Ruby Objects

Objects

- Typically a representation of something in the real world
- Have properties and methods (functions attached to the object)
- Are a convenient way to model the generic structure of something and store data about an "instance" of it

Objects

- Example: a User has a first name, a last name, and a method to return their full name based on both
- All users are not the same but they share similar properties and the information we need from them all is similar
- Therefore, we create an *object* to "model" a user and an *instance* of that *object* to model a *specific* user

Basic Object Syntax

The most basic object we can have looks like this:

```
class Dinosaur
end
```

Object names are always capitalized because objects in Ruby are **constants**, meaning their name never changes while the program is running.

Basic Object Syntax

Instances

The initialize method gets called every time a new **instance** of the object is instantiated, meaning the new method is called on the object

```
class Dinosaur
  def initialize
    puts "ROAR"
  end
end

Dinosaur.new
>"ROAR"
```

Basic Object Syntax

Instances

An instance can be stored in a variable and called later on:

```
class Dinosaur
  def initialize
    puts "ROAR"
  end
end

my_dino = Dinosaur.new
>"ROAR"

puts my_dino
>#<Dinosaur:0x007f8b2b485e78>
```

Basic Object Syntax

Instances

Each instance has a unique ID which shows where it is located in the computer's memory

```
puts my_dino
>#<Dinosaur:0x007f8b2b485e78>
# After the colon, you'll find the memory location
```

Object Instances

An instance of an object has its own traits, which are stored in **instance variables**, denoted with the @ sign.

```
class Computer
  def initialize
    @ram = rand
    puts @ram
  end
end

Computer.new
>0.4774206149841012 7
Computer.new
>0.8619859240960505
```

Object Instances

In this example, each time a new instance is created, it is given a new `@ram` value that is random.

```
class Computer
  def initialize
        @ram = rand
      puts @ram
  end
end

Computer.new
>0.4774206149841012 7
Computer.new
>0.8619859240960505
```

Exercise: Storing Information on Instances

Try creating a new object that stores a piece of information on each instance using an instance variable

Output that information when a new instance of the object is created

Methods

- A method is a function that relates to a specific object and/or instances of that object
- It is accessed through a specific object. For instance, the `User` object has a `full_name` method: `User.new.full_name`
- Methods can be created to be run on *the object* (class methods) or *on an instance of the object* (instance methods)

Basic Object Method Syntax

instance methods vs. class methods

```
class Car
  #this is an instance method. It can only be
  #used on an instance, i.e.
  #Car.new.turn_on_engine
  def turn_on_engine
    puts "engine is on"
  end


  #this is a class method, so it can be called
  #directly on the class name i.e. Car.traffic_jam
  def self.traffic_jam
    puts "All cars are involved"
  end
end
```

Exercise

Create a very basic object that models a Robot. It should have an instance method to `compute` and a class method called `destroy_humanity`.

Try instantiating a new instance of the object and calling methods on your instance as well as the class itself.

Attributes on Ruby Objects

An instance variable can only be accessed internally inside of an object, meaning that if we want to ask the object what the value of one of its instance variables is, we'll need to use a method to do so.

This is the basic concept behind **attributes** on Ruby objects

Attributes on Ruby Objects

An attribute has two operations:
write and **read**

write, or the **setter**, is used to change the value or write an initial value to the attribute

read, or the **getter**, is used to retrieve the value of the attribute

Attributes on Ruby Objects

To write to an attribute, we create a **setter** method:

```
class Car
  def engine = (engine)
    @engine = engine
  end
end
```

Attributes on Ruby Objects

To read an attribute, we create a getter method:

```
class Car
  def engine
    @engine
  end
end
```

Exercise

Create a getter and setter method for the object you made in the last exercise.

Try using it by requiring your object in IRB:

```
require './my_ruby_file_with_object'
```

Then playing around with it:

```
my_object = SomeObject.new
my_object.attribute = "Some value"
puts my_object.attribute
>"Some value"
```

Getter/Setter: Built in methods

Ruby has built in methods to create a read or write on an object attribute.

```
attr_reader :attr_name
```

This method creates a readable attribute on an object instance. Functionally equivalent to:

```
def attr_name
  @attr_name
end
```

Getter/Setter: Built in methods

Ruby has built in methods to create a read or write on an object attribute.

```
attr_writer :attr_name
```

This method creates a writable attribute on an object instance. Functionally equivalent to:

```
def attr_name=(attr_value)
  @attr_name = attr_value
end
```

Getter/Setter: Built in methods

Ruby has built in methods to create a read or write on an object attribute.

```
attr_accessor :attr_name
```

This method creates a writable and readable attribute on an object instance. Functionally equivalent to writing out the methods to do this in the last two slides.

Simplified Object Syntax

```
class Car
  attr_accessor :brand
  attr_accessor :wheels

  def turn_on_engine
   puts  "engine is on"
  end

end
```

Object Syntax

```
# declare the "Car" class/object
# (classes must have a capital name)
class Car
  # give the Car object two attributes,
  # or data about each instance: brand and wheels
  attr_accessor :brand
  attr_accessor :wheels

  # create an "instance" method called "turn_on_engine"
  # "turn_on_engine" that can be called on any *instance* of Car
  def turn_on_engine
   puts  "engine is on"
  end

end
```

Working with object instances

After an object has been declared with attributes, you can instantiate a new copy of the object to store data about a certain type of that object

```
my_prius = Car.new
```

To store information on your new instance of Car:

```
my_prius.brand = "Toyota"
my_prius.wheels = 4
```

Working with object instances

To see what information Ruby has stored on your new instance of Car:

```
my_prius
```

To call an instance method on your instance:

```
my_prius.turn_on_engine
```

Exercises

- Try creating your own object with at least two attributes and one instance method
- Instantiate a new copy of the object and assign it to a variable
- Change the attributes of your instance and try calling the method you've created on it
- Once you've been able to do this successfully, create a new method that references the two attributes

Constructors

In Ruby, a constructor is an addition to an `initialize` method to give a new instance of an object instance variable values from inception rather than on an ad-hoc basis.

Constructors

```
class Mammal
  def initialize(genus, color)
    @genus = genus
    @color = color
  end
end


rat = Mammal.new("Rattus", "gray")
puts rat.inspect
>#<Mammal:0x007fd845058f00 @genus="Rattus", @color="gray">
```

Exercise

- Create a new Ruby class with a constructor that sets at least 3 attributes on an object
- The object should have setters and getters to allow these attributes to be modified later on
- Your script should also include examples of instantiating the object, setting attributes, reading attributes and resetting them