Topic: ES -6 concepts

ES -6 concepts

ES6, or ECMAScript 2015, brought significant enhancements to JavaScript, introducing many new features and syntax improvements.
let and const:

- **let** allows declaring block-scoped variables that can be reassigned.
- **const** declares constants whose values cannot be reassigned.

Arrow Functions:

Provides a concise syntax for writing functions, with an implicit return if the function body is a single expression.

```
const add = (a, b) => a + b;
```

Template Literals:

Allow embedding expressions inside strings using ${} syntax.

```
const name = "John";
console.log(`Hello, ${name}!`);
```

Destructuring Assignment:

Allows extracting values from arrays or objects into distinct variables.
Destructuring is a process assigining a value to a single varaible
Types
Array
Object

Spread Syntax:

- Allows expanding iterable elements in places where zero or more arguments or elements are expected.it creates the deep copy

```
const arr = [1, 2, 3];
const newArr = [...arr, 4, 5];
```

Rest Parameters:

* Rest operator is introduced in ES6

* Rest operator are used in function parameters

* Rest operator is denoted by ...

* Rest operator is implicitly an array

* Rest operator will accepts from 0 to n number of arguments

* Rest operator should be the last parameter of function

* In Functions parameters rest operators can be only one

* Rest Operator can be used in array/ object destructuring

* Rest Operator is used for better readibility

* **Syntax :**

* ...varName (passed as function parameter)

Allows representing an indefinite number of arguments as an array.

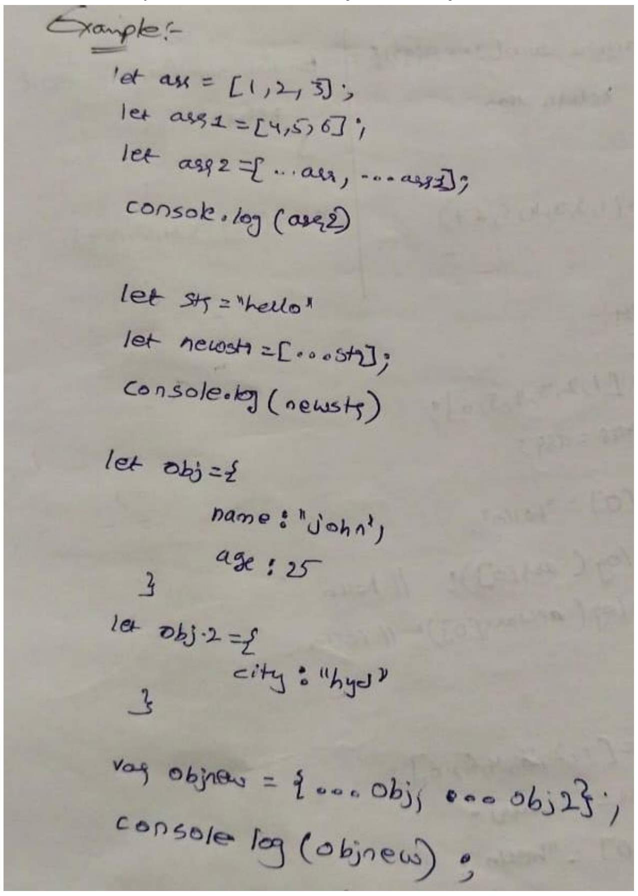
```
function sum(a,b,...args) {
   return …args
}
Sum(1,2,3,4,5,6,7)
```

Classes:
Modules:
Promises:
Async/Await:

Spread operator

spread operator (...) in JavaScript is a powerful tool used to expand elements of an iterable (like arrays, strings, or objects) into individual elements. It's often used for array manipulation, function arguments, and object spreading.

* Spread operator is used to spread the data
* which is present in array and string
* Spread operator is used as function argument
* Spread operator will create deep copy for array and object
* S+6++pread is denoted by ...
* for array [...arrRef] , for object {...objectRef}


Example:-

```
let arr = [1,2,3];
let arr1 = [4,5,6];
let arr2 =[...arr, ...arr1];
console.log (arr2)

let str = "hello"
let newstr =[...str];
console.log (newstr)

let obj ={
      name : "John",
      age : 25
 }
let obj.2 ={
      city : "hyd"
 }

var objnew = {...obj, ...obj2};
console log (objnew);
```

Expanding an Array:

```
const arr1 = [1, 2, 3];
const arr2 = [...arr, 4, 5, 6];
console.log(arr2); // Output: [1, 2, 3, 4, 5, 6]
```

Passing Arrays as Function Arguments:

```
const numbers = [1, 2, 3, 4, 5];
const sum = (a, b, c, d, e) => a + b + c + d + e;
console.log(sum(...numbers)); // Output: 15
```

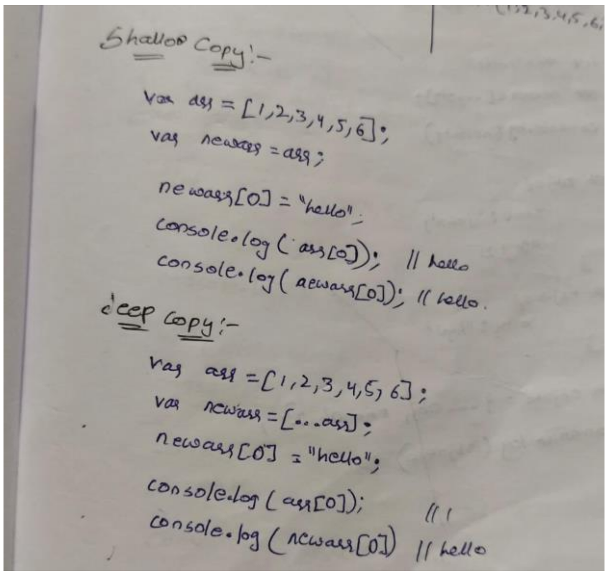
Merging Objects:

```
const obj1 = { name: 'john' };
const obj2 = { age: 23 };
const mergedObj = { ...obj1, ...obj2 };
console.log(mergedObj); // Output: { name: 'john', age: 23 }
```

shallow copy with the example

```
const originalArray = [1, 2, 3, 4, 5];
const shallowCopyArray = [...originalArray]; // Shallow copy using the rest operator
shallowCopyArray[0] = 10; // Modify the shallow copy

console.log(originalArray); // Output: [1, 2, 3, 4, 5]
console.log(shallowCopyArray); // Output: [10, 2, 3, 4, 5]
```



Shallow Copy:-

```
var arr = [1,2,3,4,5,6];
var newarr = arr;

newarr[0] = "hello".
console.log( arr[0]);   // hello
console.log( newarr[0]); // hello.
```

deep Copy:-

```
var arr =[1,2,3,4,5,6];
var newarr =[...arr];
newarr[0] ="hello";
console.log(arr[0]);        // 1
console.log(newarr[0])   // hello
```

Why spread operator introduced

```
var a=[12,2,2,22,]  //create an array
var b=a;  //assigned a array to another variable
b[0]=500; //change the second variable value
console.log(a);//500,2,2,22
```

here value of array a is also changed because here value is not assigned . memory location is assigned in order to overcome this spread operator was introduced.

In JavaScript, when we assign an array or object to another variable, we do not create a new independent copy of the original array or object. Instead, we assign a reference to the original memory location. This means that any changes made to the new variable will also reflect in the original array or object.

Understanding Shallow Copy

The behavior observed in the above example is due to shallow copying, where only the reference to the original memory location is copied, not the actual data. To overcome this and create an independent copy of the array, we can use the spread operator or other methods

By using spread operator

```
var a=[12,2,2,22,]  //create an array
var b=[...a];  //create a deep copy of an array using spread operator
b[0]=500; //change the second variable value
console.log(a);//12,2,2,22    --here value of a is not changed
console.log(b);//500,2,2,22   --value of b is only changed because spread operator creates a copy of an array
```

Call by value and call by reference

understanding the difference between call by value and call by reference is crucial for managing how data is passed and manipulated within your code

Call by Value:

Call by Value means that when a variable is passed to a function, the value of the variable is passed. If the variable is a primitive type (like number, string, boolean, null, undefined, symbol, and bigint), its value is copied to the function parameter. Changes to the parameter do not affect the original variable.

```
function changePrimitive(val) {
  val = 100;
}
let num = 50;
changePrimitive(num);
console.log(num); // Output: 50
```

num remains 50 even after calling changePrimitive because the function works with a copy of num.

Call by Reference

Call by Reference means that when a variable is passed to a function, a reference to the variable is passed. If the variable is an object (like arrays, functions, objects, etc.), its reference is passed to the function. Changes to the parameter will affect the original object.

```
function changeObject(obj) {
  obj.name = "John";
}

let person = { name: "Doe" };
changeObject(person);
console.log(person.name); // Output: John
```

the name property of the person object changes to "John" because the function works with a reference to the person object.

Key Points

1. **Primitive Types (Call by Value)**:
    - number, string, boolean, null, undefined, symbol, bigint
    - Changes inside the function do not affect the original variable.
2. **Reference Types (Call by Reference)**:
    - object, array, function
    - Changes inside the function affect the original object or array.

Destructuring

Destructuring in JavaScript is a powerful feature that allows you to extract values from arrays or properties from objects and bind them to variables in a concise and expressive way. It provides a convenient syntax for extracting data from arrays and objects.

Array Destructuring:

1. **We can destructure values individually**
2. **We can can skip elements in the array by using commas.**
3. **If the value at the specified position is undefined, you can assign a default value.**
4. **We can use rest operator (...) to collect the remaining elements into a new array.**
5. **We can destructure nested arrays**

```
const numbers = [1, 2, 3, 4, 5];
const [first, second, ...rest] = numbers; // Extracting values from the array into variables
console.log(first); // Output: 1
console.log(second); // Output: 2
console.log(rest); // Output: [3, 4, 5]
```

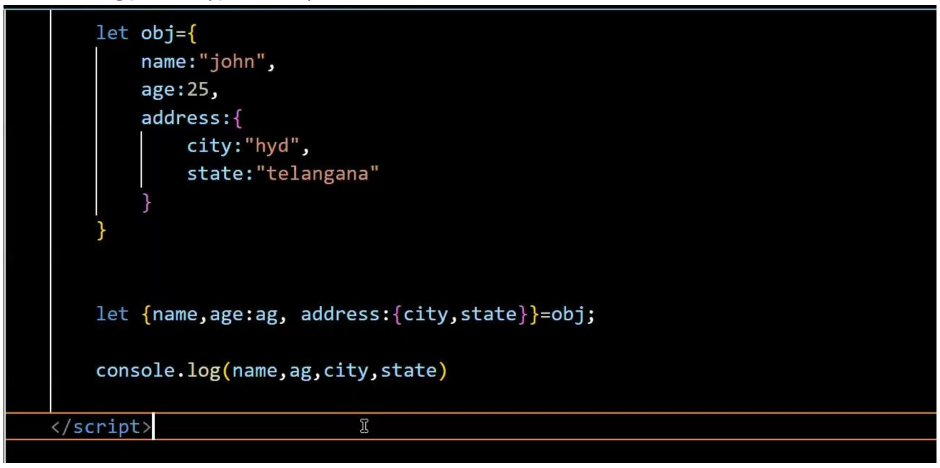
Object Destructuring:

1. **We can destructure values individually**
2. **We can rename variables while destructuring by using a colon (:).**
3. **We can set default values for properties that might be undefined.**
4. **Destructuring can be used to extract values from nested objects.**
5. **We can pass remaining values using rest operator**

const person = { name: 'John', age: 30, city: 'New York' };
const { name, age } = person; // Extracting properties from the object into variables
console.log(name); // Output: 'John'
console.log(age); // Output: 30

Nested objects destructuring

const person = { name: 'John', age: 30, address: { city: 'New York', country: 'USA' } };
const { name, address: { city, country } } = person; // Nested destructuring
console.log(name); // Output: 'John'
console.log(city); // Output: 'New York'
console.log(country); // Output: 'USA'



```
    let obj={
        name:"john",
        age:25,
        address:{
            city:"hyd",
            state:"telangana"
        }
    }


    let {name,age:ag, address:{city,state}}=obj;

    console.log(name,ag,city,state)

</script>
```