[Index | Exercise 3.1 | Exercise 3.3]

Exercise 3.2

Objectives:

- Learn about attribute access
- Learn how use getattr(), setattr(), and related functions.
- Experiment with bound methods.

Files Created: `tableformat.py`

(a) The Three Operations

The entire Python object system consists of just three core operations: getting, setting, and deleting of attributes. Normally, these are accessed via the dot (.) like this:

```
>>> s = Stock('GOOG', 100, 490.1)
>>> s.name    #  get
'GOOG'
>>> s.shares = 50    # set
>>> del s.shares     # delete
>>>
```

The three operations are also available as functions. For example:

```
>>> getattr(s, 'name')            # Same as s.name
'GOOG'
>>> setattr(s, 'shares', 50)      # Same as s.shares = 50
>>> delattr(s, 'shares')          # Same as del s.shares
>>>
```

An additional function `hasattr()` can be used to probe an object for the existence of an attribute:

```
>>> hasattr(s, 'name')
True
>>> hasattr(s, 'blah')
False
>>>
```

(b) Using getattr()

The `getattr()` function is extremely useful for writing code that processes objects in an extremely generic way. To illustrate, consider this example which prints out a set of user-defined attributes:

```
>>> s= Stock('GOOG', 100, 490.1)
>>> fields = ['name','shares','price']
>>> for name in fields:
          print(name, getattr(s, name))

name GOOG
shares 100
price 490.1
>>>
```

(c) Table Output

In Exercise 3.1, you wrote a function `print_portfolio()` that made a nicely formatted table. That function was custom tailored to a list of `Stock` objects. However, it can be completely generalized to work with any list of objects using the technique in part (b).

Create a new module called `tableformat.py`. In that program, write a function `print_table()` that takes a sequence (list) of objects, a list of attribute names, and prints a nicely formatted table. For example:

```
>>> import stock
>>> import tableformat
>>> portfolio = stock.read_portfolio('Data/portfolio.csv')
>>> tableformat.print_table(portfolio, ['name','shares','price'])
      name     shares      price
---------- ---------- ----------
        AA        100       32.2
       IBM         50       91.1
       CAT        150      83.44
      MSFT        200      51.23
        GE         95      40.37
      MSFT         50       65.1
       IBM        100      70.44

>>> tableformat.print_table(portfolio,['shares','name'])
    shares       name
---------- ----------
       100         AA
        50        IBM
       150        CAT
       200       MSFT
        95         GE
        50       MSFT
       100        IBM
>>>
```

For simplicity, just have the `print_table()` function print each field in a 10-character wide column.

(d) Bound Methods

It may be surprising, but method calls are layered onto the machinery used for simple attributes. Essentially, a method is an attribute that executes when you add the required parentheses () to call it like a function. For example:

```
>>> s = Stock('GOOG',100,490.10)
>>> s.cost           # Looks up the method
<bound method Stock.cost of <__main__.Stock object at 0x409530>>
>>> s.cost()         # Looks up and calls the method
49010.0

>>> # Same operations using getattr()
>>> getattr(s, 'cost')
<bound method Stock.cost of <__main__.Stock object at 0x409530>>
>>> getattr(s, 'cost')()
49010.0
>>>
```

A bound method is attached to the object where it came from. If that object is modified, the method will see the modifications. You can view the original object by inspecting the `__self__` attribute of the method.

```
>>> c = s.cost
>>> c()
49010.0
>>> s.shares = 75
>>> c()
36757.5
>>> c.__self__
<__main__.Stock object at 0x409530>
>>> c.__func__
<function cost at 0x37cc30>
>>> c.__func__(c.__self__)      # This is what happens behind the scenes of
calling c()
36757.5
>>>
```

Try it with the `sell()` method just to make sure you understand the mechanics:

```
>>> f = s.sell
>>> f.__func__(f.__self__, 25)     # Same as s.sell(25)
>>> s.shares
50
>>>
```

[Solution | Index | Exercise 3.1 | Exercise 3.3]

>>> Advanced Python Mastery
. . . A course by dabeaz
. . .