Read all instructions carefully. Not following instructions will result in you not earning the credit you want for the assignment.

Objectives

In Part 4 you will learn the following:

- Include a terminal-based user interface in your application
- do input validation

Structure

- Module name: desserts
  - Classes
    - Order
    - Candy
    - Cookie
    - IceCream
    - Sundae
- Module name: dessertshop
  - Functions
    - main
  - Classes
    - DessertShop

Problem

In this part of the project, you add a command line user interface to your program. To do this, you will make changes to `dessertshop.py` module.

There should be no changes to the DessertItem class and the subclasses.

Changes to dessertshop module

Add a new class **DessertShop** to the module. The DessertShop class will have 4 new methods:

- user_prompt_candy()
- user_prompt_cookie()
- user_prompt_icecream()
- user_prompt_sundae()

Each method should do the following:

1. prompt user for enough input to create the required line item on the receipt
2. get user input
3. validate the input
4. convert input values to the proper types as-needed
5. create an object of the proper type
6. return the newly created object to the caller

main() Code to Copy and Use

We give you code that runs the main loop in the provided file `ui.py`. You should be able to cut and paste it into your own main function in your `dessertshop.py` as-is.

receipt module

Upload a copy of your `receipt.py` file from Part 3 into Part 4. If you implemented it correctly in Part 3, no changes should be needed for Part 4.

Test Cases

You do not need to create any new pytest test cases for Part 4. **Do** re-run your existing test cases to make sure they still pass.

Key Program Requirements

1. A terminal-based user interface has been added to your Dessert Shop program.
2. Output should look similar to the sample run.
3. Submit the following 7 files in your workspace:

- desserts.py
- dessertshop.py
- test_dessert.py
- test_candy.py
- test_cookie.py

- test_icecream.py
- test_sundae.py

Only `dessertshop.py` should have changed in Part 4 unless you needed to correct other issues.

Sample Run

```
1: Candy
2: Cookie
3: Ice Cream
4: Sundae
What would you like to add to the order? (1-4, Enter for done): 2

Enter the type of cookie: Chocolate Chip Macadamia
Enter the quantity purchased: 5
Enter the price per dozen: 4.99
1: Candy
2: Cookie
3: Ice Cream
4: Sundae
What would you like to add to the order? (1-4, Enter for done): 3

Enter the type of ice cream: Mint Chocolate Chip
Enter the number of scoops: 4
Enter the price per scoop: .89
1: Candy
2: Cookie
3: Ice Cream
4: Sundae

What would you like to add to the order? (1-4, Enter for done):4
Enter the type of ice cream: Vanilla
Enter the number of scoops: 3
Enter the price per scoop: .89
Enter the topping: Sprinkles
Enter the price for the topping: .5
1: Candy
2: Cookie
3: Ice Cream
4: Sundae
What would you like to add to the order? (1-4, Enter for done):
```

Example Receipt

The receipt shown does not exactly match the scenario above, but shows you what would be printed on the PDF receipt if these are the items you ordered. The actual receipt should be in file `receipt.pdf`.

Name	**Cost**	**Tax**
Chocolate Chip Macadamia	$2.08	$0.15
Mint Chocolate Chip	$3.56	$0.26
Sprinkles Vanilla Sunday	$3.17	$0.14
Candy Corn	$0.38	$0.03
Gummy Bears	$0.09	$0.01
Chocolate Chip	$2.00	$0.14
Pistachio	$1.58	$0.11
Vanilla	$3.36	$0.24
Oatmeal Raisin	$0.58	$0.04

Order Subtotals	$16.77	$1.21
Order Total		$18.01
Total items in the order		9

Correctness

From your terminal, run `ruff check` on each of the 7 files above, such as:

```
ruff check dessert.py
ruff check dessertshop.py
ruff check test_dessert.py
...
```

This will check for syntax errors, violations and many issues that could lead to bugs in your code. Code will be maually graded, so any score received are partial.

Style

From your terminal, run `ruff format` on each of the 7 files above to check the format of your code, like:

```
ruff format dessert.py
ruff format dessertshop.py
ruff format test_dessert.py
```

How to Submit

From your Github assignment repository page, click Submit and enter a nontrivial commit message.

Grading

This project is manually graded. Use the following rubric.

Criteria	Mastery (100)	Developing (75)	Beginning (50)
Implementation of DessertShop class and its methods	All the methods (user_prompt_candy, user_prompt_cookie, user_prompt_icecream, user_prompt_sundae) have been implemented correctly in the DessertShop class according to the problem requirements.	Most of the methods have been implemented correctly, but there may be one or two methods missing or incorrect.	Few or none of the methods in the DessertShop class have been implemented correctly.
User input validation	All user inputs are validated properly for every method in DessertShop class.	Some of the user inputs are validated properly, but there are issues with others.	Few or none of the user inputs are validated properly.
Object creation based on user input	For each method in the DessertShop class, objects are correctly created based on user input and returned to the caller.	For most methods, objects are correctly created based on user input, but there may be one or two methods where this is not the case.	Few or none of the methods correctly create and return objects based on user input.
Program output	Program output matches exactly with the sample output provided in the problem.	Program output mostly matches the sample output, with minor discrepancies.	Program output does not match the sample output or is missing entirely.
Existing test cases still pass	All existing test cases pass without any modifications required.	Some existing test cases pass without modifications, but others fail or require modifications.	Most or all existing test cases fail or require modifications.

Students should strive for mastery level. Lower than that indicates areas where more practice is needed or more learning is needed. Code score is the average of the individual feature scores.

Total score is 1/4 * style score + 3/4 * code_score.