Section Review

Learn to Code with Ruby

Classes and Objects

- A **class** is a blueprint/template for creating objects.
- An object is a container for data (i.e. state) with methods to process or manipulate that data. The object is "concrete", the class is "abstract".
- The process of creating an object is called **instantiation**.

The initialize Method

- Ruby invokes the private **initialize** method when it instantiates an object from the class.
- We can define instance variables in **initialize** to represent the object's "state" (the data it will store).
- Our **initialize** method can accept parameters. We can assign arguments to instance variables, allowing objects to initialize with different values for state.
- When instantiating an object with the **new** class method, pass the method the arguments that you'd like to provide to **initialize**.

Readers and Writers

- An **instance method** is a method that the object will have. We define it in the object's class definition.
- All Ruby objects will initialize with some core methods such as **to_s**, which sets the string representation of the object. We can customize this output by declaring it within the class body.
- A **getter/reader** is an instance method that retrieves the value of a instance variable.
- A **setter/writer** is an instance method that writes a new value to an instance variable.
- We can manually define getters and setters but the more common approach is to use Ruby's built-in helper methods.

The attr Methods

- The **attr_reader** method defines a getter method for the specified arguments (representing instance variables)
- The **attr_writer** method defines a setter method for the specified arguments (representing instance variables)
- The **attr_accessor** method defines both a getter and setter method for the specified arguments (representing instance variables)
- Pass symbol arguments for the instance variables names. Parentheses are optional.

The self Keyword

- The **self** keyword provides a reference to the entity within which it is used. **self**'s value can vary.
- Within an instance method, **self** will refer to the instance/object that will eventually be created from the class.
- We can use **self** to invoke instance methods within other instance methods. This pattern can be helpful for breaking up complex logic.
- We can omit **self** *most* (but not all) of the time. Ruby will substitute the object. instance as the method receiver.