Advanced JS

JavaScript function method

- call() and apply() method are allows us to write a method that can be used on different object.

Example,

```
var person = {
    name: "Urvashi",
    mobileNo: 9876543210,
}
var person_address = {
    fullAddress: function () {
        return this.name + " " + this.mobileNo;
    }
}
console.log(person_address.fullAddress.call(person));
console.log(person_address.fullAddress.apply(person));
```

- bind() method introduced in ES5.
- It create a new function whose refer this keyword and with given sequence of argument.

Example,

```
var thispage = {
    name: "Current Page",
    getName: function () {
        return this.name
    }
}
var newpages = thispage.getName;
var allpage = newpages.bind(thispage);
console.log(allpage());
```

Hoisting

- It is a mechanism in JS that moves the declaration of variable and functions at the top.
- JS hoisting is applicable only for declaration not initialization. It required to initialize the variable and function before using their values.

Example,

```
x = 5;
console.log(x);
var x;
```

note : we can use hoisting using only var keyword because it is global keyword, whenever let and const gives error, because it could be declaration first.

use strict mode

- Sometime JS code display correct result even it has some errors. To overcome this problem use the JS strict mode.
- The JS provide “use strict” expression to enable the strict mode.
- If there is any silent error occur in code it throw an error.

Example,

```
"use strict"

function myFunc() {
    y = 1.34;
    console.log(y);
}
myFunc();
```

Reload() method

- It is used to reload() the webpage.
- Similar to refresh button of the browser.
- It does not return any value.

Example,

```
<body>
    <button onclick="refresh()">Refresh Page</button>

    <script>
        function refresh() {
            location.reload()
        }
    </script>
</body>
```

Array and Methods

- Array is a similar type of data.
  Some Array methods are :
  from(), keys(), find(), findIndex() etc..

  example,

```
var str = Array.from("Hello world");

console.log(str);
var newArr = new Set(["string 1", "string 2", "string 3"]);
console.log(Array.from(newArr));
```

```
var subj = ["react", "angular", "node", "vue"];
var res = subj.keys()
for (const i of res) {
    console.log(`${i} => ${subj[i]}`);
}
```

```
function isPrime(val) {
    var start = 2;
    while (start <= Math.sqrt(val)) {
        if (val % start++ < 1) {
            return false;
        }
    }
    return val > 1;
}
console.log([8, 9, 10, 13].find(isPrime));

if find index
console.log([8, 9, 10, 13].findIndex(isPrime));
```

includes() method

- It performs a case-sensitive search.
- Return true or false.

Example,

```
let sen = "Easy to learn JavaScript Language"
let word = "Easy";

let str = sen.includes(word) ? "true" : "false";
console.log(str);
```

startsWith() and endsWith() method

- startsWith() match beginning of the character or string.
- endsWith() match end of the character or string.

```
let sen = "Easy to learn JS"
console.log(sen.startsWith("Easy"));
console.log(sen.startsWith("E"));

console.log(sen.endsWith("Easy"));
console.log(sen.endsWith("JS"));
```

currying function

- it is programming technique where a function with multiple argument is transform into series of function, each take a single argument.
  Example,

```
function calculateVolume(length) {
    return function (breath) {
        return function (height) {
            return length * breath * height;
        }
    }
}
console.log(calculateVolume(4)(5)(6));
```

Optional chaining

- this feature introduced in ES2020.
- it access an object property nested within objects.
- It dealing with null or undefined value.

Syntax :

Obj?.prop
Obj?.[expr]
Obj?.[index]
Obj?.(arg)

- When we check a value of property in inside the structure.
  For ex, let perval = person.emp && person.emp.name

- So these method allows us to handle this type of cases without repeating code.
  For ex, let perval = person.emp?.name

Example,

```
const user = {
    firstname: "urvashi",
    lastname: "bhavsar",
    DoJ: "12-05-2023",
}

// return undefined if object has no key
let fname = user?.firstname;
let no = user?.mobile;
console.log(fname);
console.log(no);
```

Recursion

- It means function call itself.

Example,

```
let decrementNum = (number) => {
    if (number === 0) return;
    console.log(number);
    decrementNum(number - 1);
}
decrementNum(10)
```

JS set object

- It used to store the elements with unique values.
- Values can be any type.

Main points

- It uses the concept of key internally.
- It cannot contain the duplicate values.
- It iterates its element in insertion order.

- Set method list :
  1. add()
  2. delete()
  3. clear()
  4. entries()
  5. forEach()
  6. values()

Example with methods :

```
var lang = new Set();
lang.add("html")
lang.add("css")
lang.add("js")
console.log("=== added values ===");
for (const e of lang) {
    console.log(e);
}
console.log("=== delete one value ===");
lang.delete("html");
for (const e of lang) {
    console.log(e);
}
lang.clear();
console.log("=== after clear ===");
for (const e of lang) {
    console.log(e);
}
```

Example with methods,

```
var lang = new Set();
lang.add("html")
lang.add("css")
lang.add("js")
console.log("=== added values ===");
var i = lang.entries();
for (const e of i) {
    console.log(e);
}
console.log("=== using for each ====");
lang.forEach(element => {
    console.log(element);
});
console.log("=== has method ====");
console.log(lang.has("js"));
console.log(lang.has("nodejs"));
console.log("=== using values for of loop ====");
var lg = lang.values();
for (const e of lg) {
    console.log(e);
}
console.log("=== using values execute single value ====");
var lg = lang.values();
console.log(lg.next());
console.log(lg.next());
console.log(lg.next());
```

JS Map object

- Used to map keys to value.
- It stores each element as key-value pair.
- It operates the elements such as search, update and delete on basis of specified key.

Main Points

- It cannot contain the duplicate keys and values.
- Key and value can be any type.
- It iterates element in insertion order.

- Map method list :
  1. clear()
  2. delete()
  3. entries()
  4. forEach()

5. get()
6. has()
7. keys()
8. set()
9. values()

Example with methods :

```
var map = new Map();
map.set(1, "angular.js");
map.set(2, "vue.js");
map.set(3, "react.js");

console.log("Total map : " + map.size);
map.clear();
console.log("After clear total map : " + map.size);

console.log("Total map : " + map.size);
map.delete(3);
console.log("After clear total map : " + map.size);

let subj = map.entries();
// console.log(iterator.next().value);
// console.log(iterator.next().value);
// console.log(iterator.next().value);

// using loop
for (let i = 0; i < map.size; i++) {
    console.log(subj.next().value);
}

map.forEach(subj => {
    console.log(subj);
});

function display(value, key) {
    console.log(key + " " + value);
}
map.forEach(display)
```

Example with methods :

```
var map = new Map();
map.set(1, "angular.js");
map.set(2, "vue.js");
map.set(3, "react.js");

console.log(map.get(3));

console.log(map.has(3));
console.log(map.has(4));

let subj = map.keys();
```

```
// using loop
for (let i = 0; i < map.size; i++) {
    console.log(subj.next().value);
}

let subj = map.values();

// using loop
for (let i = 0; i < map.size; i++) {
    console.log(subj.next().value);
}
```

JS WeakSet object

- it is type of collection that allows us to store weakly held objects. Unlike Set, the WeakSet are the collection of object only.
- It doesn't contains arbitrary values.

Main points

- It contains unique object only.
- If there is no reference to store object, it target to garbage collection.
- The object aren't enumerable, it doesn't provide any method to get any specified object.

- Methods :
  1. add()
  2. has()
  3. delete()

Example,

```
var ws = new WeakSet();
var obj1 = { num: 1 }, obj2 = { num: 2 }
ws.add(obj1);
ws.add(obj2);

console.log(ws.has(obj1));
console.log(ws.has(obj2));

ws.delete(obj1)
console.log(ws.has(obj1));
```

JS WeakMap object

- it is type of collection which is almost similar to Map.
- Store element as key-pair value, where key are weakly referenced.
- Methods :
  1. delete()
  2. get()
  3. has()
  4. set()

example,

```
let wm = new WeakMap();
var obj1 = {}, obj2 = {}, obj3 = {};
wm.set(obj1, "React");
wm.set(obj2, "Angular");
wm.set(obj3, "Vue");

console.log(wm.get(obj2));
console.log(wm.has(obj2));

wm.delete(obj2);
console.log(wm.has(obj2));
```

Callback

- it can be defined as a function passed into another function as a parameter.
- It is not a keyword. it is just name of an argument that is passed to a function.
- It is useful to develop an asynchronous JavaScript code.
- they are mainly used to handle the asynchronous operations such as the registering event listeners, fetching or inserting some data into/from the file, and many more.

Example,

```
function getData(x, y, callback) {
    console.log(`<div>The multiplication of the numbers ${x} and ${y} is : (${x + y})</div>`);
    callback
}
function showData() {
    console.log("Execute function");
}
getData(20, 30, showData())
```

Closure

- it means inner function can be access to the outer function variable.
- In JS, every time a closure is created with the creation of a function.

Main points :

- It access own scope.
- It access outer function variable and global variable

Example,

```
const add = (() => {
    // i.e. means private variable
    let counter = 0;
```

```
    return function () {
        counter += 1;
        return counter;
    }
})();;
console.log(add());
console.log(add());
console.log(add());
```

Exception Handling

- It is a process or method used for handling the abnormal statements in the code and executing them.
- It also enables to handle the flow control of the program.
- Types of error:
    - Syntax error
    - Runtime error
    - Logical error
- Following standard built-in error types or constructors beside it:
    - Eval error
    - Internal error
    - Range error
    - Reference error
    - Syntax error
    - Type error
    - URI error

try…catch statement

- Use to handle the error part of the code.
- It initially tests the code for all possible errors it many contain, then it implements actions to tackle errors.
- A good programming approach is to keep the complex code within the try…catch statement.
    - try{ } → the code which needs possible error testing is kept within the try block. If any error occur, it passes to the catch{} block for taking handle the error. Otherwise, it execute the code written within.

- catch{ } → handles the error of the code by executing the set of statement written within the block. It contains either the user-defined exception handler or the built-in handler.

Example,

```
try {
    var a = [3, 4, 12, 6, 33];
    console.log(a);
    // it is undefined. so invoke catch error.
    console.log(b);
} catch (e) {
    console.log(e);
}
```

Prototype

- It is prototype-based language that facilitates the objects to acquire properties and features from one another.
- It is mechanism by which JavaScript objects inherit features from one another.
- It is a function is created the prototype property is added to that function automatically.
- It is a prototype object that holds a constructor property.
- All JS objects inherit properties and methods from a prototype.

Syntax :

ClassName.prototype.MethodName

Why we used prototype or required?

- Generally, when object is created in JS, its related functions are loaded in memory so new copy of function is created in each object creation.
- In prototype, all the object shares the same function. To ignore to create a new copy of function each object. Thus, the functions are loaded in memory at least once.

Prototype chaining

Example,

To add a new method to the constructor function.

```
<body>

    <script>
        function Student(firstName, lastName) {
            this.firstName = firstName;
```

```
            this.lastName = lastName
        }
        Student.prototype.fullName = function () {
            return this.firstName + " " + this.lastName;
        }
        var student1 = new Student("Urvashi", "Bhavsar");
        var student2 = new Student("Pooja", "Patel");
        document.write(`<div>Student Name : ${student1.fullName()}</div>`)
        document.write(`<div>Student Name : ${student2.fullName()}</div>`)
    </script>
</body>
```

Example,

```
<body>
    <script>
        function Student(firstName, lastName) {
            this.firstName = firstName;
            this.lastName = lastName;
        }
        Student.prototype.school = "Abc School";
        var stud1 = new Student("Urvashi", "Bhavsar");
        var stud2 = new Student("Pooja", "Patel");
        document.write("<div>" + stud1.firstName + " " + stud1.lastName + "
" + stud1.school + "</div>")
        document.write("<div>" + stud2.firstName + " " + stud2.lastName + "
" + stud2.school + "</div>")
    </script>
</body>
```

Constructor method

- It is a special type of method which is used to initialize and create an object.
- It is called when memory is allocated for an object.

Remember :

- Constructor keyword is used to declare a constructor method.
- It can contains one constructor method only.
- JS allows us to use parent class constructor through super keyword.

Example, using constructor keyword.

```
<body>
    <script>
        class Student {
            constructor() {
```

```
                this.id = "Emp101";
                this.name = "Urvashi";
            }
        }
        var stud = new Student();
        document.write(`id => ${stud.id}, name => ${stud.name}`);
    </script>
</body>
```

Example, using super keyword.

```
<body>
    <script>
        class Student {
            constructor() {
                this.schoolName = "Abc School"
            }
        }
        class Stud extends Student {
            constructor(fname, lname) {
                super();
                this.fname = fname;
                this.lname = lname;
            }
        }
        var stud = new Stud("Mansi", "Patel");
        document.write(stud.fname + " " + stud.lname + " " +
stud.schoolName)
    </script>
</body>
```

Destructuring

- It means to break down a structure into simpler parts.
- You can extract smaller fragments from objects and array.
- It can be used for assignments and declaration of a variable.
- It is a way to extract multiple values from data that is store in arrays or objects.
- When de-structuring an array, we use their position(or index) in an assignments.

Example, using object.

```
// create an object
const person = {
    firstname: "Raj",
    lastname: "Patel",
    mobileNo: 9856332566,
};
```

```
// destructor
const { lastname, firstname } = person;

console.log(person);
console.log(firstname);
console.log(lastname);
```

Example, using array.

```
const pcs = ["dell", "lenovo", "sony", "hp", "acer"]
const [p1, p2, p3] = pcs;
console.log(p1);
console.log(p2);
console.log(p3);
```

Higher order function – map, reduce and filter

- It is a function that takes one or more function as argument or returns a function as its result.

Example,

```
function calledCallback(callback) {
    // call the callback function
    callback();
}
function newcallFunction() {
    console.log("this is higher function example");
}
calledCallback(newcallFunction);
```

Example,

```
function hello(name) {
    return `Hi ! ${name}`;
}
function display_msg(appriciate, msg, name) {
    console.log(`${appriciate(name)}, ${msg}`);
}
display_msg(hello, "Welcome to our Home", "Urvashi");
```

- There are different type of higher order functions in array like map(), reduce(), sort() and filter().
- In object like Object.entries(). When working with object.

1. map() :
   it takes an array of an values and applied a transformation to each value in array.

```
var num = [1, 2, 3, 4, 5];
var newval = num.map((n) => n += 10);
console.log(num);
console.log(newval);
```

2. filter() :
   it takes an array and returns a new array with only the values that given in criteria.

```
var num = [11, 12, 13, 14, 15]
var task = num.filter((n) => n % 2);
console.log(num);
console.log(task);
```

3. reduce() :
   it reduce the given array into a single value by executing a reducer function.

   Syntax :
   arr.reduce(callback(accumulator, currentValue, currentIndex, array), initialValue)

```
const num = [1, 2, 3, 4, 5];
const sum = num.reduce((total, curr) => {
    return total + curr;
}, 0)
console.log(sum);
```

4. sort() :
   it is used to arrange the array elements in some order.
   By default, it is an ascending order.

```
var subj = ["react", "angular", "node", "vue"];

console.log(subj);
var res = subj.sort();
console.log(res);
```

5. Object.entries() :
   It used to return an array of a given object's own enumerable property pair.
   It is as same as that given by looping over the property values of the object manually.

   Example,

```
var myobj = {
    1: "abc",
    2: "xyz",
    3: "pqr",
}
console.log(Object.entries(myobj)[0]);
console.log(Object.entries(myobj)[1]);
console.log(Object.entries(myobj)[2]);
```

Static method

- It belongs the class instead of an instance of the class. So, an instance of no required to call static method.

- These methods call directly on the class itself.
- Note that,
    - The static keyword used to declare a static method.
    - This method has any name.
    - A class contains more than one static method.
    - If we declare more than one static method with similar name, it calling last method.
    - We can use this keyword to call a static method within another static method.
    - We can't use this keyword directly to call a static method the non-static method.

Example, more than invoke static method.

```
class student {
    static stud_info1() {
        console.log("student information");
    }
    static stud_info2() {
        console.log("another student information");
    }
}
student.stud_info1();
student.stud_info2();
```

Example, more than static method with similar name.

```
class student {
    static stud_info() {
        console.log("student information");
    }
    static stud_info() {
        console.log("another student information");
    }
}
student.stud_info();
```

example, invoke a static method within the constructor.

```
class student {
    constructor() {
        student.stud_info();
    }

    static stud_info() {
        console.log("student information");
    }
}
// calling constructor of student class
var stud = new student();
```

JS Object Oriented Concept

Class

- Classes are a special type of function.
- It defines just like function declaration and function expression.
- It contains various class members within a body including methods or constructor.
- It executed in strict mode, so the code shows silent error or mistake throws an error.
- The class syntax contains two components :
  - Class declarations
  - Class expressions

Class declarations

- It defined by using class declarations.
- class keyword is used to declare a class with any name.

example,

```
<body>
    <script>
        // declare class
        class employee {
            // initialize an object
            constructor(id, name) {
                this.id = id;
                this.name = name;
            }
            // declaring method
            EmpDetails() {
                console.log(`id : ${this.id} and name : ${this.name}`);
            }
        }
        // passing object to a variable
        var e1 = new employee(101, "Raj");
        var e2 = new employee(102, "Rohit");
        e1.EmpDetails();
        e2.EmpDetails()
    </script>
</body>
```

Example, display data in table.

```
<body>
    <table border="1" style="width: 300px; border-collapse:
collapse;">
        <tbody id="title">
            <tr>
                <th>Id</th>
                <th>Name</th>
            </tr>
        </tbody>
        <tbody id="tabledata"></tbody>
    </table>

    <script>
        let udata = document.getElementById("tabledata");
        class employee {
            constructor(id, name) {
                this.id = id;
                this.name = name;
            }

            EmpDetails() {
                udata.innerHTML += `
                    <tr>
                        <td>${this.id}</td>
                        <td>${this.name}</td>
                    <tr>
                `
            }
        }
        var e1 = new employee(101, "Raj");
        var e2 = new employee(102, "Rohit");
        e1.EmpDetails();
        e2.EmpDetails()
    </script>
</body>
```

Class expression

- define a class is by using a class expression.
- The class expression can be named or unnamed.
- It allows us to fetch the class name, however this is not possible with class declaration.

Example,

```
<body>
    <script>
```

```
        var employee = class {
            constructor(id, name) {
                this.id = id;
                this.name = name;
            }
        };
        console.log(`${employee.id} and ${employee.name}`);
    </script>
</body>
```

Encapsulation

- It is a process of binding the data (i.e. variable) with the functions acting on the data.
- It allows us to control the data and validate it.
- Important points are :
    - To make data/variables members private.
    - For set data use setter method and for get data use getter method.
- Use following properties for allow handle object :
    - Read/Write – use set data use setter method and for get data use getter method.
    - Read only – use getter method only.
    - Write only – use setter method only.

Example, prototype based example.

```
function employee(name, sal) {
    var ename = name;
    var salary = sal;

    Object.defineProperty(this, "name", {
        get: function () {
            return ename;
        },
        set: function (ename) {
            this.ename = ename;
        }
    });
    Object.defineProperty(this, "sal", {
        get: function () {
            return salary;
        },
        set: function (salary) {
            this.salary = salary;
        }
    });
}
var emp = new employee("Urvashi", 40000);
```

```
console.log(emp.name);
console.log(emp.sal);
```

inheritance

- It is a mechanism that allows us to create new classes on the basis of existing classes.
- It provides flexibility to the child class to reuse the methods and variables of a parent class.
- **extends** keyword is used to create a child class on the basis of a parent class.
- Child class take all properties and behavior of its parent class.
- Important points :
    - It maintains is IS-A relationship.
    - extends keywords used in class expression or class declarations.
    - Using extends keyword, we can extend properties and behavior of the inbuilt object as well as custom classes.
    - We can use prototype-based approach to achieve inheritance.

Example, extend inbuilt object.

```
class curr_time extends Date {
    constructor() {
        super();
    }
}
var c = new curr_time();
console.log(`Current Date => ${c.getDate()}-${c.getMonth() + 1}-
${c.getFullYear()}`);
```

Example, use custom class.

```
class person {
    constructor() {
        this.firstname = "Urvashi"
    }
}
class emp extends person {
    constructor(company, address) {
        super();
        this.company = company;
        this.address = address;
    }
}
var e = new emp("IT", "Surat");
console.log(`${e.firstname} | ${e.company} | ${e.address}`);
```

Example, prototype-based approach.

```
// constructor
function person(name) {
    this.name = name;
}
person.prototype.getName = function () {
    return this.name;
};
// another constructor
function employee(doj, sal) {
    this.doj = doj
    this.sal = sal
}
var per = new person("Urvashi");
employee.prototype = per; // person treat as a parent of emp
var emp = new employee("10-05-2024", 25000);
console.log(`${emp.getName()} | ${emp.doj} | ${emp.sal}`);
```

Polymorphism

- It concepts of an object-oriented paradigm that provides a way to perform a single action in different forms.
- It provides an ability to call the same method on different JavaScript objects.
- We can pass any type of data members with the methods.

Example, child class object invokes the parent class methods.

```
class main {
    display() {
        console.log("main class invoked");
    }
}
class sub extends main {

}
var s = new sub();
s.display()
```

Example, invoke same method in child class.

```
class main {
    display() {
        console.log("main class invoked");
    }
}
class sub extends main {
    display() {
        console.log("child class invoked");
    }
}
```

```
var cls = [new main(), new sub()];
cls.forEach(function (msg) {
    msg.display()
})
```

Abstraction

- Abstraction is a hiding detail and show only functionalities to the user.
- Important points :
    - We can't create abstract of instance class.
    - It reduce duplication of code.

Example, we can create an instance of abstract class or not.

```
function Student(studentname) {
    this.studentname = studentname;
    throw new Error("you cant create abstract of class");
}
Student.prototype.display = function () {
    return this.studentname
}
var stud = new Student();
```

Example, to archive abstraction.

```
function person() {
     this.personName = "personName";
     throw new Error("you cant create an instance of abstract class");
}
person.prototype.Result = function () {
     return "Person are : " + this.personName;
}
function student(personName) {
     this.personName = personName
}
student.prototype = Object.create(person.prototype);
var stud = new student("Pooja");
console.log(stud.Result());
```

Promise

- It is like real life express a trust.
- In JS, it is an object, which ensure that to produce a single value in a future.
- It is used for managing & tackling an asynchronous operation.
- Importance promises in JS :
    - We know about events & callback functions for handling the data, but it scope is limited.
    - Events are not able to manage & operate asynchronous operations.
    - It is simplest & better approach for handling asynchronous operations efficiency.

- Two possible differences:
  1. It can never fail or succeed twice or more. This can affect only once.
  2. If can neither switch from success to failure, failure to success.
     - If a promise has either succussed or failure & after sometime, if any success & failure callback is added, the current will be invoked, no matter the event happened earlier.

- Promise present in stages:
  1. pending – it is neither rejected nor fulfilled yet.
  2. fulfilled – related promise action is fulfilled successfully.
  3. rejected – related promise action is failed to be fulfilled.
  4. settled – either the action is fulfilled or rejected.

- It represents the completion of an asynchronous operation with its result.

- Promises of promise :
  1. Current execution of the JS event loop is not completed (success or failure). Callbacks will never be called before it.
  2. Even if the callbacks with **then()** are present but they will be called only after the execution of the asynchronous operations completely.
  3. When multiple callbacks can be included by invoking **then()** many time, each of them will be executed in a chain.

- Methods :

Method Name	Description
Promise.resolve(promise)	Returns promise only if promise.constructor == Promise.
Promise.resolve(thenable)	Makes a new promise from thenable containing then().
Promise.resolve(obj)	Makes a promise resolved for an object.
Promise.reject(obj)	Makes a promise rejection for an object.
Promise.all(array)	Makes a promise resolve when each item in an array is fulfilled or rejects when items in the array are not fulfilled.
Promise.race(array)	If any items in the array is fulfilled as soon, it resolves the promise, or if any items is rejected as soon, it reject promise

- Constructor :

Method Name	Description
new Promise(function(resolve, reject){});	Here, resolve(thenable) => the promise will be resolve with then(). resolve(obj) => promise will be fulfilled with the object. reject(obj) => promise rejected with the object.

Example,

```
var pro = new Promise(function (resolve, reject) {
    // var x = 2 + 3;
    var x = 2 + 2;
    if (x == 5) {
        resolve("executed and resolved successfully");
    } else {
        reject("rejected");
    }
});
pro.then(function (fromResolve) {
    console.log("Promise is : " + fromResolve);
}).catch(function (fromReject) {
    console.log("Promise is : " + fromReject);
})
```

Spread and Rest operator

- Spread operator is used to spread an iterable into its elements, while the Rest operator collect multiple elements into an array.

- Rest operator is represented by three dots(...). When used in a function's parameter list, it catches extra argument passed to the function and present into the array.

  Example,

```
function numbers(...nums) {
    return nums;
}
console.log(numbers(10, 1, 2, 3, 4, 5, 6));
```

- Spread operator is represented by three dots(...), works by taking all the items in an array and spreading them out.

  Example,

```
var num1 = [1, 2, 3]
var num2 = [4, 5, 6]

var nums = [...num1, ...num2]
console.log(nums);
```

  Example,

```
const person = {
    firstname: "urvashi",
    lastname: "bhavsar"
}
const per = { ...person };
console.log(per);
```

  Example,

```
function addition(num1, num2, num3) {
    return num1 + num2 + num3;
}
console.log(addition(...[45, 54, 22, 33]));
```

Rest parameter

Example,

```
// rest parameter

function sum(...input) {
    let tot = 0;
    for (let i of input) {
        tot = tot + i;
    }
    return tot
}
console.log('====================');
console.log(sum(1, 3));
console.log(sum(1, 3, 5, 8));
console.log(sum(1, 3, 5, 8, 11, 43));
```

Async

- Async function allows us to promise based code as if it were synchronous.
- This ensures that the execution thread is not blocked.
- Async functions always return a promise. If the value is return that is not a promise, JS automatically wraps it in a resolved promise.

Await

- Await is used to wait for a promise to resolve, it used in async block.
- It makes a code wait until the promise returns a result.
- It allowing the clear and more manageable async code.

Example,

```
const getdata = () => {
    let data = "Get data 1"
    return data;
}
console.log(getdata());
console.log("----------------------------");

const getdata1 = async () => {
    let data = "Get data 2"
    return data;
}
getdata1().then(data => console.log(data));
console.log("----------------------------");

const getdata2 = async () => {
    let data = await "Get data 3"
    console.log(data);
```

```
}
console.log("Hello");
getdata2();
console.log("World");
```

Local storage

- It is object of the Web Storage API provides access to the session storage or local storage for a particular domain.
- It allows to store key-value in browser.
- This allows you to read, add, modify, and delete stored data items.

- Syntax :

  mystore = window.localStorage

- Methods :

	Method	Description
1	setItem(key, value)	Store key/value pair.
2	getItem(key)	Return key value.
3	key(index)	Get keys at a given index.
4	removeItem(key)	Remove the given key with its value
5	clear()	Delete everything from storage.
6	length property	Return the number of stored items (data)

Example,

```
<body>

    <button onclick="setStorage()">Set Storage</button>
    <button onclick="getStorage()">get Storage</button>
    <button onclick="removeVal()">remove Storage</button>
    <button onclick="clearval()">clear</button>
    <button onclick="countlength()">Length</button>

    <script>
        function setStorage() {
            window.localStorage.setItem("1", "Urvashi")
        }
        function getStorage() {
            var data = window.localStorage.getItem("1")
            alert(data)
        }
        function removeVal() {
            localStorage.removeItem("1");
        }
```

```
        function clearval() {
            localStorage.clear()
        }
        function countlength() {
            alert(localStorage.length)
        }
    </script>
</body>
```