CS 61A Object Orientated Programming (OOP) Notes

Fall 2018 Jemmy Zhou

1 Classes

(a) Think of it as blueprints/a mold; *i.e. coffee machine*

(b) Creates things called objects(instances); *i.e. a cup of coffee*

2 Functions vs. Methods

(a) We call functions inside classes, **methods**

- Methods still take in 1 or parameter(s). Why at least 1?
- Executes one line at a time
- Returns some result (can be None)

(b) Magic methods; *i.e.* `__init__`

- What each magic method does is unique
- When we create an instance of a class, *i.e.* `Baller('Jemmy')`, Python by default will call the `__init__` method, which executes some code then returns the instance that was created (implicitly)

3 Attributes

(a) **Instance Attributes**

- Defined inside of methods
- Property of the instance (unique to each instance)
- Notation for defining is `self.attr_name = value`
- Notation for referencing is `self.attr_name`

(b) **Class Attributes**

- Defined outside of methods
- Property of the class (same for every instance)
- Notation for defining is `attr_name = value`
- Notation for referencing is `CLASS.attr_name` or `self.attr_name`
  - Latter only works if there is no instance attribute with the same `attr_name`
- **CANNOT** reference class attributes as just `attr_name`

(c) Instances can have instance attributes with the same name as class attributes. Python essentially "overrides" the class attribute

(d) If referencing `self.attr_name`, Python will

1. Look at self's instance attributes. If found, return. Else:
2. Look at self's class' attributes. If found, return. Else:
3. Error

4 Method Calls

(a) Either `self.method(<param1>)` or `CLASS.method(self, <param1>)`

(b) When invoking `self.method(<param1>)`, the instance `self` is implicitly included as the first parameter

(c) When invoking `CLASS.method(self, <param1>)`, we have to explicitly include `self` as the first parameter

5 Inheritance

(a) Notation for inheriting a class is `class BallHog(Baller):`

(b) Can think of as parent/child relationship

(c) A child inherits everything the parent has

- class attributes
- methods (both regular and magic)
- instance attributes (defined in the methods)

(d) The child can improve on what the parent already does

- *i.e. Overriding a method from the parent class*
- When overriding, the method in the child class has to have the **exact same signature** as the method in the parent class

(e) The child can do new things

- *i.e. Defining new methods, new class attributes, new instance attributes*
- Can invoke methods of the parent class by calling `PARENT.method(self, ...)`