# Monday: Literal Notation vs Constructors

## Literal Notation

In the previous lesson, we created objects using **literal notation**. Let's explore further examples of what literal notation looks like, and how the process of creating objects may be streamlined by using constructors.

Let's say a dog walker wants to keep track of all dogs they walk.

We'll create a dog object using literal notation:

```
var dog1 = {
  name: "Falcor",
  colors: ["black"],
  age: 4,
};
```

Now we'll create another:

```
var dog2 = {
  name: "Nola",
  colors: ["white", "black"],
  age: 6,
};
```

And another:

```
var dog3 = {
  name: "Patsy",
  colors: ["brown"],
  age: 7,
};
```

You may notice that this is getting a little repetitive. All of these dogs have the same attributes, and therefore this involves typing `name`, `colors` and `age` over and over again, each time we want to create a new dog. Good news is, there is a much faster way to make `Dog` objects! Instead of using literal notation to manually create each individual `Dog` object, we can use `Constructor` as a blueprint.

## Constructors

We will write a constructor to create `Dog` objects momentarily, but first, what *is* a constructor? The next lesson will introduce constructors and demonstrate how we can use constructors and prototypes

to make our lives easier.

After we write a constructor we'll be able to create the same dogs we created in literal notation above, with these three simple lines of code:

```
var dog1 = new Dog("Falcor", ["black"], 4);
var dog2 = new Dog("Nola", ["white", "black"], 6);
var dog3 = new Dog("Patsy", ["brown"], 7);
```

As you can see, this is *far* less code, and it's much more scaleable!