OOP - Practice 6

This practice does not need to be submitted. The goal is to provide extra hands-on practice with Python object-oriented syntax and logic.

Code exercise 1:

Define and use a class

In our bakery, we track the items we sell using Python objects.

Each of our cakes has three attributes that we need to store: what `kind` of cake it is, the `price`, and how many `slices` a full cake contains.

We also need each of our cake instances to return a string describing themselves in the format:
`The lemon cake costs $24 and is divided into 6 slices.`

Your tasks:

- Create a class called Cake with attributes `kind`, `price`, and `slices`.
- Create a method in the class called `describe()` that returns a string formatted like the example above.
- Create two instances (`spice_cake`, `chocolate_cake`) from this class using the following information:
    - kind: **spice**, price: **18**, slices: **8**
    - kind: **chocolate**, price: **24**, slices: **6**

`kind`: A string describing the variety of cake
`price`: An integer describing the cost of a whole cake
`slices`: An integer describing how many slices a full cake is divided into
`describe()`: A method that returns a formatted string describing the cake instance

Code exercise 2:

Modify a class definition

When we sell slices of cake, or eat a slice ourselves, we need to track how many slices remain. Let's expand our Cake class with a new function called `sell()` that takes as an argument the `count` of slices sold in a transaction.

It's important to put some basic reasonableness checks on this function; we can't, for example, sell 0 slices or sell a negative number of slices. And we can't sell more slices than we have remaining. If either of these conditions are true, we should return an appropriate error and avoid changing the count of slices remaining. Otherwise, we should change the count of remaining slices accordingly, and return a message showing how many slices remain.

Use these strings as the messages you return (replacing *n* with the correct value):
`Cannot sell zero or negative slices!`
`Cannot sell more slices than we have (n)!`
`This cake has n slices remaining.`

Your tasks:

- Add an attribute to keep track of the number of slices remaining in a cake.
- Add the method `sell(count)` to the Cake class to handle updating the remaining slice count.
- Ensure that the new method has some basic reasonableness checks.

`count`: An integer

Result

When called, the function will return one of three strings, as shown above. The cake instance's count of remaining slices will be decremented as slices are sold, as long as some basic requirements are met.

Constraints

- Do not allow the sale of zero slices of cake.
- Do not allow the sale of a negative number of slices of cake.
- Do not allow the sale of more slices than remain in a cake.

Example 1:

Input: `spice_cake.sell(12)`
Result: `Cannot sell more slices than are remaining (6)!`

Example 2:

Input: `spice_cake.sell(-2)`
Result: `Cannot sell zero or negative slices!`

Code exercise 3:

Compare instances

Our cakes don't all have the same price per slice. The 8-slice spice cake is $18, making it $2.25 per slice, and the 6-slice chocolate cake is $24, making it $4 per slice.

As we sell slices throughout the day, it would be nice to know which of our cakes has more value remaining in it.

We'll calculate the remaining value of a cake by finding the cost of a slice of that cake, and multiplying that by the number of slices remaining:

$$\text{value} = \left(\frac{\text{price}}{\text{slices}}\right) \times \text{remaining}$$

With this number for each cake, we're able to compare the value of remaining cake slices. But, we need to modify our `Cake` class with methods that allow us to choose what values are used to compare two instances of the class.

Your task: Modify the Cake class to include methods that compare instances of Cake. The methods are:

- isEqualTo(self, otherCake)
- isLessThan(self, otherCake)
- isGreaterThan(self, otherCake)

The above methods will return a boolean value (True or False) based on the value calculation described above.

Example 1:

```
spice_cake = Cake("spice", 18, 8)           # value = (18/8)*8 = 18
chocolate_cake = Cake("chocolate", 24, 6)   # value = (24/6)*6 = 24
print(spice_cake.isEqualTo(chocolate_cake))     # prints False
print(spice_cake.isGreaterThan(chocolate_cake)) # prints False
print(spice_cake.isLessThan(chocolate_cake))    # prints True
```

Example 2:

```
spice_cake = Cake("spice", 18, 8)          # value = (18/8)*8 = 18
chocolate_cake = Cake("chocolate", 24, 6)  # value = (24/6)*6 = 24
print(spice_cake.sell(1))                  # value = (18/8)*7 = 15.75
print(chocolate_cake.sell(3))              # value = (24/6)*3 = 12
print(spice_cake.isEqualTo(chocolate_cake))     # prints False
print(spice_cake.isGreaterThan(chocolate_cake)) # prints True
print(spice_cake.isLessThan(chocolate_cake))    # prints False
```