327: Object-oriented programming

Lecture 8

9/27/2021

Professor Barron

Today…

- HW1 Solution overview
- On to chapter 5
- When to use OOP?
- Manager objects
- Magic methods
- Context managers
- Functions/objects

When to use OOP? (Chapter 5)

- Data, no behaviors
  - Use data structures like list, set, dict, etc.
- Behavior, no data
  - Use static functions
- Both data and behaviors (or likely to grow into this case)
  - Use objects

When to use OOP?

- How many objects would be made from each class?
- OOP provides room to grow
- Do not repeat yourself (DRY)
  - cut/paste generally better than copy/paste
  - Re-use code with composition or inheritance

When to use OOP?

- OOP more verbose, but more lines of code is not inherently bad
- How readable is the code?

```
1   square = Polygon()
2   square.add_point(Point(1,1))
3   square.add_point(Point(1,2))
4   square.add_point(Point(2,2))
5   square.add_point(Point(2,1))
6   square.perimeter()
7   #-----------------------------------
8   square = [(1,1), (1,2), (2,2), (2,1)]
9   perimeter(square)
```

Manager objects

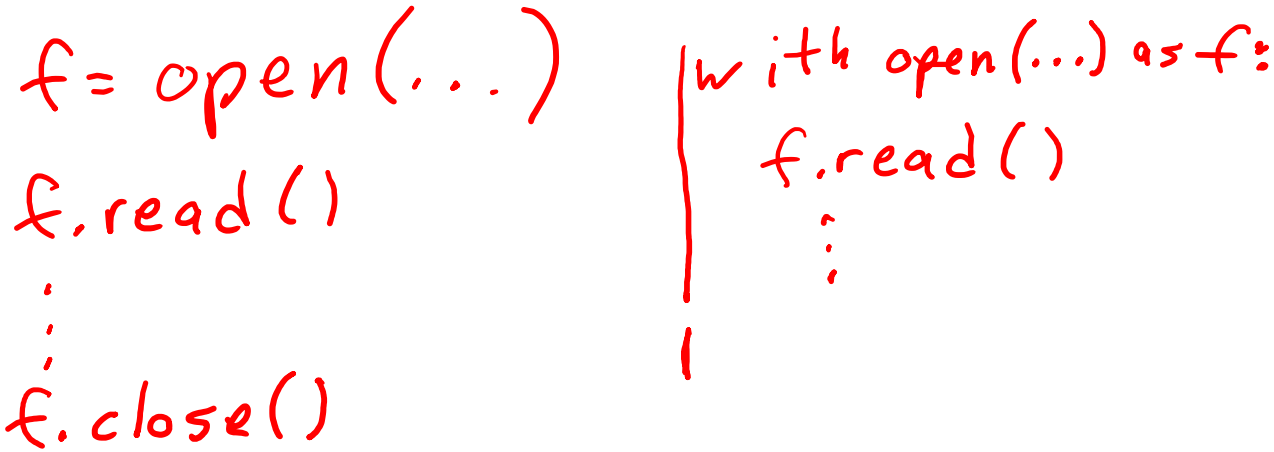
- High-level objects that manage other objects
- Often more abstract than other objects
- Don’t do much themselves
- Call methods and pass messages between objects
- Example… ZipReplace

Magic methods

- len(obj)
  - obj.__len__()
- reversed(obj)
  - obj.__reversed__()
- x in obj
  - obj.__contains__(x)
- obj1 + obj2
  - obj1.__add__(obj2)
- obj[5]
  - obj.__getitem__(5)
- obj[5:10]
  - obj.__getslice__(5,10)
- repr(obj)
  - obj.__repr__()
- str(obj)
  - obj.__str__()
- obj1 == obj2
  - obj1.__eq__(obj2)
- if obj
  - if obj.__bool__()
- for x in obj
  - for x in obj.__iter__()

Note: print() uses str(), but if str() is not overridden, then the object.__str__() method will call repr() by default

Context managers



with object as name:

    # using name in some way

- enters a local context with setup and cleanup handled automatically
- assigns object.__enter__() to name
- object.__exit__() is called when leaving the context

- locking/unlocking resources
- opening/closing files

Functions as objects/objects as functions

- Functions can be passed around, modified, called later, partially applied
- Any object can act like a function by making it callable

- foo(arg1, arg2)
  - foo.__call__(arg1, arg2)

- Methods are attributes and can be changed during runtime
  - “Monkey patching”
  - Useful for testing, but should be avoided most of the time