Dessert Shop Part 2 -- CS 1420 Version

Read instructions carefully. Not following instructions will result in you not receiving the credit you want.

Big Note: The starting point for Part 2 is the files you submitted for Part 1. Start with a copy of `dessert.py` and `test_dessert.py` from Part 1.

Objectives

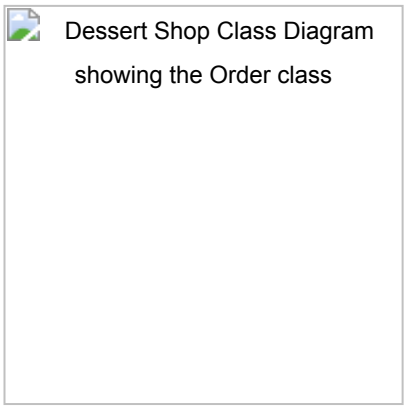
In Part 2 you will learn the following:

- Use existing classes to build an application
- Add console output to manually test your application
- Create manual test cases to test your application by inspection

Problem To solve

A Dessert Shop sells candy by the pound, cookies by the dozen, ice cream by the scoop, and sundaes (ice cream with a topping).

In Part 2, implement the ability to create an order consisting of items available at the Dessert Shop to what you already did in Part 1. These items include Candy, Cookies, Ice Cream and Sundaes. You will also implement a main function with a simple console user interface to interactively test the order entry system.



The class diagram above adds a design for an `Order` in the system and the `main()` method that is the application driver. The classes show attributes with associated data types. `main()` is in the `dessertshop` module. UML (Unified Modeling Language) is the defacto industry standard modeling tool for object-oriented programming worldwide.

You will submit 3 code files in your workspace:

- `dessert.py`
- `test_dessert.py`
- `dessertshop.py`

`test_dessert.py` is carried forward from Part 1, but not modified. `dessertshop.py` is where your main function will be.

An example run that matches the code you are given for main() is: `Candy Corn Gummy Bears Chocolate Chip Pistachio Vanilla Oatmeal Raisin Total number of items in order: 6`

Order Class

The order class is a list-like container for DessertItem objects.

The **Order** class has the following attribute and methods:

- order: list of DessertItem objects
- A constructor that creates an empty Order
- add() method that takes a single DessertItem argument and adds it to the order list
- `__len__()` magic method that returns the number of items in the order

You may add others if convenient but these `check50` will expect to be implemented.

main()

`main` should do the following:

- Create a new instance of the order class
- Add the following items to the order: `Candy("Candy Corn", 1.5, .25) Candy("Gummy Bears", .25, .35) Cookie("Chocolate Chip", 6, 3.99) IceCream("Pistachio", 2, .79) Sundae("Vanilla", 3, .69, "Hot Fudge", 1.29) Cookie("Oatmeal Raisin", 2, 3.45)`
- Print out just the name of each DessertItem in the order
- Print out the total number of items in the order

How to Test

- Run your program and verify that the order is printed to the console as indicated.

You don't need to add any new test cases to `test_dessert.py` at this point, but you can if you want to.

Correctness

From your terminal, run these commands:

```
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.

Style

From your terminal, run these commands to check the format of your code:

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

How to Submit

From your Github assignment repository page, click Commit and enter a commit message. You may want to do seprate commits on each file so that you know what changed in each, but you decide that.

Grading

Criteria	Mastery (100 points)	Proficient (85 points)	Developing (70 points)	Beginning (60 points)	Not Demonstrated (50 points)
Order Class Structure	The Order class is correctly implemented with an attribute that is a list of DessertItem objects. The constructor correctly creates an empty Order.	Order class mostly meets requirements but has minor issues in constructor or attributes.	Order class partially meets requirements but has significant issues in constructor or attributes.	Order class has been attempted, but it is fundamentally flawed.	Order class does not exist.
Order Class Methods	The Order class correctly implements the add() method, `__len__()` magic method, `__iter__()` magic method, and `__next__()` magic method. All methods work correctly.	Most of the required methods in the Order class are implemented correctly but there are minor issues.	Some of the required methods in the Order class are implemented correctly but there are major issues.	The required methods in the Order class have been attempted, but they are fundamentally flawed.	Required methods in the Order class are not implemented.

Criteria	Mastery (100 points)	Proficient (85 points)	Developing (70 points)	Beginning (60 points)	Not Demonstrated (50 points)
Main Function	The main function correctly creates an instance of the Order class, adds the required items to the order, and correctly prints out the name of each DessertItem in the order and the total number of items.	The main function does most of the required tasks correctly but has minor issues.	The main function does some of the required tasks correctly but has major issues.	The main function has been attempted but it is fundamentally flawed.	The main function is not implemented.
Program Output	The program's output exactly matches the example output given in the problem statement.	The program's output mostly matches the example output, with minor discrepancies.	The program's output somewhat matches the example output, but there are major discrepancies.	The program's output has been attempted but it significantly deviates from the example output.	The program does not produce output.
Code Quality	Code is clearly written, logically organized, and easy to follow. No syntax errors.	Code is mostly clear and logically organized, with minor syntax errors that do not affect program execution.	Code is somewhat clear and organized, but with syntax errors that may affect program execution.	Code is unclear, disorganized, and contains numerous syntax errors that prevent program from executing correctly.	No code or code cannot be evaluated due to major syntax errors.
Passes ruff checks for syntax errors, violations, and potential bugs	Code passes all `ruff` checks for syntax errors, violations, and potential bugs.	Code passes most `ruff` checks for syntax errors, violations, and potential bugs.	Code passes some `ruff` checks for syntax errors, violations, and potential bugs.	Code passes few `ruff` checks for syntax errors, violations, and potential bugs.	Code does not pass `ruff` checks for syntax errors, violations, and potential bugs.

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.