Objectives:

- *Adding methods to an object*
- *Name spacing*

In this lesson we are going to learn about JavaScript Methods. In JavaScript we have seen that we can make an Object and then we can put it in whatever data we want inside of that Object.

```
var obj = {
    name: "Chuck"           // String
    age: 45                 // Number
    isCool: False           // Boolean
    friends: ["bob", "tina"]  // Array
    }
```

What we want to show you is that we can add functions as properties to Objects, in that case they are called methods rather than properties.

A method is just a function that is a property inside of an Object.

Let's add a sum function in our Object.

```
var obj = {
    name: "Chuck",
    age: 45,
    isCool: false,
    friends: ["bob", "tina"],
    sum: function(x,y) {        // Object Method
    return x + y;
    }
```

}

The important thing here is that we are just adding a function, does not matter what the function does, any function will work here, and we are just setting it as a value for the property *sum.* After hitting Enter let's see what we get after we type *obj*.

obj

➔ *Object {name: "Chuck", age: 45, isCool: false, friends: Array[2]}*

- *sum: function (x,y)*
  age: 45
- *friends: Array[2]*
  isCool: false
  name: "Chuck"
- *__proto__: Object*

Now let's see how we can call the *sum* function.

obj.sum(10,5);

➔ *15*

The reason we need to add a method in an Object, is because it helps us keep our code organized, so we can group things logically together.

Let's say we want to implement a method called *speak.*

```
function speak() {
    return "WOOF!";
    }
```

speak();

➔ *"WOOF!"*

Let's say we also want a *speak* method for cats and it should return *"MEOW"*.

```
function speak() {
    return "MEOW!";
    }

speak()
```

➔ *"MEOW!"*

Now when we call speak it returns *MEOW!* But now we have no way of accessing our original *speak* that returned *WOOF!*

What happened here is something called the *Namespace Collision.* That is a fancy way of saying that we have two different things that have the same name.

Instead if we added these functions as methods to an Object, we could have two different things named *speak* by putting them in different namespaces.

Its very simple to do that.

```
var dogSpace = {};                  // Declaring an empty Object
dogSpace.speak = function() {       // Adding a method (function) to our Object
        return "WOOF!"
        }

var catSpace = {};                  // Declaring an empty Object
catSpace.speak = function() {       // Adding a method (function) to our Object
        return "MEOW!";
        }
```

Now if we want to get WOOF! for dog, all we need to do is as below.

dogSpace.speak()

➔ *"WOOF!"*

catSpace.speak()

➔ *"MEOW!"*

This is a nice way of organizing our code, so we can have a bunch of methods that are grouped logically together, as all dog methods go inside *dogSpace* and all the cat methods go inside *catSpace.* Its also a way of avoiding *namespace collisions.*

This same exact logic applies to real world applications where we have things like *comments, posts* and *tags.* For these keys we might have methods like *new* or *delete* and we need to have our namespace properly, so they do not all conflict. We could have those methods called like below.

post.delete()

comment.delete()

user.delete()