JavaScript ES6

The JavaScript ES6 (also called ECMAScript 2015) defines a new set of rules governing the syntax of the JavaScript code.

Feature	Description
let and const keywords	These are used to create block-level variables
Arrow Functions	It reduced the length of the code for creating a function
Classes	These are the templates to create objects
Default parameter values	It allows to pass the argument through a function by default
find() and findIndex method	It helps you to find the element or index of an array
Exponentiation (**)	It raises the first operand to the power of the second operand

Difference between ES5 and ES6

ES5 and ES6 are scripting language standards by ECMA International, with key differences summarized in the table below:

Specification	ES5	ES6
Edition	Fifth edition released in 2009	Sixth edition released in 2016
Datatypes	Supports primitive data types(strings, Number, Boolean, null, and undefined)	In addition to the ES5 primitive datatypes, it supports new datatype(symbol)
Variable Definition	In ES5, we define the variable using only var keyword	In ES6, there are three ways to define a variable with var, let and const
Arrow functions	Not supported in ES5	Supported in ES6
Default params	Not supported	Supported
Template strings	Not supported	Supported
Object destructing	Not supported	Supported

var vs let vs const

Following are the difference between var, let, and const:

behavior	var	let	const
Hoisting	Exhibits hoisting	Does not exhibit hoisting	Does not exhibit hoisting
Scope	Functional scope	Block scope	Block scope
Mutability	mutable	mutable	immutable

Arrow Functions

- Arrow functions will reduce the length of the syntax.
- There is no binding of the `this` keyword in these functions.
- In arrow functions, `this` keyword always represents the object that defined the arrow function.
- If there is only one line of code, you can ignore the brackets and return the keyword of the function.

Syntax:

```
functionName = () => {
    .......
    .......
    ........
return value;
}
```

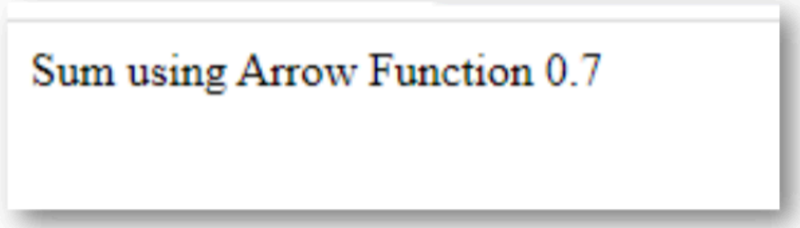
Block of code

Arrow Functions - Example

Code:

```
<script>
    const div = (a, b) => a / b;
    document.write("Div using Arrow Function " +div(7, 10));
</script>
```

Example:



Template String

- They are a new kind of string literals introduced in ES6.
- Contains features like multi-line strings and string interpolation.
- They allow embedded expressions.
- Strings are enclosed within backtick(`) instead of single or double quotes.

Example:

```
`string text`

`string text line 1
string text line 2`

`string text ${expression} string text`
```

Spread Operator

- The JavaScript spread operator (...) allows us to quickly copy all or part of an existing array or object into another array or object.
- The spread operator is often used in combination with destructuring.
- Application of Spread Operator
  - Copying Array or Object
  - Cloning Array or Object

Syntax:

```
[...iterableObj, items, ..,itemN];   //For array literals
[...iterableObj, “items”,...,”itemN”];   //For string literals
function(...iterableObj);   //passing iterable object to function
{…obj}; //For array literals
```

Spread Operator - Example

Code :

```
<script>
    let arr1 = [1, 2, 3];
    let arr2 = [4, 5, 6];
    //Copying array using spread
    let big = [...arr1, ...arr2];
    //Cloning the arrays
    let arr3 = big;
    document.write(big);
    big.push(10);
    //Will affect both the arr3 & big
    document.write(`Value of Big ${big} & arr ${arr3} `);
</script>
```

Output:

1,2,3,4,5,6
Value of Big 1,2,3,4,5,6,10
& arr 1,2,3,4,5,6,10

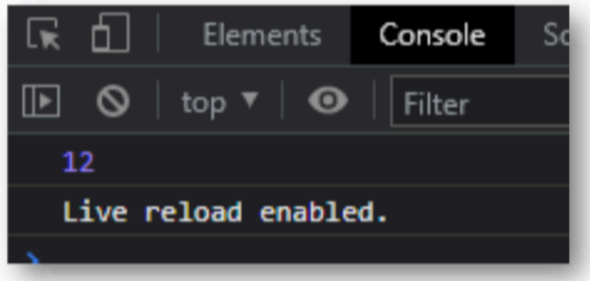
Rest Operator

- The Rest Operator in JavaScript allows a function to accept an indefinite number of arguments as an array, providing a way to represent variadic functions in JavaScript.
- It simplifies handling multiple arguments in functions, leading to cleaner and more adaptable code.

Syntax :

```
<script>
  var sum = (a, ...b) => {
    let sum = 0;
    for (let value of b) {
      sum = sum + value;
    }
    return sum + a;
  };
  let res = sum(7, 4, 1);
  console.log(res);
</script>
```

Output :



The rest operator will take an infinite number of values and store them in an array.

Object Destructuring

- Object destructuring allows you to extract properties from objects and bind them to variables.
- There are two ways of object destructuring:
  - Using dot(.) operator.
  - By assigning properties of an object to variables.

Syntax:

```
keyword variableName = objectName.property;

keyword {propertyName,…,propertyN} = objectName;
```

Example:

```
var rno = studObj.rollNo;

let {rollNo, name} = studObj;
```

Object Destructuring - Example

Code:

```
<script>
     var studObj = {
          rno : 101,
         name : "Alexa"
     }
     // old way
     const roll = studObj.rno;
     document.write(`Value using dot operator ${roll}
<br>`);

     // new way
     const {rno, name} = studObj;
     document.write(`Value using new way ${rno},${name}`);
```

Output:

Value using dot operator 101
Value using new way 101,Alexa