upGrad

Introduction to JavaScript and TypeScript

Course: Introduction to JavaScript and TypeScript

Lecture On: ES6 - I

Instructor: Aquib Ajani

upGrad

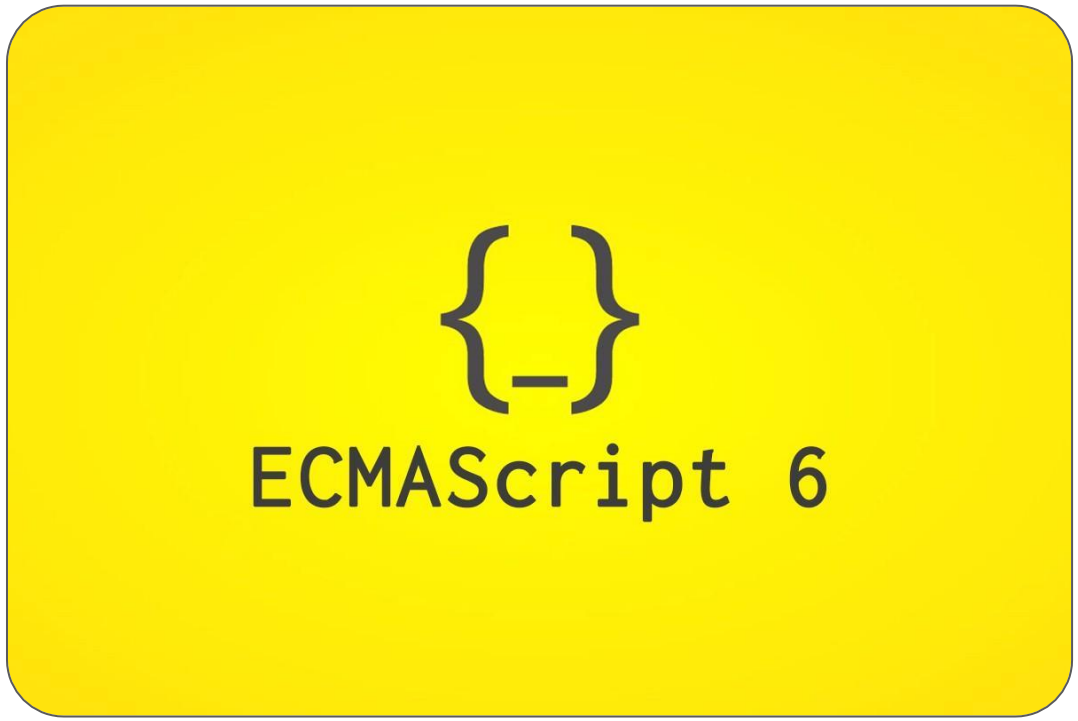
COURSE ROADMAP

- Introduction to JavaScript
- Strings in JS
- Hands-on coding Session- I
- Arrays, Objects and Functions in JS - I
- Arrays, Objects and Functions in JS - II
- Classes in JS
- OOPs using JS
- Hoisting, Scope and Closure - I
- Callbacks
- Hands-on coding Session- II
- Promises
- Hands-on coding Session- III
- **ES6 - I**
- ES6 - II

TODAY'S AGENDA

- Introduction to ES6
- Introduction to let and const
- Template Literals
- ES6 Modules

Introduction to ES6



INTRODUCTION TO ES6

- ECMAScript 2015, also known as ES2015, is a big update to JavaScript and is generally called ES6. The previous version was called ES5
- ES6 introduced a variety of new features in JavaScript (JS). These features further streamlined and simplified JS
- Top features of ES6:
  - Introduction of variables **let** and **const**
  - Arrow functions (**=>**)
  - Array and object destructuring
  - Array methods like **map()**, **filter()**, and **reduce()**
  - Template Literals
  - Import & Export modules

INTRODUCTION TO ES6

The 'let' keyword

- Before ES6, JavaScript had only two scopes: Global Scope and Function (Local) Scope. ES6 introduced a new scope called Block Scope. For example, in the screenshot below on the left-hand side, the variable declared with var inside the block {} can be accessed outside the block. However, if you use **let**, the variable number cannot be accessed outside the block. This mechanism is called **block scope**

```
{
    var number = 2;
}
//The variable number can be accessed/used here.
```

```
{
    let number = 2;
}
//The variable number CANNOT be accessed/used here.
```

- Similarly, redeclaring the variable using the var keyword will redeclare the variable outside the block. However, using let will not redeclare the variable outside the block

```
var number = 5;
//Here, number = 5
{
    var number = 2;
    //Here, number = 2
}
//Here, number = 2
```

```
let number = 5;
//Here, number = 5
{
    let number = 2;
    //Here, number = 2
}
//Here, number = 5
```

INTRODUCTION TO ES6

The 'let' keyword

- The **let** keyword was mainly introduced to disallow multiple declarations of the same variable in the same scope

```
var number = 2; //The variable number is 2
var number = 3; //The variable number is 3
```

```
let number = 5;
let number = 3; //Not allowed
{
    let number = 6; //Allowed
    let number = 7; //Not allowed
}
```

Also, redeclaring a variable declared with **let**, in another block or another scope, is **allowed**. Additionally, variables declared with the let keyword are **NOT HOISTED**

Poll 1 (15 Sec)

What will be the output of the code snippet given in the screenshot?

1. 4
2. 3
3. Uncaught ReferenceError: index is not defined
4. Undefined

```
let arr = [1, 2, 3, 4, 5], numberToFind =
4;
// finding index of the given numberToFind
in the given array arr
for (let index = 1; index <= 10; index++) {
   if (arr[index] === numberToFind) {
       break;
   }
}
console.log(index);
```

Poll 1 (15 Sec)

What will be the output of the code snippet given in the screenshot?

```
let arr = [1, 2, 3, 4, 5], numberToFind =
4;
// finding index of the given numberToFind
in the given array arr
for (let index = 1; index <= 10; index++) {
   if (arr[index] === numberToFind) {
       break;
   }
}
console.log(index);
```

1. 4
2. 3
3. **Uncaught ReferenceError: index is not defined**

   The variable index is declared and initialised within the for loop, which has the block scope. This means that the variable index is not accessible outside the for loop, thereby throwing a ReferenceError.
4. Undefined

INTRODUCTION TO ES6

The ‘let’ keyword

- The keyword let allows you change the value. However, the keyword **const** is stubborn and does not allow you to change the value stored in it
- If you try to change the value stored in a const variable, an error will be thrown (for example, ***‘Uncaught TypeError: Assignment to constant variable’).***
- You should declare a variable using the let keyword if you want to change its value in the future. If you do not want to change the value of a variable **ever**, you should declare it with the **const** keyword
- Here is the general naming convention for constants: A constant should be written in upper case with words separated by underscore (_)

INTRODUCTION TO ES6

The ‘const’ keyword

- Variables defined with the **const** keyword behave similar to the let variable, except they cannot be reassigned. The variable const allows you to declare a JavaScript variable with a constant value, a value that cannot be changed. However, if a variable has already been declared with var or let, it cannot be reassigned with const

```
const number = 2; //The variable number is 2
const number = 3; //Not allowed
number = 3; //Not allowed
```

```
var number = 2; //The variable number is 2
const number = 3; //Not allowed
```

- Variables with the const keyword need to be assigned when they are declared

```
const number; //Not allowed
number = 3; //Not allowed
```

```
const number = 3; //The variable number is 3
```

- Objects and arrays defined with const can change their properties, but the variable holding the object/array cannot be changed

```
const person = { firstName: “John”, lastName: “Doe”, age: 27 };
person.age = 40;  person.nationality = “English”;  //Allowed
person = { firstName: “Jane”, lastName: “Dias”, age: 25 };   //Not allowed
```

Poll 2 (15 Sec)

Select the correct identifier with the given properties must be declared.

- name (name of the user logging in to the website)
- PI (gravitational acceleration)

1. const name;
   const PI = 3.14;
2. let name;
   let PI = 3.14;
3. let name;
   const PI = 3.14;
4. const name;
   let PI = 3.14;

Poll 2 (Answer)

Select the correct identifier with the given properties must be declared.

- name (name of the user logging in to the website)
- PI (gravitational acceleration)

1. const name;
   const PI = 3.14;
2. let name;
   let PI = 3.14;
3. **let name;**
   const PI = 3.14;
4. const name;
   let PI = 3.14;

Poll 3 (15 Sec)

What will be the output of the code snippet given in the screenshot?

1. Gupta
2. Agrawal
3. undefined
4. Error

```
const employee = {
   firstName : "Prachi",
   lastName : "Agrawal",
}
employee.lastname = "Gupta";
console.log(employee.lastname);
```

Poll 3 (15 Sec)

What will be the output of the code snippet given in the screenshot?

1. **Gupta**
2. Agrawal
3. undefined
4. Error

```
const employee = {
   firstName : "Prachi",
   lastName : "Agrawal",
}
employee.lastname = "Gupta";
console.log(employee.lastname);
```

Spread Operator and Rest Parameters

SPREAD OPERATOR AND REST PARAMETERS

Spread Operator

- As per the official documentation of MDN, **spread syntax** allows an iterable such as an array expression or string to be expanded in places where zero or more arguments (for function calls) or elements (for array literals) are expected, or an object expression to be expanded in places where zero or more key-value pairs (for object literals) are expected
- As arrays and objects are copied by reference, the spread operator copies the value (not the reference) of the object/array into a new variable

```
const contact = [“john@gmail.com”, +66345678001];
const person = [“John”, “Doe”, ...contact, “English”];
console.log(person); //["John", "Doe", "john@gmail.com", 66345678001, "English"]
```

```
let car = { name: “Mercedes”, model: “C200”, color: “white”, weight: 500 };
console.log(car.name); //Mercedes
let car2 = {...car}; //Object car is copied by value into object car2
car.name = “Audi”;
console.log(car.name); //Audi
console.log(car2.name); //Mercedes
```

SPREAD OPERATOR AND REST PARAMETERS

Rest Parameters

- At times, you may not be sure about the number of arguments that a function might receive. In such a case, you can use the **rest parameters**
- Its syntax is similar to that of the spread operator ("...")

```
let fun = (...numbers) => console.log(numbers.length);


fun(10, 2, 6, 7); //4
fun(1); //1
fun(20, 32); //2
```

Poll 4 (15 Sec)

What will be the output of the code snippet given in the screenshot?

1. 124
2. Compile time error
3. Run-time error
4. No Output

```
let score = [122,27,124];
Math.max(...score);
```

Poll 4 (15 Sec)

What will be the output of the code snippet given in the screenshot?

1. 124

   Passing score with the spread operator to the Math.max() method is equivalent to calling method with the apply() method having the parameter score (Math.max.apply(score)).
   The maximum value will be calculated based on the array elements passed. Here, the output will be 124, which is of the maximum value among all members in an array.

2. Compile time error
3. Run-time error
4. No Output

```
let score = [122,27,124];
Math.max(...score);
```

Template Literals

Template Literals

- As per MDN, **template literals** are string literals that allow embedded expressions. You can use multi-line strings and **string interpolation** features with template literals
- For example, in ES5, we use + to concatenate strings. In ES6, we can include this in the template literals. With the help of template strings, we need not use + for concatenation
- You can simply add a string with backtick (``), and if you wish to add placeholders like variables/expressions, you can use ${}.
- You can also have multiple lines of text without adding + ”\n” for a new line

```
let user = “John Doe”;
let age = 27;
let str = `The name of the user is ${user}
            and his age is ${age}`;
console.log(str);

// "The name of the user is John Doe
            and his age is 27"
```

Modules:
Import & Export

IMPORT AND EXPORT MODULES

- JavaScript modules are essentially libraries that are included in the given program. So, you can call the functions present in other files without having to recreate the function in the file. In the example below, the variable lib stores the modules **imported** from the file utility.js. Now, if there is a function, say, area, inside utility.js, it can be accessed using lib.area()

utils.js

```
const sum = (...arr) => {
  return arr.reduce((accumulator, currentValue) => accumulator + currentValue);
}

export default sum;
```

script.js

```
import sum from 'utils.js';
console.log(sum(1, 2, 3, 4));
```

IMPORT AND EXPORT MODULES

- Similarly, you can export functions to be imported in other files using the **module.exports** functionality. In the example below, we are exporting the variable area, which stores the function to calculate area

utils.js

```
export default sum = (...arr) => {
  return arr.reduce((accumulator, currentValue) => accumulator + currentValue);
}
```

script.js

```
import sum from 'utils.js';
console.log(sum(1, 2, 3, 4));
```

HANDS-ON EXERCISE

1. Write a JavaScript program to compare two objects in order to determine if the first one contains the property values equivalent to the second one
2. Write a JavaScript program to convert a specified number into an array of digits

KEY TAKEAWAYS

- Introduction to ES6
- Introduction to let and const
- How template literals work
- How ES6 modules work

TASKS TO COMPLETE AFTER THE SESSION

- MCQs
- Coding Questions

upGrad

Thank You!