assign the inputs to the 4 instance attributes listed above.
    - Note: "type" is a Python keyword, so avoid naming any variables "type".
  - Define **getters and setters** for each of the 4 attributes. You should have 4 getters and 4 setters. Note that not all of these may be used.
  - **`__str__`**
    - Inputs: none
    - Return value: a string
    - Construct a message containing all 4 attributes, formatted in a readable manner. For example:

```
Fresh Spring Rolls (Appetizer): $3.99
      4 vegetarian rolls wrapped in rice paper either fresh or fried
```

Menu Class

- Next, define the **`Menu`** class in a file titled **`Menu.py`**. This class will make use of **`MenuItem`** objects, so be sure to include the proper import statement.
- This class represents the restaurant's menu which contains four different categories of menu items diners can order from.
- The class will have a single class (or static) variable:
  - **`MENU_ITEM_TYPES`**: a list containing 4 strings, representing the 4 possible types of menu items: "Drink", "Appetizer", "Entree", "Dessert".

- The class will use the following instance attribute (should be private):
  - **`self.__menuItemDictionary`**: a **`dictionary`** containing all the menu items from the menu. The **keys** are strings representing the types of the menu item, and the **values** are a **`list`** of **`MenuItem`** objects.
- Define the following methods:
  - **`__init__`**
    - Inputs (1): a string representing the name of the csv file that contains information about the menu items for the restaurant's menu
    - Return value: none
    - Initialize the single instance attribute to an empty **`dictionary`**.
    - Open and read the csv file and create a **`MenuItem`** object from each line in the file. Add the new object to the dictionary by using its type as the key. Note that each line in the file contains the 4 pieces of information needed to create a **`MenuItem`** object. Close the file object.
  - **`getMenuItem`**
    - Inputs (2): a string representing a type of menu item (will be one of the four types listed in **`MENU_ITEM_TYPES`**) and an integer representing the index position of a certain menu item
    - Return value: a **`MenuItem`** object from the dictionary
    - Get the correct **`MenuItem`** from the dictionary using its type and index position in the list of items.
  - **`printMenuItemsByType`**
    - Inputs (1): a string representing a type of menu item (will be one of the four types listed in **`MENU_ITEM_TYPES`**)
    - Return value: None
    - Print a header with the type of menu items, followed by a numbered list of all the menu items of that type.
  - **`getNumMenuItemsByType`**
    - Inputs (1): a string representing a type of menu item (will be one of the four types listed in **`MENU_ITEM_TYPES`**)
    - Return value: an integer representing the number of **`MenuItems`** of the input type
  - You do not need to define any getters and setters for this class.
- At this point, you should be able to test the methods in the **`Menu`** class.

Part 2 – Creating Diners

- Next, define the **`Diner`** class in a file titled **`Diner.py`**. This class represents one of the diners at the restaurant and keeps tracks of their status and meal.
- It will make use of **`MenuItem`** objects, so be sure to add the proper import statement.
- The class will have the following class (or static) variable:
  - **`STATUSES`**: a list of strings containing the possible statuses a diner might have: "seated", "ordering", "eating", "paying", "leaving"
- The class will use the following instance attributes (all should be private):
  - **`self.__name`**: a string representing the diner's name
  - **`self.__order`**: a list of the **`MenuItem`** objects ordered by the diner
  - **`self.__status`**: an integer corresponding the diner's current dining status
- Define the following methods:
  - **`__init__`**
    - Inputs (1): a string representing the diner's name
    - Return value: none
    - Set the diner's name attribute to the input value. Set the diner's order attribute to an empty list (the diner has not ordered any menu items yet), set the status attribute to 0 (corresponding to a seated status).
  - Define **getters and setters** for all the instance attributes.
  - **`updateStatus`**
    - Inputs: none
    - Return value: none
    - Increase the diner's status by 1.
  - **`printOrder`**
    - Inputs: none
    - Return value: none
    - Print a message containing all the menu items the diner ordered.
  - **`calculateMealCost`**
    - Inputs: none
    - Return value: a float representing the total cost of the diner's meal
    - Total up the cost of each of the menu items the diner ordered.
  - **`__str__`**
    - Inputs: none
    - Return value: a string
    - Construct a message containing the diner's name and status, formatted in a readable manner. Examples:

```
Diner Paul is currently seated.
Diner Faith is currently ordering.
```

- At this point, you should be able to test the methods associated with the **`Diner`** class.

Part 3 – Creating a Waiter

- Next, define the **`Waiter`** class in a file titled **`Waiter.py`**. This class will make use of **`Menu`** and **`Diner`** objects, so be sure to include the proper import statements.
- This class will represent the restaurant's waiter. The waiter maintains a list of the diners it is currently taking care of, and progresses them through the different stages of the restaurant. The waiter in the simulation will repeat multiple cycles of attending to the diners. In each cycle, the waiter will seat a new diner, if one arrives, take any diners' orders if needed, and give diners their bill, according to each diner's status.
- The class will use the following instance attributes (all should be private):
  - **`self.__diners`**: a list of **`Diner`** objects the waiter is attending to
  - **`self.__menu`**: a **`Menu`** object representing the restaurant's menu
- Also define the following methods:
  - **`__init__`**
    - Inputs (1): a **`Menu`** object
    - Return value: none
    - Assign the input parameter to the corresponding attribute. Initialize the list of diners to an empty list.
  - **`addDiner`**
    - Inputs (1): a **`Diner`** object
    - Return Value: none
    - Add the new **`Diner`** object to the waiter's list of diners.
  - **`getNumDiners`**
    - Inputs: none
    - Return Value: an integer representing the number of diners the waiter is currently keeping track of
  - **`printDinerStatuses`**
    - Inputs: none
    - Return Value: none
    - Print all the diners the waiter is keeping track of, grouped by their statuses.
    - Loop through each of the possible dining statuses a **`Diner`** might have (hint: use the Diner class attribute) to group the diners.
  - **`takeOrders`**
    - Inputs: none
    - Return Value: none
    - Loop through the list of diners and check if the diner's status is "ordering"
    - For each diner that is ordering, loop through the different menu types (hint: use the class attribute from the Menu class).

            - With each menu type, print the menu items of that type, then ask the diner to order a menu item by selecting a number (be sure to include error checking).
            - After the diner selected a menu item, add the item to the diner. Once the diner orders one menu item of each type, print the diner's order.
            - Each diner must order **exactly one** item of each type
    - **`ringUpDiners`**
        - Inputs: none
        - Return Value: none
        - Loop through the list of diners and check if the diner's status is "paying".
        - For each diner that is paying, calculate the diner's meal cost and print it out in a message to the diner.
    - **`removeDoneDiners`**
        - Inputs: none
        - Return Value: none
        - Loop through the list of diners and check if the diner's status is "leaving".
        - For each diner that is leaving, print a message thanking the diner.
        - To remove the "leaving" diners from the list of diners, you will need to loop over the list of diners backwards. This should be done in its own loop.
            - Looping through the list in reverse will allow you to iterate over items in the list while also deleting items from it.
            - Hint: use a range based for loop.
    - **`advanceDiners`**
        - Inputs: none
        - Return Value: none
        - This method allows the waiter to attend to the diners at their various stages as well as move each diner on to the next stage.
        - First, call the **`printDinerStatuses()`** method.
        - Then, in order, call the **`takeOrders()`**, **`ringUpDiners()`**, and **`removeDiners()`** methods.
        - Finally, update each diner's status.
    - You do not need to define any getters and setters for this class.
- At this point, if your classes are all correctly implemented, you should be able to run the program in its entirety.

Extra Credit

- Create a GUI using **tkinter** for the simulation. The design of the GUI can be as complex or simple as you decide.
    - Any user input should be handled in the GUI. All text outputs should also appear in the GUI, i.e. there should be no need for the user to interact with the console.
- Add an attribute called **`progress`** and an attribute called **`diningSpeed`** to the **`Diner`** class.
    - **`progress`** represents how far a **`Diner`** has progressed in each stage that they are in. **`diningSpeed`** is how quickly a diner progresses to each stage/status.
        - A diner's progress starts at a value equal to their dining speed and decreases over time. Once the progress reaches -1, you can reset the progress and advance the diner's status.
    - You will need to modify the constructor to take in an additional input (dining speed) to be assigned to this variable. Set the progress initially to the same value as the **`diningSpeed`**.
    - Add the appropriate getters and setters.
    - In **`RestaurantHelper.py`**, you will need to uncomment the version of **`randomDinerGenerator()`** that uses the Diner constructor with the diner's progress, and comment out the old version of that method.
    - Also add a Diner class attribute called **`DINING_SPEED`**, a list with the values: "hurried", "normal", and "slow".
    - Modify the string method to include the dining speed. Example:

`Diner Tod is dining at a hurried speed and is currently seated.`

    - The challenging step is to incorporate the logic for advancing the diners forward. Inside of **`advanceDiners`** (**`Waiter`** class), instead of updating the diner's statuses, you will need to decrement each diner's progress by 1, and then check for diner's who have a progress of -1 to advance their statuses and reset their progress counters.
    - In takeOrders and ringUpDiners, you will now need to check that the diner's status is correct and that their progress counters reached -1 before executing the necessary steps.
    - In removeDoneDiners, you do not need to check for progress, just remove the diners who are done per the original directions.

Sample Output

(separate file)

Deliverables

Project – code due on Blackboard 5/4/18 at 11:59 pm

- **Project that do not run will subject to an automatic 50% penalty.**
- **Late submissions will not be accepted.**

Submission Instructions

- Compress your Python project folder which contains your Python source code.
- You should have a zip file entitled **ITP115_project_*lastname_firstname*.zip** where you replace ***lastname*** with your last/family name and ***firstname*** with your first name.
- Under Assignments on Blackboard, upload the zip file.

Grading

Item	Points
Menu Item Class	10
Menu Class	25
Diner Class	15
Waiter Class	30
Project executes successfully and has error checking	10
Proper coding style and detailed comments	10
Total*	**100**

** Points will be deducted for poor code style, or improper submission.*