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

Objectives

In Part 6 you will learn the following:

- Add an inherited property defined by a Protocol class (Interface in other languages)

Structure

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

Problem

Suppose you want to add functionality to your Dessert Shop application to identify what type of packaging a dessert item should be placed in. These packaging types will show up on the receipt. Each item will show what type of packaging it will be place in. All items of a similar type will use the same type of packaging.

Because a property like packaging could be used in multiple applications, we put it in its own module and define an explicit type.

To do this, you will add an interface to your Dessert Shop application and update your receipt output as described below.

Packaging Protocol Class

Define a new Protocol class `Packaging` that has one attribute `packaging` of type `str`. It should be in file `packaging.py`.

In other programming languages like Java, we would define a getter and setter in this interface. But in Python, we'll just define an attribute and use it directly.

Changes to DessertItem class

Change the DessertItem class to inherit from the new `Packaging` interface. Do not change the constructor signature, because you can create a dessert and add packaging later. Default value is `None`.

Changes to the Concrete Subclasses of DessertItem

- set the superclass packaging type within the constructor; the default value for each respective type is as follows:
  - for Candy: packaging = "Bag"
  - for Cookie: packaging = "Box"
  - for Ice Cream: packaging = "Bowl"
  - for Sundae: packaging = "Boat"
- modify the `__str__` magic method to include the packaging type as shown in the sample run

Test Cases

Add automated test cases to test correct packaging for each concrete dessert type in their respective pytest test files.

Key Program Requirements

The new Protocol class Packaging has been added to your system.

Test code has been updated to check for the packaging type.

The receipt has been updated to show packaging.

Example Run

1: Candy

2: Cookie

3: Ice Cream

4: Sunday

What would you like to add to the order? (1-4, Enter for done): 2

Enter the type of cookie: Oatmeal Raisin

Enter the quantity purchased: 4

Enter the price per dozen: 3.45

1: Candy

2: Cookie

3: Ice Cream

4: Sunday

What would you like to add to the order? (1-4, Enter for done):

```
-----------------------------------------------Receipt----------------------------
Oatmeal Raisin Cookies (Box)
      4 cookies @ $3.45/dozen:                        $1.15           [Tax: $0.08]
Candy Corn (Bag)
      1.50 lbs. @ $0.25/lb.:                          $0.38           [Tax: $0.03]
Gummy Bears (Bag)
      0.25 lbs. @ $0.35/lb.:                          $0.09           [Tax: $0.01]
Chocolate Chip Cookies (Box)
      6 cookies @ $3.99/dozen:                        $2.00           [Tax: $0.14]
Pistachio Ice Cream (Bowl)
      2 scoops @ $0.79/scoop:                         $1.58           [Tax: $0.11]
Vanilla Sundae (Boat)
      3 scoops of Vanilla ice cream @ $0.69/scoop
      Hot Fudge topping @ $1.29:                      $3.36           [Tax: $0.24]
Oatmeal Raisin Cookies (Box)
      2 cookies @ $3.45/dozen:                        $0.58           [Tax: $0.04]
-------------------------------------------------------------------------------
Total number of items in order: 7
Order Subtotals:                                      $9.12           [Tax: $0.66]
Order Total:                                                                $9.78
-------------------------------------------------------------------------------
```

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 (85)	Beginning (70)	Low (50)
Implementation of Packaging Protocol Class	The Packaging Protocol class is correctly implemented with the packaging attribute of type str.	The Packaging Protocol class is implemented but with minor errors in the packaging attribute.	The Packaging Protocol class is partially implemented with significant errors in the packaging attribute.	The Packaging Protocol class is either not implemented or incorrectly implemented.
Changes to DessertItem class	The DessertItem class correctly inherits from the Packaging Protocol class with a default value of None for packaging.	The DessertItem class inherits from the Packaging Protocol class, but there are minor errors in handling default value or inheritance.	The DessertItem class attempts to inherit from the Packaging Protocol class, but with significant errors or omissions.	The DessertItem class does not inherit from the Packaging Protocol class or has serious implementation errors.
Changes to DessertItem Subclasses	All subclasses correctly set the superclass packaging type within the constructor and modify the `__str__` method appropriately.	Most subclasses correctly set the superclass packaging type within the constructor and modify the `__str__` method, but there are minor errors.	Some subclasses set the superclass packaging type within the constructor and attempt to modify the `__str__` method, but with significant errors.	Few or none of the subclasses correctly implement the required changes.

Criteria	Mastery (100)	Developing (85)	Beginning (70)	Low (50)
Test Cases	All required test cases are implemented correctly and pass.	Most required test cases are implemented and pass, but there may be a few missing or failing tests.	Some required test cases are implemented and pass, but there are significant gaps in test coverage.	Few or no required test cases are implemented, or most tests fail.
Receipt Output	The receipt correctly includes packaging information for each item.	The receipt includes packaging information for most items, but there may be a few errors or omissions.	The receipt includes packaging information for some items, but there are significant errors or omissions.	The receipt does not include correct packaging information for the items or has serious errors.
Use of Packaging Protocol Class	Packaging Protocol class is implemented and used correctly across the system.	Packaging Protocol class is mostly implemented and used correctly, but there are minor errors or omissions.	Packaging Protocol class is used in some parts of the system, but there are significant errors or omissions.	Packaging Protocol class is either not used or incorrectly used across the system.

Students should aim for the Mastery level in all categories to ensure they have fully understood and implemented the concepts covered in the project. Overall score is the average of the individual criteria scores.