Object-Oriented Design

csc148, Introduction to Computer Science
Diane Horton
Winter 2015

UNIVERSITY OF TORONTO

DCS50

Recap: Using Objects in Python

- *Examples in the Python shell.*
- An object bundles together:
  - data and
  - methods that can be applied to it.
- Objects are stored at a location in memory.
- Every value in Python is an object.
  - Even simple things like integers.
- Python defines many useful types of object.
  - `str`, `list`, `dict`, etc.

Problem

- We need a program that will work with points:

  In two dimensions, a point is two numbers (coordinates) that are treated collectively as a single object. Points are often written in parentheses with a comma separating the coordinates. For example, (0, 0) represents the origin, and (x, y) represents the point x units to the right and y units up from the origin. Some of the typical operations that one associates with points might be calculating the distance of a point from the origin, or from another point, or finding a midpoint of two points, or asking if a point falls within a given rectangle or circle.

How should we represent a point?

- Options?

Designing a class: *what*

- First priority is to define what the class does.
- What does the class represent?
  - Record this in a class docstring.
- What things must we be able to do to instances of the class?
  - Define the header and docstring of a method for each.
- Once we have this, we have enough to . . .

The public interface

- . . . we have enough to use `help(Point)`
- This is all that any client code needs to know in order to use the class.
  - Just like when you use `str` or `dict`.
- We have the public interface of the class.
  - interface: a point where two things meet and interact.
  - public: meant to be seen by programmers who are not working on the class itself.

How do you decide?

- A good strategy:
  - the most important noun → the class name
  - other nouns → attributes
  - verbs → methods
- *Example: A public interface for class* `Point`.

We can start writing client code

- With only the public interface, we are good to go.
- *Example: A simple program that uses class* `Point`.

Designing a class: *how*

- Next we figure out how to implement the public interface.
  - Create instance variables in the `__init__` method.
  - Write the body of each method.
- Of course, helper methods may be helpful.
- *Example: An implementation of class* `Point`.

Recap: special methods

- These methods are called implicitly.
- `__init__` is called when you construct an instance of the class.
- `__str__` is called when you print an instance.
- `__eq__` is called when you compare two instances with `==`.
- Your class inherits these methods from `Object`.
- When you define your own version, you are overriding the inherited one.

Recap: calling special methods

- Examples:

```
>>> p1 = Point([6, 1, 2])
>>> print(p1)
>>>(6.0, 1.0, 2.0)
>>> str(p1)  # The str function calls __str__
>>> '(6.0, 1.0, 2.0)'
>>> p2 = Point([6, 1])
>>> p1 == p2
>>> False
```

- You can also call them explicitly (but we don't):

```
>>> p1.__str__()
>>> '(6.0, 1.0, 2.0)'
>>> p1.__eq__(p2)
>>> False
```

Another special method: `__repr__`

- Method `__str__` returns a human-friendly string describing the object.
- But `__repr__` returns a string that can construct an object with the same value.
- It is the "official" string representation of the object.
- *Examples in the shell.*
- Note: If you don't define a `__str__` method, `__repr__` is called in its place.

Controlling access

- We saw that client code does not need to know any implementation details.
- But it can, inadvertently or maliciously, access the attributes of our class.
- *Examples.*
- We can prevent this by using the property mechanism.
- *Example: using property in class* `Point`.

Using the property mechanism

- For each instance variable `x`, define methods `set_x` and `get_x`.
- Call function `property` to re-direct any:
  - attempt to assign to `x` → call to `set_x`
  - attempt to evaluate `x` → call to `get_x`
- Now client code can only do with `x` what your `set_x` and `get_x` methods allow.
- Important: Client code that did not directly touch the attributes still works as is.