**features of ES6 like arrow functions, callbacks, promises, async/await**

ECMA : European Computer Manufacturers Association abbreviated as ECMA

Features of ES6:

1.Arrow functions:

ES6 Arrow functions enable us to write functions with simpler and shorter syntax and make our code more readable and organised. The arrow functions are introduced in the ES6 version. Arrow functions provides us with a more precise approach to writing JavaScript Functions.

Simple function example ---without arrow

function myFunction(g1, g2) {

return g1 / g2;

}

const value = myFunction(8, 2); // Calling the function

console.log(value);

---------------------------------------------------------------------

function calcAddition(number1, number2) {

return number1 + number2;

}

console.log(calcAddition(6,9));

with arrow

Syntax:

For Single Argument:

let function\_name = argument1 => expression

example program

function multiply(a, b) {

return a \* b;

}

console.log(multiply(3, 5));

----------------------------------------------------------

// Arrow function for multiplying two numbers

value = (a, b) => a \* b;

console.log(value(3, 5));

number = (a, b) => {

c = 5;

return (a + b) \* c;

};

console.log(number(2, 3));

2.callbacks:

*I will call back later!*

A callback is a function passed as an argument to another function

This technique allows a function to call another function

A callback function can run after another function has finished

function mainFunction(callback) {

console.log("Performing operation...");

setTimeout(function() {

callback("Operation complete");

}, 1000);

}

function callbackFunction(result) {

console.log("Result: " + result);

}

mainFunction(callbackFunction);

Output

Performing operation...

Result: Operation complete

var numbers = [1, 2, 3, 4, 5];

function mainFunction(callback) {

console.log("Performing operation...");

numbers.forEach(callback);

}

function callbackFunction(number) {

console.log("Result: " + number);

}

mainFunction(callbackFunction);

Output :

Performing operation...

Result: 1

Result: 2

Result: 3

Result: 4

Result: 5

3.promises:

“*I Promise a Result!"*

A Callback is a great way when dealing with basic cases like minimal asynchronous operations. But when you are developing a web application that has a lot of code, then working with Callback will be messy. This excessive Callback nesting is often referred to as Callback hell.

To deal with such cases, we have to use Promises instead of Callbacks.

Creating a Promise

In JavaScript, we can create a Promise by using the Promise() constructor.

Syntax

const Promise = new Promise((resolve,reject) => {....});

Example

let Promise = new Promise((resolve, reject)=>{

let a = 3;

if(a==3){

resolve('Success');

}

else{

reject('Failed');

}

})

Promise.then((message)=>{

console.log("It is then block. The message is: ?+ message)

}).catch((message)=>{

console.log("It is Catch block. The message is: ?+ message)

})

Output

It is then block. The message is: Success

4.asysnc/await:

The Async/Await functionality is one of them. Async/Await is an extension of promises that we get as language support.

The async and await keywords allow asynchronous, **promise-based** behavior to be written more easily and avoid configured promise chains.

<script>

let dataDisplay = async () => {

let data = await "world";

console.log(data); };

console.log(1);

dataDisplay();

console.log("Hello");

</script>

o/p:

1

Hello

World