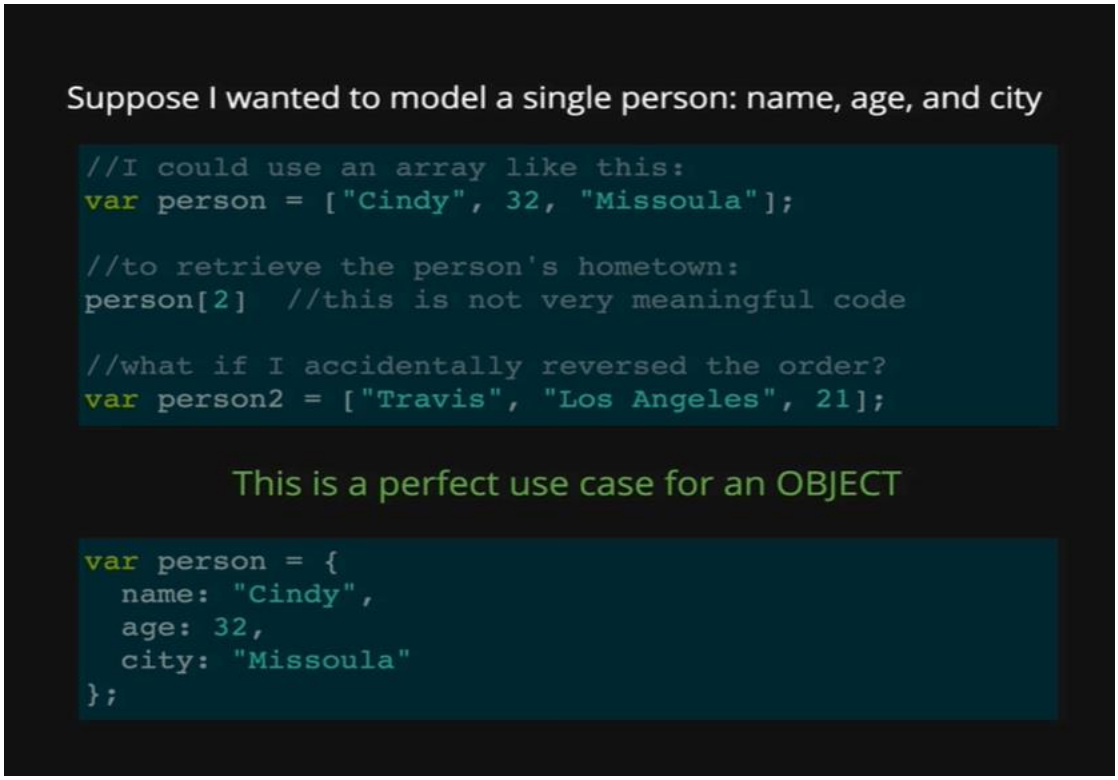
Objectives:

- *Understand objects conceptually*
- *Write code using JS objects*

In this lesson we are going to learn about a new data structure called the *Objects*.



Here is a hypothetical situation, suppose that we want to model a single person. In JavaScript each person has a name, an age and a city. There are lot of ways we can do that, we could have three different variables, name, age and city but then they are not related to one another they are totally separate. So, if we want to connect them, we can use an array like we are doing it by assigning it to the person variable on the first line where the first item is name “Cindy”, second item is age “32” and third item is city “Missoula”. This is not a great use of an array because this data is not really a list. Yes, we can force it to be an array, but this data does not lend itself to the format of an array where we have list that often has a logical order, there is no logical order here.

To access the city out of this array, we need to write person[2], but that requires us to know that the city is at index 2. But if we accidentally reverse the order and assign it to the person2 variable where we have “Travis” as the name upfront with index 0, and at index 1, we have the city Los

Angeles, and age 21 at index 2 and then if we access person[2], we would get age 21, instead of city “Missoula”. This is all to show that an Array is not the perfect solution for every situation.

There is a much better data structure for us to use here, which is the JavaScript Object. You can already see an example at the second block of the picture, as how we will take the person array with name, age and city and turn it into a JavaScript Object.

JS Array:

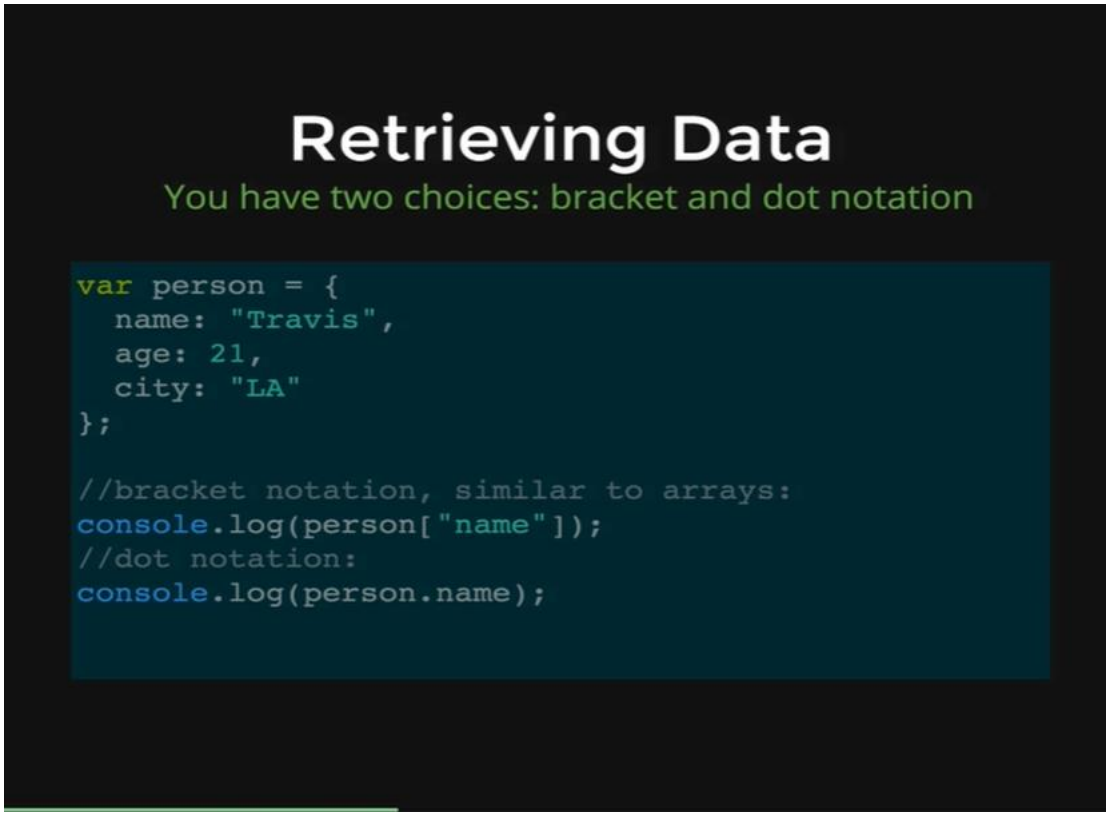
```
var person = [“Cindy”, 32, “Missoula”];
```

JS Object:

```
var person = {
    name: “Cindy”,
    age: “32”
    city: “Missoula”
    };
```

The first thing we can notice is that we have curly braces, rather than square brackets for an Object. The next important piece is that every item in this object is a key-value pair. We have a property or a key (name) and a value (“Cindy”) for each of the items.

This picture shows the exact same thing, we have a different person object. Inside the curly braces we are setting the name to be Travis, age to be 21 and city to be LA. Below that we also have a simple diagram of what this data structure looks like. We have three different slots in this object, and it is very important to note that objects do not have any built-in orders, unlike an array where there is a first item, a second item and then a third item. Think of an item inside of a given object, as just floating around in there, there is no order, no property comes first or second, it does not matter how we declare them and in what order, they are all treated the same; but this diagram shows the object order because we had assumed this order for this one. We can see "Travis" is stored under the key 'name', "21" is stored under the key 'age', and "LA" is stored under key 'city'.



To retrieve the data out of an object, we have two choices, we can either use the name key of the person object like 'person["name"]', in this case we are getting "Travis" out of the person object. This is very similar to arrays, the only difference is "name" is not a number, it is a string.

The other option is to use dot notation. Dot notation is little bit shorter and simpler; we write person.name and the name must match the name key and that would also give us "Travis".

var dog = {

name: "Rusty",

breed: "Mutt",

age: 3

}

dog

➔ *Object {name: "Rusty", breed: "Mutt", age: 3}*

dog["age"]

➔ *3*

dog.age

➔ *3*

Its up to you, which method you want to use to get the value out of the object; but there are few differences.

We cannot use dot notation if the property or the key of the value starts with a number. We also cannot use dot notation if the property or the key value has a space in it, like on the picture you can see JavaScript cannot accept "someObject.fav color", as JavaScript thinks we are accessing *fav* instead of *fav color*. That is if we wanted a property with a space which not a good practice, then we need to use ["fav color"].

The middle example shows that we can look up things using a variable if we use square bracket notation. If we have a variable called *str* and it is equal to *name* in quotes, if we do someObject.str it will just look for the property str, but if we do someObject[str], its going to evaluate str which gives us *name* in quotes then JavaScript will look for the name property in someObject. This is something we will see occasionally, that is why its worth knowing the difference here. We can use square bracket notation using the variable name to look up a property.

The next thing that we want to do is, be able to Update data inside of an object. Its very similar to arrays where we access the data and then reassign it with an equal sign. You can see on the picture we have our same person object with the name Travis, age is 21 and city is LA. If we want to add 1 to Travis' age all we need to do is access age, we can use the square brackets or the dot notation and then reassign it, person["age"] += 1, and then that will add 1 to age, and now the age is 22; or we can use person.city and reassign that to London, so person.city equal to London will now set person to have city of London.

Let's demonstrate that,

dog

➔ *Object {name: "Rusty", breed: "Mutt", age: 3}*

dog.age = 4;

➔ *4*

dog

➔ *Object {name: "Rusty", breed: "Mutt", age: 4}*

dog.age += 1;

➔ *5*

dog

➔ *Object {name: "Rusty", breed: "Mutt", age: 5}*

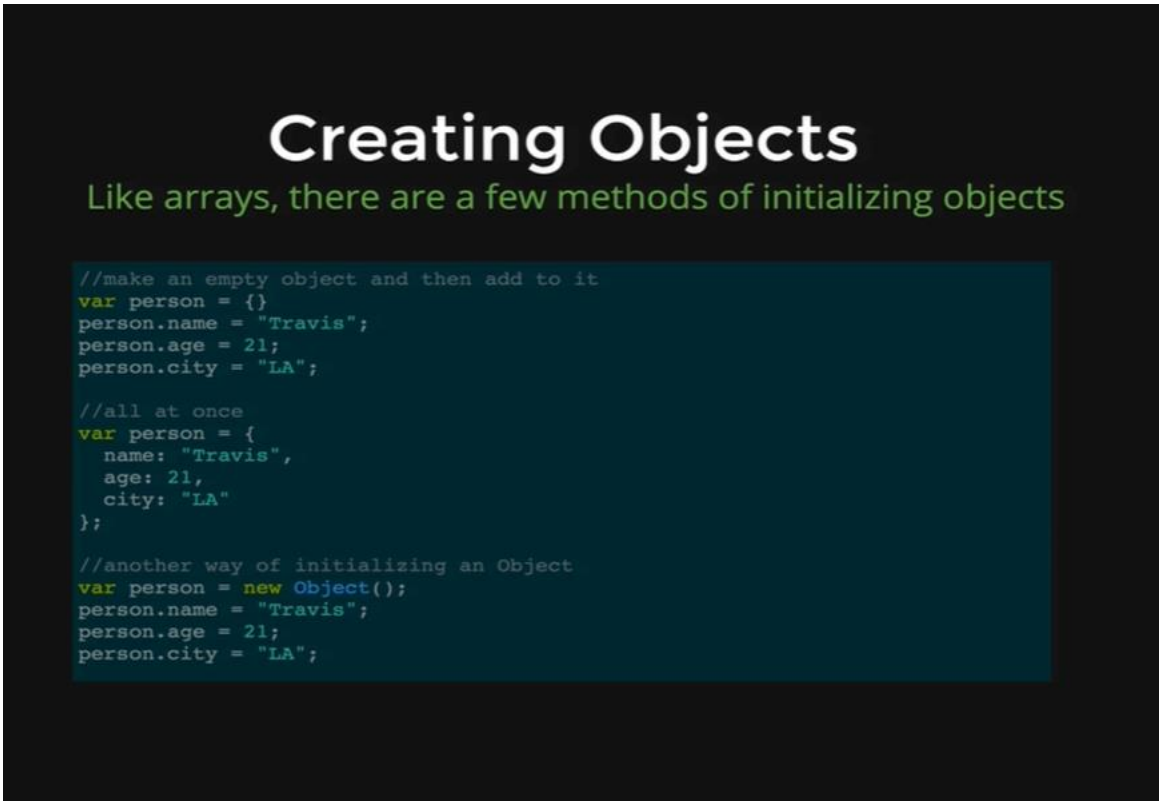
dog["name"]

➔ *"Rusty"*

dog["name"] = "Tater";

dog

➔ *Object {name: "Tater", breed: "Mutt", age: 5}*



There are few different ways of initializing objects, just like we saw arrays, so we can either make an empty object first using curly braces, *var person = {},* and then we can add the data after the set, one piece at a time,

```
var person = {}
person.name = "Travis";
person.age = 21;
 person.city = "LA".
```

We can also do it all at once, which we saw so far, it is called object literal notation,

```
var person = {
        name: "Travis",
        age = 21,
        city: "LA"
        };
```

The last thing that we will not see very often, until much later in JavaScript is that we can use *new* object which is a function just like new array that will make us a new object and return it to us as an empty object and then we can add person.name, person.age and person.city.

```
var person = new Object();
person.name = "Travis";
person.age = 21;
person.city = "LA";
```

```
var person = new Object();
person["name"] = "Travis";
person["age"] = 21;
person["city"] = "LA";
```

Another point we would like to make about an object, is that just like arrays, they can hold any sort of data. Our data can be numbers or strings or Booleans or an array or even another object as you see inside the *junkObject* we have another object *pet.* Just like arrays we can mix and match as much as we like.