upGrad

#RahoAmbitious

Full Stack Software Development

upGrad

Course: JavaScript and Server-Side Communication

Lecture On : ES6

Instructor : Siddhesh Prabhugaonkar

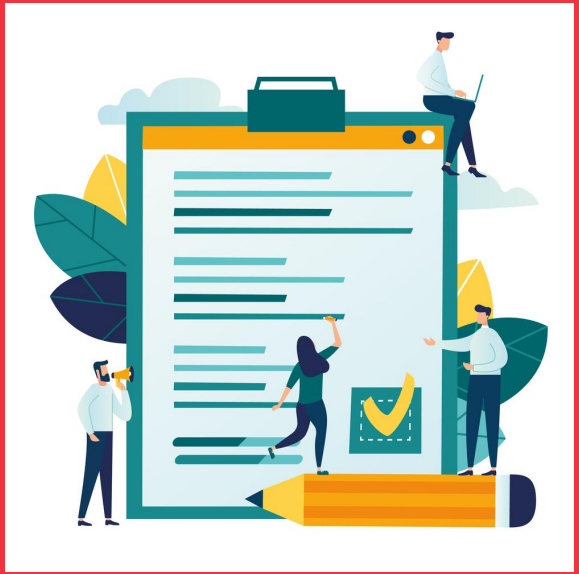
In the previous session, we covered….

- DOM manipulation using JavaScript
- Web storage

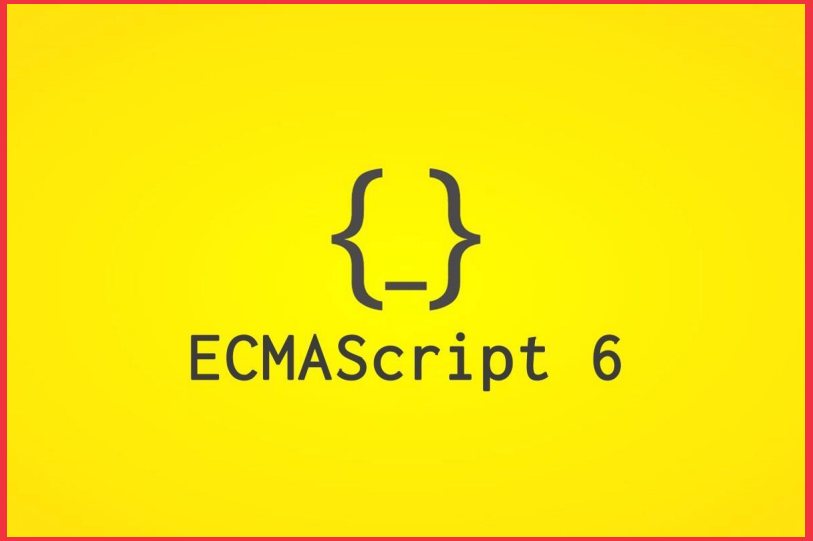
Today's Agenda

upGrad

- Introduction to ES6
- Introduction to let and const
- The arrow functions of ES6
- Array Methods - map(), filter(), and reduce()
- Array and Object Destructuring - Spread Operator and Rest Parameters
- Template Literals
- Import and Export



Introduction to ES6



Introduction to ES6

upGrad

- ECMAScript 2015, also known as ES2015, is a significant update to JavaScript. This update is called ES6, as the earlier standard update was called ES5.
- ES6 introduced a host of new features in JavaScript (JS), which makes working with JS more streamlined and simplified.
- Some of the top features include:
    1. Introduction of variables **let** and **const**
    2. Arrow functions (**=>**)
    3. Array and object destructuring
    4. Array methods like **map()**, **filter()**, and **reduce()**
    5. Template Literals
    6. Import and Export modules

The 'let' keyword

Before ES6, JavaScript (JS) had only two scopes – Global Scope and Function (Local) Scope. ES6 introduces something called ***Block Scope***. For instance, in the example 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 is called a **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
```

The 'let' keyword

let keyword was majorly introduced to not allow 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, re-declaring a variable declared with **let**, in another block or another scope, is **allowed**.
Additionally, variables declared with **let** are **NOT HOISTED**.

upGrad

Poll 1 (15 Sec)

Correct output of the code snippet is:

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 (Answer)

Correct output of the code snippet is:

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.

1. 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);
```

The ‘const’ keyword

- We now know that, ***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 changing the value stored in a ***const*** variable, an error is thrown saying ***‘Uncaught TypeError: Assignment to constant variable’***.

- Thus, you must declare a variable using the ***let*** keyword when you want to change its value in future and you must declare a variable with ***const*** keyword when you do not want to change its value **ever**!

- The general naming convention being followed for constants is: A constant should be in uppercase with words separated by underscore (_).

The ‘const’ keyword

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

The **const** variables 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 which the below properties must be declared.

- name (name of the user who logs into 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 which they must be declared.

- name (name of the user who logs into 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;

Difference between const and Object.freeze()

Objects that are declared with **const** can have their properties changed.

```
var person = { firstName: “John”, lastName: “Doe”, age: 27 };
person.age = 40;  //Allowed
person.nationality = “English”;  //Allowed
```

To have objects that cannot have their properties changed, i.e. to make objects immutable, we can use ***Object.freeze()*** method.

```
var person = { firstName: “John”, lastName: “Doe”, age: 27 };
Object.freeze(person);
person.age = 40;
person.nationality = “English”;  //Not allowed
```

However, ***Object.freeze()*** works on a shallow level. The properties of nested objects can be changed.

```
var person = { firstName: “John”, lastName: “Doe”, contact: {number: 9999999999 } };
Object.freeze(person);
person.firstName = “Jane”; //Not allowed
person.contact.number = 7777777777;  //Allowed
```

upGrad

Poll 3 (15 Sec)

What is the output here?

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

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

Poll 3 (Answer)

What is the output here?

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

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

upGrad

Poll 4 (15 Sec)

What is the output here?

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

```
const details = {
   firstname : "Prachi",
   lastname : "Agrawal",
}
Object.freeze(details);
details.lastname = "Gupta";
console.log(details.lastname);
```

Poll 4 (Answer)

What is the output here?

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

```
const details = {
   firstname : "Prachi",
   lastname : "Agrawal",
}
Object.freeze(details);
details.lastname = "Gupta";
console.log(details.lastname);
```

Arrow functions

Arrow functions have two advantages over the regular functions – they allow us to write functions with a much shorter syntax, and they also do not have any binding with **this**.

- For instance, using arrow functions, you can remove the function keyword and replace it with a fat arrow (=>) symbol. Additionally, if your function has only one statement, you can remove the curly braces as well as the return statement.

```
var display = function() {
    return "Hello World";
} //traditional ES5 function
declaration
```

```
let display = () => {
    return "Hello World";
}
//Arrow function
```

```
let display = () => "Hello World";
//Single line arrow function
```

- If your function has parameters, you can simply add them inside the parentheses (**"()"**). And if you have only one parameter, then you can even remove the parentheses.

```
let display = (name) => "Hello" + name;
// Here, name is an argument to the arrow function.
```

```
let display = name => "Hello" + name;
// Here, as name is the only single argument. the ()
are removed
```

Arrow functions

Finally, in the arrow functions, the **'this'** keyword always represents the object that defined the function. For example, if a button calls a function, in regular JavaScript (JS), **this** would refer to the button element, but in ES6, it will be either the class in which the function is defined or the window object.

ES5

```
let person = {
   firstName: "Prachi",
   lastName: "Agrawal",
   get: function() {
     console.log("Outer: " + this.firstName + " " + this.lastName);
      let print = function() {
          console.log("Outer: " + this.firstName + " " + this.lastName);
       };
       print();
    }
}
person.get();
//Outer: Prachi Agrawal
//Outer: undefined undefined
```

Arrow functions

Finally, in the arrow functions, the **'this'** keyword always represents the object that defined the function. For example, if a button calls a function, in regular JavaScript (JS), **this** would refer to the button element, but in ES6, it will be either the class in which the function is defined or the window object.

ES6

```
let person = {
   firstName: "Prachi",
   lastName: "Agrawal",
   get: function() {
        console.log("Outer: " + this.firstName + " " + this.lastName);
        let print = () => {
             console.log("Outer: " + this.firstName + " " + this.lastName);
        };
        print();
   }
}
person.get();
//Outer: Prachi Agrawal
//Outer: Prachi Agrawal
```

Array Methods:
map(), filter(), reduce()

map(), filter(), and reduce()

upGrad

The map() function

- ES6 introduced a new method of traversing an array called **map()**.
- The map() method basically creates a new array by calling a function for every element in the array.
- The map() method does not change the original array.

```
let numbers = [3, 4, 6, 5];

let twice = numbers.map(function(number) {
    return number * 2;
})

console.log(twice); //6, 8, 12, 10
```

map(), filter(), and reduce()

upGrad

The filter() function

- ES6 introduced a new method of filtering values of an array called **filter()**.
- The filter() method basically creates a new array by calling a function for every element in the array, and by checking a condition against each element in the array.
- If the condition is met, i.e. the condition is true, then that element is added to the resultant array.
- The filter() method does not change the original array.

```
let numbers = [3, 4, 6, 5];

let even = numbers.filter(function(number) {
    return number % 2 === 0;
})

console.log(even); //4, 6
```

map(), filter(), and reduce()

upGrad

The reduce() function

- ES6 introduced a new method of reducing values of an array to a single value called **reduce()**.
- The reduce() method executes a provided function for each value of the array, from left to right.
- The return value of the function is stored inside an accumulator or in other words, a result variable.
- The reduce() method does not change the original array.

```
let numbers = [3, 4, 6, 5];

let sum = numbers.reduce(function(accumulator, element) {
    return accumulator + element;
})

console.log(sum); //18
```

Poll 5 (15 Sec)

Suppose you have a very large dataset of numbers stored in an array and you have to find out which numbers in that large dataset are multiples of 10. Which of the following array operations are useful in this case?

1. map
2. reduce
3. filter
4. None of the above

Poll 5 (Answer)

Suppose you have a very large dataset of numbers stored in an array and you have to find out which numbers in that large dataset are multiples of 10. Which of the following array operations are useful in this case?

1. map
2. reduce
3. **filter**
4. None of the above

Poll 6 (15 Sec)

Suppose you have a large dataset array consisting of the yearly revenue of a company (in billion dollars) from 2007 to 2017. You have to calculate the total revenue of the company in the past years, i.e. from 2007 to 2017. Which of the following array operations would you use to calculate the total revenue of that company in the past 10 years?

1. map
2. reduce
3. filter
4. None of the above

Poll 6 (Answer)

Suppose you have a large dataset array consisting of the yearly revenue of a company (in billion dollars) from 2007 to 2017. You have to calculate the total revenue of the company in the past years, i.e. from 2007 to 2017. Which of the following array operations would you use to calculate the total revenue of that company in the past 10 years?

1. map
2. **reduce**
3. filter
4. None of the above

upGrad

Spread Operator and Rest Parameters

Spread Operator

- As per the official docs 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 and 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
```

Rest Parameters

- There would be times when you might not be sure about the number of arguments that a function might receive. In such a case, the **rest parameters** come to the rescue.
- 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 7 (15 Sec)

What will be the output of the following code?

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

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

Poll 7 (Answer)

What will be the output of the following code?

1. 124

Passing score with spread operator to *Math.max()* method is equivalent to calling method with an *apply()* method having parameter *score*. << Math.max.apply(score)>>. Maximum value will be calculated on the basis of passed array elements. So, here, the output would be 124, which is the maximum among all members in an array.

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

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

upGrad

Template Literals

Template Literals

upGrad

- As per MDN, **template literals** are string literals that allow embedded expressions. You can use multi-line strings and string interpolation features with them.
- For instance, in ES5, we use **+** to concatenate strings. Instead, we can now include that in the template literals. With the help of template strings, there is no need to use **+** for concatenation.
- You can simply add a string with backtick (``), and if you wish to add placeholders like variables/expressions, then you can do that using the ${}.
- You can also have multiple line 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"
```

upGrad

Modules: Import and Export

Import and Export Modules

upGrad

JavaScript (JS) modules are basically libraries which are included in the given program. So, you can call the functions existing in other files without having to re-create the function in the file. For instance, 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

upGrad

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));
```

upGrad

Project Work

(Let’s convert our code to ES6 syntax)

You can refer to the solution here.

Hands-On Exercise 1 (3 mins)

upGrad

You are given an object named *movie*. This object should have a setter method named ***set()*** which should set the *title* and *director* of the movie. The *movie* object has another method named ***get()***. The ***get()*** method should contain a normal function named ***print()*** which should print the *title* and *director* of the movie.

The output should be:
Title = Inception, Director = Christopher Nolan

Stub Code: here

Solution: here

Hands-On Exercise 2 (3 mins)

upGrad

Implement the filter function to convert a given array into one in which all elements are multiples of 10.

Stub Code: here

Solution: here

Key Takeaways

- ES6 comes with a host of new features which simplify coding in JavaScript.
- The variable **let** introduces block scope, where its existence is restricted to only the block in which it is defined, and it cannot be redeclared. The variable **const** allows us to add values to variables which cannot be changed.
- In ES6, functions can be converted to a shorter syntax using the arrow function.
- The method **map()** is used to create a function which is applied on every element of an array, the method **filter()** is used to filter values in an array, and the method **reduce()** is used to reduce the values of an array to a single value.
- As arrays and objects are copied by reference, the spread operator copies the value and not the reference of the object/array into a new variable.

upGrad

Doubts Clearance (5 mins)

Task to complete after today's session

MCQs
Coding Questions
Project - Checkpoint 3

In the next class, we will discuss...

- JavaScript callbacks
- JavaScript promises

upGrad

#RahoAmbitious

Thank You!