2: Objects in JS

Objects

github repo Objects in javscript are a way of storing data/information in Javascript. It is similar to a harddrive, or a memory card. In javascript they can be used to describe properties of a specific element.

- Primitive data types String, number, bigint, boolean, undefined, symbol and null are all primitive data types. If a harddrive doesn't have data stored in it, it still is a harddrive. The same thing is for arrays and objects, regardless if they have information stored in them or not, they are still persistant in being arrays and objects

Arrays, much like most things in javascript, are also objects.

> Objects are data structures

- Objects Fundamentals This object has a key named dog, with the value "Bob". `javascript const obj = { dog: "Bob" }`

Object keys can have arrays, functions, strings, boolean, and numbers as values. `javascript const dog = { age: 2, breed: "Labrador", furColor: "Grey", size: "Medium" }`

With our object car, we can: * Retrieve data * Add new data(key value pairs) * Edit data/ changing existing key value pairs

Retrieving data

Given the car object we can access information two ways `javascript const dog = { age: 2, breed: "Labrador", furColor: "Grey", size: "Medium" }` Using dot notation

```
dog.breed; // -> "Labrador"
```

Or using square bracket notation

```
dog["breed"]; // -> "Labrador"
```

Dot notation is very specific, if you are referring to a variable, it will not work. If the key is set as a variable, you will have to use square bracket notation.

Adding/Editing data

Given the car object we can add information in two ways `javascript const dog = { age: 2, breed: "Labrador", furColor: "Grey", size: "Medium" }` Using dot notation

```
dog.age = 4; // -> "4"
```

Or using square bracket notation

```
dog["age"] = 4; // -> "4"
```

- We can't use dot notation, if we are referencing a variable. But we can with square bracket notation ```javascript let obj = { cat: "apple", dog: "lola" }; let thing = "dog";

obj[thing]; //=> "lola" //vs obj.thing; //=> null ```

Methods

We can also assign functions as values in objects. These are called methods `javascript const dog = { age: 2, breed: "Labrador", furColor: "Grey", size: "Medium", bark: function(){ console.log("Bark!") } } //to call the function dog.bark(); // -> "Bark!"`

Sharing

When we are creating variables as objects, we are just creating references `javascript let a = { thing: 5 }; let b = a;` Object a and b are both the reference to the same object. If we change the property of one, the other it will reflect on both ends. `javascript let a = { thing: 5 }; let b = a; b.thing = 6; console.log(b.thing) //6 console.log(a.thing) //6`

One way to prevent this from happening is by using spread operators `javascript let a = { thing: 5 }; let b = {...a}; b.thing = 6; console.log(b.thing) // 6 console.log(a.thing) //5`

We can't modify an object directly through its reference, but we can modify its values. We can change the content inside of the object. ```javascript const replace = function (ref) { ref = {}; // this code does *not* affect the object passed };

const update = function (ref) { ref.key = "newvalue"; // this code *does* affect the *contents* of the object };

const a = { key: "value" }; replace(a); // a still has its original value - it's unmodified update(a); // the _contents_of 'a' are changed ```

This

This is a special keyword within javascript.

This has different values depending on where it is used.

This wants to refer to an owner object.

If this is used in a method then it is happy and can refer to the object that the method exists in. `javascript const obj = { thing: 1, myFunc: function () { console.log(this); }, }; // this refers to obj` We can also call upon properties

```
const dog = {
  age: 2,
  breed: "Labrador",
  furColor: "Grey",
  size: "Medium",
  howOldIsThisDog: function(){
    console.log(this.age)
  }
}
dog.howOldIsThisDog() // -> 2
```

View on Compass | Leave Feedback