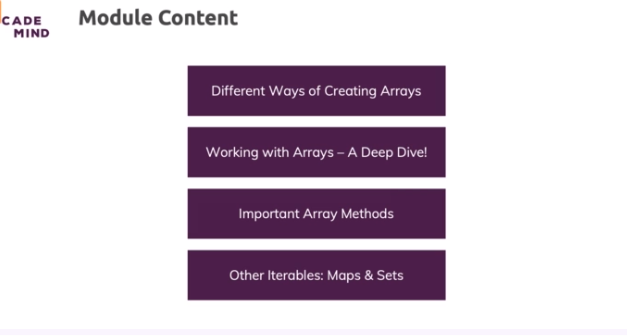
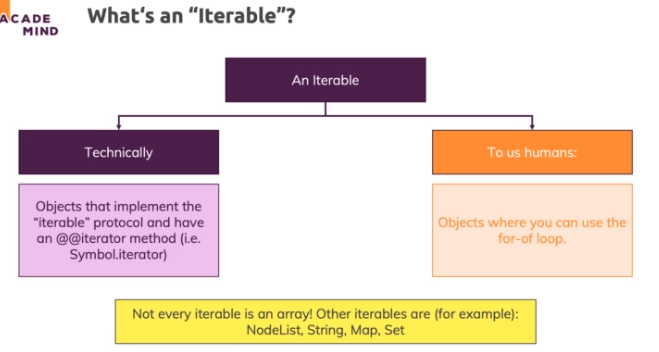
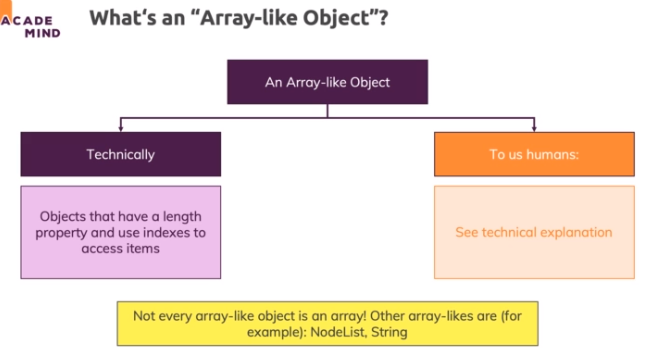
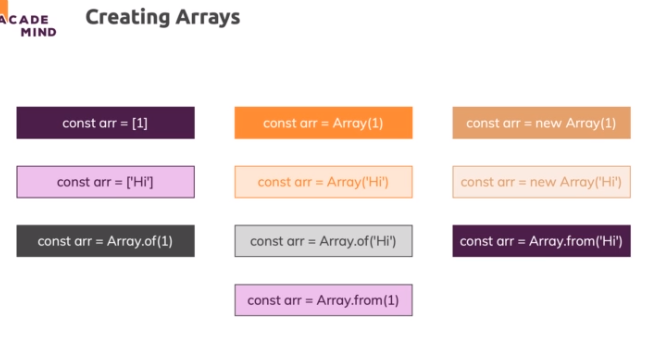
**Arrays & Iterables**

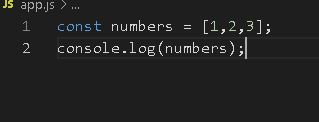


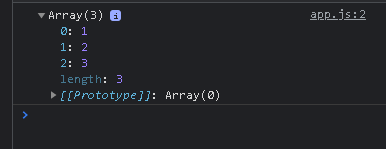
**What are "Iterables" and "Array-like Objects"?**

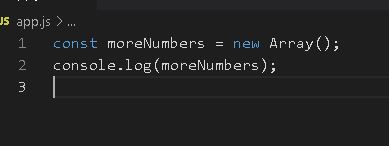


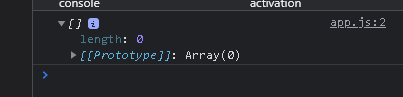


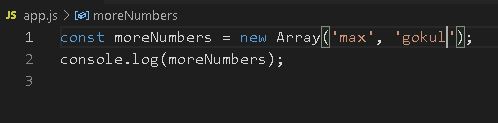


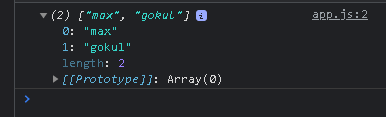


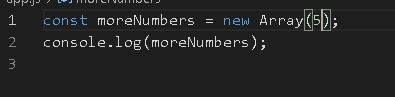


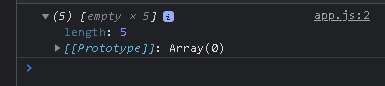


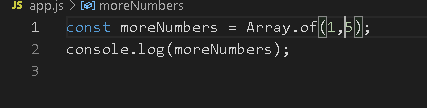






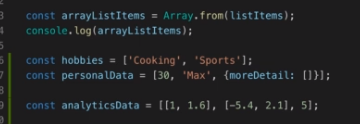






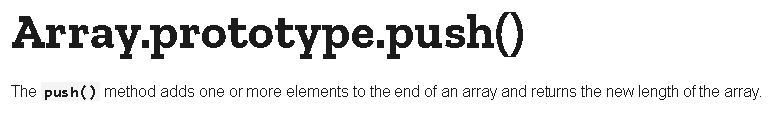
**Which Data Can You Store In Arrays?**

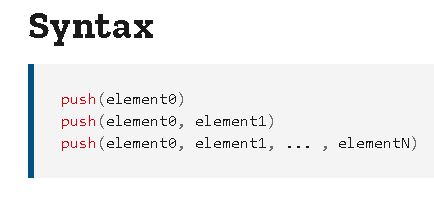
* You can store numbers as you see, you can store strings, you can store objects
* you can have arrays which are uniform or which have different types of data in there.
* You can also have nested arrays or multi-dimensional arrays
* you can have a loop in a loop

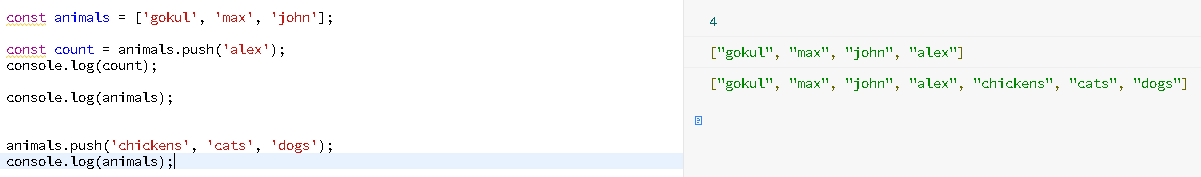


* arrays are index based, so you can access data by index
* Index based, index starts at zero,
* the length of an array is max index plus 1 though.

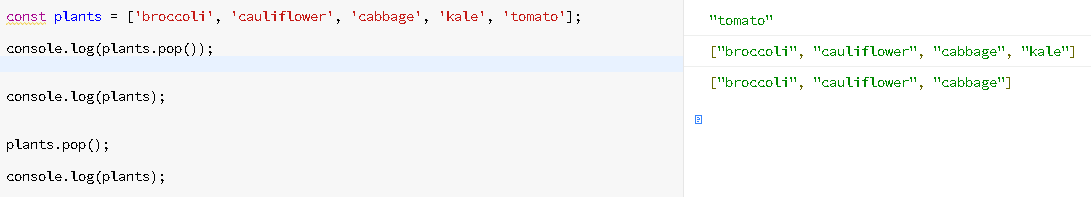
**push(), pop(), unshift(), shift() - Adding & Removing Elements**



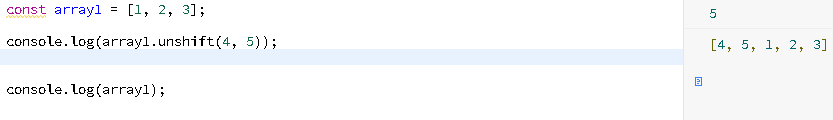


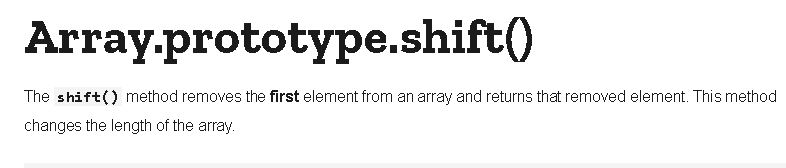


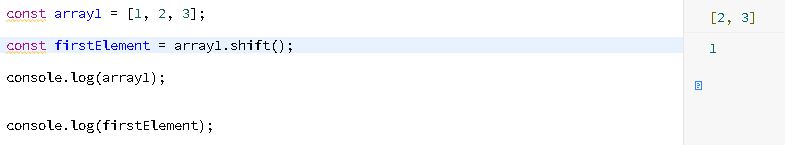




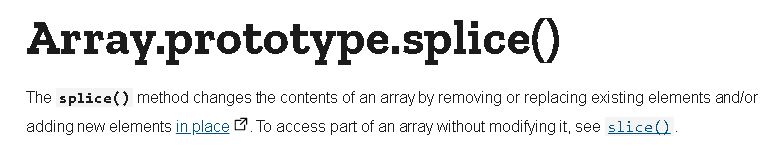


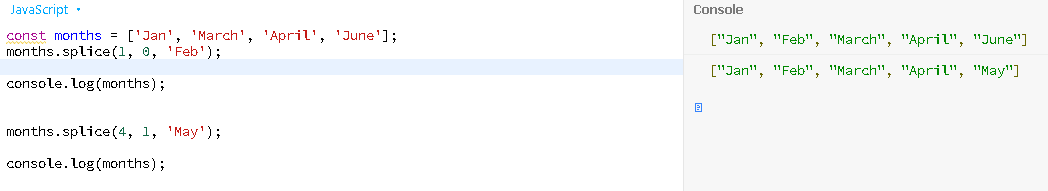






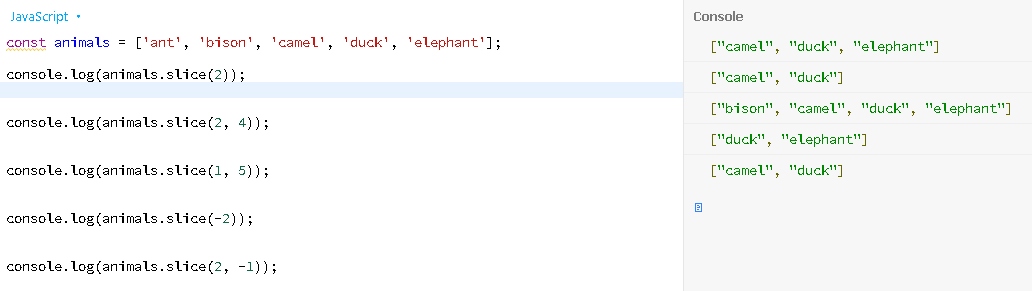
**The splice() Method**

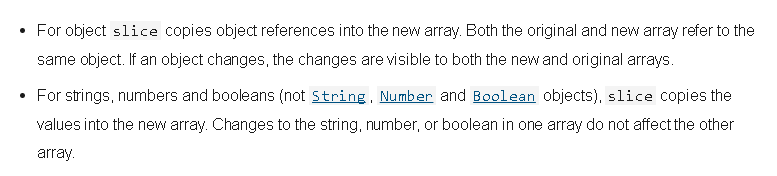




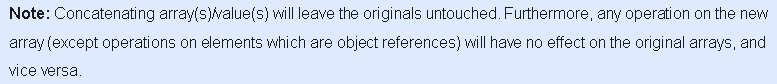
**Selecting Ranges & Creating Copies with slice()**

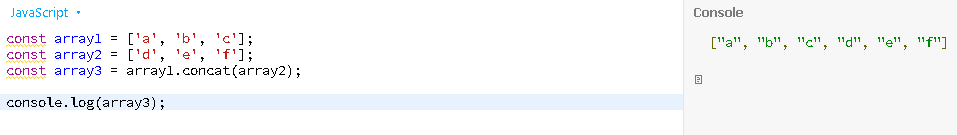






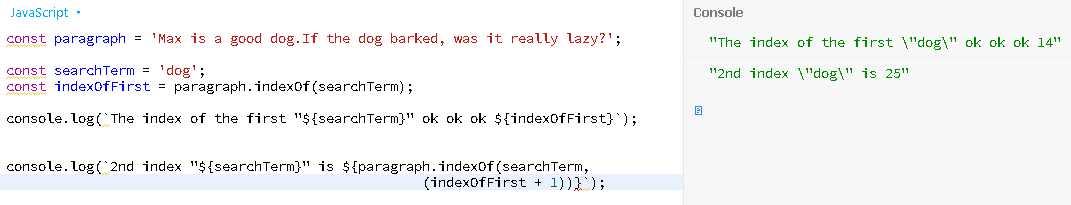


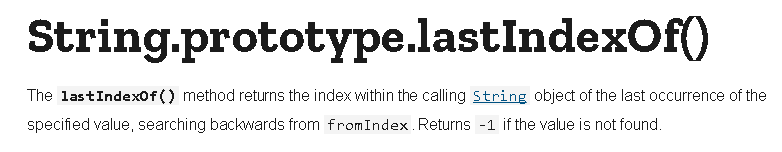


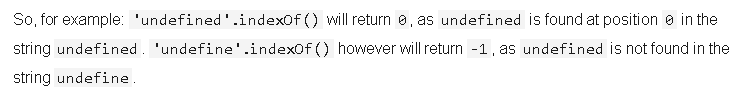


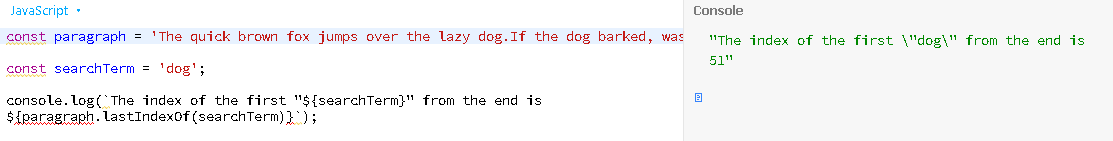
**Retrieving Indexes with indexOf() /& lastIndexOf()**





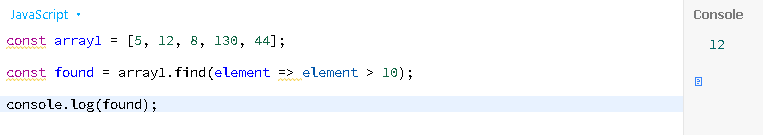


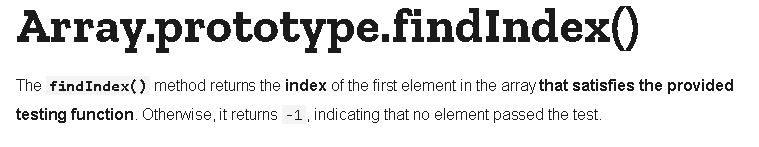


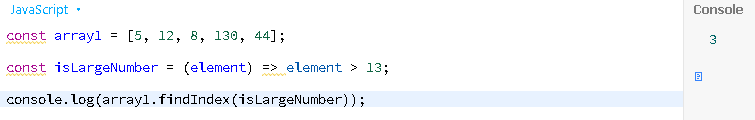


**Finding Stuff: find() and findIndex()**

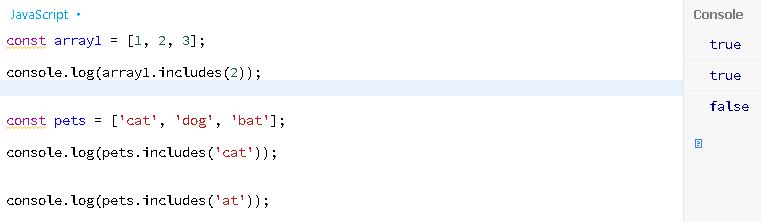






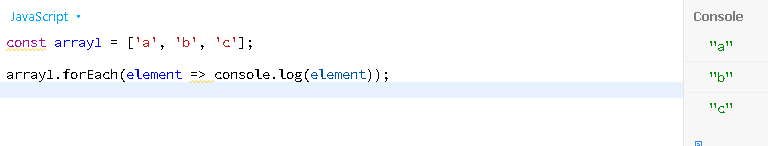




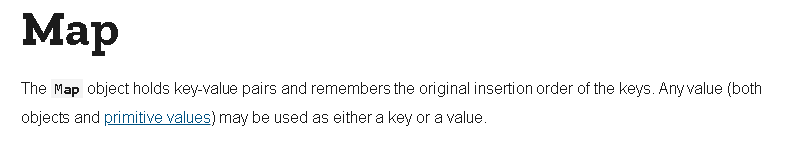


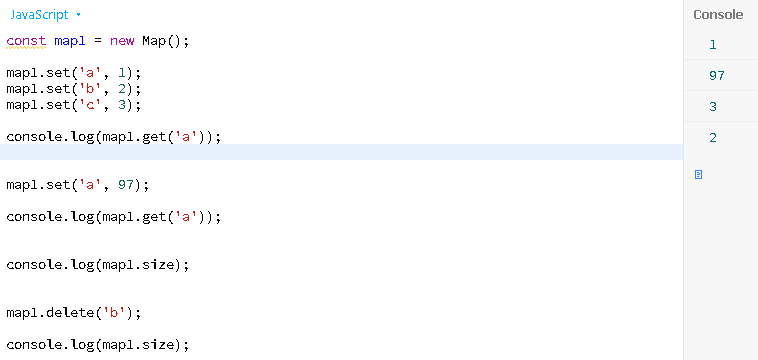
**The forEach() Method**



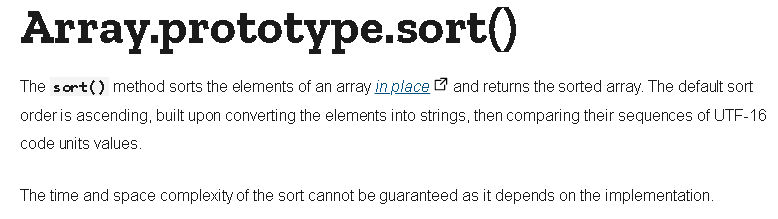


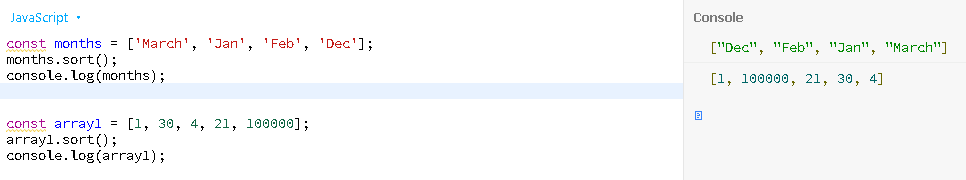
**Transforming Data with map()**



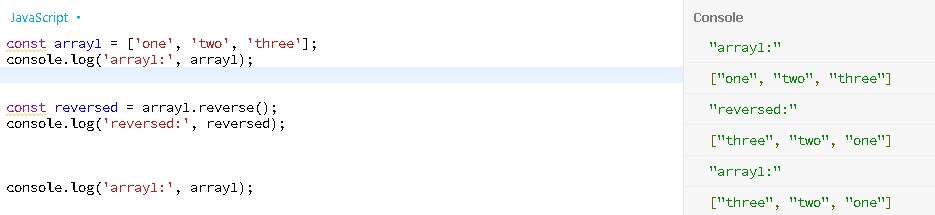


**sort()ing and reverse()ing**

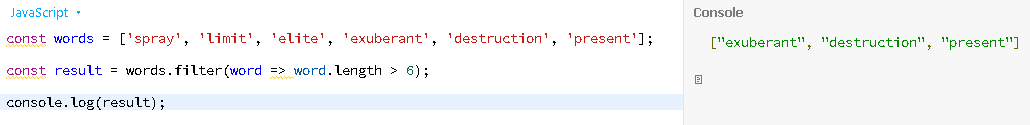


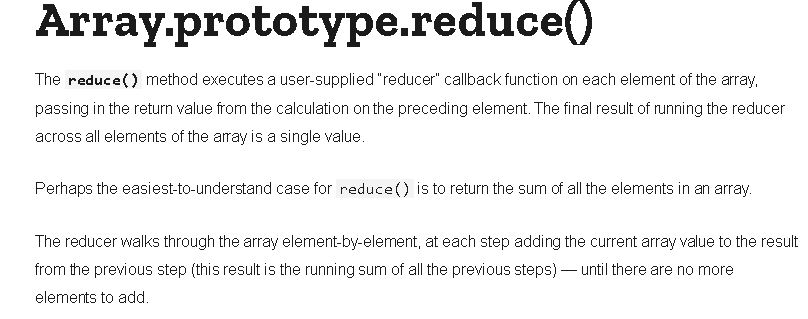


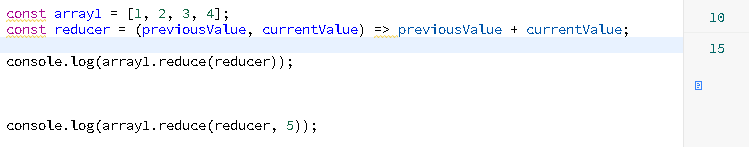












**Chaining Methods in JavaScript**

With all these useful array methods you learned about, it's important to understand how you can combine them. Let's take map() and reduce() as an example:

    1. const originalArray = [{price: 10.99}, {price: 5.99}, {price: 29.99}];

    2. const transformedArray = originalArray.map(obj => obj.price); // produces [10.99, 5.99, 29.99]

    3. const sum = transformedArray.reduce((sumVal, curVal) => sumVal + curVal, 0); // => 46.97

Of course, you could skip the map step and just add the extraction logic to reduce():

    1. const originalArray = [{price: 10.99}, {price: 5.99}, {price: 29.99}];

    2. const sum = originalArray.reduce((sumVal, curVal) => sumVal + curVal.price, 0); // => 46.97

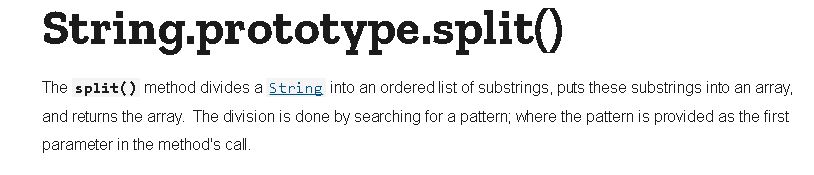
But let's say you have a more complex extraction logic and hence want to split this into multiple method calls. Or you have a re-usable map function which you want to be able to use in different places of your app. Then you can still write the initial example in a more concise way if you **leverage method chaining**:

    1. const originalArray = [{price: 10.99}, {price: 5.99}, {price: 29.99}];

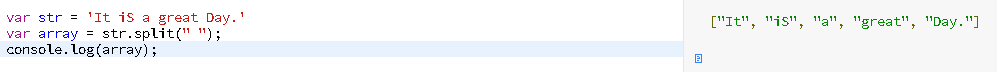
    2. const sum = originalArray.map(obj => obj.price)

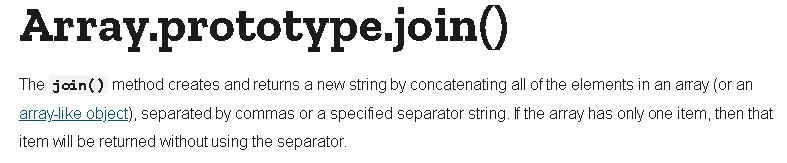
    3.     .reduce((sumVal, curVal) => sumVal + curVal, 0); // => 46.97

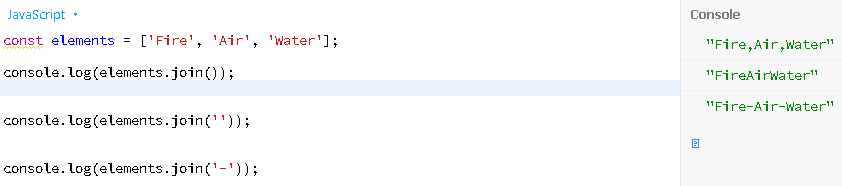
We call .reduce() directly on the result of map() (which produces an array, that's why this is possible). Hence we can avoid storing the mapped array in a separate constant or variable that we might not need in any other place.

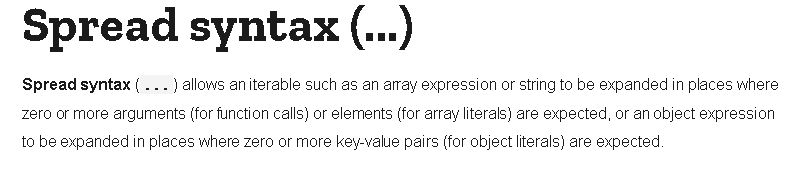


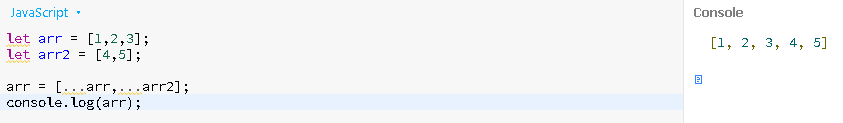
https://paper-attachments.dropbox.com/s_B811018CBBBCE4D46FA98506EC59066A443D8A20F6FB89871C7CAC57F62BCE92_1632494574736_image_2021-09-24_201254.png

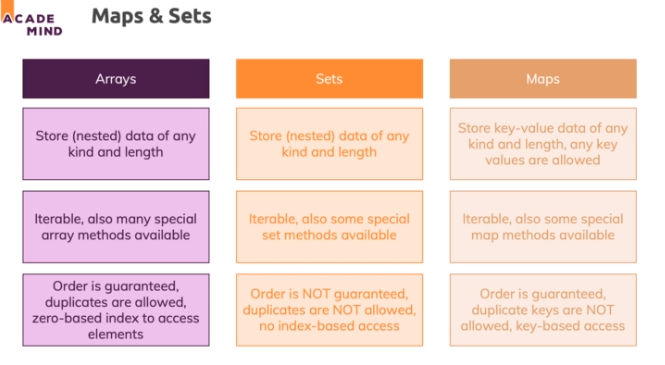












* An array can be used as a key value in maps

**Working with Sets**

* Sets data structures which help you manage unique values and in some cases, that can be useful.
* let's say ID which are already in use by logged in users, then you could use a set to keep track

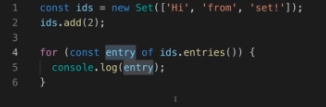
of these IDs and you might want to use a set instead of an array because you want to ensure that a single ID can't be part of the set more than once

* IDs should be unique and therefore you might want to store them in a data structure where you can't have any duplicates

https://paper-attachments.dropbox.com/s_B811018CBBBCE4D46FA98506EC59066A443D8A20F6FB89871C7CAC57F62BCE92_1632496266999_screenshot-www.udemy.com-2021-09-24-20-40-44-750.png

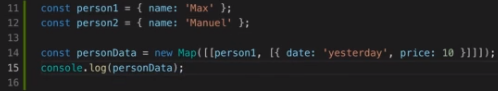
https://paper-attachments.dropbox.com/s_B811018CBBBCE4D46FA98506EC59066A443D8A20F6FB89871C7CAC57F62BCE92_1632496270710_screenshot-www.udemy.com-2021-09-24-20-41-01-166.png

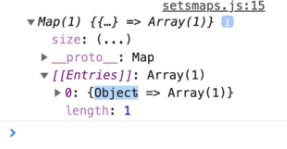
* entries is a method which you can execute and it returns, as you can see, an iterable



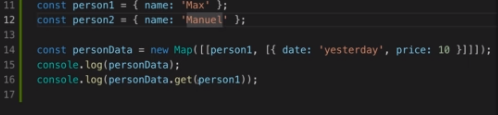
**Working with Maps**

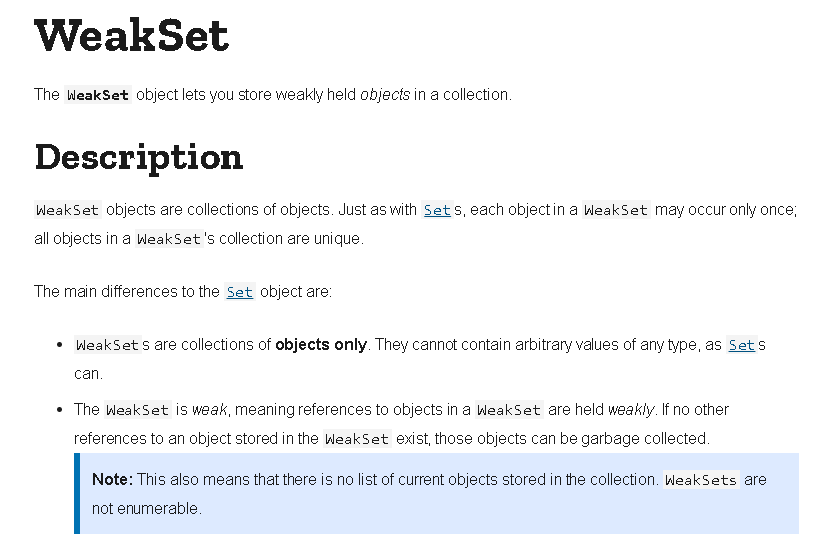
* map constructor here can be initialized with an array of arrays and it's an array of array because each array in that array is one key-value pair

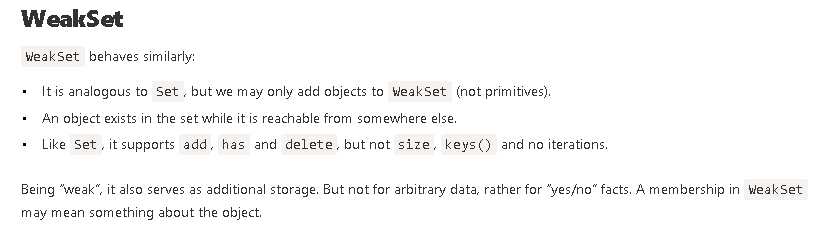




* we can use that map in the part of the application where we need that extra information







* it can only store objects so that it can actually clear these objects for you, release them to garbage collection

