Object-Oriented Programming

CSC148, Introduction to Computer Science
Fall 2016
Diane Horton
A scenario

• Suppose we have data from Twitter:
  – For users: userid, name, and bio
  – Who follows whom
  – For tweets: content, date, tweeter’s userid

• Suppose we want to compute, e.g., a user’s:
  – Average tweet’s per day
  – Date with the most tweets
  – Average tweet length
  – Longest period with no tweets
  – All tweets mentioning someone they follow
Discussion
v0: with dicts

• Each user is represented by a dict, e.g.:
  
  ```
  {'userid': 'dianelynnhorton',
   'bio': 'Does cartwheels on special occasions',
   'follows': ['karenreid', 'potus', 'drake']}
  ```

• Each tweet is represented by a dict, e.g.:
  
  ```
  {'userid': 'Cmdr_Hadfield',
   'date': 'Sep 11',
   'content': 'It’s wallpapered with Velcro.'}
  ```

• Pass these to functions that compute stats.
• Problems?
v1: with classes

• Each tweet is an instance of a class.
  – userid, created_at and content are instance variables
  – The constructor initializes them
  – Client code uses dot notation to access them

• We have added a new type to Python!

• Similarly, each user can be an instance of a class (to be written).

• Pass these to functions that compute stats.

• Benefits?
Special method \_
   \_
   \_

- \_
   \_
   \_
   is a **special method**. **We don’t have to call it explicitly.**

- **When we create a new object**, for example

```python
t = Tweet(‘mo123’, date(2016, 9, 16), ‘cool’)
```

**three things happen:**

1. The object is created.
2. \_
   \_
   \_
   is called. The object’s id is passed to the first parameter.
3. The object’s id is returned.

- \_
   \_
   \_
   is called a **constructor**.
Best practises

1. Method `__init__` should initialize every instance attribute.

2. No other code should create attributes.
   - In fact, it is possible for other code to add attributes.
     Example:
     ```python
t = Tweet(‘Rukhsana’, date(2016, 9, 16), ‘Hey!’)
t.topping = ‘bacon’
```
   - But this should never be done.

Effect: Client code can be sure every instance has the same attributes.
More terminology

• Class
• Object, or instance of the class
• Instance attribute, instance variable, or data member
• Dot notation
Representation Invariants

• A representation invariant is
  – a boolean statement about the attributes of the class
  – which must be true for any instance of the class

• The type contract is a simple representation invariant.

• But we can generalize this idea to express other kinds of things.
  – For example: \( \text{len(content)} \geq 140 \)
RIs are important to clients

• Client code has full access to our instance variables. (More on this later.)
• So it must know the RIs in order to keep instances in a consistent & meaningful state.
Using objects

• We’re used to doing things like this:
  >>> word = 'supercalifragilisticexpealidocious'
  >>> word.count('i')
  6

• ”Hey word, you’re a string. Could you please count the occurrences of ’i’ in you?”

• How does Python know which string to count?
  – As with our initializer, one value is passed implicitly:
    The value of word is passed to the first parameter of count.

• Aside: We could, but don’t, call count like this:
  >>> str.count(word, 'i')
Using our class

• We can’t use it in that way because it has no methods.
• It provides no services other than storage.
• We can change that by moving functions inside the class.
v2: with classes that include methods

- Each class both holds data and provides services related to that data.
Methods and self

• A function defined in a class is called a method.

• The first parameter is passed implicitly. E.g.:

```python
david = User('davidLiu', 'Lover of chocolate.')
david.verbosity(2016)
```

Method verbosity receives

- The value of `david` into the first parameter
- The value `2016` into the second parameter

• By convention, we always call that first parameter `self`.
Class Design Recipe

• The Function Design Recipe taught us to
  – Separate interface from implementation.
  – Define the interface and test cases *before* implementing.

• The Class Design Recipe does also.
Class User

• Notice that an instance of User contains references to 0 or more instances of Tweet.

• We say that:
  – Class User is in a **contains** relationship to class Tweet.
  – User **has-a** Tweet.
Choose method or function?

• Methods are for behaviours that
  – are essential to any client of the class
  – are integral to the very definition of that thing

• Functions are for behaviours that
  – are more specialized
  – not every client of the class would need
  – it feels reasonable to expect the client code to implement

• The choice may not be clear cut.
RIs are helpful with methods

• Representation invariants are very helpful now that we have methods.

• They help you write your methods:
  – You can assume the RIs are true when the method begins.
  – (They may become temporarily false during the method.)
  – You know you must restore the RIs by the end.
    This gives you a target that may suggest part of your algorithm.
All attributes are openly available

• The Python philosophy is to welcome client code to access the data unless there is good reason not to.

• Some good reasons not to welcome client code to access the data:
  – The structures storing the data are complex. Client code shouldn’t have to think about that.
  – There are important constraints on the data. We don’t want client code to mess it up!
When we want data to be private

- Indicate that intention with an underscore. Example: `_follows`
- This tells client code:
  - You shouldn’t access this attribute directly.
  - You should get the information by calling methods.
- The underscore does not prevent access; it only signals our intention.
- These methods create an interface that:
  - Provides an abstraction for the client code.
  - Ensures the structures storing the data are not messed up.
Separating Interface from Implementation

• The distinction helps with two critical things.
  • Management of complexity.
    – The class *encapsulates* all details about the implementation.
    – Client code can be written at a higher level.
    – Cognitive psychology: limited working memory; ”chunking” helps.
    – This is essentially chunking.
  • Plug-out / plug-in compatibility of code.
    – We can change the implementation of a class and client code runs the same.
Modelling a domain with classes

• When we tackle a new problem we must
  – Choose what the classes should be
  – Decide what each class will be responsible for

• In other words: we must decide how to model the domain.

• Simple rule of thumb:
  – An important noun suggests a class
  – A verb suggests a method