Instance vs Class Variables

In our exploration of classes so far we have used plenty of **instance variables** or **attributes**. Similar to how we can have *class methods* we can also have *class variables*. Let's compare the two.

Instance Variables

Like we learned previously, `instance variables` are denoted with `@` and are typically assigned inside `#initialize`:

```
class Car
  def initialize(color)
    @color = color
  end

  def color
    @color
  end
end

car_1 = Car.new("red")
p car_1.color     # "red"

car_2 = Car.new("black")
p car_2.color     # "black"
```

Nothing new here. If we want cars to vary in the property of color, then we simply make the relevant instance variable for `@color`. Great, but what if we wanted to have a property that is shared among all cars? Let's accomplish this next using a class variable.

Class Variables

Let's say we wanted all car instances to have the same number of wheels. We can add a class variable, `@@num_wheels`:

```
class Car
  @@num_wheels = 4

  def initialize(color)
    @color = color
  end

  # getter for @color instance variable
  def color
    @color
  end

  # getter for @@num_wheels class variable
  def num_wheels
    @@num_wheels
  end
end

car_1 = Car.new("red")
p car_1.num_wheels    # 4

car_2 = Car.new("black")
p car_2.num_wheels    # 4
```

Notice that we use `@@` to denote class variables and typically assign these variables right inside of the class, but *not* inside of `#initialize`. This means that any car instance we create will be able to refer to this single, shared `@@num_wheels` variable. An important distinction to have in mind is that instances `car_1` and `car_2` have their own/separate `@color` variables, but share a single `@@num_wheels` variable.

As a result of all instances sharing this single variable, a change to this variable will effect all instances. Let's create a class method that sets `@@num_wheels`:

```
class Car
  @@num_wheels = 4

  def self.upgrade_to_flying_cars
    @@num_wheels = 0
  end

  def initialize(color)
    @color = color
  end

  def num_wheels
    @@num_wheels
  end
end

car_1 = Car.new("red")
car_2 = Car.new("black")

p car_1.num_wheels    # 4
p car_2.num_wheels    # 4

Car.upgrade_to_flying_cars

p car_1.num_wheels    # 0
p car_2.num_wheels    # 0

car_3 = Car.new("silver")
p car_3.num_wheels    # 0
```

The future is now! Changing class variables is really powerful since it effects every instance that we created and *will create in the future* (see `car_3` above). However, with great power comes great responsibility, so be very cautious when writing such code.

Class Constants

Often times, we'll want to prevent class variables from being changed for safety. In this scenario we'll want to create a **class constant** instead. As it's name suggests, a constant cannot be reassigned. Let's redo the last example with a class constant:

```
class Car
  NUM_WHEELS = 4

  def self.upgrade_to_flying_cars
    NUM_WHEELS = 0    # SyntaxError: dynamic constant assignment
  end

  def initialize(color)
    @color = color
  end

  def num_wheels
    NUM_WHEELS
  end
end

car_1 = Car.new("red")
car_2 = Car.new("black")

p car_1.num_wheels    # 4
p car_2.num_wheels    # 4

Car.upgrade_to_flying_cars
```

Class constant names must be capitalized. Notice that reassigning the constant will fail with an error, exactly what we wanted!

- an `@instance_variable` will be a distinct variable in each instance of a class; changing the variable will only effect that one instance
- a `@@class_variable` will be shared among all instances of a class; changing the variable will effect all instances because all instances of the class
- a `CLASS_CONSTANT` will be shared among all instances of a class, but cannot be changed