[Index | Exercise 3.3 | Exercise 3.5]

Exercise 3.4

Objectives:

- Learn how to encapsulate object internals using private attributes, properties, and slots

Files Modified: `stock.py`

(a) Private attributes

As a general rule, attributes that are internal to a class should have a leading underscore. In the previous exercise, the `Stock` class had a `types` class variable that was used for converting rows of data. Change the code so that this variable has a leading underscore on it.

(b) Properties for computed attributes

Earlier, you defined a class `Stock`. For example:

```
>>> s = Stock('GOOG',100,490.10)
>>> s.name
'GOOG'
>>> s.shares
100
>>> s.price
490.1
>>> s.cost()
49010.0
>>>
```

Using a property, turn `cost()` into an attribute that no longer requires the parentheses. For example:

```
>>> s = Stock('GOOG', 100, 490.1)
>>> s.cost               # Property. Computes the cost
49010.0
>>>
```

(c) Enforcing Validation Rules

Using properties and private attributes, modify the `shares` attribute of the `Stock` class so that it can only be assigned a non-negative integer value. In addition, modify the `price` attribute so that it can only be assigned a non-negative floating point value.

The new object should work almost exactly the same as the old one except for extra type and value checking.

```
>>> s = Stock('GOOG', 100, 490.10)
>>> s.shares = 50          # OK
>>> s.shares = '50'
Traceback (most recent call last):
...
TypeError: Expected integer
>>> s.shares = -10
Traceback (most recent call last):
...
ValueError: shares must be >= 0

>>> s.price = 123.45       # OK
>>> s.price = '123.45'
Traceback (most recent call last):
...
TypeError: Expected float
>>> s.price = -10.0
Traceback (most recent call last):
...
ValueError: price must be >= 0
>>>
```

(d) Adding `__slots__`

Modify your new `Stock` class to use `__slots__`. You will find that you have to use a different set of attribute names than before--specifically, you will have to list the private attribute names (e.g., if a property is storing a value in an attribute `_shares`, that is the name you list in `__slots__`). Verify that the class still works and that you can no longer add new attributes.

```
>>> s = Stock('GOOG', 100, 490.10)
>>> s.spam = 42
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Stock' object has no attribute 'spam'
>>>
```

(e) Reconciling Types

In the current `Stock` class, there is a `_types` class variable that gives conversions when reading from a file, but there are also properties that are enforcing types. Who is in charge of this show? Fix the property definitions so that they use the types specified in the `_types` class variable. Make sure the properties work when types are changed via subclassing. For example:

```
>>> from decimal import Decimal
>>> class DStock(Stock):
        _types = (str, int, Decimal)
```

```
>>> s = DStock('AA', 50, Decimal('91.1'))
>>> s.price = 92.3
Traceback (most recent call last):
...
TypeError: Expected a Decimal
>>>
```

Discussion

The resulting `Stock` class at the end of this exercise is a muddled mess of properties, type checking, constructors, and other details. Imagine how unpleasant it would be to maintain code that featured dozens or hundreds of such class definitions.

We're going to figure out how to simplify things considerably, but it's going to take some time and some more advanced techniques. Stay tuned.

[Solution | Index | Exercise 3.3 | Exercise 3.5]
