Features introduced in Javascript from 2009 – 2022

**“use strict”**

* It defines that the below codes should be executed under strict mode
* You can use strict mode in all your programs. It helps you to write cleaner code, like preventing you from using undeclared variables.
* Introduced in ES5

**chartAt()**

* Inbuilt string method which takes index number as argument and returns the character of the specified index
* Introduced in ES5
* Example :

var str = "ECMA 2005";

console.log(str.charAt(0)); // prints E

**trim()**

* An inbuilt string method which removes the whitespaces from both side of the respective string
* Introduced in ES5
* Example :

var str = "  ECMA 2005  ";

console.log(str); // prints "  ECMA 2005  "

console.log(str.trim()); // prints "ECMA 2005"

**Array.isArray()**

* This method checks whether the passed argument is an array or not
* Return type will be an Boolean value (true/false)
* Introduced in ES5
* Example :

var element = [1, 2, 3, 4, 5, 6, 7, 8, 9];

console.log(Array.isArray(element)); // prints true

**ForEach() – Loop**

* One of the loop methods introduced in ES5
* Takes a call back function has an argument,
* That function will have two parameters, 1st argument will be the individual element and 2nd argument will the index of the element
* Introduced in ES5
* Example :

var array = [1, 2, 3, 4];

array.forEach(function (element, index) {

  console.log("Index of " + index + " Array Element is " + element);

});

/\* Prints

Index of 0 Array Element is 1

Index of 1 Array Element is 2

Index of 2 Array Element is 3

Index of 3 Array Element is 4 \*/

**map()**

* An array method which creates a new array by populating the current array elements individually as per the given call back function
* Introduced in ES5
* Example :

var array = [1, 2, 3, 4];

var newArray = array.map(function (element, index) { //index parameter is optional

  return element \* 2; // here each array element is iterated through this function

});

console.log(newArray); // Prints : [2,4,6,8]

**filter()**

* This also creates a new array, but the result array will have elements only where the current element passes the function statement
* Introduced in ES5
* Example :

var array = [1, 2, 3, 4];

var filteredArray = array.filter(function (element, index) {

  return element > 2; // here each array element is iterated through this function

});

console.log(filteredArray); // Prints only [3,4]

**reduce()**

* It finds the sum of the entire array
* Same as map, filter methodwhich takes function as argument and the function itself takes four parameters
* 1st Parameter will be accumulator which will be the initial Value
* 2nd Parameter will the current array element
* 3rd Parameter will the current index of the array element
* 4th Parameter will the entire array itself
* Example

var array = [1, 2, 3, 4];

var initialAccumulatorValue = 0;

var TotalSum = array.reduce(function (accumulator, currentValue, index, array) {

  return accumulator + currentValue; // each array element is iterated through this function

}, initialAccumulatorValue);

console.log(TotalSum); // Prints 10

**every()**

* This method also takes an function as argument and the function takes the individual array value as parameter
* Return type will be an Boolean value based on the function statement
* Example

var array = [1, 2, 3, 4];

var result = array.every(function(element){

  return element > 3 // Checks whether all the array element value is great than 3

})

console.log(result); // Prints false, because only 4 passes the function statement

**some()**

* Works similar to every method, but if even one value passes the function statement the return type will be true.
* Example:

var array = [1, 2, 3, 4];

var result = array.some(function (element) {

  return element > 3; // Checks whether all the array element value is great than 3

});

console.log(result); // Prints true, because 4 passes the function statement

**indexOf()**

* Search an complete array for an specific element’s index value
* Example :

var array = ["alpha", "beta", "gamma", "omega"];

var indexValue = array.indexOf("gamma");

console.log(indexValue); // Returns 2 as the index value

**lastIndexOf()**

* Works similar to indexOf() method, but the search will commence for the end of the array
* Example :

var array = ["alpha", "beta", "gamma", "omega"];

var indexValue = array.lastIndexOf("gamma");

console.log(indexValue); // Returns 2 as the index value, because index of value omega becomes 1

**Variable declarations [ let, const ]**

* We use **let** keyword for declaring variables in block-scoped declarations
* The value of the variable can be changed later, within the block
* **const** keyword is also used for variable declarations but the value is immutable.

Example :

let x = 1;

const y = 1;

let z;

const a; // Value must be assigned during initialization itself

{

  const b = 3; // Will be available only inside this scope alone

  x = 2; // Allowed

  y = 2; // Not Allowed (Error : Assignment to constant variable)

}

console.log(b); // wont be available outside the scope (same works for let as well)

console.log(x);

console.log(y);

console.log(z); // returns undefined hence variable is declared before itself

**Arrow functions**

* Arrow functions allows a short syntax for writing function expressions
* If it’s an single line you don’t need a return keyword & curly braces
* The function keyword is also not required
* Arrow functions are not hoisted. They must be defined before they are used.
* Example :

const normalFunction = function (x, y) {

  return x + y;

};

console.log(normalFunction(2, 3)); // prints 5 as value

const arrowFunction = (x, y) => x + y;

console.log(arrowFunction(2, 3)); // prints 5 as value

**For of** Loop

* One of the loops introduced in ES6 versions
* A variable is assigned in the loop time, and that variable name will assigned to each array value during the iteration period
* This loop method can be used on strings as well
* Example :

const array = ["alpha", "beta", "gamma", "omega"];

for (let value of array) {

  console.log(value); // value is an variable assigned to each array element value individually during iteration

}

**Maps**

* The Map object holds key-value pairs of any data Type
* It remembers the original insertion order of the keys
* It even has its own built in methods like :
* set() – sets the value for a key in Map
* get() –gets the value for a key
* clear() – removes all elements | delete() – removes the specified key
* size – returns the number of key value pairs present inside
* Example :

const radiations = new Map();

radiations.set("alpha", 1); // alpha becomes the key with value 1

radiations.set("beta", 2); // beta becomes the key with value 2

radiations.set("gamma", 3); // gamma becomes the key with value 3

radiations.set("omega", 4); // omega becomes the key with value 4

console.log(radiations.get("beta")); // prints 2 as value

console.log(radiations.size); // prints 4 as value

radiations.delete("beta"); // deletes the beta key

console.log(radiations); // prints = Map(3) { 'alpha' => 1, 'gamma' => 3, 'omega' => 4 }

**Spread Operator**

* Spread expands an array of its elements even if it’s an string, it’ll spread it to an array with individual values
* It is also used for copying arrays & objects
* Example :

const string = "Spread";

console.log([...string]); // prints [ 'S', 'p', 'r', 'e', 'a', 'd' ]

const radiations = ["alpha", "beta"];

const allRadiations = ["beta", ...radiations, "gamma"];

console.log(allRadiations); // prints ["beta","alpha","beta","gamma"]

**Sets**

* Set is a collection of unique values
* Each value can only occur once in a set
* It can hold any value of data types

const array = [1, 2, 1, 2, 3, 4, 45, 6, 7];

const unique = new Set(array);

console.log(unique); // prints Set(7) { 1, 2, 3, 4, 45, 6, 7 }

**Rest Operator**

* Rest operator works just like spread in opposite manner
* It bundles the entire elements as an array and also can be used to indefinite elements as parameters

function sum(...params) {

  let sum = 0;

  params.forEach((el) => console.log(el));

}

sum(1, 2, 3, 4, 5, 6, 7, 8); // all the argument passed will be bundled as an single array

**Includes ()**

* Checks whether a string contains a specified value,
* Return type will be a Boolean

let string = "alpha";

console.log(string.includes("x")); // returns false since x is not in "alpha"

**startsWith()**

* returns true if a string begins with a specified value, otherwise false

let string = "alpha game";

console.log(string.startsWith("game")); // returns false since game is not in beginning

**endsWith()**

* returns true if a string begins with a specified value, otherwise false

let string = "alpha game";

console.log(string.endsWith("game")); // returns true since game is in the end

**find()**

* works similar to filter method, but if the element meets the function statement, it will only the first occurrence of the element

let array = [1, 2, 3, 4, 5, 6, 7, 8];

let result = array.find((element) => element > 3);

console.log(result); // Returns only 4

**findIndex()**

* Works very similar to find but instead of returning the first occurrence element it returns the index of the element

let array = [1, 2, 3, 4, 5, 6, 7, 8];

let result = array.findIndex((element) => element > 3);

console.log(result); // Returns only 3

**Number.isInteger()**

* Method returns true if the argument is an integer.
* Number.isInteger(10) => returns true
* Number.isInteger(10.5) => returns false

**isFinite()**

* Method returns true if the argument is an finite or NaN.
* isFinite(10/10) => returns false
* isFinite(10/1) => returns true

**isNaN()**

* Method returns true if the argument is not an number.
* isNaN(“10”) => returns false
* isNaN(10) => returns true

**Exponentiation Operator**

* exponentiation operator (\*\*) raises the first operand to the power of the second operand
* let x = 5 \*\* 2 // returns 25

**padStart()**

* This method adds the current string with another string which was passed as an argument
* Takes two arguments – 1st Length of the string and 2nd will the string to be added

let str = "2";

console.log(str.padStart(3, "4")); // returns "442"

**padEnd()**

* This method also adds the current string with another string which was passed as an argument but from the end
* Takes two arguments – 1st Length of the string and 2nd will the string to be added

let str = "2";

console.log(str.padEnd(3, "4")); // returns "244"

**Object.Entries()**

* This method returns an array of array holding key,value pairs

const user = {

  firstName: "John",

  lastName: "Doe",

  age: 50,

};

console.log(Object.entries(user)); // returns [ [ 'firstName', 'John' ], [ 'lastName', 'Doe' ], [ 'age', 50 ] ]

**flat()**

* This method converts deeply nested array like 2D arrays into flat arrays by taking an integer as an argument
* That argument will determine the depth of the nested array

var array = [[1, 2],[3, 4],[[5, 6],[6, 7],[7, 8],[8, 9]]];

console.log(array.flat(2)); // Returns [1,2,3,4,5,6,6,7,8,8,9]

**Numberic Separators**

* This feature allows to use underscore as numeric separators to help improve readability between group of numbers

const value = 1\_00\_000 // returns 100000

**replaceAll()**

* Similar to replace method but, it replaces the entire string occurrence

const string = "hello+world+this+replaces+all";

console.log(string.replaceAll("+", " ")); // returns "hello world this replaces all"

**at()**

* With this feature we can access any position of an array(or) string, even the last element by passing negative values as argument (negative indexing)

var array = [1, 2, 3, 4, 5, 6, 7, 8, 9];

console.log(array.at(-1)); // returns the last element