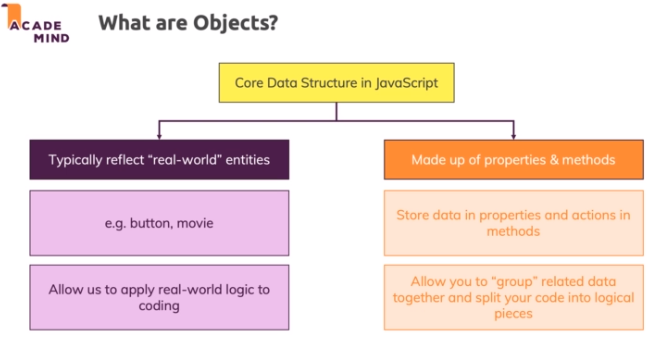
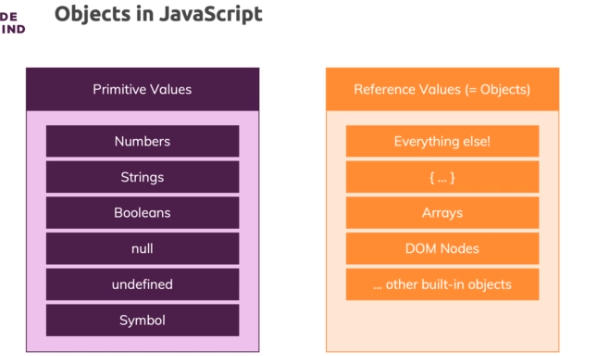
**Objects**





* Objects in Javascript are structures made up of properties
* Methods are key-value pairs but the value is a function that's tied to that object
* So you could say properties are variables in an object
* methods are functions in an object

**Objects & Primitive Values**

Objects are **reference values** - you learned that.

O**bjects are of course made up of primitive values**.

Here's an example:

    1. const complexPerson = {

    2.     name: 'Max',

    3.     hobbies: ['Sports', 'Cooking'],

    4.     address: {

    5.         street: 'Some Street 5',

    6.         stateId: 5,

    7.         country: 'Germany',

    8.         phone: {

    9.             number: 12 345 678 9,

    10.             isMobile: true

    11.         }

    12.     },

    13. };

Event though complexPerson has multiple nested reference values (nested arrays and objects), you end up with primitive values if you drill into the object.

**name** holds a string ('Max') => **Primitive value**

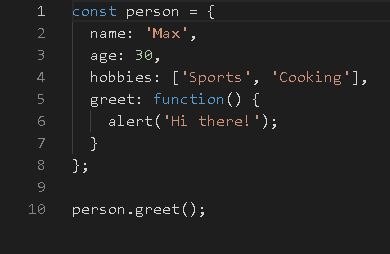
**hobbies** holds an array (i.e. a reference value) which is full of strings ('Sports', 'Cooking') => **Primitive values**

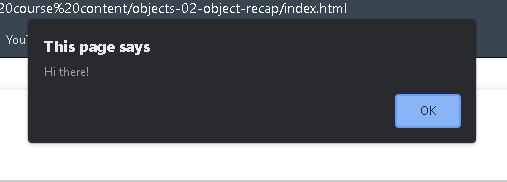
**address** holds an object which in turn holds a mixture of primitive values like 'Some Street 5' and nested objects (phone), but if you dive into phone, you find only numbers and booleans in there => **Primitive values**

So you could say: **Primitive** values are the **core building blocks** that hold your data, **objects (and arrays)** are helpful for **organizing** and **working** with that data.

**Objects - Recap**

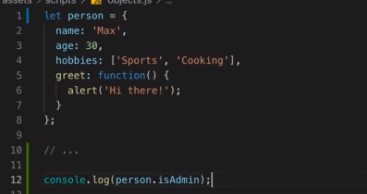
* if you store a function as a value on a certain key, then this is called a method

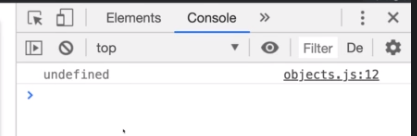




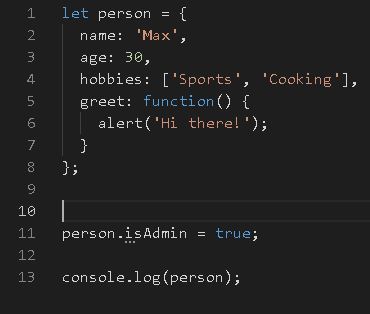
**Adding, Modifying & Deleting Properties**

* you can use the dot notation and simply as a property, which you haven't added before
* If you access a property on an object which doesn't exist, you're not getting an error.



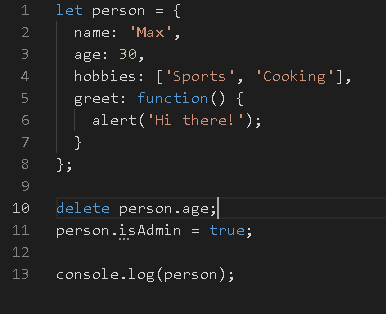


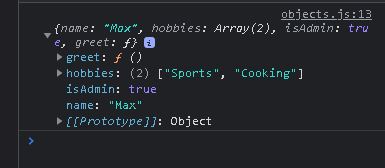
**To add an object**





**To delete an object**

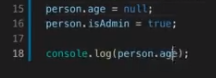


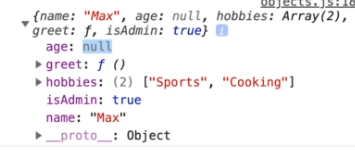


* it is a good rule to keep in mind that you should actually never assign undefined to any value

**To reset something**

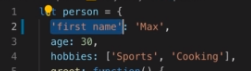
* null is the value we use if we want to reset something, if we want to clear something undefined as a value





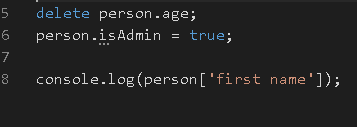
**Special Key Names &  Access**

* everything which you could use on a variable can be used as a key but not everything which you can use as a key name can be used as a variable



* Quotes must be included if key values are gonna be more than one word

**Square Bracket Property**



**Property Types**

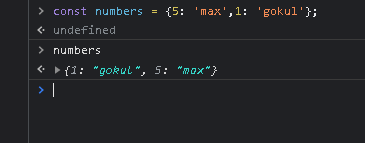
https://paper-attachments.dropbox.com/s_0338A6EE9D43AA217E7A9914827B18F51A6124746E8379AF260127D79397AF26_1632409309508_screenshot-www.udemy.com-2021-09-23-20-31-26-060.png

https://paper-attachments.dropbox.com/s_0338A6EE9D43AA217E7A9914827B18F51A6124746E8379AF260127D79397AF26_1632409322384_screenshot-www.udemy.com-2021-09-23-20-31-42-508.png

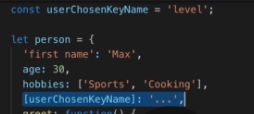
* Integers will get presumed as a string if used as a key value to store an object and it can be acessed by using an array at the console.log

**Property Order**

* numbers will be sorted, otherwise if you have string keys, the insertion order will be kept.
* That's how Javascript object properties are ordered by Javascript.

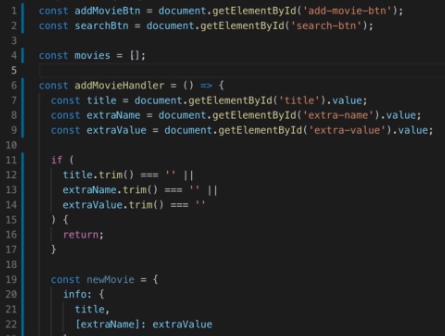


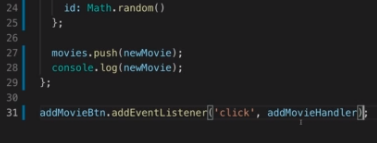
**Dynamic Property Access & Setting Properties Dynamically**

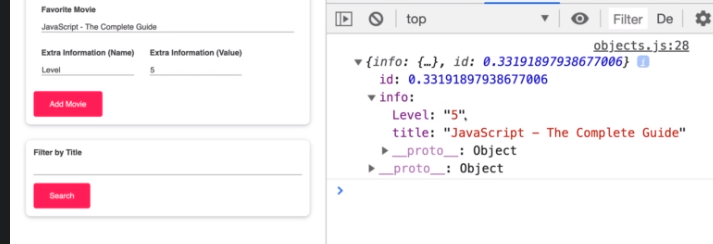


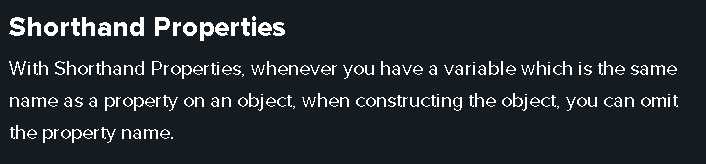
* Sometimes we’ve to search for a keyname that holds a specific value
* so we can access it again by using a square bracket also can modify the value present in it

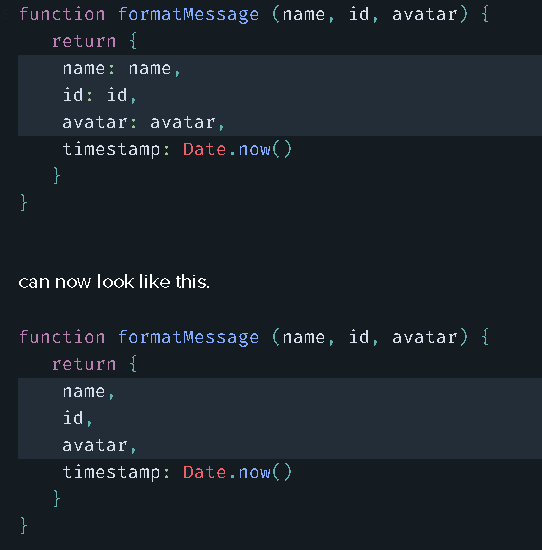
**Demo App**



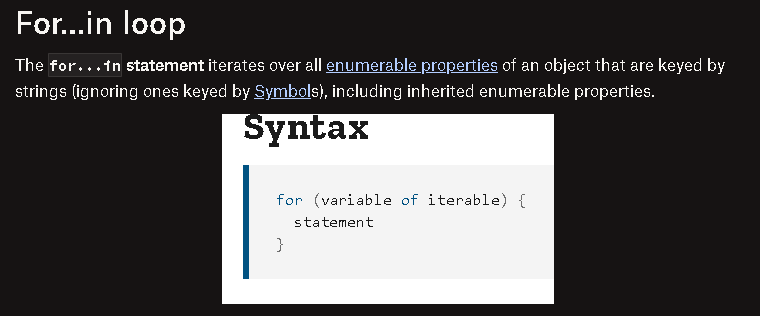


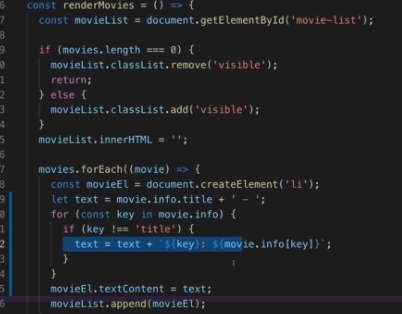


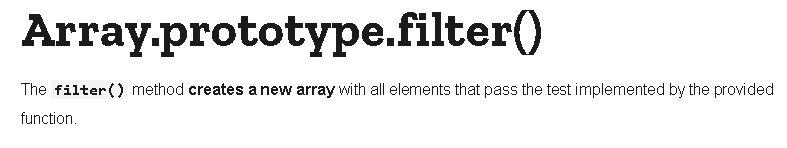


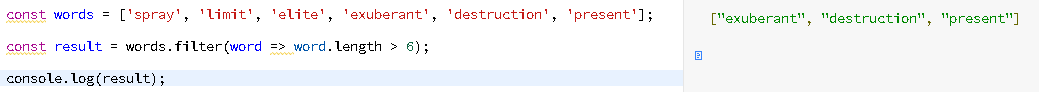


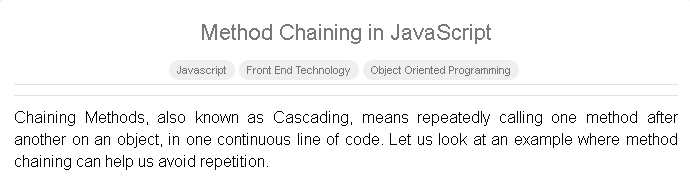
**for-in Loops & Outputting Dynamic Properties**



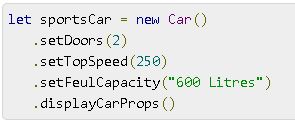




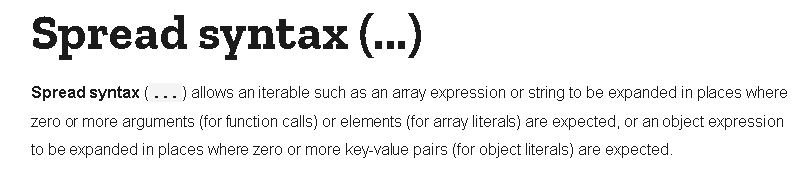


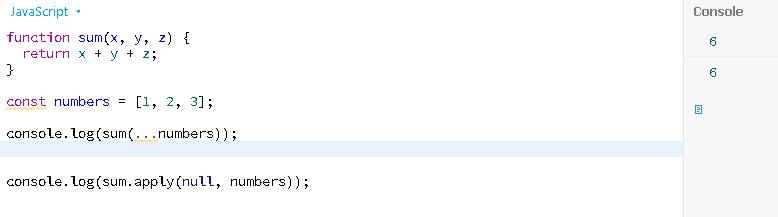


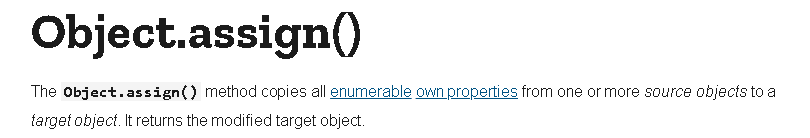


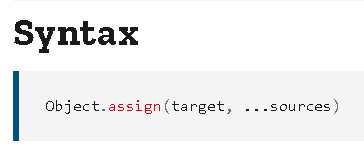


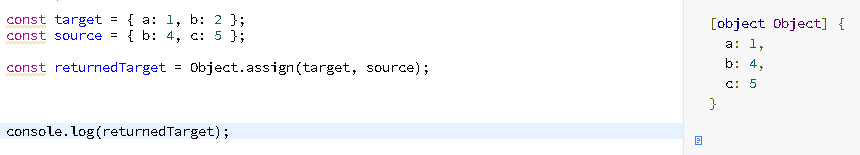
**The Object Spread Operator (...)**



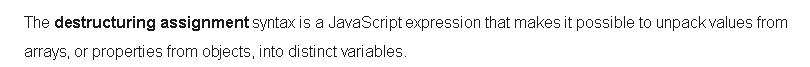


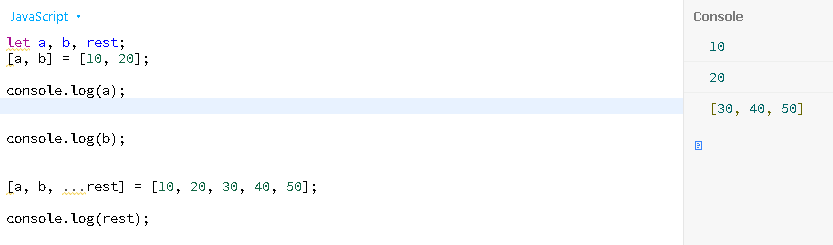


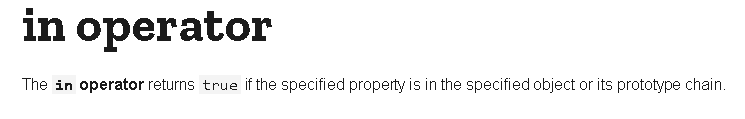


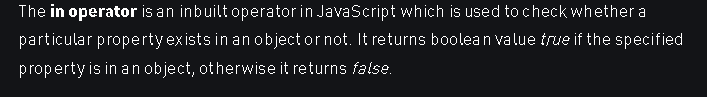


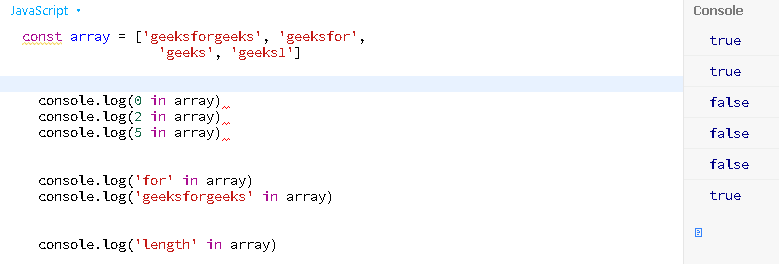
**Object Destructuring**

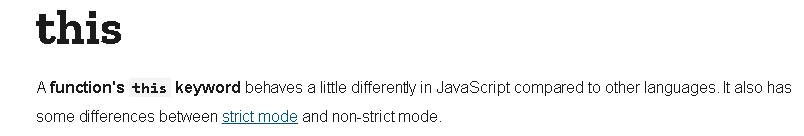


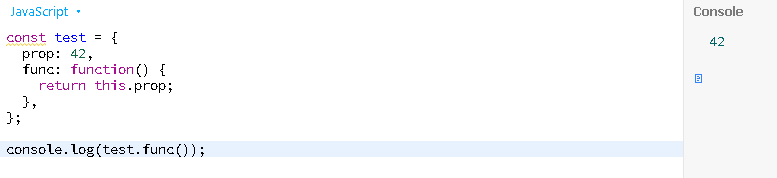




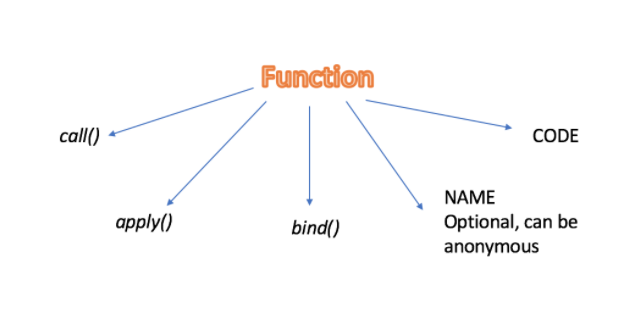


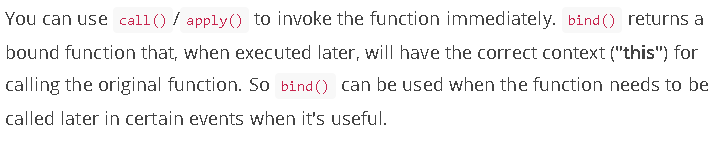


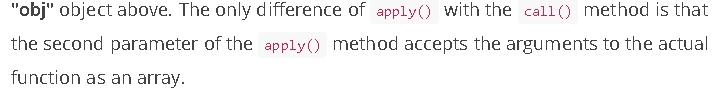


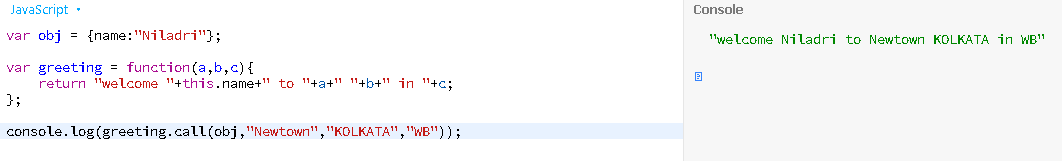


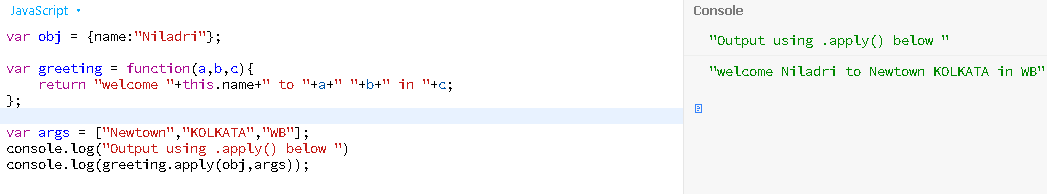
**call(), apply(), bind()**

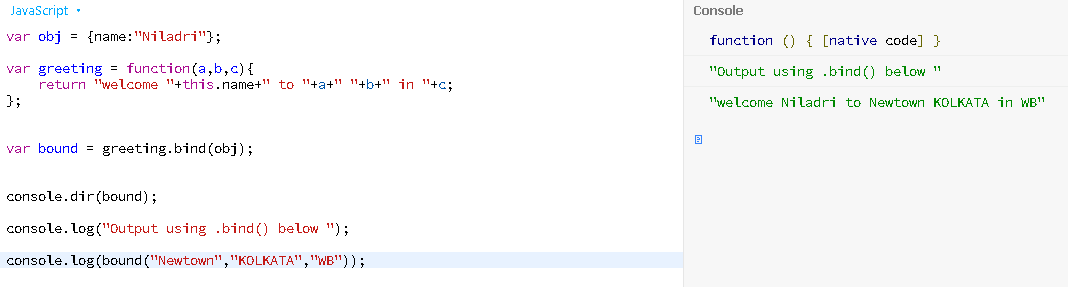












**"this" - Summary**

this refers to different things, depending on where it's used and how (if used in a function) a function is called.

Generally, this refers to the "thing" which called a function (if used inside of a function). That can be the global context, an object or some bound data/ object (e.g. when the browser binds this to the button that triggered a click event).

**1) this in Global Context (i.e. outside of any function)**

    1. function something() { ... }

    2.

    3. console.log(this); // logs global object (window in browser) - ALWAYS (also in strict mode)!

**2) this in a Function (non-Arrow) - Called in the global context**

    1. function something() {

    2.     console.log(this);

    3. }

    4.

    5. something(); // logs global object (window in browser) in non-strict mode, undefined in strict mode

**3) this in an Arrow-Function - Called in the global context**

    1. const something = () => {

    2.     console.log(this);

    3. }

    4.

    5. something(); // logs global object (window in browser) - ALWAYS (also in strict mode)!

**4) this in a Method (non-Arrow) - Called on an object**

    1. const person = {

    2.     name: 'Max',

    3.     greet: function() { // or use method shorthand: greet() { ... }

    4.         console.log(this.name);

    5.     }

    6. };

    7.

    8. person.greet(); // logs 'Max', "this" refers to the person object

**5) this in a Method (Arrow Function) - Called on an object**

    1. const person = {

    2.     name: 'Max',

    3.     greet: () => {

    4.         console.log(this.name);

    5.     }

    6. };

    7.

    8. person.greet(); // logs nothing (or some global name on window object), "this" refers to global (window) object, even in strict mode

this can refer to unexpected things if you call it on some other object, e.g.:

    1. const person = {

    2.     name: 'Max',

    3.     greet() {

    4.         console.log(this.name);

    5.     }

    6. };

    7.

    8. const anotherPerson = { name: 'Manuel' }; // does NOT have a built-in greet method!

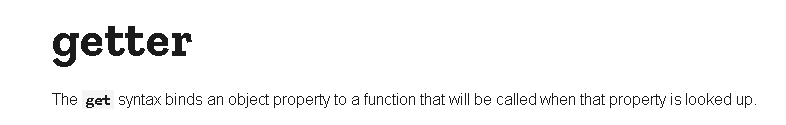
    9.

    10. anotherPerson.sayHi = person.greet; // greet is NOT called here, it's just assigned to a new property/ method on the "anotherPerson" object

    11.

    12. anotherPerson.sayHi(); // logs 'Manuel' because method is called on "anotherPerson" object => "this" refers to the "thing" which called it

If in doubt, a console.log(this); can always help you find out what this is referring to at the moment!



* sometimes we want to control how a property can be set
* how a value can be assigned or how you can get it, so how you can retrieve it.

