Object Oriented JavaScript

What is Procedural programming?

- Most of the programming we've been doing so far has been of the procedural kind, meaning step-by-step.
- In procedural programming, we simply write out exactly what our program is doing in a line-by-line manner
- This makes it difficult to reuse any code we've written

What is Object Oriented programming?

- Rather than storing all of our code in one monolithic block, we organize it into objects
- These objects have attributes and methods, which is where all of the code we had written procedurally before ends up
- Rather than duplicating code every time we want some functionality, we instead call the object containing that functionality and invoke it

What is Object Oriented programming?

- Objects have awareness of their own functionality, attributes, and methods
- Objects, in almost all cases, shouldn't have to refer to other objects within their methods and attributes, there should be as much "separation of concerns" as possible
- This will increase the ease of testing later on in the development process

Creating your first JavaScript objects

JavaScript objects can be created in a few different ways, the simplest is the object literal:

```
var myCar = {
  brand: "Prius",
  wheels: 4
}
// More on accessing attributes later on
// but here's a simple example
myCar.brand
>>"Prius"
```

Exercise: Object Literals

- Try creating an object that represents something in the "real world" using the object literal notation (`Car`, `Person`, `User`)
- Create this object in the Chrome Developer Console or in a .js file that is loaded by a web page - then attempt to access the object in the developer console (type the variable name the object is stored in and press enter)

Creating your first JavaScript objects - constructor function

You can also create an object through a function, known as a constructor:

```
function Car(brand, wheels){
  this.brand = brand;
  this.wheels = wheels;
}

// creating a new instance of Car known as myCar
var myCar = new Car("toyota", 4);
```

This way, your objects have clearly defined attributes at instantiation (creation of an instance)

Difference between objects and their instances

- When we refer to an object, we refer to the prototypal definition of it
- For example, a `Car` has a `brand` and `wheels`
- When we refer to an **instance** of an object, we're talking about a specific example of that object
- For example, an instance of the `Car` object has a `brand` of `"toyota"` and a `wheels` count of 4
- Take a moment to process this, it is confusing

Creating your first JavaScript objects - what is `this` ?

`this` simply refers to the object instance we're currently within, in this case we're referring to any instance of `Car`:

```
function Car(brand, wheels){
  this.brand = brand;
  this.wheels = wheels;
}
```

Creating your first JavaScript objects - instantiation review

Instantiation is the invocation of a new copy, or instance, of your object

```
// defining what a car is
function Car(brand, wheels) {
  this.brand = brand;
  this.wheels = wheels;
}

// here we are instantiating a new instance of Car
var myCar = new Car("toyota", 4);

// we can create as many cars as we'd like!
var someOtherCar = new Car("maserati", 4);
```

Exercise

- Create a new JavaScript file with a constructor function for an object of your choosing
- Instantiate two copies of that object without getting syntax errors

Reading attributes, using methods

If you'd like to access the attributes or methods stored on your object, there are two different ways:

- dot notation
- square-bracket notation

Reading attributes, using methods

Square bracket notation can be used to retrieve attributes whose names must be notated with quotes

```
 var myCar = {
   brand: "Prius",
  "has-leather": false
 }

myCar["has-leather"]
 > false
```

Reading attributes, using methods

Dot notation can be used for attribute names that can be notated without using quotes

```
var myCar = {
  brand: "Prius",
  "has-leather": false
}

myCar.brand
> "Prius"
```

This is why you want to use simple names for attributes: the dot notation is far easier to use.

Storing functions in object literals

Functions can also be easily stored on attributes of an object:

```
var myCar = {
  brand: "Prius",
  friendlyBrand: function() {
    return "Your car's brand is " + this.brand;
  }
}
```

In this context, functions are known as methods. To access the method, just use () after the attribute name the function has:

```
myCar.friendlyBrand()
>>"Your car's brand is Prius"
```

Exercise

- Define an object using a constructor function that has at least one attribute and one method
- Instantiate a new instance of that object and try to access its attributes and methods

Prototypes

- JavaScript is actually an entirely object oriented language.
- The only "basic" types in JavaScript, the simple types, are numbers, strings, booleans, null, and undefined
- Everything else is an object! And in fact, the above are actually also objects, they are simply immutable, meaning unchangeable

Prototypes

- You can actually extend a JavaScript object, meaning add functionality through object attributes, fairly easily using its prototype
- This allows you to modify an object and its instances after they have been defined initially
- Don't forget that an object (`function Car`) is different from an object instance (`var myCar`)

Prototypes: sample code

```
function Car(brand, wheels) {
  this.brand = brand;
  this.wheels = wheels;
}

// add a method or attribute to the parent object
//by using the object's prototype
Car.prototype.friendlyBrand = function() {
  return "This car's brand is " + this.brand;
}

// define the new Car object
myCar = new Car("toyota", 4);

// now you can use that method in all instances of the object
myCar.friendlyBrand()
> "This car's brand is Toyota"
```

Exercise

- Take the object you created in the last exercise and extend it using its prototype
- Add one more attribute and one more method to it
- Try to access these attributes and methods on both an already created instance and a brand new instance of the object

Prototypal Objects

- We can also choose to have one object start out with another as its template
- For instance, an Admin object could extend from a User object but have a few different traits from the User object

Prototypical Objects: Example Part 1

```
// define the User constructor
function User(fname, lname, email){
  this.fname = fname;
  this.lname = lname;
  this.email = email;
}

// define the prototype methods
User.prototype.fullName = function() {
  return this.fname + " " + this.lname;
}
```

Prototypical Objects: Example Part 2

```
// define the Admin constructor
function Admin(fname, lname, email, admin){
  this.admin = admin

  // call User constructor
  User.call(this, fname, lname, email);
}

// admin now has all of the prototype methods of a User
Admin.prototype = Object.create(User.prototype);

// create a new admin
var newAdmin = new Admin("Orlando", "Caraballo", "o@email.com", true)

// because the Admin prototype was set to mirror
//  the User prototype we can now use the fullName() method on newAdmin
newAdmin.fullName()
```

Exercise

Try creating an object, then extend another object from it. Some examples:

Animal -> Mammal

House -> Room

Resources

Codecademy

Javascript: Objects I, Objects II

TeamTreeHouse

Javascript Loops, Arrays and Objects: Tracking Data Using Objects

Programming Step-By-Step: Object-Oriented Javascript Part 1