Monday: JavaScript Objects

JavaScript is an object-oriented programming language. Though we may not have realized it, we've already been working with JavaScript objects when we use data types like strings and numbers, or when we use a data collection like arrays. Even the functions we write are also objects in JavaScript. In this lesson, we will explore what it means to be an object in JavaScript.

Previously, we have had variables that have stored strings, numbers, Booleans, arrays and functions. What happens when we want a variable to store much more information about the "thing" the variable represents? For example, if *you* were a variable and we wanted to store information about you (your name, your course level, your planned track, your enrollment status) in the single variable "moringaStudent" - we'd need more than a single string or a single array, we'd need an object!

In technical terminology, objects are containers that **encapsulate** data - meaning all of the relevant data and functions for the thing that the variable name represents are kept together in a "capsule", better known as an **object** that can be created and manipulated in our programs as a single unit.

Here is an example of a `moringaStudent` object:

```
var moringaStudent = {
  firstName: "Charlie",
  lastName: "Obina",
  level: 1,
  track: ["Prep","JavaScript","Python", "Django"],
  enrollmentStatus: true
};
```

Let's take a look at how this object is defined. We have our variable `moringaStudent`. We assign it the value of an object by using the curly braces, `{ }`. This is called **literal notation** and we have used it previously to create strings by using quotes, `" "` and arrays by using brackets, `[ ]`.

Inside the curly braces are five properties for our `moringaStudent` object: first name, last name, level, track, and enrollment status. Every property of a JavaScript object consists of a **key-value** pair. The **key** is the variable that describes the kind of information to be stored. The **value** is the specific value of the key. So, in our example, the first name property has a key called `firstName` and a value of "Charlie", the last name property has a key `lastName`, with a value of "Obina" and so on, with the remaining keys `level`, `track` and `enrollmentStatus`.

Each key-value pair is separated by a colon. And pairs are separated from each other with a comma.

We could write our object like this and it would also work:

```
var moringaStudent = {firstName: "Charlie", lastName: "Obina", level: 1, track: ["Prep","JavaScr
ipt","Python", "Django"], enrollmentStatus: true};
```

However, the formatting of the object with each property indented two spaces on a separate line is a convention used when writing JavaScript objects to make it easy to see each property. Imagine an object with hundreds of properties written on the same line. It would be a bit of a challenge to sort out the details.

Property keys are always a JavaScript string (though quotes are not needed in this context) that starts with a letter. Property values can be any data type: strings, numbers, Booleans, arrays or even functions. When the value of a property is a function, we call it a **method**.

Here is an object with one property and one method. This method when called will make my cat "speak" by writing "Meow" to the console. (Previously, we have used `console.log` just for debugging, but we can use it for any message we'd like to see in the console.)

```
var myCat = {
  name: "Kitty Kanyau",
  speak: function() {
    console.log("Meow!");
  }
};
```

To keep it simple, you can think of properties as nouns, and methods as verbs or actions.

So, once we have an object, how do we use it? What do we do with the `name` and `speak` method?

To access properties and methods on objects, we can use either **dot notation** or **bracket notation**.

```
> myCat.name
  "Kitty Kanyau"
> myCat['name']
  "Kitty Kanyau"
> myCat.speak()
  Meow!
> myCat['speak']()
  Meow!
```

Dot notation is easier to write and read, but bracket notation will additionally allow us to use properties with special characters, or select properties using variables. Until we need the additional functionality, we'll stick with dot notation.

Let's create an empty new dog object. We use the curly braces to signal JavaScript to create a new object.

```
> var dog = {};
  undefined
```

The built-in JavaScript function that creates a new dog object returns the value of `undefined` but if we type `dog` we can see that an empty object has been created for the `dog` variable.

Now, let's give our dog some properties using dot notation. Here, our values are a string and a number

```
> dog.name = "Bark Simba"
  "Bark Simba"
> dog.age = 5
  5
```

Now, let's add an array for a property value:

```
> dog.colors = ["brown","black","white"]
  ["brown","black","white"]
```

The value of a property comes with all of the functionality of its type. For example, we are able to use indexing on the `colors` array as we've done with other arrays.

```
> dog.colors[0]
  "brown"
> dog.colors[1]
  "black"
```

We can use array methods on `colors` like `push`, which returns the new length of the array:

```
> dog.colors.push("gray");
  4
> dog.colors
  ["brown","black","white","gray"]
```

Number methods on `age`:

```
> dog.age = 5
  5
> dog.age + 10
  15
```

We can also update any property by reassigning its value:

```
> dog.name = "Rex"
  "Rex"
```

Let's add a method to our dog. This will be a property with a function as a value. In this case, we'll give our dog some `howl` functionality.

```
> dog.howl = function() { console.log("Aaaaaaaaaaaoooooooooooo!") }
> dog.howl()
  Aaaaaaaaaaaoooooooooooo!
```

With objects, we can use properties within other properties. What if we decided we wanted to calculate our dog's age in human years? Let's add another method to our `dog` object.

```
> dog.humanYears = function() {return this.age * 7}
  function(){return this.age * 7}
```

Notice that the `humanYears` function has a keyword of `this`. When `this` is used in an object's method, it always refers to the object on which the method is called. So, when we run `dog.humanYears()`, `this` will always refer to the object, `dog`. (`this` can also be used in other places, but it gets tricky depending on its context and we won't cover it in detail here.)

Now when we run `dog.humanYears()`, we get 35.

Additional Practice

In the JavaScript console, practice creating objects of your own. Feel free to follow along with this lesson or explore some objects of your own. Here is some guided practice to try.

Create an object that stores information about a flower including name, color, and height.

- Change the color of your flower using dot notation.
- Change the height of your flower using bracket notation.
- Add a property that indicates what kind of creatures help the flower with pollination. This includes bees, butterflies, and birds.
- Add one more creature to your list: humans
- Write a method that allows the flower to grow. After the method is run, the height value should be increased.
- View all of the properties and methods for your flower object.
- Explore viewing, adding and updating more properties and methods on your flower object.
- Try using some of the string, number and array methods you have used before on the properties that store these types of data.