## Introduction

In this step, we're going to have a go at creating our own classes.

Create the file *animal.rb*:

```
class Animal
  def initialize(name="Animal")
    @name = name
    puts "An animal has been born"
  end

end
```

We create a new `Animal` class.

One of the most important methods in a class is the `initialize` method. This will be called every time a new `Animal` object is created. In this case, the initialize method has a parameter `name` and has a default value of "Animal". This means that all instances of the `Animal` class will have a name that will be stored in the instance variable here that starts with the `@` symbol (`@name`). We saw instance variables used earlier in Sinatra.

An **instance variable** differs from a normal type of variable, because instead of being available just inside the scope of the method where it was created, like normal variables, an instance variable is available throughout the scope of the class. So all of the methods inside the `Animal` class will have access to it.

## Initialize Method

The last thing we do in the `initialize` method is use `puts` to output a message.

Now boot IRB and require your file to proceed.

## .new Method

The new method is used to create instances of a class:

```
animal = Animal.new
```

The `initialize` method will be run.

## Method say_hello

Let's create another object:

```
dog = Animal.new('Bob')
```

Note that the unique ID is slightly different from the one before.

Create a new method:

```
class Animal
  def initialize(name="Animal")
    @name = name
    puts "An animal has been born"
  end

  def say_hello
    "Hello! My name is #{ @name }"
  end
end
```

It uses string interpolation to insert the name of the animal using the instance variable:

```
dog.say_hello
```

## Add More Properties to Animal Objects

Now say we wanted to add more properties to our animal objects. We do this by adding what is called a **getter method** to the class. The name of getter methods is always the same as the property that we want to add.

So say we wanted to add a property of `speed` to our animal class. We do this by creating a method like so:

```
def speed
  @speed
end
```

This returns a value of an instance variable `@speed`.

## Query Bob Object in IRB

However currently `speed` will return `nil` because it is not set anywhere:

```
dog.speed
```

## Setter and Getter method.

So, what we need is a **setter method**. A setter method looks very similar to a getter method. Note the equal symbol along with a parameter after the name of the method:

```
def speed=(value)
  @speed = value
end
```

Again the method has exactly the same name as the property we're setting. Inside the method, we set the instance variable. It has the same name as the property. It doesn't actually have to have the same name, but it's useful to do so.

```
dog.speed = 10
dog.speed
```

## Shortcuts for Setters and Getters.

Ruby supplies some shortcuts for setters and getters. Instead of having to write out the methods, all we need to do is use these lines of code:

```
attr_reader :strength
attr_writer :strength
```

So say we wanted to have a getter method to read animal's strength. For this we are using `attr_reader` with a symbol that equals to the name of the desired attribute. This will create a getter method automatically for property called `strength`.

If we want to then set this method, we use the `attr_writer` shortcut that we can see here.

## Attribute Accessor

Because it's very common to have both get and set properties, there's also another shortcut

```
attr_accessor :agility
```

This will allow you to get and set properties at the same time, and you only need one line.