Dessert Shop Part 1

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

Objectives

- Make Python classes to represent items in the real world
- Build several subclasses
- Write and run automated test cases for constructors, attributes and methods

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).

For this part of the project, you will create the structure for a Dessert Shop program. There will be no user interface nor input/output at this time, but there will be later. Your code will be tested using automated test cases.

Test code can be simple asserts, unittest code, pytest code, or doctests as long as your set of test cases meets the requirements specified for the project. Whichever you choose, be consistent. `unittest` is built-in with Python, as are doctests and asserts. `pytest` is a 3rd-party module but is simpler to write code for than unittest, and it will run unittest code. Whichever you choose, be consistent.

To create this framework, you will implement an inheritance hierarchy of classes derived from a DessertItem superclass. Candy, Cookie, and IceCream classes will derive from the DessertItem class. The Sundae class will derive from the IceCream class. Implement the classes in a file called `dessert.py`.

The classes will be structured as below.

DessertItem Superclass

The DessertItem superclass contains:

- name: str
- a constructor with one parameter that sets the name attribute to the passed-in value.

Derived Subclasses Candy, Cookie, IceCream and Sundae

All derived subclasses (Candy, Cookie, IceCream and Sundae) contain:

- Attributes as described below with reasonable default values
- A constructor with enough parameters to initialize all the attributes of the object, including the superclass attribute. *The constructor must call the superclass constructor.*

The **Candy** attributes are:

- candy_weight: float

- price_per_pound: float

The **Cookie** attributes are:

- cookie_quantity: int
- price_per_dozen: float

The **IceCream** attributes are:

- scoop_count: int
- price_per_scoop: float

The **Sundae** attributes are:

- topping_name
- topping_price

Shown here is a UML class diagram where attributes have names in snake case and declared types. Specifying types is good in design, even if the language does not require us to explicitly declare the types of variables in our code.

In a file called `dessert.py` implement the classes as described. In a file called `test_dessert.py` write test code that tests your classes.

How to Test

You must write test cases to test your code too. Use the built-in `unittest` module or the third-party module `pytest`. I recommend pytest. To install `pytest` in your programming environment execute the following from a terminal with administrative rights:

```
pip3 install pytest
```

Your test code should test the following 3 cases for each class:

1. call the default constructor
2. call the constructor with nominal values
3. modify the attributes after construction

Do not test DessertItem class directly; test it through the subclasses only. This will result in 12 unique but very similar test cases total.

Correctness

From your terminal, run these commands: `ruff check dessert.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 test_dessert.py
```

How to Submit

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

Grading

Criteria	Mastery (100 points)	Proficient (85 points)	Developing (70 points)	Beginning (60 points)	Not Demonstrated (50 points)
DessertItem Superclass	DessertItem superclass correctly contains 'name' attribute, and constructor correctly sets 'name' with a passed-in value.	DessertItem superclass is mostly correct, but has minor issues.	DessertItem superclass is present but has major issues with attributes or constructor.	DessertItem superclass has been attempted, but it is fundamentally flawed.	DessertItem superclass does not exist.
Candy Class	Candy class correctly inherits from DessertItem, contains required attributes (candy_weight, price_per_pound), and its constructor initializes all attributes correctly.	Candy class mostly meets requirements but has minor issues with attributes or constructor.	Candy class partially meets requirements but has significant issues with attributes or constructor.	Candy class has been attempted, but it is fundamentally flawed.	Candy class does not exist.
Cookie Class	Cookie class correctly inherits from DessertItem, contains required attributes (cookie_quantity, price_per_dozen), and its constructor initializes all attributes correctly.	Cookie class mostly meets requirements but has minor issues with attributes or constructor.	Cookie class partially meets requirements but has significant issues with attributes or constructor.	Cookie class has been attempted, but it is fundamentally flawed.	Cookie class does not exist.

Criteria	Mastery (100 points)	Proficient (85 points)	Developing (70 points)	Beginning (60 points)	Not Demonstrated (50 points)
Ice Cream Class	Ice Cream class correctly inherits from DessertItem, contains required attributes (scoop_count, price_per_scoop), and its constructor initializes all attributes correctly.	Ice Cream class mostly meets requirements but has minor issues with attributes or constructor.	Ice Cream class partially meets requirements but has significant issues with attributes or constructor.	Ice Cream class has been attempted, but it is fundamentally flawed.	Ice Cream class does not exist.
Sundae Class	Sundae class correctly inherits from Ice Cream, contains required attributes (topping_name, topping_price), and its constructor initializes all attributes correctly.	Sundae class mostly meets requirements but has minor issues with attributes or constructor.	Sundae class partially meets requirements but has significant issues with attributes or constructor.	Sundae class has been attempted, but it is fundamentally flawed.	Sundae class does not exist.
Implementation of Inheritance	All classes correctly and logically inherit from the appropriate superclasses. Superclass constructors are correctly called in derived class constructors.	Most classes correctly and logically inherit from the appropriate superclasses, but there are minor issues.	Some classes correctly and logically inherit from the appropriate superclasses, but there are significant issues.	Attempt at implementing inheritance is made, but with numerous errors and misunderstanding.	No use or incorrect use of inheritance.
Test Cases	All test cases are correctly written and successfully test all functionalities of the classes. All required tests for each class are present.	Most test cases are correctly written and successfully test most functionalities. A few required tests may be missing.	Some test cases are correctly written and test some functionalities, but many required tests are missing.	Test cases are present, but they are fundamentally flawed and do not test class functionalities correctly.	No test cases are provided.

Criteria	Mastery (100 points)	Proficient (85 points)	Developing (70 points)	Beginning (60 points)	Not Demonstrated (50 points)
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	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 fails many `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.