Notes on Object Orientated Programming

Last updated Fall 2019 by Jemmy Zhou

Disclaimer: These are not official class notes. They're just meant to be a quick reference. Please let me know if there are any typos or mistakes.

1. Classes

- Think of them as blueprints/molds; *i.e. coffee machine*
- Creates things called **objects** (or **instances** of a class); *i.e. a cup of coffee*

2. Functions vs. Methods

- We call functions inside classes **methods**
  - Similarities:
    - Executes one line at a time
    - Returns some result (can be None)
  - Differences:
    - Functions take in 0 or more parameter(s).
    - Methods take in 1 or more parameter(s). Why at least 1?
- Magic methods, *i.e.* `__init__(...)`
  - What each magic method does/returns is unique
    - For `__init__(...)`, when we create an instance of a class, i.e. `Baller('Tom')`, Python will call something like `Baller.__init__(self, 'Tom')`, which executes some code then returns `self` (implicity)
  - Some other magic methods:
    - `__str__(self)`
    - `__repr__(self)`
    - `__iter__(self)`
    - and many more!

3. Attributes

1. **Instance Attributes**
   - Defined inside methods**
   - Property of the instance (unique to each instance)
   - Notation for defining is `self.attr_name = value`
   - Notation for referencing is `self.attr_name`
2. **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`
     - NOTE: Latter only works if there is no instance attribute with the same `attr_name`.
3. Notes:
   - **CANNOT** reference attributes as just `attr_name`.
   - Instances can have instance attributes with the same name as class attributes. Python will "override" the class attribute with the instance attribute.
   - 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

1. Either `self.method(<params>)` or `CLASS.method(self, <params>)` work.
2. When invoking `self.method(<params>)`, the instance `self` is implicity passed in as the first parameter.
3. When invoking `CLASS.method(<params>)`, we have to explicity pass in `self` as the first parameter.

5. Inheritance

1. Notation for inheriting class `Baller` in class `BallHog` is `class BallHog(Baller):`
2. Can think of as a parent/child relationship
3. A child inherits everything the parent has
    - class attributes
    - methods (both regular and magic)
    - instance attributes (defined in the methods)
4. 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.
5. The child can do new things
    - *i.e. Define new methods, new class attributes, new instance attributes...*
    - Can invoke methods of the parent class by calling `PARENT.method(self, ...)`