JavaScript Array

# 1 - concat()

const hege = ["Cecilie", "Lone"];  
const stale = ["Emil", "Tobias", "Linus"];  
const children = hege.concat(stale);

## **//** Cecilie,Lone,Emil,Tobias,Linus

## **Definition and Usage**

* The concat() method concatenates (joins) two or more arrays.
* The concat() method does not change the existing arrays, but returns a new array, containing the values of the joined arrays.

## **Syntax**

## array1.concat(array2, array3, ..., arrayX)

const hege = ["Cecilie", "Lone"];  
const stale = ["Emil", "Tobias", "Linus"];  
const kai = ["Robin"];  
const children = hege.concat(stale, kai);

// Cecilie,Lone,Emil,Tobias,Linus,Robin

# 2 - constructor

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.constructor;  
// Returns function Array() { [native code] }

## **Definition and Usage**

The constructor property returns the function that created the Array prototype.

For JavaScript arrays the constructor property returns:  
function Array() { [native code] }

For JavaScript objects the constructor property returns:  
function Object() { [native code] }

# 3 - copyWithin()

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.copyWithin(2, 0);

// Banana,Orange,Banana,Orange

## **Definition and Usage**

The copyWithin() method copies array elements to another position in an array, overwriting the existing values.

The copyWithin() does not add items to the array.

copyWithin() overwrites the original array.

## **Syntax**

## Array.copyWithin(target, start, end)

# 4 - entries()

const fruits = ["Banana", "Orange", "Apple", "Mango"];

const f = fruits.entries(); *// entries object değer döndürüyor*

for (let x of f) {

  console.log(x);

}

## **Definition and Usage**

[0, "Banana"]  
[1, "Orange"]  
[2, "Apple"]  
[3, "Mango"]

entries() does not change the original array.

# 5 – every()

const ages = [32, 33, 16, 40];  
ages.every(checkAge)    // Returns false  
function checkAge(age) {  
  return age > 18;  
}

## **Definition and Usage**

The every() method returns true if all elements in an array pass a test (provided as a function).

The method executes the function once for each element present in the array:

* If it finds an array element where the function returns a *false* value, every() returns false (and does not check the remaining values)
* If no false occur, every() returns true

every() does not execute the function for empty array elements.

every() does not change the original array

## **Syntax**

## *array*.every(function(currentValue, index, arr), thisValue)

**currentValue** : Required. The value of the current element

**index** : Optional. The array index of the current element

**arr** : Optional. The array object the current element belongs to

# 6 – fill()

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.fill("Kiwi");

// Kiwi,Kiwi,Kiwi,Kiwi

## **Definition and Usage**

The fill() method fills specified elements in an array with a static value.

You can specify the position of where to start and end the filling. If not specified, all elements will be filled.

fill() overwrites the original array.

## **Syntax**

*array*.fill(*value, start, end*)

**value** Required. The value to fill the array with

**start** Optional. The index to start filling the array (default is 0)

**end** Optional. The index to stop filling the array (default is array.length)

# 7 – filter()

const ages = [32, 33, 16, 40];  
ages.filter(checkAdult)    // Returns [32, 33, 40],  
function checkAdult(age) {  
  return age >= 18;  
}

## **Definition and Usage**

The filter() method creates an array filled with all array elements that pass a test (provided by a function).

filter() does not execute the function for empty array elements.

filter() does not change the original array.

## **Syntax**

*array*.filter(*function(currentValue, index, arr), thisValue*)

**currentValue** Required. The value of the current element

**index** Optional. The array index of the current element

**arr** Optional. The array object the current element belongs to

**thisValue** Optional. A value to be passed to the function to be used as its "this" value. If this parameter is empty, the value "undefined" will be passed as its "this" value

# 8 – find()

const ages = [3, 30, 18, 20];  
  
function checkAge(age) {  
  return age > 18;  
}  
  
function myFunction() {  
  document.getElementById("demo").innerHTML = ages.find(checkAge);  
}

// 30

## **Definition and Usage**

The find() method returns the value of the array element that passes a test (provided by a function).

The method executes the function once for each element present in the array:

* If it finds an array element where the function returns a *true* value, find() returns the value of that array element (and does not check the remaining values)
* Otherwise it returns undefined

find() does not execute the function for empty array elements.

find() does not change the original array.

## **Syntax**

*array*.find(*function(currentValue, index, arr),thisValue*)

**currentValue** Required. The value of the current element

**index** Optional. The array index of the current element

**arr** Optional. The array object the current element belongs to

**thisValue** Optional. A value to be passed to the function to be used as its "this" value. If this parameter is empty, the value "undefined" will be passed as its "this" value

# 9 – findIndex()

const ages = [3, 10, 18, 20];  
ages.findIndex(checkAge)   // Returns 3  
function checkAge(age) {  
  return age > 18;  
}

## **Definition and Usage**

The findIndex() method returns the index of the first array element that passes a test (provided by a function).

The method executes the function once for each element present in the array:

* If it finds an array element where the function returns a *true* value, findIndex() returns the index of that array element (and does not check the remaining values)
* Otherwise it returns -1

findIndex() does not execute the function for empty array elements.

findIndex() does not change the original array.

## **Syntax**

*array*.findIndex(*function(currentValue, index, arr),thisValue*)

**currentValue** Required. The value of the current element

**index** Optional. The array index of the current element

**arr** Optional. The array object the current element belongs to

**thisValue** Optional. A value to be passed to the function to be used as its "this" value. If this parameter is empty, the value "undefined" will be passed as its "this" value

# 10 – forEach()

  const fruits = ["Banana", "Orange", "Apple", "Mango"];

  fruits.forEach((*element*, *i*) => {

      console.log(*i*, *element*);

  });

*4) ['Banana', 'Orange', 'Apple', 'Mango']*

1. 0: "Banana"
2. 1: "Orange"
3. 2: "Apple"
4. 3: "Mango"
5. length: 4
6. *[[Prototype]]*: Array(0)

## **Definition and Usage**

The forEach() method calls a function once for each element in an array, in order.

forEach() is not executed for array elements without values.

forEach’in dönüş parametresi yoktur. Varolan dizi üzerinde işlem yapmaktadır.

## **Syntax**

*array*.findIndex(*function(currentValue, index, arr),thisValue*)

**currentValue** Required. The value of the current element

**index** Optional. The array index of the current element

**arr** Optional. The array object the current element belongs to

**thisValue** Optional. A value to be passed to the function to be used as its "this" value. If this parameter is empty, the value "undefined" will be passed as its "this" value

# 11 - from ()

Create an Array from a String:

Array.from("ABCDEFG")   // Returns [A,B,C,D,E,F,G]

## **Definition and Usage**

The from() method returns an Array object from any object with a length property or any iterable object.

## **Syntax**

Array.from(object, mapFunction, thisValue)

**object** Required. The object to convert to an array

**mapFunction** Optional. A map function to call on each item of the array

**thisValue** Optional. A value to use as this when executing the mapFunction

# 12 – includes()

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.includes("Mango")   // Returns true

## **Definition and Usage**

The includes() method returns true if an array contains a specified element, otherwise false.

includes() is case sensitive.

## **Syntax**

*array*.includes(element, start)

**element** **Required**. The element to search for

**start** **Optional**. Default 0. At which position in the array to start the search

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.includes("Banana", 3);   // Returns false

# 13 – indexOf()

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.indexOf("Apple")   // Returns 2

## **Definition and Usage**

The indexOf() method searches an array for a specified item and returns its position.

The search will start at the specified position (at 0 if no start position is specified), and end the search at the end of the array.

indexOf() returns -1 if the item is not found.

If the item is present more than once, the indexOf method returns the position of the first occurence.

## **Syntax**

*array*.indexOf(item, start)

**item** **Required**. The item to search for

**start** **Optional**. Where to start the search. Negative values will start at the given position counting from the end, and search to the end.

const fruits = ["Banana", "Orange", "Apple", "Mango", "Apple"];  
fruits.indexOf("Apple", 4)   // Returns 4

# 14 – isArray()

Check whether an object is an array:

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
Array.isArray(fruits)   // Returns true

## **Definition and Usage**

The isArray() method returns true if an object is an array, otherwise false.

## **Syntax**

Array.isArray(obj)

**obj** **Required**. The object to be tested

# 15 – join()

Convert the elements of an array into a string:

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.join()   // Returns "Banana,Orange,Apple,Mango"

## **Definition and Usage**

The join() method returns an array as a string.

The elements will be separated by a specified separator. The default separator is comma (,).

join() does not change the original array.

## **Syntax**

*array*.join(separator)

**separator** **Optional**. The separator to be used. If omitted, the elements are separated with a comma

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.join(" and ");

Banana and Orange and Apple and Mango

# 16 – keys()

Create an Array Iterator object, containing the keys of the array:

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
const keys = fruits.keys();  
  
for (let x of keys) {  
  text += x + "<br>";  
}

// 0  
1  
2  
3

## **Definition and Usage**

The keys() method returns an Array Iterator object with the keys of an array.

keys() does not change the original array.

## **Syntax**

*array*.keys()

## **Parameter Values**

No parameters.

# 17 – lastIndexOf()

Search an array for the item "Apple":

const fruits = ["Apple", "Orange", "Apple", "Mango"];  
fruits.lastIndexOf("Apple")   // Returns 2

## **Definition and Usage**

The lastIndexOf() method returns the last index (position) of a specified value.

The search starts at a specified position (at the end if no start position is specified), and ends the search at the beginning of the array.

lastIndexOf() returns -1 if the item is not found.

If the search value is present more than once, the method returns the position of the last occurence.

## **Syntax**

*array*.lastIndexOf(item, start)

**item** **Required**. The item to search for

start Optional. Where to start the search. Negative values will start at the given position counting from the end, and search to the beginning

# 18 – map()

Return a new array with the square root of all the original values:

const numbers = [4, 9, 16, 25];  
const newArr = numbers.map(Math.sqrt)

Map ise dönüş parametresine sahiptir. Bir değişkene atanır. Her döngüde yeni bir dizi oluşturur. ForEach’e oranla daha hızlı çalışmaktadır.

## **Definition and Usage**

The map() method creates a new array with the results of calling a function for every array element.

The map() method calls the provided function once for each element in an array, in order.

map() does not execute the function for empty elements.

map() does not change the original array.

## **Syntax**

*array*.map(function(currentValue, index, arr), thisValue)

**Syntax :** *array*.map(*function(currentValue, index, arr),thisValue*)

**currentValue** Required. The value of the current element

**index** Optional. The array index of the current element

**arr** Optional. The array object the current element belongs to

**thisValue** Optional. A value to be passed to the function to be used as its "this" value. If this parameter is empty, the value "undefined" will be passed as its "this" value

const persons = [  
  {firstname : "Malcom", lastname: "Reynolds"},  
  {firstname : "Kaylee", lastname: "Frye"},  
  {firstname : "Jayne", lastname: "Cobb"}  
];  
  
document.getElementById("demo").innerHTML = persons.map((item) => {  
  return [item.firstname,item.lastname].join(" ");  
}

// Malcom Reynolds,Kaylee Frye,Jayne Cobb

# 19 – pop()

Remove the last element of an array:

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.pop();   // Removes "Mango"

//"Banana", "Orange", "Apple"

## **Definition and Usage**

The pop() method removes the last element of an array.

pop() returns the element it removes.

pop() changes the length of the array.

## **Syntax**

*array*.pop()

# 20 – push()

Add a new item to an array:

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.push("Kiwi");   // Adds "Kiwi"

// Banana,Orange,Apple,Mango,Kiwi

## **Definition and Usage**

The push() method adds new items to the end of an array.

push() changes the length of the array and returns the new length.

## **Syntax**

*array*.push(item1, item2, ..., itemX)

# 21 - reduce()

Subtract the numbers in the array, starting from the beginning:

const numbers = [175, 50, 25];  
  
document.getElementById("demo").innerHTML = numbers.reduce(myFunc);  
  
function myFunc(total, num) {  
  return total - num;  
}

// 100

## **Definition and Usage**

The reduce() method executes a reducer function for each value of an array.

reduce() returns a single value which is the function's accumulated result.

reduce() does not execute the function for empty array elements.

reduce() does not change the original array.

## **Syntax**

*array*.reduce(function(total, currentValue, currentIndex, arr), initialValue)

**total** **Required**. The initialValue, or the previously returned value of the function

**currentValue** **Required**. The value of the current element

**currentIndex** **Optional**. The array index of the current element

**arr** **Optional**. The array object the current element belongs to

**initialValue** **Optional**. A value to be passed to the function as the initial value

# 22 – reduceRight()

Subtract the numbers in the array, starting from the end:

const numbers = [175, 50, 25];  
  
document.getElementById("demo").innerHTML = numbers.reduceRight(myFunc);  
  
function myFunc(total, num) {  
  return total - num;  
}

// -200

## **Definition and Usage**

The reduceRight() method executes a reducer function for each value of an array , from right to left.

reduceRight() returns a single value which is the function's accumulated result.

reduceRight() does not execute the function for empty array elements.

# 23 – reverse()

Reverse the order of the elements in an array:

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.reverse();

// Mango,Apple,Orange,Banana

## **Definition and Usage**

The reverse() method reverses the order of the elements in an array.

reverse() overwrites the original array.

## **Syntax**

*array*.reverse()

# 24 – shift()

Remove the first item of an array:

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.shift();

// Orange,Apple,Mango

## **Definition and Usage**

The shift() method removes the first item of an array.

shift() returns the element it removes.

shift() changes the original array.

## **Syntax**

*array*.shift()

# 25 – slice()

Select elements from an array:

const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
const citrus = fruits.slice(1, 3); // 1 den 3 e kadar

// Orange,Lemon

## **Definition and Usage**

The slice() method returns selected elements in an array, as a new array.

slice() selects the elements starting at the given start argument, and ends at, but does not include, the given end argument.

slice() does not change the original array.

## **Syntax**

*array*.slice(start, end)

var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
var myBest = fruits.slice(-3, -1);

// Lemon,Apple

# 26 – some()

Check if any values in the ages array are 18 or over:

const ages = [3, 10, 18, 20];  
  
ages.some(checkAdult)   // Returns true  
  
function checkAdult(age) {  
  return age >= 18;  
}

## **Definition and Usage**

The some() method checks if any of the elements in an array pass a test (provided as a function).

some() executes the function once for each element in the array:

* If it finds an array element where the function returns a true value, some() “returns true (and does not check the remaining values)
* Otherwise it returns false

some() does not execute the function for empty array elements.

some() does not change the original array.

## **Syntax**

*array*.some(function(currentValue, index, arr), thisValue)

**currentValue** Required. The value of the current element

**index** Optional. The array index of the current element

**arr** Optional. The array object the current element belongs to

**thisValue** Optional. A value to be passed to the function to be used as its "this" value. If this parameter is empty, the value "undefined" will be passed as its "this" value

# 27 – sort()

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.sort();

## **Definition and Usage**

The sort() method sorts the elements of an array.

The sort order can be either alphabetic or numeric, and either ascending (up) or descending (down).

By default, the sort() method sorts the values as strings in alphabetical and ascending order.

This works well for strings ("Apple" comes before "Banana"). However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1".

Because of this, the sort() method will produce an incorrect result when sorting numbers.

sort() overwrites the original array.

## **Syntax**

*array*.sort(compareFunction)

**compareFunction** **Optional**. A function that defines an alternative sort order. The function should return a negative, zero, or positive value, depending on the arguments, like:

function(a, b){return a-b}

When the sort() method compares two values, it sends the values to the compare function, and sorts the values according to the returned (negative, zero, positive) value.

Example:

When comparing 40 and 100, the sort() method calls the compare function(40,100).

The function calculates 40-100, and returns -60 (a negative value).

The sort function will sort 40 as a value lower than 100.

# 28 – splice()

Add elements to an array:

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
  
// At position 2, add 2 elements:  
fruits.splice(2, 0, "Lemon", "Kiwi");

// Banana,Orange,Lemon,Kiwi,Apple,Mango

fruits.splice(2, 1, "Lemon", "Kiwi");

// Banana,Orange,Lemon,Kiwi,Mango

## **Definition and Usage**

The splice() method adds and/or removes array elements.

splice() overwrites the original array.

## **Syntax**

*array*.splice(index, howmany, item1, ....., itemX)

**index** **Required**. The position to add/remove items. Negative values a the position from the end of the array.

**howmany** **Optional**. Number of items to be removed.

**item1**, ..., **itemX** Optional. New elements(s) to be added

# 29 – toString()

Convert an array to a string:

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
let text = fruits.toString();

// Banana,Orange,Apple,Mango

## **Definition and Usage**

The toString() method returns a string with all array values separated by commas.

toString() does not change the original array.

## **Syntax**

*array*.toString()

# 30 – unshift()

Add new items to the beginning of an array:

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.unshift("Lemon","Pineapple");

// Lemon,Pineapple,Banana,Orange,Apple,Mango

## **Definition and Usage**

The unshift() method adds new items to the beginning of an array, and returns the new length.

unshift() overwrites the original array.

## **Syntax**

*array*.unshift(item1, item2, ..., itemX)

**item1**, **item2**, ..., **itemX** **Required**. The item(s) to add to the beginning of the array

# JavaScript Boolean

# Boolean prototype

Make a new method for JavaScript booleans:

Boolean.prototype.myColor = function() {  
  if (this.valueOf() == true) {  
    return "green";  
  } else {  
    return = "red";  
  }  
};

let a = true;

document.getElementById("demo").innerHTML = a.myColor();

//green

## **Definition and Usage**

The prototype is a global constructor available for all JavaScript objects.

Boolean.prototype referres to the global Boolean() object.

The prototype constructor allows you to add new properties and methods to booleans.

When constructing a new property, ALL arrays will get the property and its value.

When constructing a new method, ALL arrays will get the method.

# Boolean.toString()

Convert a Boolean value to a string:

let bool = true;  
bool.toString()    // Returns "true"

## **Definition and Usage**

The Boolean.toString() method returns a boolean value as a string.

The Boolean.toString() is called by JavaScript automatically whenever a boolean is used in a string operation.

# Boolean.valueOf()

Return the primitive value of a Boolean object:

let bool = false;  
bool.valueOf()    //Returns false

## **Definition and Usage**

The valueOf() method returns the primitive value of a boolean.

valueOf() is usually called by JavaScript behind the scenes, and not explicitly in code.

# Class

# constructor Method

Create a Car class, and then create an object called "mycar" based on the Car class:

class Car {  
  constructor(brand) {  // Constructor  
    this.carname = brand;  
  }  
}  
mycar = new Car("Ford");

The constructor() method is a special method for creating and initializing objects created within a class.

The constructor() method is called automatically when a class is initiated, and it has to have the exact name "constructor", in fact, if you do not have a constructor method, JavaScript will add an invisible and empty constructor method.

You can use the super() method to call the constructor of a parent class.

class Car {  
  constructor(brand) {  
    this.carname = brand;  
  }  
  present() {  
    return 'I have a ' + this.carname;  
  }  
}  
  
class Model extends Car {  
  constructor(brand, mod) {  
    **super(brand);**  
    this.model = mod;  
  }  
  show() {  
    return this.present() + ', it is a ' + this.model;  
  }  
}  
  
mycar = new Model("Ford", "Mustang");  
document.getElementById("demo").innerHTML = mycar.show();

The super() method refers to the parent class.

By calling the super() method in the constructor method, we call the parent's constructor method and gets access to the parent's properties and methods.

The extends keyword is used to create a child class of another class (parent).

The child class inherits all the methods from another class.

# static

Create a static method and call it on the class:

class Car {  
  constructor(brand) {  
    this.carname = brand;  
  }  
  static hello() {  // static method  
    return "Hello!!";  
  }  
}  
  
mycar = new Car("Ford");  
  
//Call 'hello()' on the class Car:  
document.getElementById("demo").innerHTML = Car.hello();

## **Definition and Usage**

The static keyword defines static methods for classes.

Static methods are called directly on the class (Car from the example above) - without creating an instance/object (mycar) of the class.

If you want to use the mycar object, inside the static method, you can send it as a parameter:

Send "mycar" as a parameter:

class Car {  
  constructor(brand) {  
    this.carname = brand;  
  }  
  static hello(x) {  
    return "Hello " + x.carname;  
  }  
}  
  
mycar = new Car("Ford");  
  
document.getElementById("demo").innerHTML = Car.hello(mycar);

# Error

The Error object provides error information when an error occurs.

try {  
  adddlert("Welcome guest!");  
}  
catch(err) {  
  document.getElementById("demo").innerHTML = err.name;

  document.getElementById("demo").innerHTML = err.message;  
}

## **Definition and Usage**

The name property sets or returns the name of an error.

Six different values can be returned by the error name property:

EvalError An error has occurred in the eval() function. Note: Newer versions of JavaScript does not throw any EvalError. Use SyntaxError instead.

**RangeError** A number "out of range" has occurred

**ReferenceError** An illegal reference has occurred

**SyntaxError** A syntax error has occurred

**TypeError** A type error has occurred

**URIError** An error in encodeURI() has occurred

# JSON Reference

JSON is a format for storing and transporting data.

JavaScript Objects can be converted into JSON, and JSON can be converted back into JavaScript Objects.

This way we can work with the data as JavaScript objects, with no complicated parsing or translations.

Sending JSON:

// a JavaScript object...:  
var myObj = { "name":"John", "age":31, "city":"New York" };  
  
// ...converted into JSON:  
var myJSON = JSON.stringify(myObj);  
  
// send JSON:  
window.location = "demo\_json.php?x=" + myJSON;

JSON values **cannot**be one of the following data types:

* a function
* a date
* *undefined*

**parse()** Parses a JSON string and returns a JavaScript object

**stringify()** Convert a JavaScript object to a JSON string

The JSON.stringify() method converts JavaScript objects into strings.

When sending data to a web server the data has to be a string.

const myJSON = '{ "name":"John", "age":31, "city":"New York"  }';

const myObj = JSON.parse(myJSON);

document.getElementById("demo").innerHTML = myObj.name;

// John

Storing data as JSON, using localStorage

// Storing data:  
const myObj = { "name":"John", "age":31, "city":"New York" };  
const myJSON = JSON.stringify(myObj);  
localStorage.setItem("testJSON", myJSON);  
  
// Retrieving data:  
const text = localStorage.getItem("testJSON");  
const obj = JSON.parse(text);  
document.getElementById("demo").innerHTML = obj.name;

// John

/\*replace the value of "city" to upper case:\*/  
const text = '{ "name":"John", "age":"39", "city":"New York"}';  
const obj = JSON.parse(text, function (key, value) {  
  if (key == "city") {  
    return value.toUpperCase();  
  } else {  
    return value;  
  }  
});  
  
document.getElementById("demo").innerHTML = obj.name + ", " + obj.city;

John, NEW YORK

# String

* Return the first character of a string:

let str = "HELLO WORLD";  
str.charAt(0)    // Returns "H"

* Return the Unicode of the first character in a string (the Unicode value for "H"):

let str = "HELLO WORLD";  
str.charCodeAt(0)    // Returns 72

* Join two strings:

let str1 = "Hello ";  
let str2 = "world!";  
let res = str1.concat(str2);

// Hello world!

* constructor

let str = "Hello World!";  
str.constructor   // Returns function String() {[native code]}

* Check if a string ends with "universe.":

let str = "Hello world";  
str.endsWith("world")   // Returns true  
str.endsWith("World")   // Returns false

* Convert a Unicode number into a character:

String.fromCharCode(65)   // Returns "A"

* Check if a string includes "world":

let str = "Hello world, welcome to the universe.";  
str.includes("world")   // Returns true

* Search a string for "welcome":

let str = "Hello world, welcome to the universe.";  
str.indexOf("welcome")   // Returns 13  
str.indexOf("Welcome")   // Returns -1

* Search a string for the last occurrence of "planet":

let str = "Hello planet earth, you are a great planet.";  
str.lastIndexOf("planet")   // Returns 36  
str.lastIndexOf("Planet")   // Returns -1

* Search a string for "ain":

let str = "The rain in SPAIN stays mainly in the plain";  
str.match(/ain/g)   // Returns ain,ain,ain

The match() method searches a string for a match against a regular expression, and returns the matches, as an Array object.

**Note:** If the regular expression does not include the g modifier (to perform a global search), the match() method will return only the first match in the string.

* Make a new string by copying a string twice:

let str = "Hello world!";  
str.repeat(2);

// Hello world!Hello world!

* Return a string where "Microsoft" is replaced with "W3Schools":

let str = "Visit Microsoft!";  
str.replace("Microsoft", "W3Schools");

replace() does not change the original string.

* Search for "W3Schools":

let str = "Visit W3Schools!";  
str.search("W3Schools")   // Returns 6

* Extract parts of a string:

let str = "Hello world!";  
str.slice(0, 5)   // Returns "Hello"

***string*.slice(*start*,*end*)**

* Split a string into an array of substrings:

let str = "How are you doing today?";  
const myArr = str.split(" ");

// How,are,you,doing,today?

The split() method splits a string into an array of substrings, and returns the new array.

If an empty string ("") is used as the separator, the string is split between each character.

The split() method does not change the original string.

***string*.split(*separator*,*limit*)**

let str = "How are you doing today?";  
const myArr = str.split(" ",3);

// How,are,you

* Check if a string starts with "Hello":

let str = "Hello world, welcome to the universe.";  
str.startsWith("Hello")   // Returns true

* Extract parts of a string:

let str = "Hello world!";  
str.substr(1, 4)   // Returns "ello"

The substr() method extracts parts of a string, beginning at the character at a specified position, and returns a specified number of characters.

**Tip:** To extract characters from the end of the string, use a negative start number.

substr() method does not change the original string.

***string*.substr(*start*,*length*)**

* Begin the extraction at position 2, and extract the rest of the string:

str.substr(2); // llo world!

* Extract only the first character:

str.substr(0, 1); // H

* Extract only the last character:

res = str.substr(11, 1); // !

* Extract characters from a string:

let str = "Hello world!";  
str.substring(1, 4)   // Returns "ell"

The substring() method extracts characters, between to indices (positions), from a string, and returns the substring.

The substring() method extracts characters between "start" and "end", not including "end".

If "start" is greater than "end", substring() will swap the two arguments, meaning (1, 4) equals (4, 1).

If "start" or "end" is less than 0, they are treated as 0.

The substring() method does not change the original string.

***string*.substring(*start, end*)**

* Convert the string to lowercase letters:

let str = "Hello World!";  
str.toLocaleLowerCase()   // Returns "hello world!"

* Convert the string to uppercase letters:

let str = "Hello World!";  
let res = str.toLocaleUpperCase(); // Returns "HELLO WORLD!"

* Convert the string to lowercase letters:

let str = "Hello World!";  
str.toLowerCase(); // hello world!

* Convert the string to uppercase letters:

let str = "Hello World!";  
str.toUpperCase()   // Returns "HELLO WORLD!"

* Remove whitespace from both sides of a string:

let str = "       Hello World!        ";  
str.trim()   // Returns "Hello World!"

* Return the primitive value of a string object:

let str = "Hello World!";  
str.valueOf()   // Returns "Hello World!"