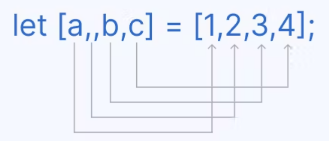
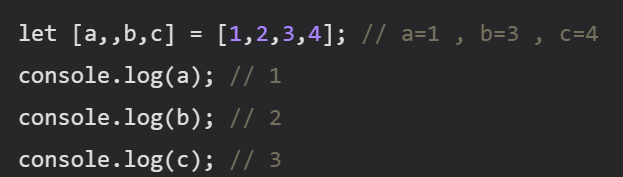
Data Structures, Modern Operators and Strings

There are 4 data structures that are in-built in JavaScript. We will cover all 4 data structures in depth. Even there are non-built-in data structures. But those data structures are not within the scope of this blog. If you are a beginner in JavaScript do check out our in-depth JavaScript series JavaScript Series. We will mostly focus on practical learning instead of theoretical learning. We will include only the required theory for the practical.

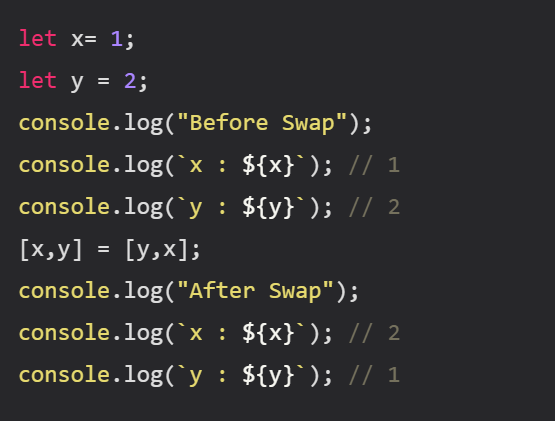
**Destructuring Arrays**

The destructuring in JavaScript makes it possible to unpack arrays into distinct variables.

While destructuring we use the same type of brace on both sides. if you take a look at the above image we destructured an array in 3 variables called a,b, and c. After b we have left blank space which will skip that term. Let's try to understand with an example.



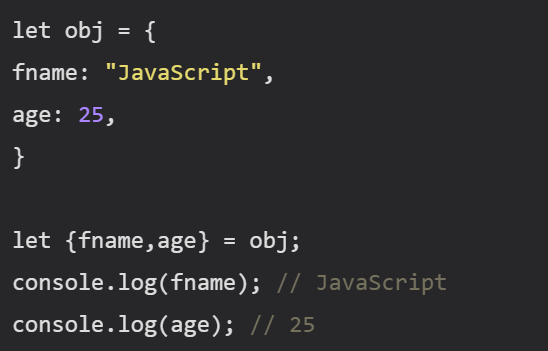
Now with the help of destructuring, we can swap 2 variables' values.If we have x=1 and y=2 after swaping it becomes x=2 and y=1.



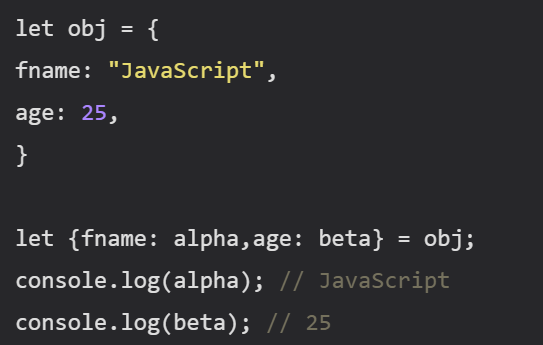
**Destructuring Objects**

Destructuring objects is similar to arrays but there are a few changes we need to do. We have to mention the same name of the key mentioned in an object. We can change the name but it needs some changes. we will try to understand both approaches with help of examples.

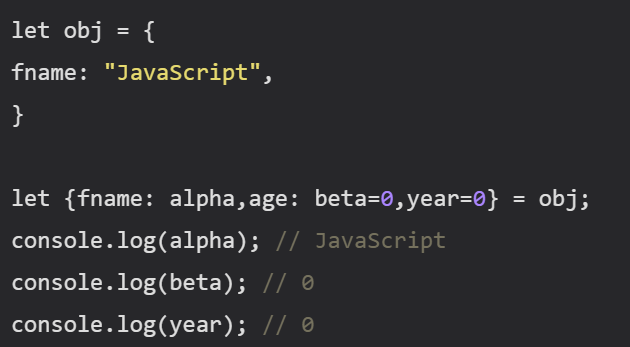
Without changing the name mentioned in the object



changing name mentioned in object

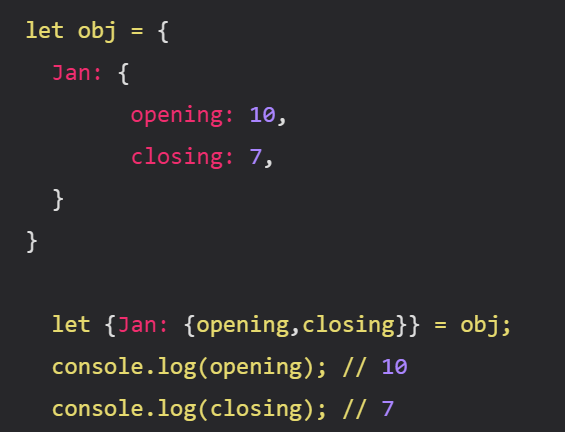


We can also give default values while destructuring object. They are useful when there is no matching property of object



**Nested Destructuring**

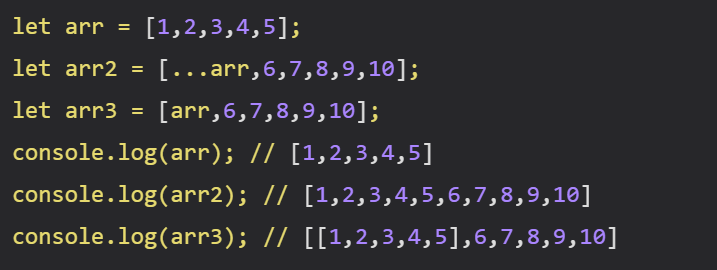
Suppose there is an object which has one more object in it. How will you destructure that object. It's simple we can do nested destructuring. Refer below example.



**Spread Operator**

Spread syntax (...) allows an iterable, such as an array or string, to be expanded in places where zero or more arguments (for function calls) or elements (for array literals) are expected.

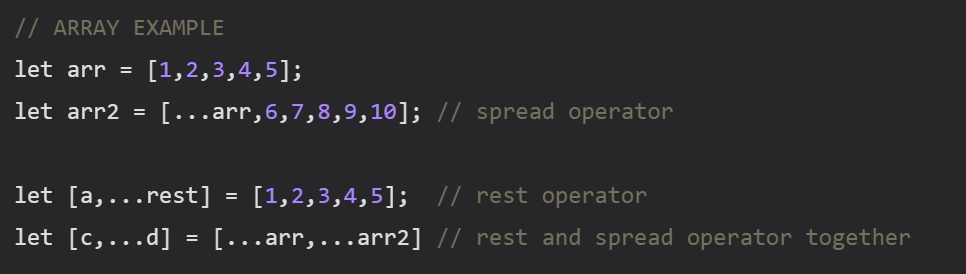
Let's understand it by example.



In the above example, arr is copied and pasted in arr2 element by element. But if we directly mentioned arr in arr2 it would be a nested array instead of the array.

**Rest Operator**

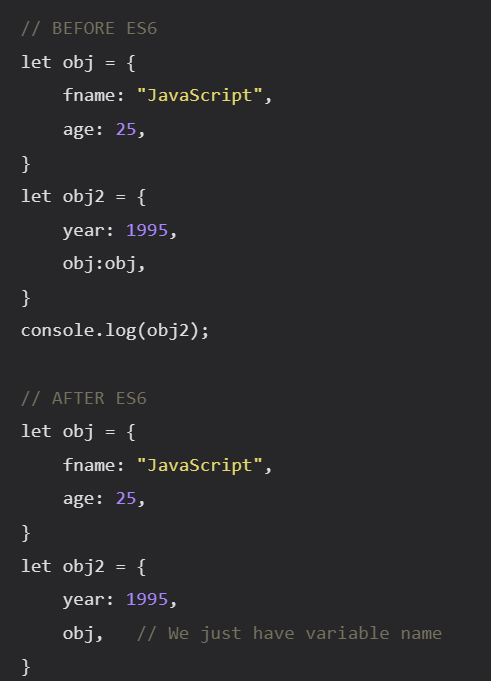
The rest operator is the same as spread operator. But when we use (...) on the left-hand side of assignment operator(=) it is the rest operator and when we use (...) on the right and side of assignment operator(=) it is spread operator. Let's understand with an example.



**Enhanced Object Literals**

In ES6, there are some advanced object literals introduced to make working with objects easier.

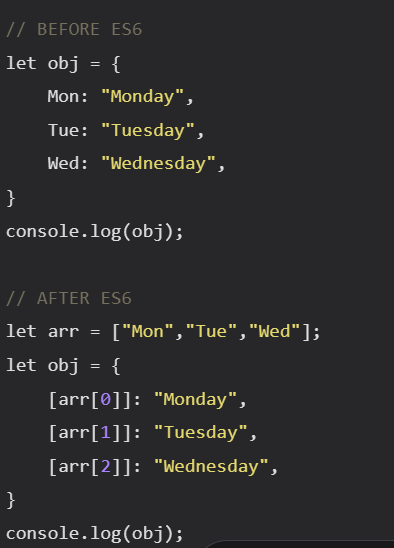
Before ES6 when we wanted to copy one object into another object, we have to give a key: value pair. But in ES6 we just have to give a variable name. It's better to understand with examples.



We used to write function in object with key: value pair, instead of value we use to write function. But new way to write function in object was introduced. We just have to write function name and braces. Refer below example to understand better.

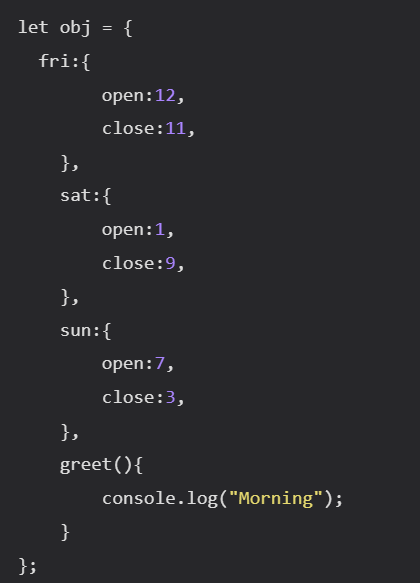


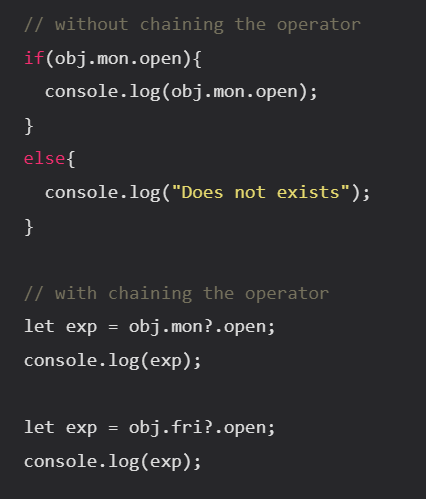
We had to write key names before ES6, but after ES6 we can take array values directly as a key name in an object. Refer to the below example to understand better.



**Optional Chaining**

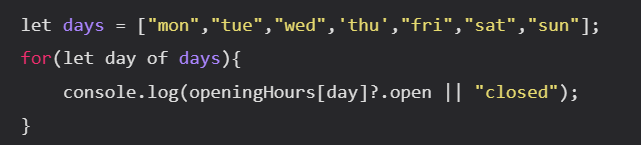
The optional chaining operator (?.) enables you to read the value of a property located deep within a chain of connected objects without having to check that each reference in the chain is valid. Let's take an example to understand it better.



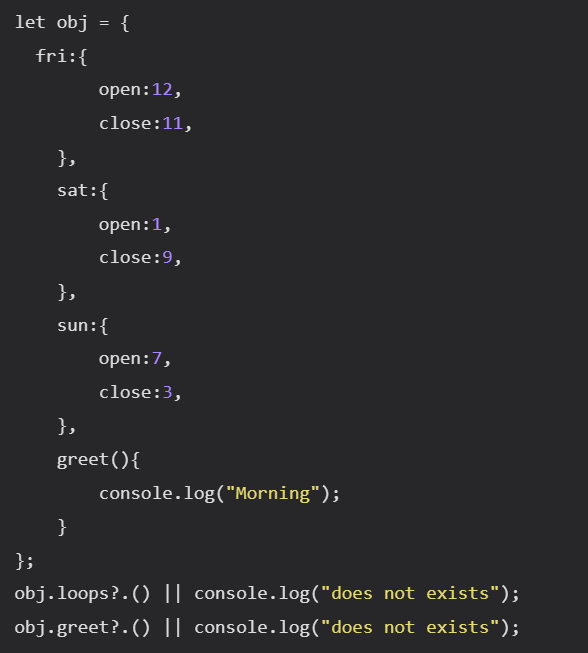


Now in the above example, if everything before optional chaining (?.) exists then only it will print, or else it will give undefined.

optional chaining with array

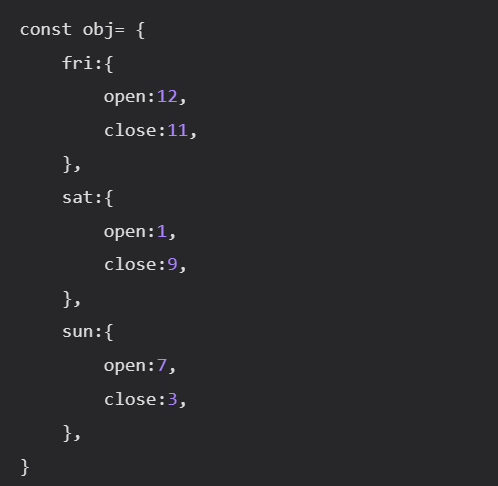


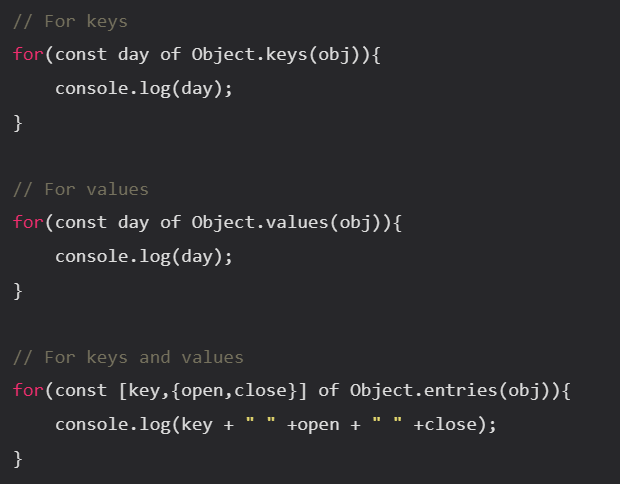
optional chaining on methods



Looping objects

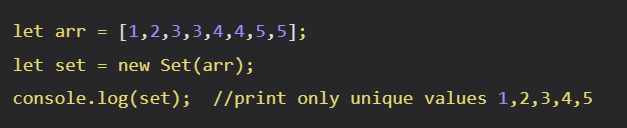
Objects are not iterable still we can loop them in an indirect way. There is an inbuild object method we can use to loop through objects.





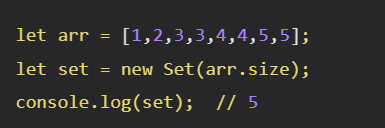
**Sets**

The set is a collection of unique elements. There is no duplication in a set. We can pass iterable to set. The most common iterable passed to a set is an array. Let's try it using an example.



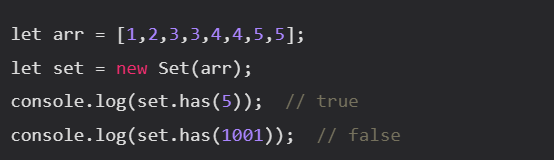
**size property**

It returns size of set.



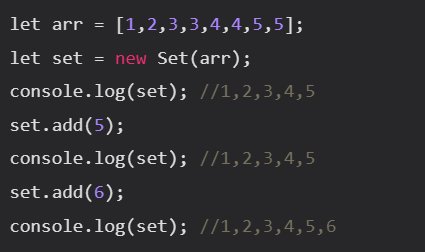
**has property**

It return true if set contain value given by user. It is similar to includes property of array.



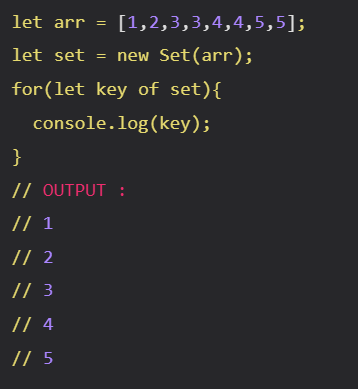
**add property**

We can add a new value to the set using add. If the value we are adding already exists in the set it won't add. But the value we passed doesn't exist it will add at end of a set.



**looping through set**

It is very easy to loop through set. We just have to use for loop like any other data structure.

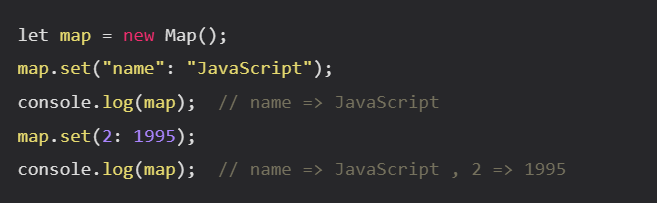


**Maps**

The map is a data structure that helps in storing the data in the form of pairs. The pair consists of a unique key and a value mapped to the key. It helps prevent duplicity.

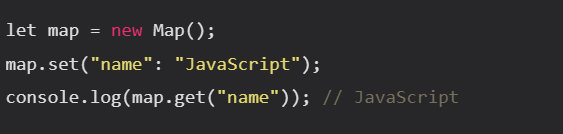
A Map is ordered and iterable, whereas an object is not an ordered and not an iterable property.

We have to use set property to push elements into map. We can use any data type as key in maps.



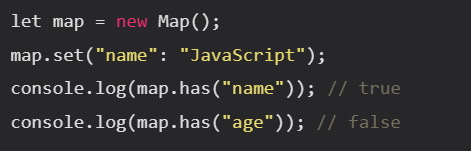
**get property**

We can pass key name to get value of that key using get property.



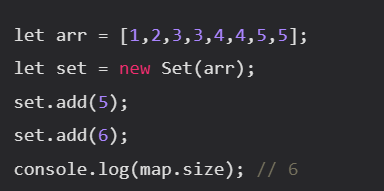
**has property**

It returns a Boolean value. It checks if a key is present given by the user and then returns true or else false.

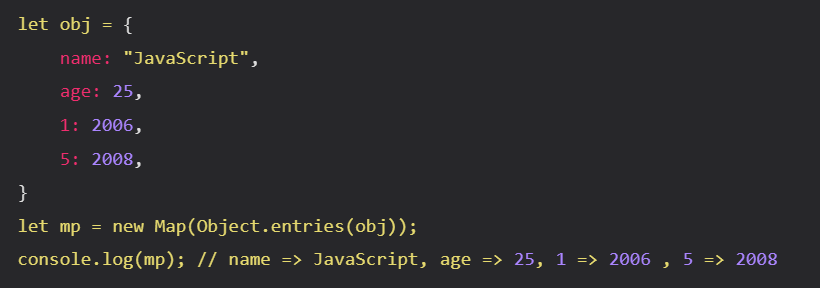


size property

It returns count of pair of key,value present in map.



**object to map**

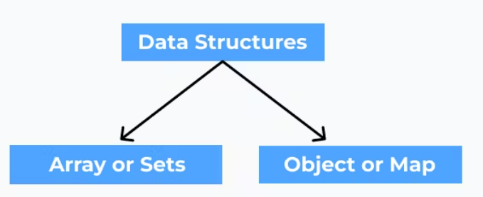
We can convert object to map. But for that we need to get key and value pair from objects which can be done using Object.entries().

**Iterate through map**

We can use for of loop to iterate through map.



**When to use which data structure**

There are two types of data structure. There are different use cases for different data structures. 

* Simple list
* key-value pair
* <https://kedar98.hashnode.dev/data-structures-and-modern-operators#heading-get-property>