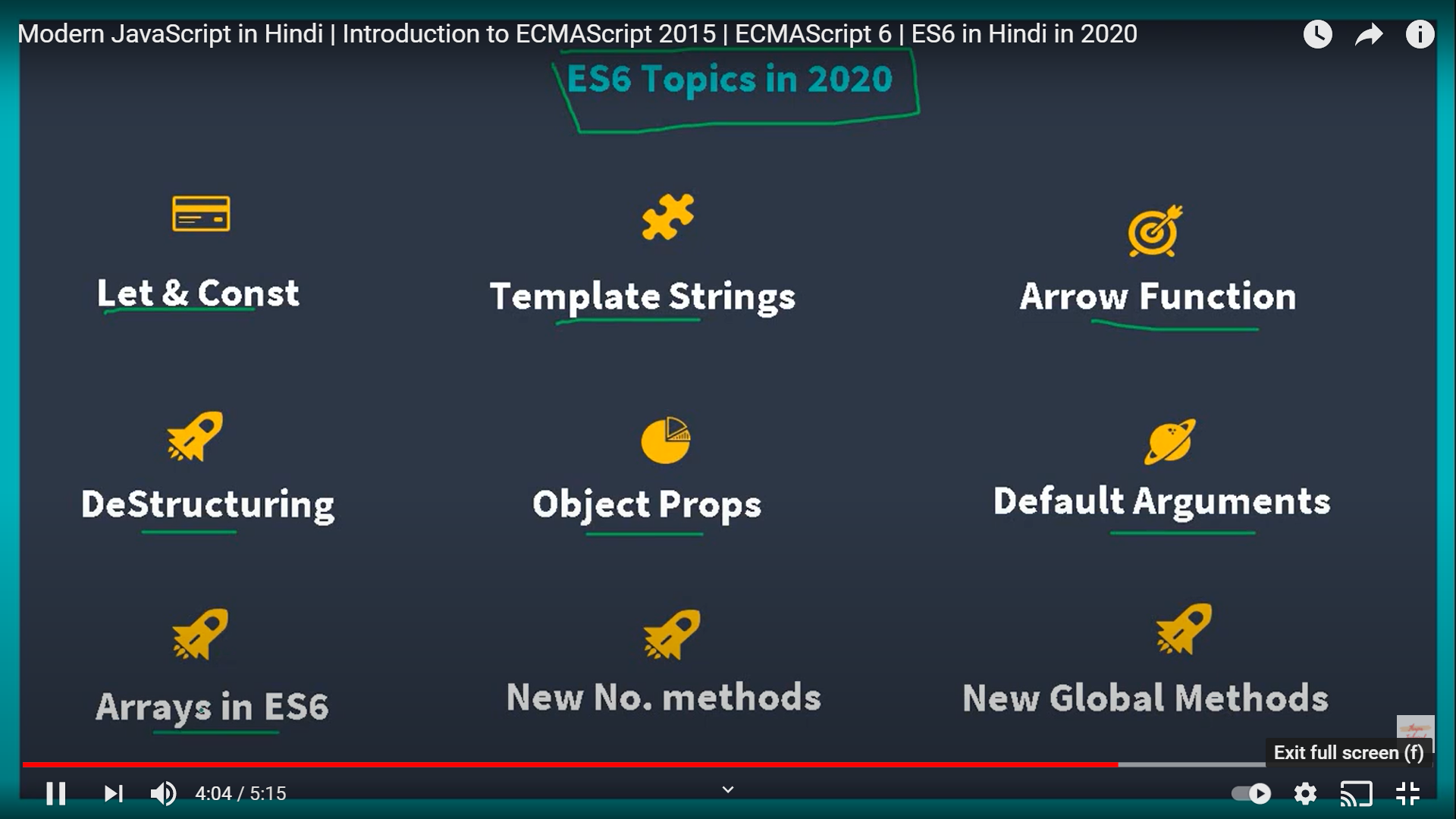
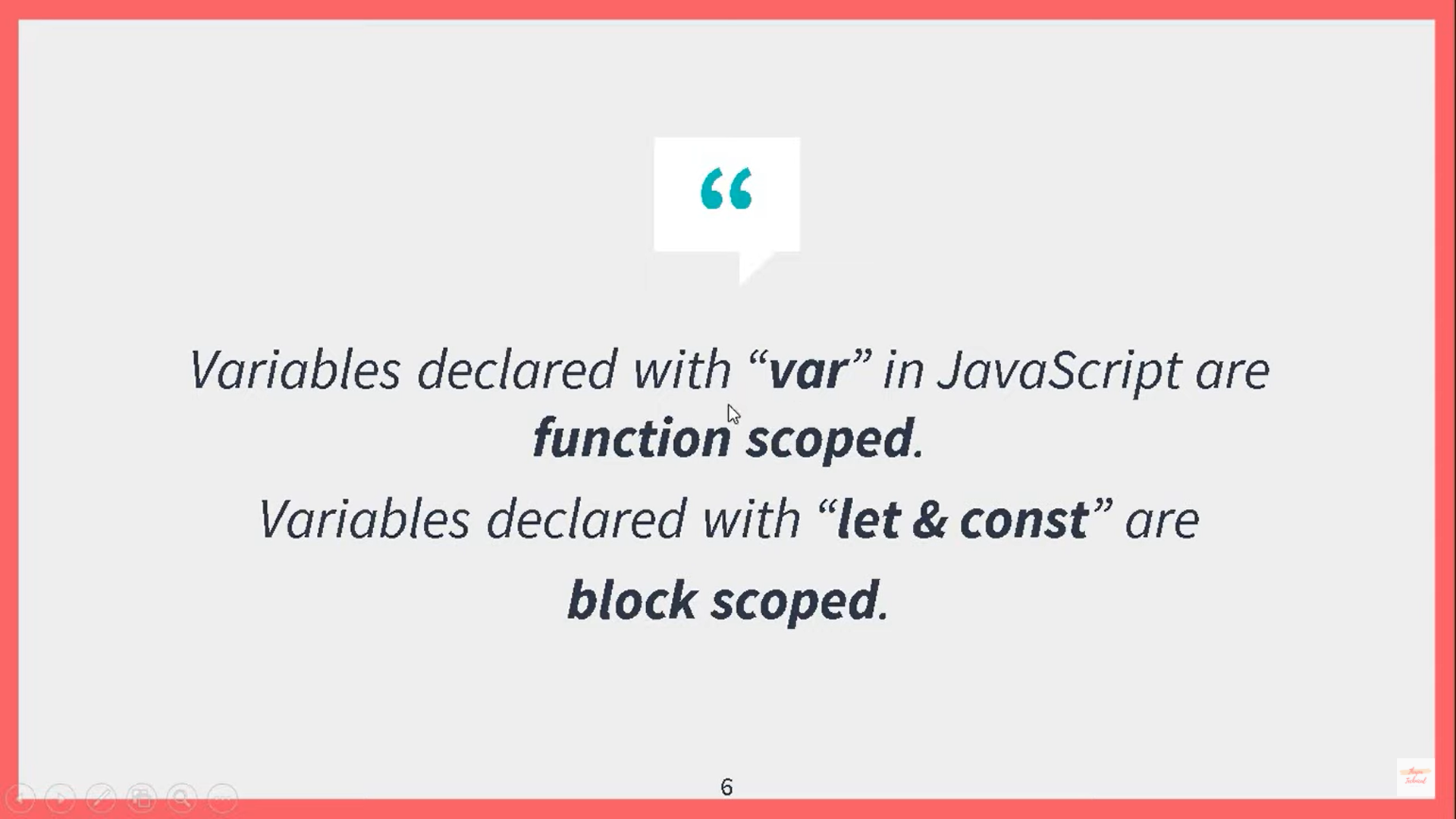


ECMAScript 6 was the second major revision to JavaScript.

ECMAScript 6 is also known as ES6 and ECMAScript 2015.





Before ES2015, JavaScript had only two types of scope: **Global Scope** and **Function Scope**.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<script>

**var x = 20;** // It has global scope

function one() {

var x = 10;

console.log(x);

}

**console.log(x);**

</script>

</head>

<body onload="one()">

Working with Variables

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<script>

function one() {

**var x = 10; // local scope**

console.log(x);

}

**console.log(x); // Will give error here**

</script>

</head>

<body onload="one()">

Working with Variables

</body>

</html>

Variables declared with the var keyword cannot have **Block Scope**.

Variables declared inside a block **{}** can be accessed from outside the block.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<script>

var x = 20;

function one() {

**{**

**var x = 10;**

**console.log(x);**

**}**

**x = 100;**

console.log(x);

}

console.log(x);

</script>

</head>

<body onload="one()">

Working with Variables

</body>

</html>

Let

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<script>

function one() {

{

let x = 10;

console.log(x);

}

x = 100;

console.log(x);

}

console.log("There is error" + x);

</script>

</head>

<body onload="one()">

Working with Variables

</body>

</html>

Const

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<script>

function one() {

{

const x = 10;

console.log(x);

x = 1000;

console.log(x);

}

x = 100;

console.log(x);

}

console.log("There is error" + x);

</script>

</head>

<body onload="one()">

Working with Variables

</body>

</html>

Arrow Functions

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

</head>

<body>

Working with Variables

<p id="p1"></p>

**<script>**

**var res = (x, y) =>**

**x+y;**

**console.log(res);**

**document.getElementById("p1").innerHTML = res(3, 5);**

</script>

() => {}

</body>

</html>

The For/Of Loop

The For/Of Loop

The JavaScript for/of statement loops through the values of an iterable objects.

for/of lets you loop over data structures that are iterable such as Arrays, Strings, Maps, NodeLists, and more.

The for/of loop has the following syntax:

for (*variable* of *iterable*) {  
  // *code block to be executed*  
}

*variable* - For every iteration the value of the next property is assigned to the variable. *Variable* can be declared with const, let, or var.

*iterable* - An object that has iterable properties.

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript For/Of Loop</h2>

<p>The for/of statement loops through the values of an iterable object.</p>

<p id="demo"></p>

<script>

let cars = ["BMW", "Volvo", "Mini"];

let text = "";

for (let x of cars) {

text += x + "<br>";

}

document.getElementById("demo").innerHTML = text;

</script>

</body>

</html>

# Babel

As any language, Javascript also has versions named ECMAScript (short for ES). Currently, most browsers support ES5. ES5 used to be good even though it was painful to code in it. Remember, this not reading from inside callback functions? The new version of Javascript, ES6, also known as ES2015 (specs of the language were finalized in June 2015) makes Javascript great again. If you want to learn about ES6, check out the links at the end of this article. All the great features of ES6 come with one big problem — majority of browsers do not fully support them. That’s when Babel comes to play. Babel is a JS transpiler that converts new JS code into old ones. It is a very flexible tool in terms of transpiling. One can easily add presets such as es2015, es2016, es2017, or env; so that Babel compiles them to ES5.

# Webpack

Now that we know what Babel and ES6+ are, we would like to use that. We would also like to use SASS for our styles, PostCSS for autoprefixing. Plus, we would like to minify and uglify both our CSS and Javascript code. Webpack solves all of these problems using one config file (named webpack.config.js) and one CLI command webpack.

Webpack is a modular build tool that has two sets of functionality — Loaders and Plugins. Loaders transform the source code of a module. For example, style-loader adds CSS to DOM using style tags. sass-loader compiles SASS files to CSS. babel-loader transpiles JS code given the presets. Plugins are the core of Webpack. They can do things that loaders can’t. For example, there is a plugin called UglifyJS that minifies and uglifies the output of webpack.