Homework 3: Coffee Coupon Dispenser

You will create a small program that assigns one random coffee coupon to each unique customer.

Overview

In this homework, you will write a small **coupon dispenser** program for a campus coffee shop.

The program will loop asking you to enter a customer name (or names), show, or exit.

- If you type **`exit`** (exact spelling) the program will print `"Goodbye!"` and stop.
- If you enter one or more customer **names** (separated by commas).
    - **A coupon will be picked at random from a list of coupons for each name if that name doesn't already have an assigned coupon.**
- If you type **`show`** (exact spelling) it will display a string with each customer's name and coupon.

Important: You must store and manage your data using **only lists** (no dictionaries).

One example run (Informal):

1. At the start of the program, a **CouponDispenser** object is initialized with a predefined list of coupons which are defined in the starter code. The coupons will be stored in the object's **coupon_cards** list (you can change these if you wish in the starter code).
   For example, the predefined list of coupons is as follows:

```
[
        "10% off",
        "Free small coffee",
        "Buy 1 get 1 half off",
        "Free extra espresso shot",
 ]
```

2. The user enters customers' names: `Ava, Ben`
3. Ben and Ava are each assigned one coupon at random from the **coupon_cards** list if they don't already have a coupon. It is possible that they are assigned the same type of coupon.
4. The user types **`show`** to display the assignments. This shows the coupon for each name on a single line (Hint: print the newline character \n to force a new line).

```
Ava: 10% off
```

```
Ben: Buy 1 get 1 half off
```

5. The user can repeat steps 2-4 above with other names.
6. The user types **`exit`**, the program prints the line below and ends:

```
"Goodbye!"
```

Note: You can call **test()** on the last line in the **main** function to test each function.

Requirements

In this assignment, you will implement a **`CouponDispenser`** class with the following methods:

1. **`__init__(self, coupon_cards)`**

Initialize a new **`CouponDispenser`** object.

- **Set `coupon_cards`** to the **`coupon_cards`** parameter(a list of strings).

This list contains **all available coupons**.

- **Set `customer_roster`** to an empty list.

This list will store **customer names**, in the order they received a coupon.

- **Set `issued_indices`** to an empty list.

This list will store the **index** of the coupon given to each customer.
The index at position **`i`** in **`issued_indices`** matches the customer at position **`i`** in **`customer_roster`**.

Example: **`customer_roster[0] == "Amy"`** and **`issued_indices[0] == 2`**, then Amy was assigned **`coupon_cards[2]`**.

2. **`__str__(self)`**

Returns the string representation of **`CouponDispenser`**.

- Return a single string with all coupons in **`coupon_cards`** joined by pipes (|).

For example:

```
box_multi = CouponDispenser(["10% off", "Free small coffee"])
```

Then

```
print(box_multi)
```

The terminal displays:

`"10% off|Free small coffee"`

- If **`coupon_cards`** is empty, return an empty string ""

3. **`issue_coupon(self, name)`**

If **`name`** is already assigned a coupon, return it. Otherwise assign a random coupon to the name.

- If the list **`coupon_cards`** is empty, return:
  - `"The box is empty."`
- If **`name`** already exists in **`customer_roster`**:
  - Find the existing coupon assigned to **`name`**. (**Hint**: Find the index where name exists in **`customer_roster`** and use that to find the corresponding coupon in **`issued_indices`**)
  - Return: "That name already has a coupon: <coupon>" (where <coupon> is the existing coupon assigned to **`name`**)
  - Do NOT change any lists.
- If **`name`** is new:
  - Pick a random coupon index from all coupons (repeats allowed):
- Append the **`name`** to **`customer_roster`** and the randomly chosen index to **`issued_indices`**, and return the assigned coupon text.

Important: Do not use dictionaries in this method.

4. **`distribute_session(self)`**

Run the interactive “coupon dispenser” session.

Loop behavior

- Maintain a loop counter in a variable called **`round_number`** - This should be 1 on the first iteration of the loop.
- In each iteration of the loop, prompt the user for input **exactly** like this (including punctuation and spacing):

`"Round <round_number> - Enter a name (or a comma-separated list), or type 'show' or 'exit': "`

Handle the user input

Use **`user_input`** `to` store the value the user entered (don’t change its capitalization).

1. **If `user_input` is exactly `"exit"`**
    - Print: **`"Goodbye!"`**
    - Stop the loop (end the session).
2. **If `user_input` is exactly `"show"`**
    - Print all current coupon assignments in the same order as **`customer_roster`**, one per line:
        - **`"<name>: <coupon>"`**
3. **Otherwise: treat the input as names**
    - Split the line by commas.
        - Hint: Use pieces = text.split(“,”) to generate a list of text splitted by commas.
    - For each piece:
        - Strip leading/trailing whitespace.
        - Ignore it if it becomes an empty string.
        - Hint: Use stripped_text = text.strip() to strip leading/trailing whitespace.
    - For each remaining name:
        - Call **`issue_coupon(name)`**
        - Print the returned string.

Reminder: Use **lists only** (no dictionaries).

5. **`tally_distribution(self) (Extra Credit)`**

Print the number of times each coupon was distributed (using **ONLY lists**).

Steps

1. **If no coupons have been issued**
    - Print exactly: **`"Empty"`**
    - Return immediately (no value).
3. **Otherwise**
    - For each coupon in **`coupon_cards`**, print a line in the following format:
      `"<coupon> distribution count: <count>."`
4. **Return**

tally_distribution example output:

Assume **box** is a coupon dispenser object

The **`coupon_cards, customer_roster,`** `and issued_indices are`

```
box.coupon_cards = [
```

```
            "10% off",
            "Free small coffee",
            "Buy 1 get 1 half off",
            "Free extra espresso shot",
        ]
      box.customer_roster = ["Amy", "Ben", "Candy", "Dan"]
      box.issued_indices = [0, 2, 3, 2]

The use run:
      box.tally_distribution()
```

In the terminal:

10% off distribution count: 1.

Free small coffee distribution count: 0.

Buy 1 get 1 half off distribution count: 2.

Free extra espresso shot distribution count: 1.

6. main()

- Define a list of strings called **coupon_cards.**

```
[
            "10% off",
            "Free small coffee",
            "Buy 1 get 1 half off",
            "Free extra espresso shot",
        ]
```

- Create a **CouponDispenser** object that passes the list **coupon_cards**, and calls **distribute_session()**, and **tally_distribution()**.

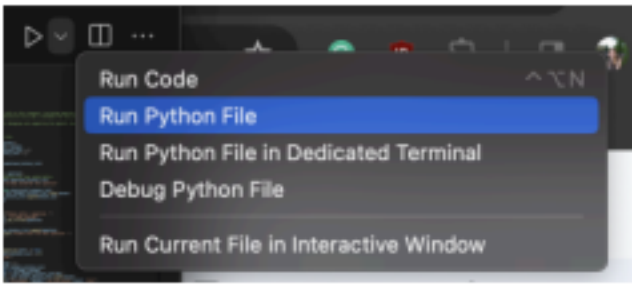
Grading Rubric (60 points + 6 points bonus)

Points	Item
6 pts	__init__ correctly initializes coupon_cards, customer_roster, and issued_indices

6 pts	3 pts: __str__: joins non-empty coupon_cards with pipes; 3 pts: returns "" for empty
6 pts	issue_coupon returns “The box is empty.” when coupon_cards is empty.
6 pts	issue_coupon correctly re-reports an already assigned name (no state change)
6 pts	issue_coupon randomly selects from all coupon indices (repeats allowed) and appends chosen index for new names
6 pts	issue_coupon appends new names to customer_roster (kept aligned with issued_indices)
3 pts	distribute_session starts with the specified prompt format
6 pts	distribute_session correctly implements comma-separated names
3 pts	distribute_session correctly implements the exit command
6 pts	distribute_session correctly implements the show command
2 pts	coupon_cards list is properly defined and used in main()
2 pts	CouponDispenser is properly constructed and used in main()
2 pts	distribute_session is called correctly in main()
Extra Credit	
2 pts	tally_distribution prints coupon distribution counts correctly.
2 pts	tally_distribution prints the distribution count in format "{coupon} distribution count: {count}. "
2 pts	tally_distribution is called and its output is displayed in main()

Running and Submission

Run your code locally and make sure the provided tests pass. If you are using VS Code, you can use “Run Python File”.



Follow the instructions on Canvas to submit your GitHub repository link by Friday, January 30 at 11:59 pm.