JS ECMA Script 2015

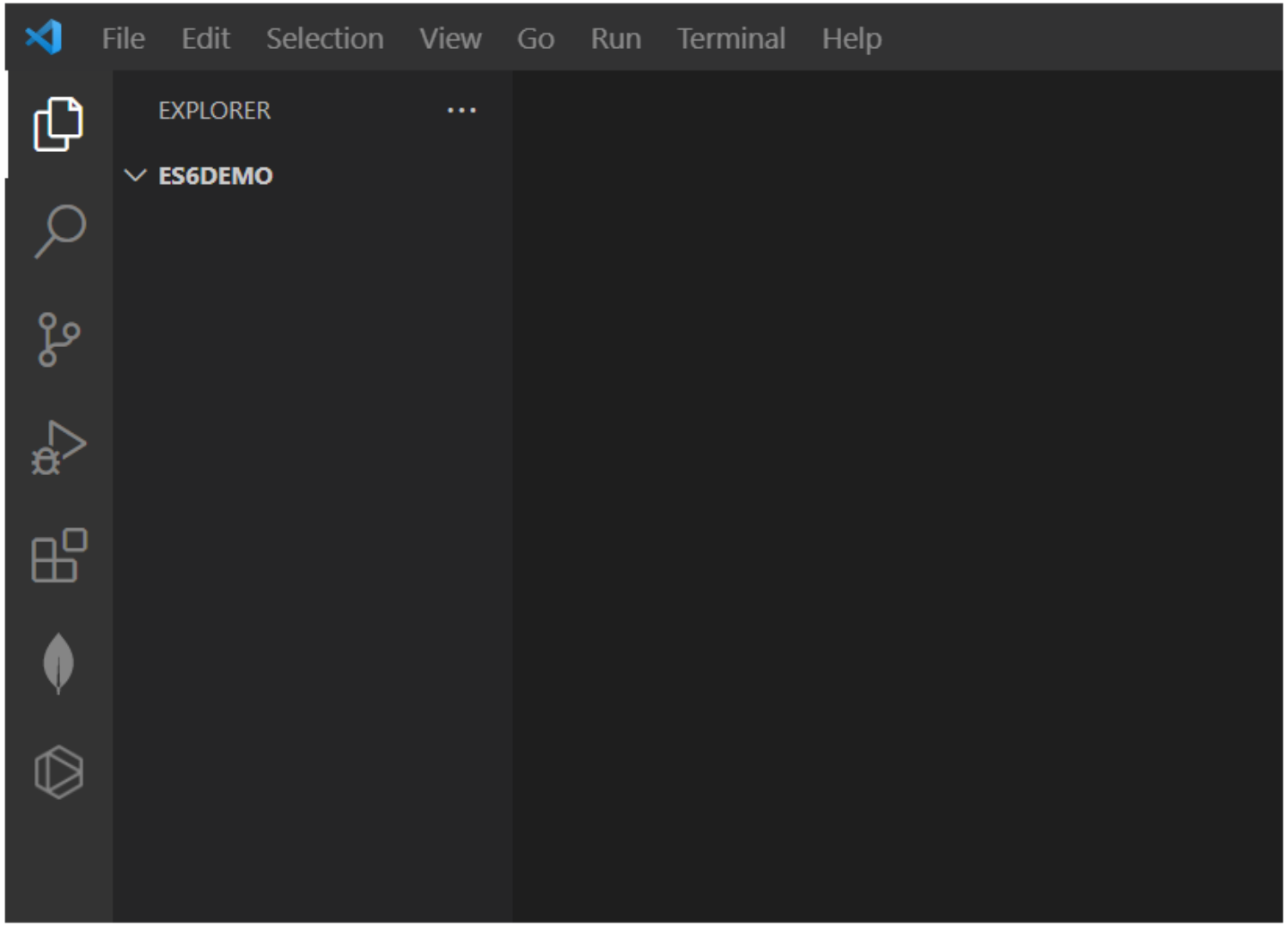
#JavaScript Notes

ECMA Script 2015
ES6

JS

Demo

Create new Folder and Open in VS Code



Create Demo.html File Write Internal JS Code

File Edit Selection View Go Run Terminal Help demo.html - es6demo - Visua

EXPLORER

ES6DEMO

demo.html

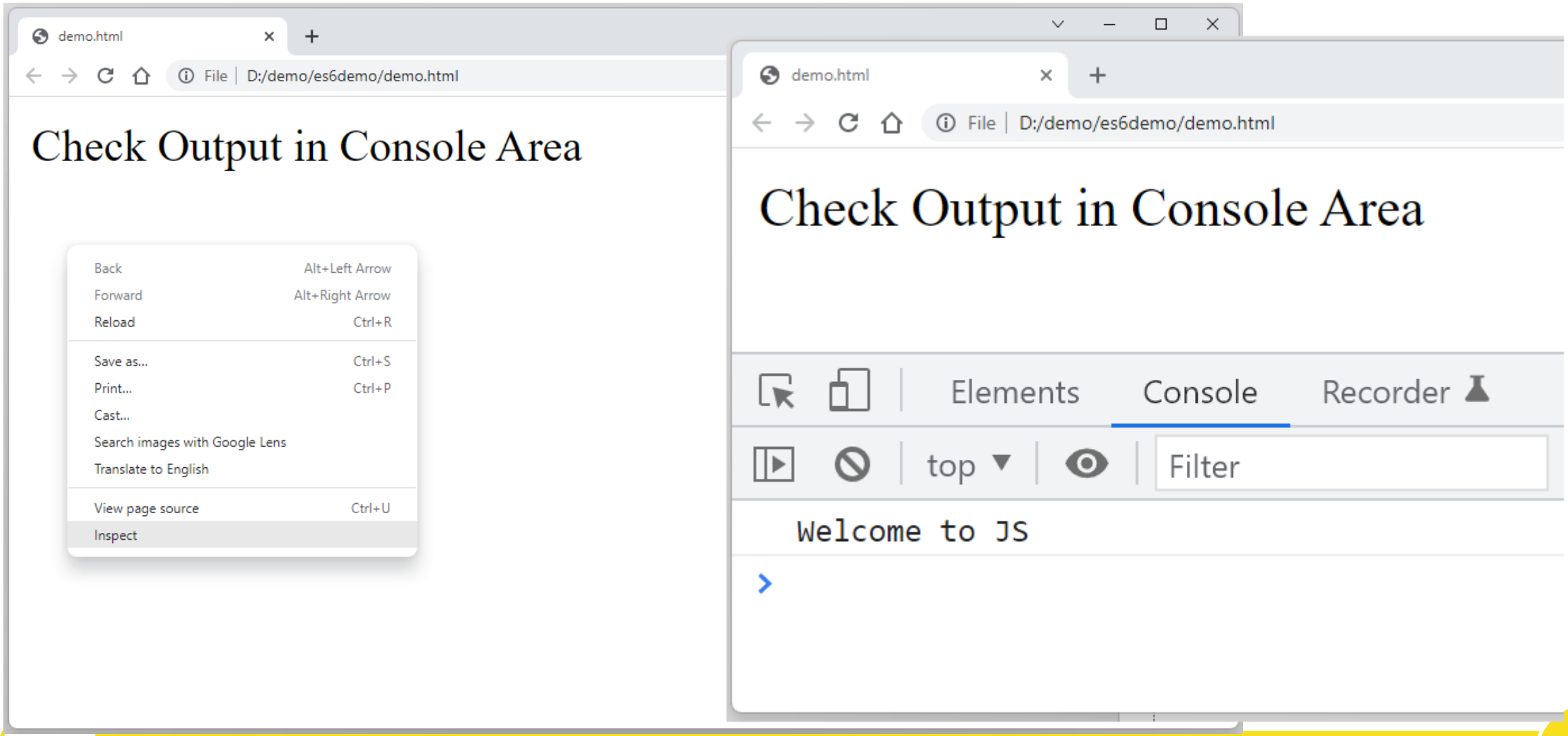
demo.html

demo.html > html > body

```html
<html>
    <head>
        <script>
            console.log("Welcome to JS");
        </script>

    </head>
    <body>
            Check Output in Console Area
    </body>
</html>
```

Console Area



External JS Example

EXPLORER

ES6DEMO

- demo.html
- demo.js

demo.html > html

```html
<html>
<head>
    <script type="text/javascript" src="demo.js"></script>
</head>
    <body>
        Check Output in Console Area
    </body>
</html>
```

demo.js

```js
console.log("Hello Js")
```

Console Log,Error,Warn

Console Log,Error,Warn

- The console provides a simple debugging console.
- Used to print information on stdout and stderr.
- It uses built-in methods for printing informational, warning and error messages.

Console Log,Error,Warn

demo.html > html > head

```html
<html>
<head>
    <script>
        console.log('hello world');
        // Prints hello world, to stdout

        console.log('hello %s', 'world');
        // Prints hello world, to stdout

        console.error(new Error('Error'));
        // Prints Error: 'Error', to stderr

        const name = 'Global Objects';
        console.warn(`Danger ${name}! Danger!`);

    </script>

</head>
    <body>
            Check Output in Console Area
    </body>
</html>
```

Elements Console Sources Network

```
<html>
▶ <head>…</head>
… <body> Check Output in Console Area </body> == $
```

html body

Console Issues

top ▼ Filter

```
hello world
hello world
▶ Error: Error
    at demo.html:10:23
▶ Danger Global Objects! Danger!
>
```

Call Back

JavaScript Callbacks

- Callbacks are a great way to handle something after something else has been completed. By something here we mean a function execution.
- Callback Function : “A function is a block of code that performs a certain task when called. “

Benefit of Callback Function

- The benefit of using a callback function is that you can wait for the result of a previous function call and then execute another function call.

```
// function
function greet(name, callback) {
   console.log('Hi' + ' ' + name);
   callback();
}
// callback function
function callMe() {
   console.log('I am callback function');
}
// passing function as an argument
greet('Akash', callMe);
```

demo.js > ...

```javascript
// Function
function greet(name, callback) {   //2
    console.log('Hi' + ' ' + name);
    callback();
}

// Callback Function
function callMe() {  //3
    console.log('I am callback function');
}

// Passing function as an argument
greet('Akash', callMe);  //1
```

Console | Issues ×

top | Filter

```
Hi Akash
I am callback function
>
```

setTimeout

setTimeout(cb, ms)

- The setTimeout() calls a function (cb) after a specified number of milliseconds (ms).
- The timeout must be in the range of 1-2,147,483,647 inclusive.
- If the value is outside that range, it's changed to 1 millisecond.

- setTimeout(function, milliseconds);
  - function - a function containing a block of code
  - milliseconds - the time after which the function is executed

Example

```
setTimeout(function(){
    console.log('I have come after 500 miliseconds')},500);
```

```
demo.js > ...
setTimeout(function(){
console.log('I have come after 500 miliseconds')},500);

```

Call Back and SetTimeout Example

demo.js > ...

```js
// program to display time every 3 seconds
function showTime() {

    // return new date and time
    let dateTime= new Date();
    // returns the current local time

    let time = dateTime.toLocaleTimeString();
    console.log(time)

    // display the time after 3 seconds
    setTimeout(showTime,3000);
}

// calling the function
showTime();
```


```js
// program to display time every 3 seconds
function showTime() {

    // return new date and time
    let dateTime= new Date();
    // returns the current local time

    let time = dateTime.toLocaleTimeString();
    console.log(time)

    // display the time after 3 seconds
    setTimeout(showTime,3000);
}

// calling the function
showTime();
```

clearTimeout(t)

- The clearTimeout() is used to cancel a timeout that was set with setTimeout(). The callback will not execute.

setInterval

setInterval(cb, ms)

- setinterval() calls a function (cb) repeatedly at specified intervals (in milliseconds (ms)).
- The interval must be in the range of 1-2,147,483,647 inclusive.
- If the value is outside that range, it's changed to 1 millisecond.

Example

```
setInterval(function(){
    console.log('Welcome to Node.js')
    },500);
```

demo.js > ...

```
setInterval(function(){
    console.log('Welcome to Node.js')
    },500);
```

92 messages
92 user mes...
No errors
No warnings
92 info
No verbose

Expression
undefined

```
Welcome to Node.js
Welcome to Node.js
Welcome to Node.js
Welcome to Node.js
Welcome to Node.js
Welcome to Node.js
```

clearInterval(t)

- The clearInterval() is used to stop a timer that was set with setInterval(). The callback will not execute.

ES 6

Index

- Let Const
- Template Literals
- Arrow Functions
- Spread Operator
- Rest Parameters
- Promises

Let,Const

JavaScript let

- let is similar to var but let has scope.
- let is **only accessible in the block** level it is defined.

```
let a = 0;
console.log(a); //0

if (true) {
    let a = 10;
    console.log(a); //10
  }
console.log(a); // 0
```

```
1   let a = 0;
2   console.log(a); //0
3
4   if (true) {
5       let a = 10;
6       console.log(a); //10
7     }
8   console.log(a); // 0
```

JavaScript const

- Const is used to assign a constant value to the variable.
- And the value cannot be changed. Its fixed.

```
const a = 10;
console.log(a); //Print 10

a = 50; //Error

```

JavaScript Template Literals (Template Strings)

JS

Template Literals

- Template literals provide an easy and clean way create multi-line strings and perform string interpolation.
- Now we can embed variables or expressions into a string easily .
- They are enclosed in backticks ``.

Template Literals Example

demo.js > ...

```js
let str = `Template literal in ES6`;

console.log(str);// Template literal in ES6
console.log(str.length); // 23
console.log(typeof str);// string
```

Multiline Strings Using Template Literals

- Template literals also make it easy to write multiline strings.

```js
// using the + operator
const message1 = 'This is a long message\n' +
'that spans across multiple lines\n' +
'in the code.'

console.log(message1)
```

demo.js > ...

```js
// Simple multi-line string
let a = `Hello
            How Are you ?.
                Thanks`;

console.log(a); //Print Data
```

Variable Expression

- Variables or expressions can be placed inside the string using the ${...}

```
demo.js > ...
1   const myname = 'Akash';
2   console.log(`Hello ${myname}!`); // Hello Akash!
3
```

```
const myname = 'Akash';
console.log(`Hello ${myname}!`); // Hello Akash!
```

Sum of 2 Numbers

```
demo.js > ...
1   // String with embedded variables and expression
2   let a = 10;
3   let b = 20;
4   let result = `The sum of ${a} and ${b} is ${a+b}.`;
5   console.log(result); // The sum of 10 and 20 is 30.
```

```
// String with embedded variables and expression
let a = 10;
let b = 20;
    let result = `The sum of ${a} and ${b} is ${a+b}.`;
    console.log(result); // The sum of 10 and 20 is 30.
```

Without Template Literals

- In the earlier versions of JavaScript, you would use a single quote '' or a double quote "" for strings.
- To use the same quotations inside the string, you can use the escape character \.

```
const str1 = 'This is a string';

// cannot use the same quotes
const str2 = 'A "quote" inside a string';  // valid code
const str3 = 'A 'quote' inside a string';  // Error

const str4 = "Another 'quote' inside a string"; // valid code
const str5 = "Another "quote" inside a string"; // Error
```

```
// escape characters using \
const str3 = 'A \'quote\' inside a string';  // valid code
const str5 = "Another \"quote\" inside a string"; // valid code
```

With Template Literals

- Instead of using escape characters, you can use template literals. For example,

```
const str1 = `This is a string`;
const str2 = `This is a string with a 'quote' in it`;
const str3 = `This is a string with a "double quote" in it`;
```


```
const str1 = `This is a string`;
const str2 = `This is a string with a 'quote' in it`;
const str3 = `This is a string with a "double quote" in it`;
```


- As you can see, the template literals not only make it easy to include quotations but also make our code look cleaner.

Arrow

JavaScript Arrow Function

- In the ES6 version, you can use arrow functions to create function expressions.
- Use the (...args) => expression; to define an arrow function.
- Use the (...args) => { statements } to define an arrow function that has multiple statements.

```
// Function expression
let x = function(x, y) {
    return x * y;
}

//Arrow Function
let x = (x, y) => x * y;
```

demo.js > ...

```
// Function expression
let x = function(x, y) {
    return x * y;
}

//Arrow Function
let x = (x, y) => x * y;
```

Example 1: Arrow Function with No Argument

- If a function doesn't take any argument, then you should use empty parentheses.

```
demo.js > ...
// Function expression
let msg = () => console.log("Hello World")
msg(); // Hello World
```

```
// Function expression
let msg = () => console.log("Hello World")
msg(); // Hello World
```

Example 2: Arrow Function with One Argument

- If a function has only one argument, you can omit the parentheses.

demo.js > ...

```js
// Function expression
let msg = x => console.log(x)
msg("Hello World"); // Hello World
```

```
// Function expression
let msg = x => console.log(x)
msg("Hello World"); // Hello World
```

Arrow Function with Argument

```
demo.js > ...
1   let add = (x, y) => x + y;
2
3   console.log(add(10, 20)); // 30;
```

let add = (x, y) => x + y;

console.log(add(10, 20)); // 30;

Example 3: Arrow Function as an Expression

- You can also dynamically create a function and use it as an expression.

```
let age = 5;

let welcome = (age < 18) ?
  () => console.log('Baby') :
  () => console.log('Adult');

welcome(); // Baby
```

demo.js > ...

```
1  let age = 5;
2
3  let welcome = (age < 18) ?
4    () => console.log('Baby') :
5    () => console.log('Adult');
6
7  welcome(); // Baby
8
```

Example 4: Multiline Arrow Functions

- If a function body has multiple statements, you need to put them inside curly brackets {}.

```
let sum = (a, b) => {
    let result = a + b;
    return result;
}

let result1 = sum(5,7);

console.log(result1); // 12
```

demo.js > ...

```
1   let sum = (a, b) => {
2       let result = a + b;
3       return result;
4   }
5
6   let result1 = sum(5,7);
7
8   console.log(result1); // 12
9
```

Spread operator

Spread operator

- ES6 provides a new operator called **spread operator that consists of three dots (...)**. The spread operator allows you to spread out elements of an iterable object such as an array, map, or set.

```
demo.js > ...
1   const odd = [1,3,5];
2   const combined = [2,4,6, ...odd];
3   console.log(combined); // [ 2, 4, 6, 1, 3, 5 ]
```

```
const odd = [1,3,5];
const combined = [2,4,6, ...odd];
console.log(combined); // [ 2, 4, 6, 1, 3, 5 ]
```

1) Constructing array literal

- The spread operator allows you to insert another array into the initialized array when you construct an array using the literal form.

```
demo.js > ...
1   let initialChars = ['A', 'B'];
2   let chars = [...initialChars, 'C', 'D'];
3   console.log(chars); // ["A", "B", "C", "D"]
```

```
let initialChars = ['A', 'B'];
let chars = [...initialChars, 'C', 'D'];
console.log(chars); // ["A", "B", "C", "D"]
```

2) Concatenating arrays

- Also, you can use the spread operator to concatenate two or more arrays:

```
demo.js > ...
1   let numbers = [1, 2];
2   let moreNumbers = [3, 4];
3   let allNumbers = [...numbers, ...moreNumbers];
4   console.log(allNumbers); // [1, 2, 3, 4]
```

```
let numbers = [1, 2];
let moreNumbers = [3, 4];
let allNumbers = [...numbers, ...moreNumbers];
console.log(allNumbers); // [1, 2, 3, 4]
```

3) Copying an array

- In addition, you can copy an array instance by using the spread operator:

```
demo.js > ...
1   let scores = [80, 70, 90];
2   let copiedScores = [...scores];
3   console.log(copiedScores); // [80, 70, 90]
```

```
let scores = [80, 70, 90];
let copiedScores = [...scores];
console.log(copiedScores); // [80, 70, 90]
```

Rest Parameter

Rest Parameter

- When the spread operator is used as a parameter, it is known as the rest parameter.
- You can also accept multiple arguments in a function call using the rest parameter.

Example

- When a single argument is passed to the func() function, the rest parameter takes only one parameter.
- When three arguments are passed, the rest parameter takes all three parameters.
- Using the rest parameter will pass the arguments as array elements.

demo.js > ...

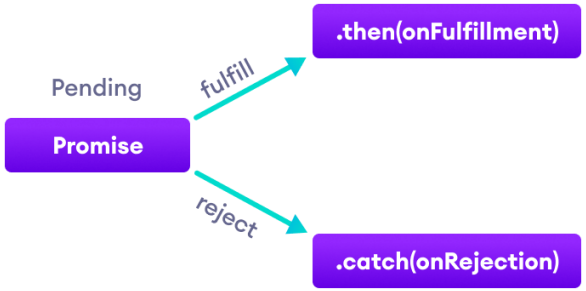
```js
let myfunc = function(...args) {
    console.log(args);
}

myfunc(3); // [3]
myfunc(4, 5, 6); // [4, 5, 6]
```

Promises

Promises

- A promise is basically an advancement of callbacks in Node. While developing an application you may encounter that you are using a lot of nested callback functions.
- A promise is an object that allows you to handle asynchronous operations. It's an alternative to plain old callbacks.
- Promises have many advantages over callbacks. To name a few:
  - Make the async code easier to read.
  - Provide combined error handling.
  - Better control flow. You can have async actions execute in parallel or series.
  - Promises are used to handle asynchronous http requests.



Promise Object Properties

- Pending
  - While a Promise object is "pending" (working), the result is undefined.
- Fulfilled
  - When a Promise object is "fulfilled", the result is a value.
- Rejected
  - When a Promise object is "rejected", the result is an error object.

Call Back vs Promises

```
a(() => {
  b(() => {
    c(() => {
      d(() => {
        // and so on ...
      });
    });
  });
});
```

```
Promise.resolve()
  .then(a)
  .then(b)
  .then(c)
  .then(d)
  .catch(console.error);
```

Syntax

- **then()** : is invoked when a promise is either resolved or rejected.
- **catch()** : is invoked when a promise is either rejected or some error has occured in execution.
- Syntax :

```
.then(function(result){
        //handle success
    }, function(error){
        //handle error
    })
```

- *catch()* is invoked when a promise is either rejected or some error has occured in execution.

Example

```
var mypromise = new Promise(function(resolve, reject) {
  const x = 100;
  const y = 100;
  if(x === y) {
    resolve();
  } else {
    reject();
  }
});
mypromise.
  then(function () {
    console.log('Success');
  }).
  catch(function () {
    console.log('Error');
  });
```

JS demo.js

JS demo.js > ...

```
1  var mypromise = new Promise(function(resolve, reject) {
2    const x = 100;
3    const y = 100;
4    if(x === y) {
5      resolve();
6    } else {
7      reject();
8    }
9  });
10
11 mypromise.
12     then(function () {
13         console.log('Success');
14     }).
15     catch(function () {
16         console.log('Error');
17     });
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Success
```

Get Exclusive Video Tutorials

www.aptutorials.com
https://www.youtube.com/user/Akashtips

Get More Details

www.akashsir.com

If You Liked It !

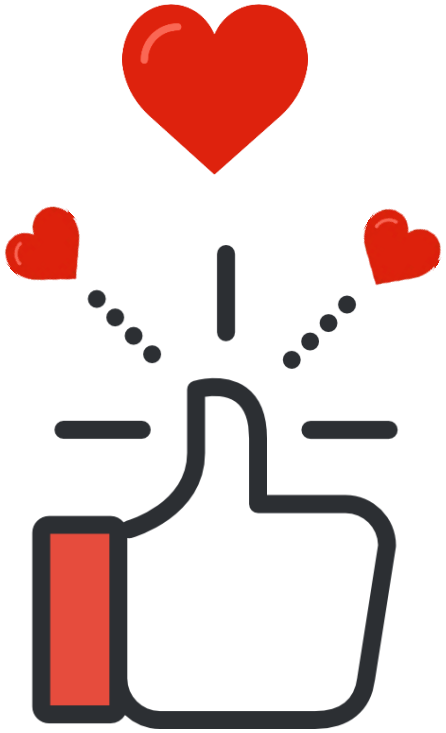
Rating Us Now

Just Dial

https://www.justdial.com/Ahmedabad/Akash-Technolabs-Navrangpura-Bus-Stop-Navrangpura/079PXX79-XX79-170615221520-S5C4_BZDET

Sulekha

https://www.sulekha.com/akash-technolabs-navrangpura-ahmedabad-contact-address/ahmedabad



Connect With Me

Akash Padhiyar
#AkashSir

www.akashsir.com
www.akashtechnolabs.com
www.akashpadhiyar.com
www.aptutorials.com

Social Info

Akash.padhiyar

Akashpadhiyar

Akash_padhiyar

+91 99786-21654

#Akashpadhiyar
#aptutorials