4

Array Destructing and Spread Operators

Objectives

After completing this lesson, you should be able to do the following:

- Let and Const
- Arrays
- Destructuring
- Spread Operator
- Rest Parameter

Initialization of Variable

- It is the process to store the value in the variable. A variable can be initialized at any time before its use.
- The ES6 syntax used the keyword **var** to declare a variable and also the variables are declared by:
  - Using **let**.
  - Using **const**.

Let and Const

- Let
  - Any variable which is declared by using the **let** keyword is assigned the block scope. **Block scope** is nothing but a section where the **let** variable gets declared whether it is a **function{}**, a **block{}**, or a **global** **(script)**.

- Const:
  - ES6 gives a new way to declare a constant using the **const** keyword. The keyword **const** creates a read-only reference to the value. There are some properties of the **const** that are as follows:
  - **Properties:**
    - It cannot be reassigned a value.
    - It is block scoped.
    - A constant cannot be re-declared.
    - Constants must be initialized at the time of declaration.

TS

Typescript Arrays

Arrays

➢ Arrays in Typescript is Same as Arrays in JavaScript

Creating Arrays

- Initializing an array:

```
var myNewEmptyArray = [];

var numbers = [1, 2, 3, 4, 'five'];
```

- Creating an array with `new`:

```
var myNewEmptyArray = new Array();

var myNewNonEmptyArray = new Array(15);

var numbers = new Array(1, 2, 3, 4, 'five');
```

Accessing Array Elements

➢ Length of an array:

```
var c = ["blue","red","green","purple"];
c.length;  ← 4
c[20] = "white";
c.length;  ← 21
```

➢ Iterating arrays:

```
for (var i = 0; i < c.length; i += 1)
{
    console.log(c[i]);
}
```

```
for (var i in c)
{
    console.log(c[i]);
}
```

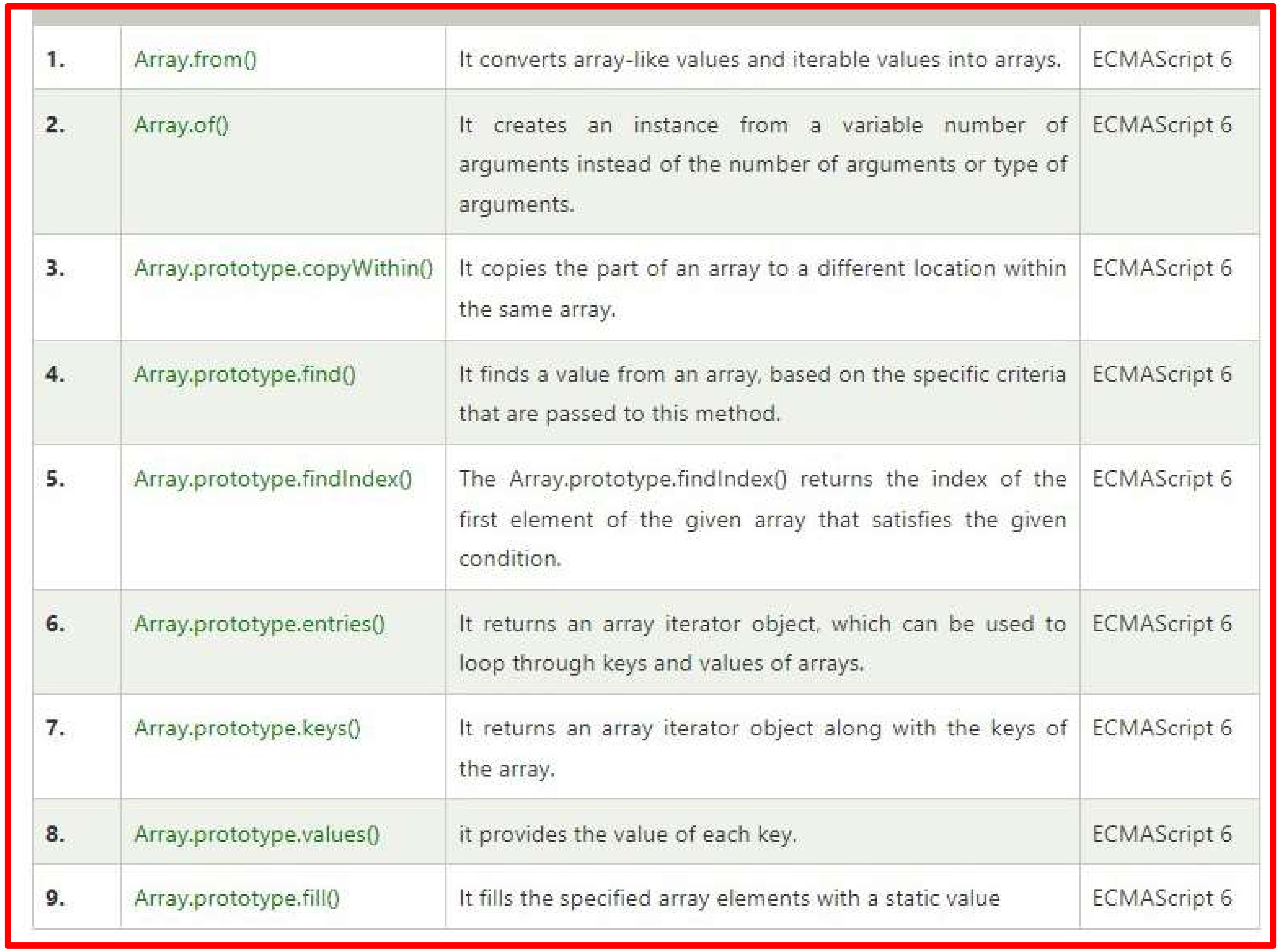
Multidimensional Arrays

- Creating multidimensional arrays:

```
var myArr = new Array(10);
for (i = 0; i < 10; i++) {
    myArr[i] = new Array(10);
    for (j = 0; j < 10; j++) {
      myArr[i][j] = "[" + i + j + "]";
  }
}
```

```
var tic_tac_toe = [
[ 0, 0,'X'],
['X',0,'X'],
['X',0, 0]
];
```

ES6 Array methods

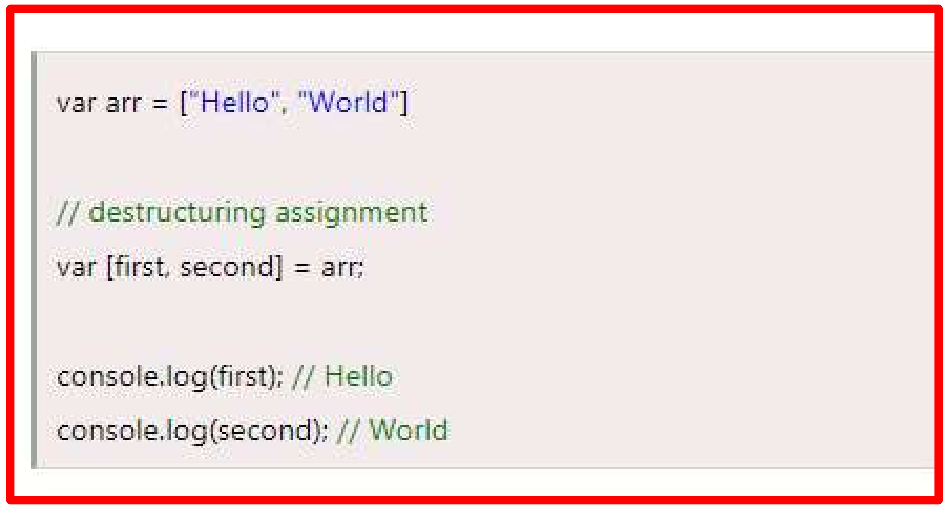


1.	Array.from()	It converts array-like values and iterable values into arrays.	ECMAScript 6
2.	Array.of()	It creates an instance from a variable number of arguments instead of the number of arguments or type of arguments.	ECMAScript 6
3.	Array.prototype.copyWithin()	It copies the part of an array to a different location within the same array.	ECMAScript 6
4.	Array.prototype.find()	It finds a value from an array, based on the specific criteria that are passed to this method.	ECMAScript 6
5.	Array.prototype.findIndex()	The Array.prototype.findIndex() returns the index of the first element of the given array that satisfies the given condition.	ECMAScript 6
6.	Array.prototype.entries()	It returns an array iterator object, which can be used to loop through keys and values of arrays.	ECMAScript 6
7.	Array.prototype.keys()	It returns an array iterator object along with the keys of the array.	ECMAScript 6
8.	Array.prototype.values()	it provides the value of each key.	ECMAScript 6
9.	Array.prototype.fill()	It fills the specified array elements with a static value	ECMAScript 6

Array destructuring

- Destructuring means to break down a complex structure into simpler parts. With the syntax of destructuring, you can extract smaller fragments from objects and arrays. It can be used for assignments and declaration of a variable.
- Destructuring is an efficient way to extract multiple values from data that is stored in arrays or objects. When destructuring an array, we use their positions (or index) in an assignment.

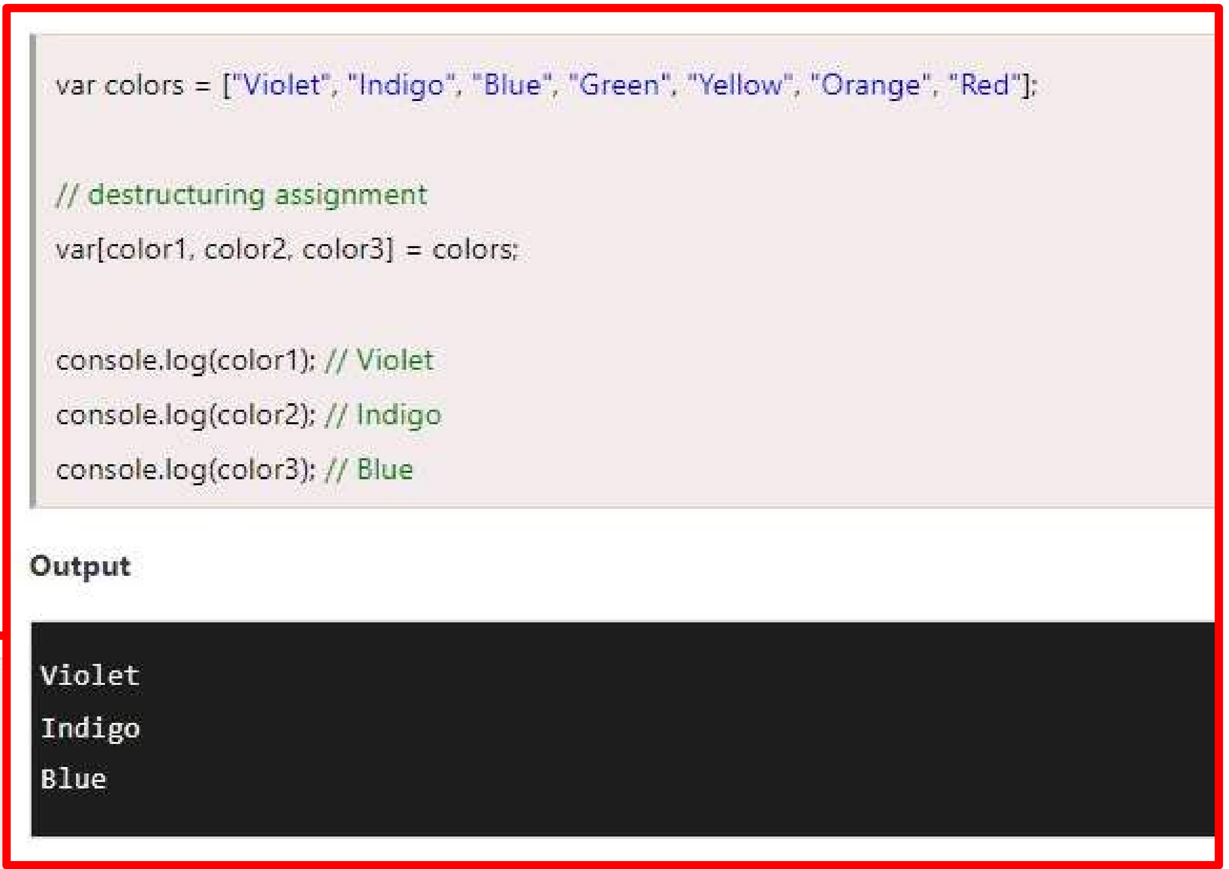
Sample Illustrations



```
var arr = ["Hello", "World"]

// destructuring assignment
var [first, second] = arr;

console.log(first); // Hello
console.log(second); // World
```



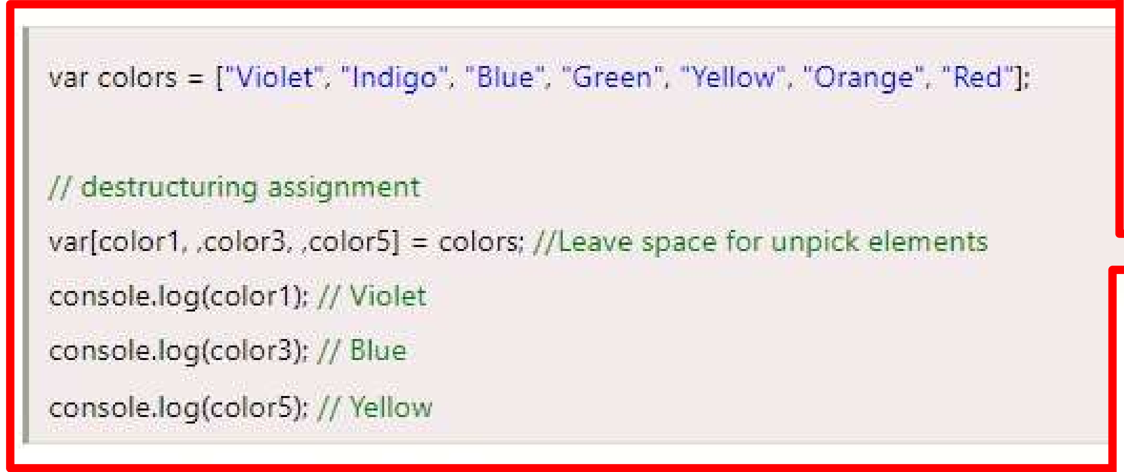
```
var colors = ["Violet", "Indigo", "Blue", "Green", "Yellow", "Orange", "Red"];

// destructuring assignment
var[color1, color2, color3] = colors;

console.log(color1); // Violet
console.log(color2); // Indigo
console.log(color3); // Blue
```

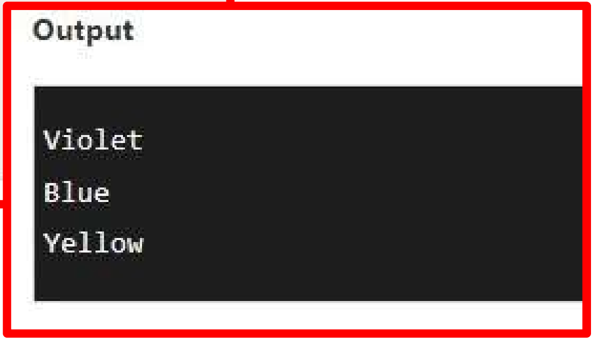
Output

```
Violet
Indigo
Blue
```



```
var colors = ["Violet", "Indigo", "Blue", "Green", "Yellow", "Orange", "Red"];

// destructuring assignment
var[color1, ,color3, ,color5] = colors; //Leave space for unpick elements
console.log(color1); // Violet
console.log(color3); // Blue
console.log(color5); // Yellow
```



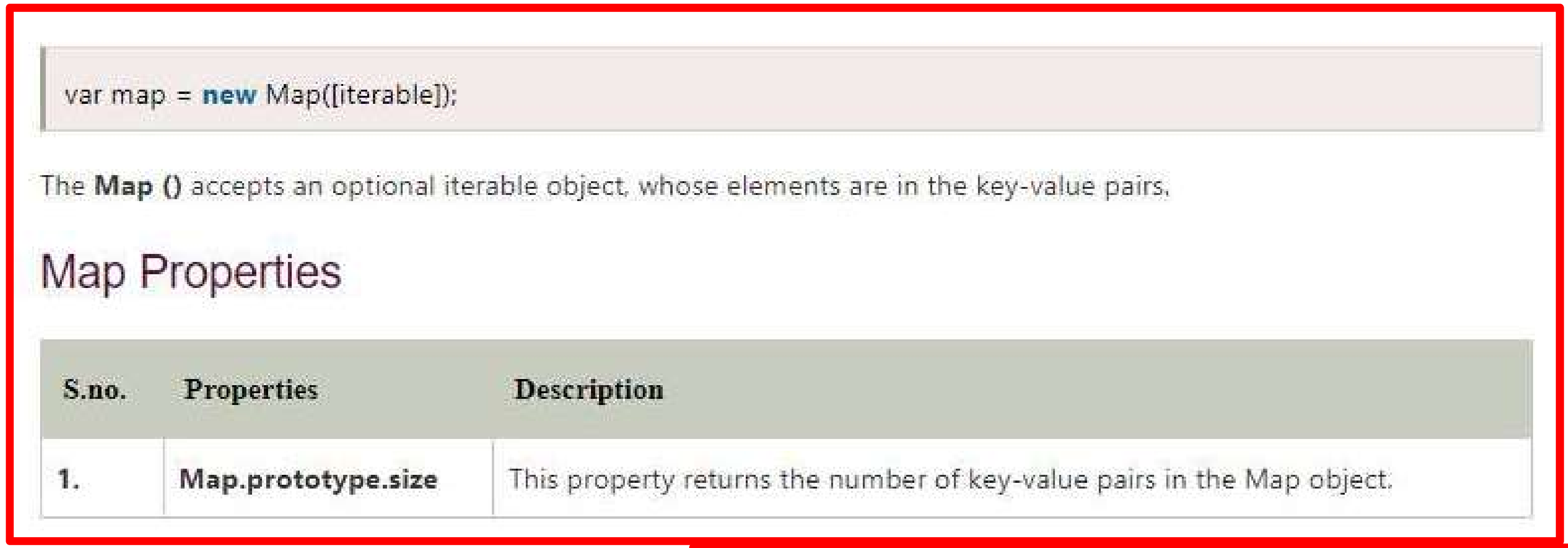
Output

```
Violet
Blue
Yellow
```

Map

- ES6 is a series of new features that are added to the JavaScript. Prior to ES6, when we require the mapping of keys and values, we often use an object. It is because the object allows us to map a key to the value of any type.
- ES6 provides us a new collection type called **Map**, which holds the key-value pairs in which values of any type can be used as either keys or values. A Map object always remembers the actual insertion order of the keys. Keys and values in a Map object may be primitive or objects. It returns the new or empty Map.
- Maps are ordered, so they traverse the elements in their insertion order.

Syntax and Creation

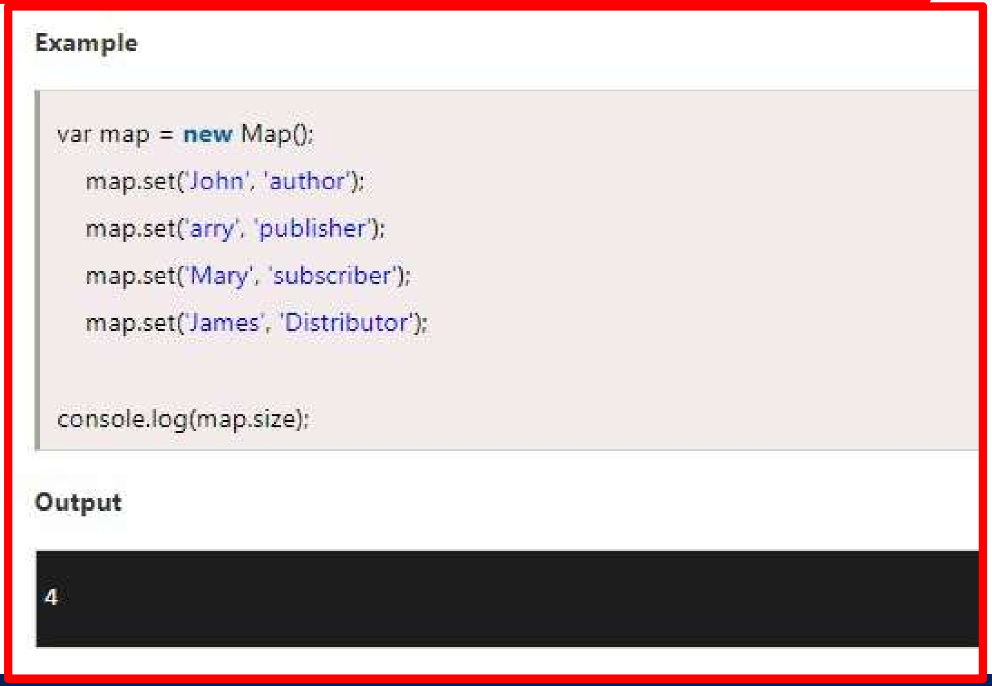


```
var map = new Map([iterable]);
```

The **Map ()** accepts an optional iterable object, whose elements are in the key-value pairs.

Map Properties

S.no.	Properties	Description
1.	**Map.prototype.size**	This property returns the number of key-value pairs in the Map object.



Example

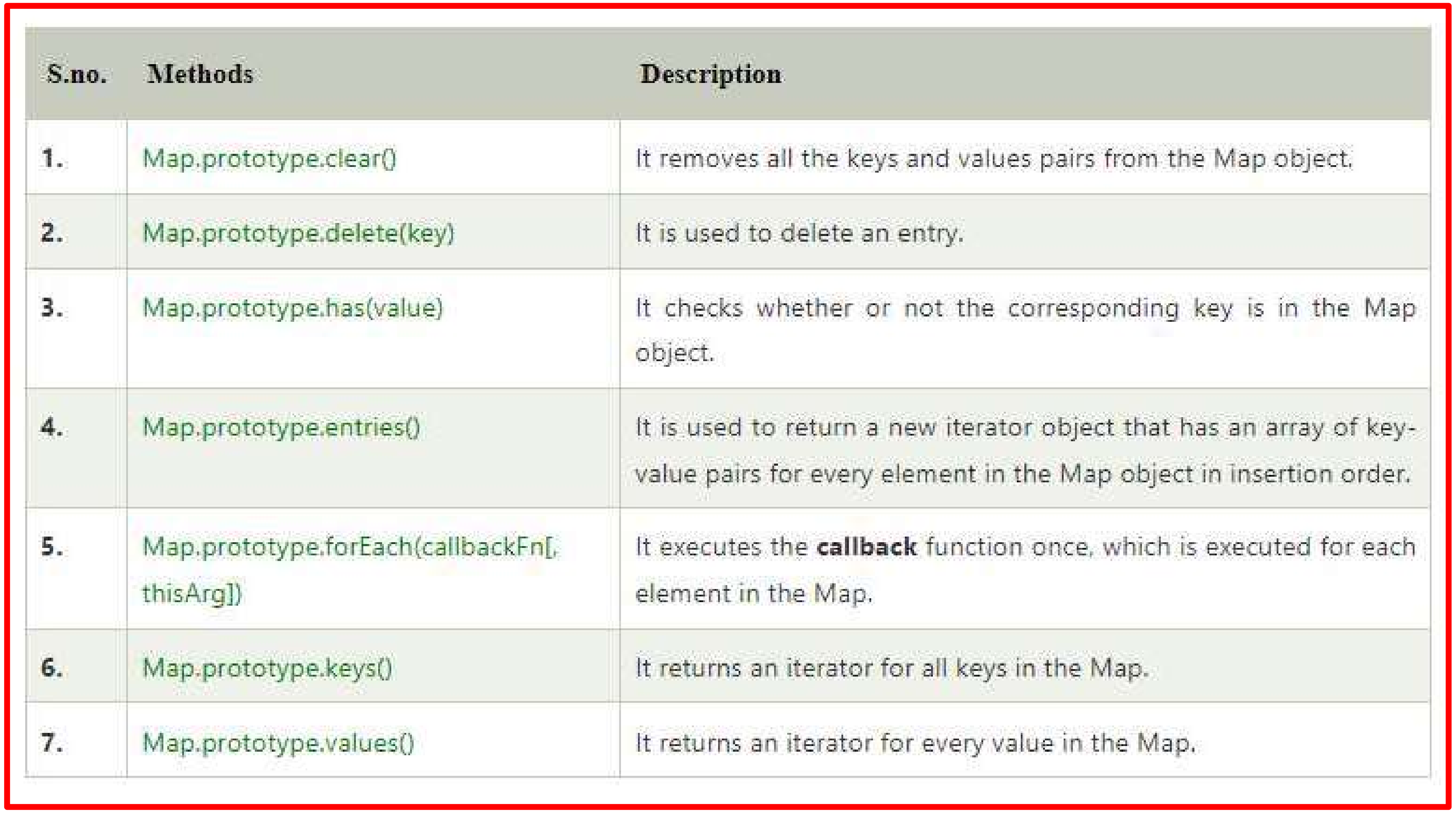
```
var map = new Map();
   map.set('John', 'author');
   map.set('arry', 'publisher');
   map.set('Mary', 'subscriber');
   map.set('James', 'Distributor');

console.log(map.size);
```

Output

```
4
```

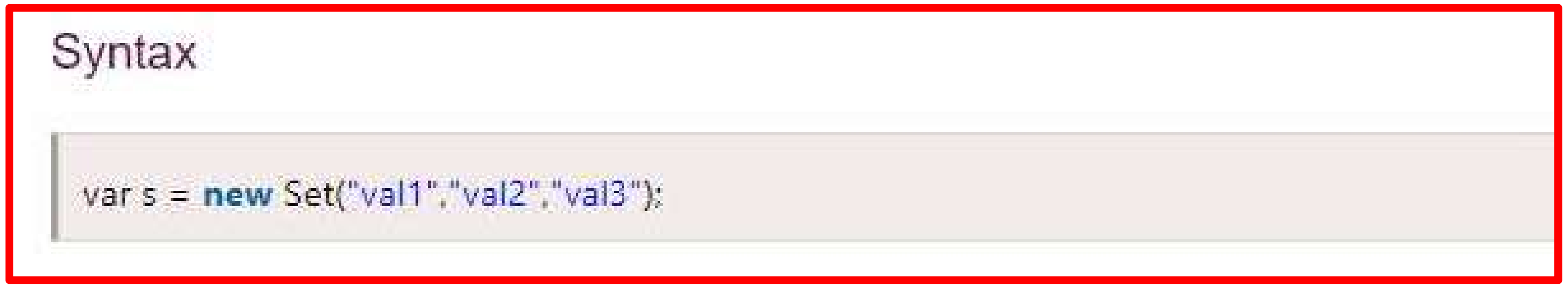
Map Methods



S.no.	Methods	Description
1.	Map.prototype.clear()	It removes all the keys and values pairs from the Map object.
2.	Map.prototype.delete(key)	It is used to delete an entry.
3.	Map.prototype.has(value)	It checks whether or not the corresponding key is in the Map object.
4.	Map.prototype.entries()	It is used to return a new iterator object that has an array of key-value pairs for every element in the Map object in insertion order.
5.	Map.prototype.forEach(callbackFn[, thisArg])	It executes the **callback** function once, which is executed for each element in the Map.
6.	Map.prototype.keys()	It returns an iterator for all keys in the Map.
7.	Map.prototype.values()	It returns an iterator for every value in the Map.

Set

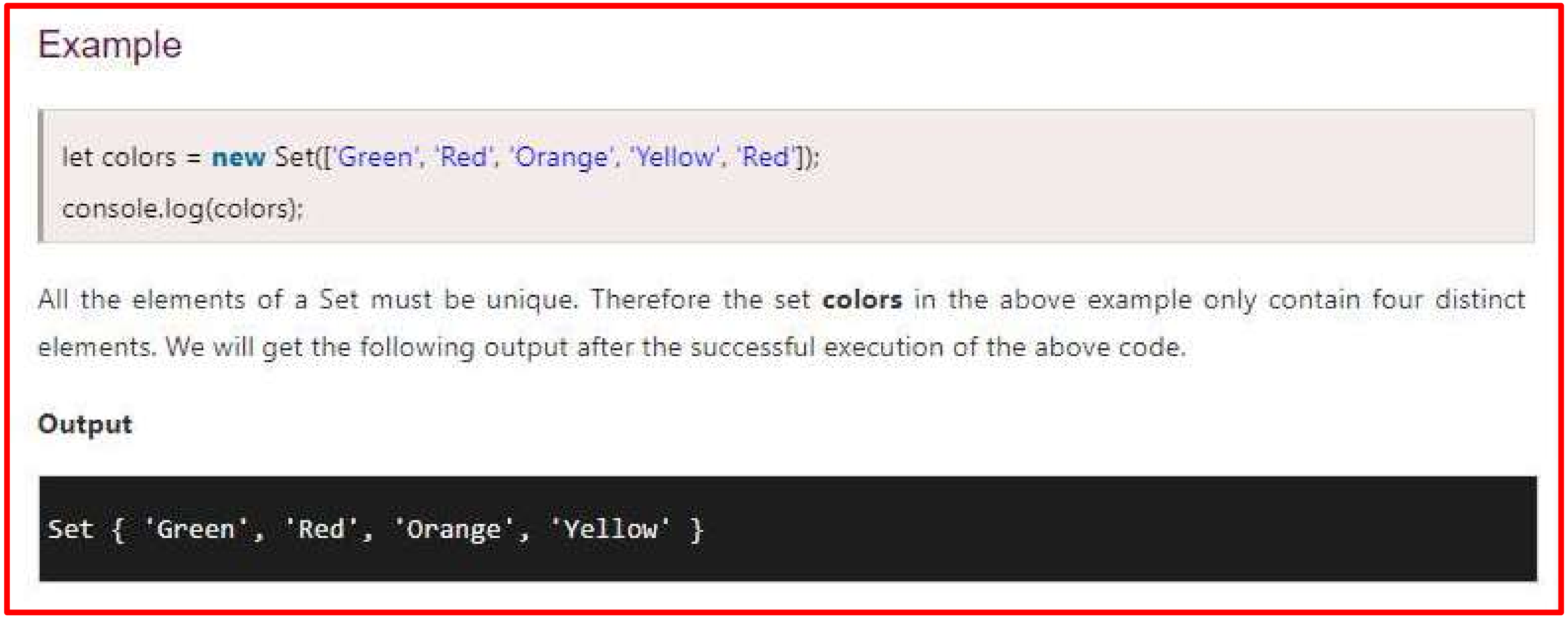
- A set is a data structure that allows you to create a collection of unique values. Sets are the collections that deal with single objects or single values.
- Set is the collection of values similar to arrays, but it does not contain any duplicates. It allows us to store unique values. It supports both primitive values and object references.
- As similar to maps, sets are also ordered, i.e., the elements in sets are iterated in their insertion order. It returns the set object.



Syntax

```
var s = new Set("val1","val2","val3");
```

Example



Example

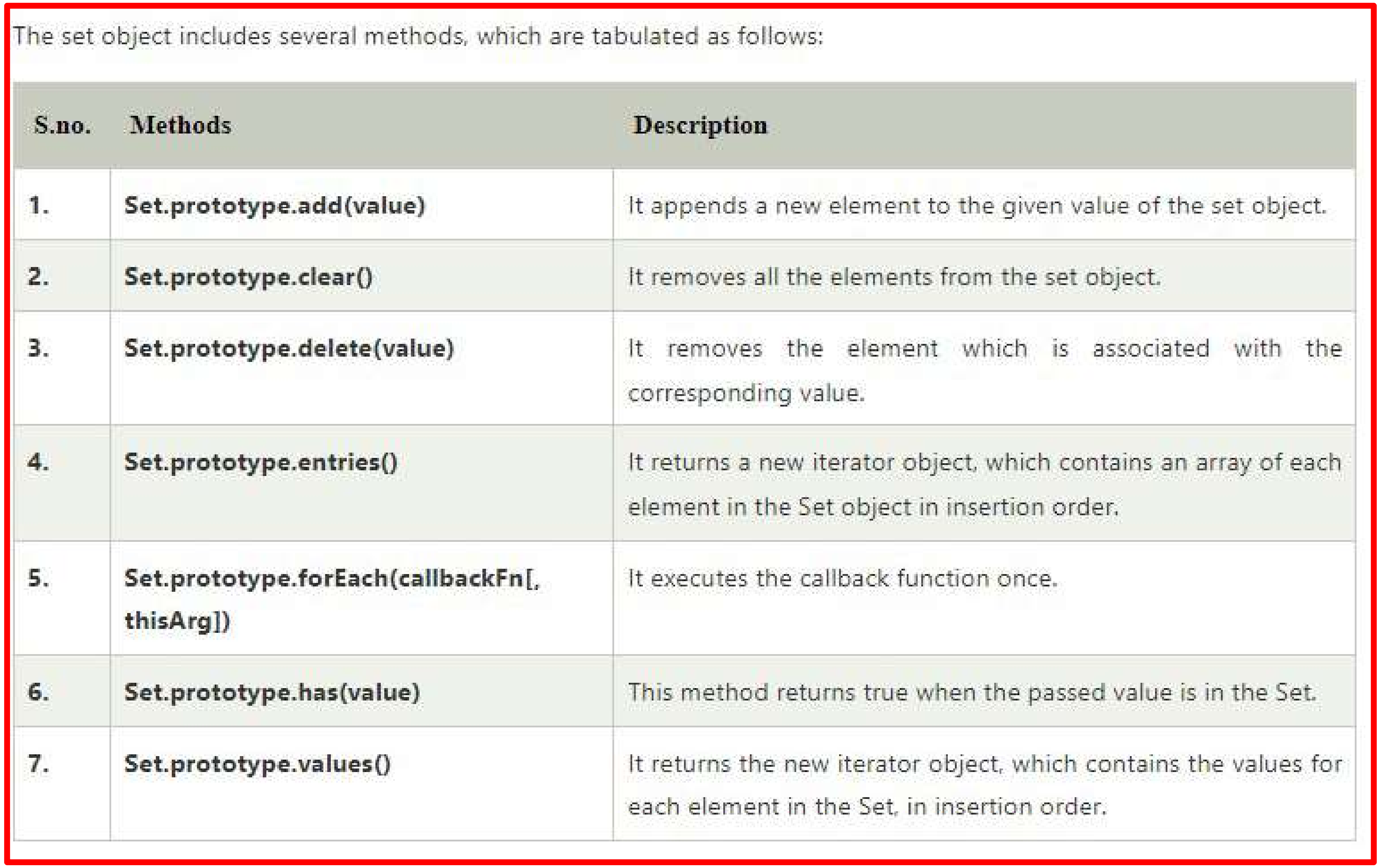
```
let colors = new Set(['Green', 'Red', 'Orange', 'Yellow', 'Red']);
console.log(colors);
```

All the elements of a Set must be unique. Therefore the set **colors** in the above example only contain four distinct elements. We will get the following output after the successful execution of the above code.

Output

```
Set { 'Green', 'Red', 'Orange', 'Yellow' }
```

Set Methods

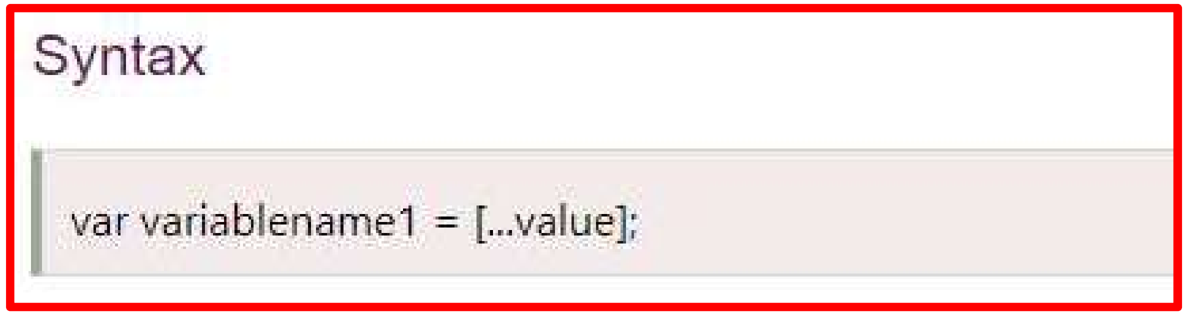


The set object includes several methods, which are tabulated as follows:

S.no.	Methods	Description
1.	**Set.prototype.add(value)**	It appends a new element to the given value of the set object.
2.	**Set.prototype.clear()**	It removes all the elements from the set object.
3.	**Set.prototype.delete(value)**	It removes the element which is associated with the corresponding value.
4.	**Set.prototype.entries()**	It returns a new iterator object, which contains an array of each element in the Set object in insertion order.
5.	**Set.prototype.forEach(callbackFn[, thisArg])**	It executes the callback function once.
6.	**Set.prototype.has(value)**	This method returns true when the passed value is in the Set.
7.	**Set.prototype.values()**	It returns the new iterator object, which contains the values for each element in the Set, in insertion order.

Spread Operator

- Typescript introduced a new operator referred to as a spread operator, which consists of three dots (...). It allows an iterable to expand in places where more than zero arguments are expected. It gives us the privilege to obtain the parameters from an array.
- Spread operator syntax is similar to the rest parameter, but it is entirely opposite of it. Let's understand the syntax of the spread operator.


Syntax

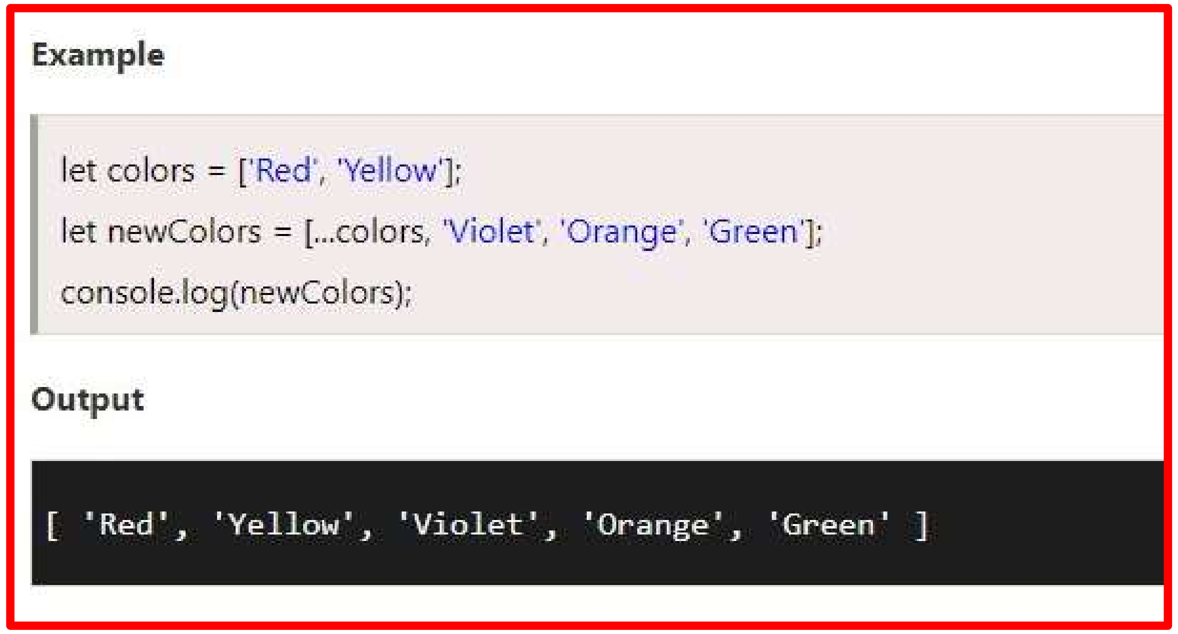
```
var variablename1 = [...value];
```

- The three dots (...) in the above syntax are the spread operator, which targets the entire values in the particular variable.

Spread Operator and Array Manipulation

Constructing array literal

- When we construct an array using the literal form, the spread operator allows us to insert another array within an initialized array.



Example

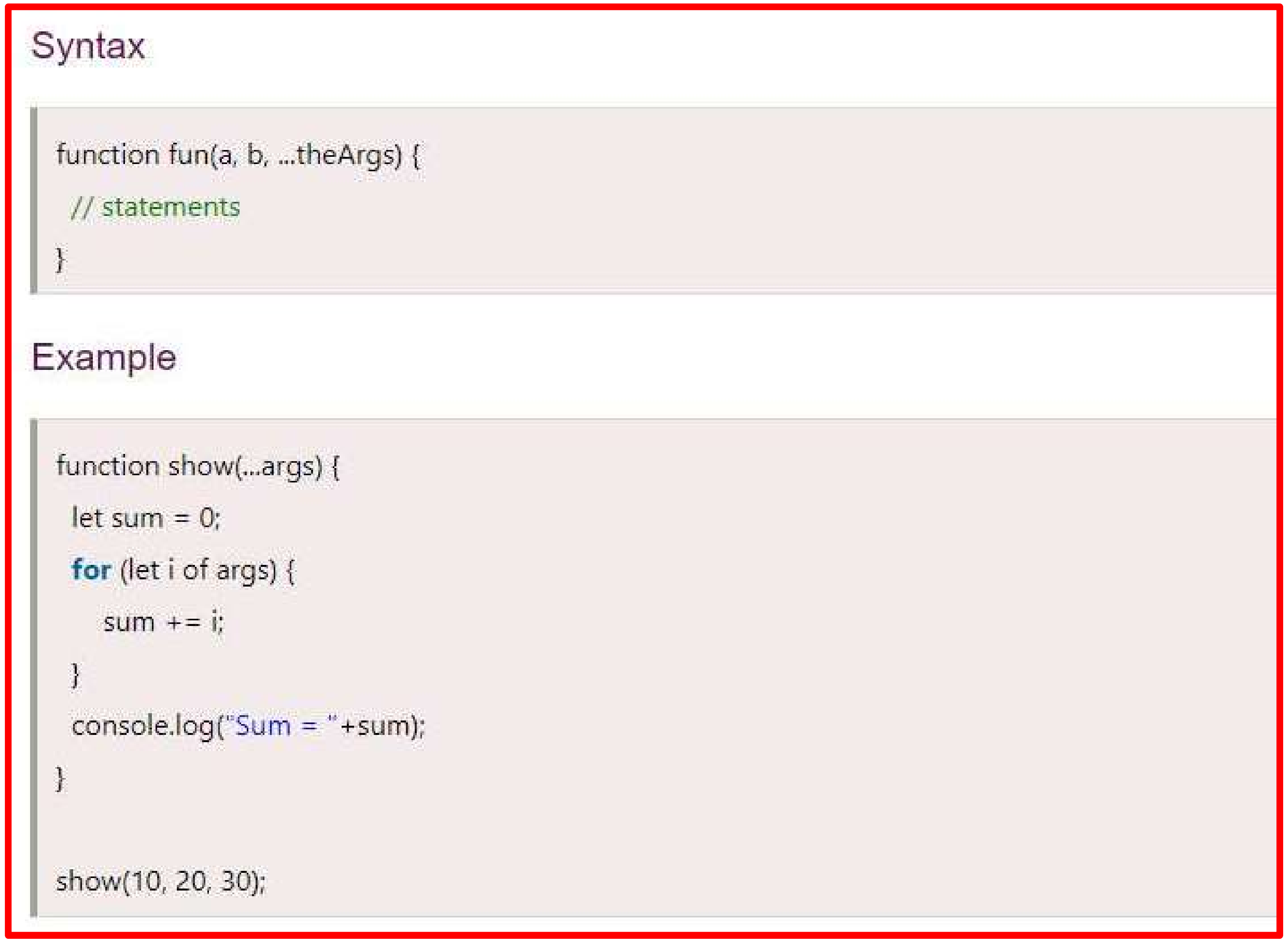
```
let colors = ['Red', 'Yellow'];
let newColors = [...colors, 'Violet', 'Orange', 'Green'];
console.log(newColors);
```

Output

```
[ 'Red', 'Yellow', 'Violet', 'Orange', 'Green' ]
```

Rest Parameter

- The rest parameter is introduced in ECMAScript 2015 or ES6, which improves the ability to handle parameters. The rest parameter allows us to represent an indefinite number of arguments as an array. By using the rest parameter, a function can be called with any number of arguments.
- Before ES6, the **arguments** object of the function was used. The **arguments** object is not an instance of the Array type. Therefore, we can't use the **filter()** method directly.
- The rest parameter is prefixed with three dots (...). Although the syntax of the rest parameter is similar to the spread operator, it is entirely opposite from the spread operator. The rest parameter has to be the last argument because it is used to collect all of the remaining elements into an array.



Syntax

```
function fun(a, b, ...theArgs) {
  // statements
}
```

Example

```
function show(...args) {
  let sum = 0;
  for (let i of args) {
     sum += i;
  }
  console.log("Sum = "+sum);
}

show(10, 20, 30);
```

Summary

In this lesson, you should have learned how to:

- Let and Const
- Arrays
- Destructuring
- Spread Operator
- Rest Parameter