Homework 4: Farmers Market

For this assignment, you will be writing methods so that customers can successfully order and pay for a pickup order at the Ann Arbor Farmers Market. You will also be writing and fixing test cases, so you can guarantee that every step within the checkout i.e., placing the order and the order being processed is accurate, and your customers are happy!

Review the starter code thoroughly before beginning this assignment, as understanding how the classes interact with each other is important. Take notes or draw a diagram if necessary.

Overview

<u>**Customer** Class</u>

The ***Customer*** class represents a customer who will order from the vendors. Most of the class has been provided for you. You will write the ***place_order*** method.

Each *Customer* object has 3 instance variables:

- **name -** a string representing a customer's name
- **wallet -** a float showing how much money is in the customer's farmer's market account wallet. The default value is $10.
- **cust_id** – an integer representing a customer's account ID number.

The *Customer* class also includes several methods:

- ***__init__*** - which initializes the customer's attributes
- ***__str__*** - which returns the customer's name, wallet and customer ID as a string
- ***reload_wallet(self, amount)*** - which adds a passed amount to the wallet
- ***place_order(self, vendor, order)*** - which takes a vendor object and an 'order', which is a dictionary where the keys are Produce objects and the values are another dictionary with keys of *quantity* and *fresh_pick*. It returns a boolean value that indicates if the order was successfully placed or not.

<u>Produce Class</u>

The ***Produce*** class represents a produce item at a vendor. The code for this class has been provided for you.

A *Produce* object has 2 instance variables:

- **name** - a string representing the name of produce
- **cost** - a float representing how much a *regular pick* of a produce item costs per pound versus a *fresh pick*, which costs 1.50 more per pound

The *Produce* class includes 2 methods:

- ***__init__*** - which initializes the produce's name and cost

__str__ **-** which returns the name and cost as a string

Vendor Class

The ***Vendor*** class represents a vendor.. You will write several methods for this class. See the Tasks to Complete for which methods.

Each *Vendor* object has 3 instance variables:

name - a string which is the name of the vendor

earnings - a float representing the earnings the vendor currently has

inventory - a dictionary which holds the produce objects as the keys and the quantities (in pounds) in stock of each produce as the values

The *Vendor* class also includes several methods:

__init__ - which initializes the attributes

__str__ - which returns a string with the vendor's name and the current menu

receive_payment(self, amount) - which takes an amount and adds it to the vendor's earning

calculate_cost(self, produce, quantity, fresh_pick, customer) - takes the quantity, produce object, whether a fresh_pick has been requested, and returns the total cost

stock_up(self, produce, quantity) **-** which adds the quantity of produce to the inventory

process_order(self, order) **-** which takes a dictionary that has produce objects as the key and quantity as a value and returns a boolean value

Tasks to Complete

- **Complete the *Customer* Class**
    - Complete the ***place_order(self, vendor, order)*** method
        - Call the ***calculate_cost*** method on the vendor object to calculate the total cost of the order
        - Check if the customer has the total cost or more in their wallet. If they don't have enough money, print **"Insufficient funds"** and return **False**
        - Call the ***process_order*** method on the vendor object. If ***process_order*** returns **True**, remove the total cost from the customer's wallet and call the ***receive_payment*** method to add it to the vendor's earnings and return **True**. Otherwise, return **False**.

- **Complete the Vendor Class**

    - A ***calculate_cost(self, produce, quantity, fresh_pick, customer)*** method that takes the produce object, quantity and fresh_pick: a Boolean variable that specifies if the customer has requested a fresh pick, and returns the total cost.
    Note: A fresh picks of any produce costs **$1.50 more** per pound than the regular picks

    - A ***stock_up(self, produce, quantity)*** method that takes the produce object and the quantity. It adds the quantity to the existing quantity if the item exists in the inventory dictionary or creates a new item in the inventory dictionary with the item name as the key and the quantity as the value.

    - A ***process_order(self, order)*** method that takes a dictionary that represents the order and checks that there is enough produce in the inventory for the order and if not returns **False**. Otherwise, it subtracts the quantity of the produce ordered from the quantity in the inventory and returns **True**.
    - An order should only be fulfilled if everything that is requested for it is present in the inventory.

- **Write and Fix Test Cases**

  Note: Many test cases have already been written for you. **Please do not edit any test cases other than the ones that have comments above them explicitly asking you to do so.** These test case-related tasks are also explained below:

  - Write test cases for the following scenarios for the ***place_order*** method in ***test_customer_place_order***:
    - The customer doesn't have enough money in their wallet to place the order
    - The vendor doesn't have enough produce in stock
    - The vendor doesn't sell the produce item mentioned in the order

  - Fix the test cases in ***test_customer_place_order_2***

  As you are working on one test case, feel free to comment out the test cases that you are not working on, but **be sure to uncomment all test cases before you turn in your homework.**

Grading Rubric (60 points)

Note that if you use hardcoding (specify expected values directly) in any of the methods by way of editing to get them to pass the test cases, or you edit any test cases other than the ones you have been directed to, you will NOT receive credit for those related portions.

- 10 points for correctly completing the ***Customer*** class (5 for ***place_order***, 5 for the rest)
- 30 points for correctly completing the ***Vendor*** class (10 points per method)
- 15 points for completing ***test_customer_place_order*** (5 points per scenario)
- 5 points for fixing ***test_customer_place_order_2***

Extra Credit (6 points)

To gain extra credit on this assignment, please check for the following condition in *calculate_cost()*

Every customer with a customer ID that is a multiple of 100 (eg: 100, 200 etc) gets a 15% discount on each produce item.