[Index | Exercise 3.2 | Exercise 3.4]

Exercise 3.3

Objectives:

- Learn about class variables and class methods

Files Modified: `stock.py`, `reader.py`

Instances of the `Stock` class defined in the previous exercise are normally created as follows:

```
>>> s = Stock('GOOG', 100, 490.1)
>>>
```

However, the `read_portfolio()` function also creates instances from rows of data read from files. For example, code such as the following is used:

```
>>> import csv
>>> f = open('Data/portfolio.csv')
>>> rows = csv.reader(f)
>>> headers = next(rows)
>>> row = next(rows)
>>> row
['AA', '100', '32.20']
>>> s = Stock(row[0], int(row[1]), float(row[2]))
>>>
```

(a) Alternate constructors

Perhaps the creation of a `Stock` from a row of raw data is better handled by an alternate constructor. Modify the `Stock` class so that it has a `types` class variable and `from_row()` class method like this:

```
class Stock:
    types = (str, int, float)
    def __init__(self, name, shares, price):
        self.name = name
        self.shares = shares
        self.price = price

    @classmethod
    def from_row(cls, row):
        values = [func(val) for func, val in zip(cls.types, row)]
        return cls(*values)
    ...
```

Here's how to test it:

```
>>> row = ['AA', '100', '32.20']
>>> s = Stock.from_row(row)
>>> s.name
'AA'
>>> s.shares
100
>>> s.price
32.2
>>> s.cost()
3220.0000000000005
>>>
```

Carefully observe how the string values in the row have been converted to a proper type.

(b) Class variables and inheritance

Class variables such as `types` are sometimes used as a customization mechanism when inheritance is used. For example, in the `Stock` class, the types can be easily changed in a subclass. Try this example which changes the `price` attribute to a `Decimal` instance (which is often better suited to financial calculations):

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

>>> row = ['AA', '100', '32.20']
>>> s = DStock.from_row(row)
>>> s.price
Decimal('32.20')
>>> s.cost()
Decimal('3220.0')
>>>
```

Design Discussion

The problem being addressed in this exercise concerns the conversion of data read from a file. Would it make sense to perform these conversions in the `__init__()` method of the `Stock` class instead? For example:

```
class Stock:
    def __init__(self, name, shares, price):
        self.name = str(name)
        self.shares = int(shares)
        self.price = float(price)
```

By doing this, you would convert a row of data as follows:

```
>>> row = ['AA', '100', '32.20']
>>> s = Stock(*row)
>>> s.name
'AA'
>>> s.shares
100
>>> s.price
32.2
>>>
```

Is this good or bad? What are your thoughts? On the whole, I think it's a questionable design since it mixes two different things together--the creation of an instance and the conversion of data. Plus, the implicit conversions in `__init__()` limit flexibility and might introduce weird bugs if a user isn't paying careful attention.

(c) Reading Objects

Change the `read_portfolio()` function to use the new `Stock.from_row()` method that you wrote.

Point of thought: Which version of code do you like better? The version of `read_portfolio()` that performed the type conversions or the one that performs conversions in the `Stock.from_row()` method? Think about data abstraction.

Can you modify `read_portfolio()` to create objects using a class other than `Stock`? For example, can the user select the class that they want?

(d) Generalizing

A useful feature of class-methods is that you can use them to put a highly uniform instance creation interface on a wide variety of classes and write general purpose utility functions that use them.

Earlier, you created a file `reader.py` that had some functions for reading CSV data. Add the following `read_csv_as_instances()` function to the file which accepts a class as input and uses the class `from_row()` method to create a list of instances:

```
# reader.py
...

def read_csv_as_instances(filename, cls):
    '''
    Read a CSV file into a list of instances
    '''
    records = []
    with open(filename) as f:
        rows = csv.reader(f)
        headers = next(rows)
        for row in rows:
```

```
            records.append(cls.from_row(row))
    return records
```

Get rid of the `read_portfolio()` function--you don't need that anymore. If you want to read a list of `Stock` objects, do this:

```
>>> # Read a portfolio of Stock instances
>>> from reader import read_csv_as_instances
>>> from stock import Stock
>>> portfolio = read_csv_as_instances('Data/portfolio.csv', Stock)
>>> portfolio
[<__main__.Stock object at 0x100674748>,
<__main__.Stock object at 0x1006746d8>,
<__main__.Stock object at 0x1006747b8>,
<__main__.Stock object at 0x100674828>,
<__main__.Stock object at 0x100674898>,
<__main__.Stock object at 0x100674908>,
<__main__.Stock object at 0x100674978>]
>>>
```

Here is another example of how you might use `read_csv_as_instances()` with a completely different class:

```
>>> class Row:
        def __init__(self, route, date, daytype, numrides):
            self.route = route
            self.date = date
            self.daytype = daytype
            self.numrides = numrides
        @classmethod
        def from_row(cls, row):
            return cls(row[0], row[1], row[2], int(row[3]))

>>> rides = read_csv_as_instances('Data/ctabus.csv', Row)
>>> len(rides)
577563
>>>
```

Discussion

This exercise illustrates the two most common uses of class variables and class methods. Class variables are often used to hold a global parameter (e.g., a configuration setting) that is shared across all instances. Sometimes subclasses will inherit from the base class and override the setting to change behavior.

Class methods are most commonly used to implement alternate constructors as shown. A common way to spot such class methods is to look for the word "from" in the name. For example, here is an example on built-in dictionaries:

```
>>> d = dict.fromkeys(['a','b','c'], 0)     # class method
>>> d
{'a': 0, 'c': 0, 'b': 0}
>>>
```

[Solution | Index | Exercise 3.2 | Exercise 3.4]

>>> Advanced Python Mastery
. . . A course by dabeaz
. . . Copyright 2007-2023

 . This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License