# ECMAScript

ECMAScript (ES) is a scripting language specification standardized by ECMAScript International. It is used by applications to enable client-side scripting. Languages like JavaScript, Jscript and ActionScript are governed by this specification. This introduces you to ES6 implementation in JavaScript.

ECMA Script6's implementation discussed here

1) Support for constants

2) Block Scope

3) Arrow Functions

4) Extended Parameter Handling

5) Template Literals

6) Extended Literals

7) Enhanced Object Properties

8) De-structuring Assignment

9) Modules

10) Classes

11) Iterators

12) Generators

13) Collections

14) New built in methods for various classes

15) Promises

## ES6 Variables:



A variable, by definition, is "a named space in the memory" that stores values. In other words, it acts as a container for values in a program.

Identifiers cannot be keywords.

Identifiers can contain alphabets and numbers.

Identifiers cannot contain spaces and special characters, except the underscore (_) and the dollar ($) sign.

Variable names cannot begin with a number.

## ES5 :

ES5 syntax used the var keyword to achieve the same. The ES5 syntax for declaring a variable is as follows.

```
//Declaration using var keyword
var  variable_name
```

## ES6:

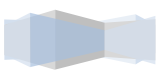
ES6 introduces the following variable declaration syntax

Using the let.

Using the const.

Variable initialization refers to the process of storing a value in the variable. A variable may be initialized either at the time of its declaration or at a later point in time.





## JavaScript and Dynamic Typing:

JavaScript is an un-typed language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically. This feature is termed as dynamic typing.

## JavaScriptVariable Scope:

- Global Scope : A variable with global scope can be accessed from within any part of the JavaScript code.
- Local Scope : A variable with a local scope can be accessed from within a function where it is declared.

```
var a =10
function test() {
    var b =100
    console.log(b); //100
}

console.log(a); //10
test()
//100
//100
```

## The Let and Block Scope:

The block scope restricts a variable's access to the block in which it is declared. The var keyword assigns a function scope to the variable. Unlike the var keyword, the let keyword allows the script to restrict access to the variable to the nearest enclosing block.

```
  if(true){
    var a= 2;

  }

  console.log(a);
//let is block scope

if (true) {
   let a =2;
   console.log(a); //2 anwes
}
//console.log(a); // ReferenceError: a is not defined
```

## The const :

The const declaration creates a read-only reference to a value. It does not mean the value it holds is immutable, just that the variable identifier cannot be reassigned. Constants are block-scoped, much like variables defined using the let statement. The value of a constant cannot change through re-assignment, and it can't be re-declared.

## Operators

- Operands : Represents the data.
- Operator : Defines how the operands will be processed to produce a value.

## ES6 Arithmetic Operators

```
let num1 = 10
let num2 = 2
let res = 0
```

```
res = num1+num2
console.log("Sum:        "+res); //12

res = num1-num2;
console.log("Difference: "+res)  //8

res = num1*num2
console.log("Product:    "+res) //20

res = num1/num2
console.log("Quotient:   "+res) //5

res = num1%num2
console.log("Remainder:  "+res) //0

num1++
console.log("Value of num1 after increment "+num1)  //11

num2--
console.log("Value of num2 after decrement "+num2) //1
```

## Relational Operators:

Relational operators test or define the kind of relationship between two entities. Relational operators return a boolean value, i.e. true/false.

Assume the value of A is 10 and B is 20.

```
let num1 = 5;
let num2 = 9;

console.log("Value of num1: " + num1); //5
console.log("Value of num2: " + num2); //9

var res = num1 > num2;
console.log("num1 greater than num2: " + res); //false

res = num1 < num2;
```

```
console.log("num1 lesser than num2: " + res); //true

res = num1 >= num2;
console.log("num1 greater than or equal to  num2: " + res); //false

res = num1 <= num2;
console.log("num1 lesser than or equal to num2: " + res); //true

res = num1 == num2;
console.log("num1 is equal to num2: " + res); //false

res = num1 != num2;
console.log("num1 not equal to  num2: " + res); //true
```

## Logical Operators

Logical operators are used to combine two or more conditions. Logical operators, too, return a Boolean value. Assume the value of variable A is 10 and B is 20.

```
let a=true;
let b =false

if (a &&b) {
    console.log("true");
} else {
    console.log("false");
}
//out put false

if (a ||b) {
    console.log("true");
} else {
    console.log("false");
}

//output True
```

```
if (!a ||b) {
    console.log("true");
} else {
    console.log("false");
}

//output false
```

## Assignment Operators

```
let a = 12;
let b = 10;
a = b; //10
console.log("a = b: " + a);  //10
a += b;

console.log("a+=b: " + a); //20
a -= b;

console.log("a-=b: " + a); //10
a *= b;

console.log("a*=b: " + a); //100
a /= b;

console.log("a/=b: " + a); //10
a %= b;

console.log("a%=b: " + a);//0
```

## Decision Making

## if Statement:

If the Boolean expression evaluates to true, then the block of code inside the if statement will be executed. If the Boolean expression evaluates to false, then the first set of code after the end of the if statement (after the closing curly brace) will be executed.

```
let  num = 5
if (num < 10) {
   console.log("number is positive")
}
```

## if…else Statement:

An if can be followed by an optional else block. The else block will execute if the Boolean expression tested by if evaluates to false.

Following is the syntax.

```
if(boolean_expression) {
   // statement(s) will execute if the Boolean expression is true
} else {
   // statement(s) will execute if the Boolean expression is false
}
```

```
let a =true
let b =false
if(a && b) {
  console.log("true")
} else {
   // statement(s) will execute if the Boolean expression is false
   console.log("false")
}
```

## else…if Ladder :

The else…if ladder is useful to test multiple conditions. Following is the syntax of the same.

```
if (boolean_expression1) {
   //statements if the expression1 evaluates to true
} else if (boolean_expression2) {
   //statements if the expression2 evaluates to true
} else {
   //statements if both expression1 and expression2 result to false
}
```

```
var num = 2
if(num > 0) {
   console.log(num+" is positive")
} else if(num < 0) {
   console.log(num+" is negative")
} else {
   console.log(num+" is neither positive nor negative")
}
//output 2 is positive
```

## switch…case Statement

The switch statement evaluates an expression, matches the expression's value to a case clause and executes the statements associated with that case.

Following is the syntax.

```
switch(Apptron) {
   case constant_expr1: {
      //statements;
      break;
```

```
  }
  case constant_expr2: {
    //statements;
    break;
  }
  default: {
    //statements;
    break;
  }
```

```
let APPTRON = "A";
switch(APPTRON) {
  case "A": {
    console.log("Excellent");
    break;
  }
  case "B": {
    console.log("Good");
    break;
  }
  case "C": {
    console.log("Fair");
    break;
  }
  case "D": {
    console.log("Poor");
    break;
  }
  default: {
    console.log("Invalid choice");
    break;
  }
}
```

# ES6 Loops

At times, certain instructions require repeated execution. Loops are an ideal way to do the same. A loop represents a set of instructions that must be repeated. In a loop's context, a repetition is termed as an iteration.

There are two types loops definite and indefinite:

## Definite loops:

a) for loop
b) in loop
c) of loop

## Definite loops:

a) while loop
b) do while loop

# ES6 - The 'for' loop

The for loop executes the code block for a specified number of times. It can be used to iterate over a fixed set of values, such as an array. Following is the syntax of the for loop.

```
for (let i=0; i<10 ; i++){
  console.log("Apptron "+i);
}

var num = 5
var factorial = 1;
for( let i = num ; i >= 1; i-- ) {
   factorial *= i ;
}
```

```
console.log(factorial);
```

## ES6 The for…in loop

The for…in loop is used to loop through an object's properties.

Following is the syntax of 'for…in' loop.

In each iteration, one property from the object is assigned to the variable name and this loop continues till all the properties of the object are exhausted.

```
for (variablename in object) {
   //statement or block to execute
}

var obj = {a:1, b:2, c:3};
for (var prop in obj) {
   console.log(obj[prop]);
}
```

## ES6 The for…of loop

The for…of loop is used to iterate iterables instead of object literals.

Following is the syntax of 'for…of' loop.

```
for (variablename of object) {
  // statement or block to execute
}


for (let val of[12 , 13 , 123]) {
   console.log(val)
```

```
}

// 12
// 13
// 123
```

## Indefinite Loop :

An indefinite loop is used when the number of iterations in a loop is indeterminate or unknown.

## ES6 The while loop

The while loop executes the instructions each time the condition specified evaluates to true.

```
while (expression) {
  //  Statement(s) to be executed if expression is true
  }

  Let num = 5;
  Let factorial = 1;

  while(num >= 1) {
  factorial = factorial * num;
  num--;
  }
   console.log("The factorial  is "+factorial);
```

## ES6 The do…while loop

The do…while loop is similar to the while loop except that the do…whileloop doesn't evaluate the condition for the first time the loop executes. However, the condition is evaluated for the subsequent iterations. In other words, the code block will be executed at least once in a do…while loop.

```
do {
// Statement(s) to be executed;
}
while (expression);

var n = 10;
do {
console.log(n);
n--;
}
while(n>=0);
```

## The Loop Control Statements:

The break statement is used to take the control out of a construct. Using break in a loop causes the program to exit the loop. Following is an example of the break statement

```
    var i = 1
    while(i<= 10) {
    if (i == 5 ) {
        console.log("The first multiple of 5  is : "+i) ;
        break     //exit the loop if the first multiple is found
    }
    i++
    }
```

## Arrays



a) Features of an Array:

b) An array declaration allocates sequential memory blocks.

c) Arrays are static. This means that an array once initialized cannot be resized.

d) Each memory block represents an array element.

e) Array elements are identified by a unique integer called as the subscript/index of the element.

f) Arrays too, like variables, should be declared before they are used.

g)Array initialization refers to populating the array elements.

h) Array element values can be updated or modified but cannot be deleted.

Declaring and Initializing Arrays

var array_name; //declaration

array_name = [val1,val2,valn..] //initialization

OR

var array_name = [val1,val2…valn]

```
let apptron;
apptron = ["1","2","3","4"]
console.log(apptron[0]);
console.log(apptron[1]);

let names = new Array("app","react","ionic","js")
for(var i = 0;i<names.length;i++) {
console.log(names[i])
```

```
}
```

# Array Methods

## 1) Method concat() :

Syntax

array.concat(value1, value2, ..., valueN);

```
let app = ["a", "b", "c"];
const numeric = [1, 2, 3];
var alphaNumeric = app.concat(numeric);
console.log("alphaNumeric : " + alphaNumeric );
```

## 2) Method filter() :

Syntax

array.filter(callback[, thisObject]);

Parameters:

callback : Function to test for each element.

thisObject: Object to use as this when executing callback.



```
function isBigEnough(element, index, array) {
```

```
return (element >= 10);
}
var passed = [12, 5, 8, 130, 44].filter(isBigEnough);
console.log("Test Value : " + passed );

// output:
// 12,130,44
```

## 3) Method forEach() :

Syntax

array.forEach(callback[, thisObject]);

```
nums = new Array(12,13,14,15)
console.log("Printing original array.....")

nums.forEach(function(val,index) {
console.log(val)
})
nums.reverse()  //reverses the array element
console.log("Printing Reversed array....")

nums.forEach(function(val,index){
console.log(val)
})
```

## 4) Method indexOf() :

indexOf() method returns the first index at which a given element can be found in the array, or -1 if it is not present.

Syntax

array.indexOf(searchElement[, fromIndex]);

Parameters:

searchElement : Element to locate in the array.

fromIndex : The index at which to begin the search. Defaults to 0, i.e. the whole array will be searched. If the index is greater than or equal to the length of the array, -1 is returned.

```
var index = [12, 5, 8, 130, 44].indexOf(44);
  console.log("index is : " + index )

 //OUTPUT  its serach equal values
```

## 5) Array Method join() :

join() method joins all the elements of an array into a string.

Syntax

array.join(separator);

Parameters

separator : Specifies a string to separate each element of the array. If omitted, the array elements are separated with a comma

```
var arr = new Array("First","Second","Third");
 var str = arr.join();
 console.log("str : " + str );
 var str = arr.join(", ");
 console.log("str : " + str );

 var str = arr.join(" + ");
 console.log("str : " + str );

 // str : First,Second,Third
 // str : First, Second, Third
 // str : First + Second + Third
```

## 6) Method lastIndexOf()

lastIndexOf() method returns the last index at which a given element can be found in the array, or -1 if it is not present. The array is searched backwards, starting at fromIndex

Syntax

array.lastIndexOf(searchElement[, fromIndex]);

Parameters:

searchElement : Element to locate in the array.

fromIndex : The index at which to start searching backwards. Defaults to the array's length, i.e., the whole array will be searched. If the index is greater than or equal to the length of the array, the whole array will be searched. If negative, it is taken as the offset from the end of the array.

```
var index = [12, 5, 8, 130, 44].lastIndexOf(8);
console.log("index is : " + index );

// Output
// index is : 2
```

## 7) Method map() :

map() method creates a new array with the results of calling a provided function on every element in this array.

Syntax

array.map(callback[, thisObject]);

Parameters:

callback: Function that produces an element of the new Array from an element of the current one.

This Object : Object to use as this when executing callback.

```
var numbers = [1, 4, 9];

var roots = numbers.map(Math.sqrt);
console.log("roots is : " + roots );
```

```
// Output
// roots is : 1,2,3
```

## 8) Method pop()

pop() method removes the last element from an array and returns that element.

Syntax

array.pop();

```
var numbers = [1, 4, 9];
var element = numbers.pop();
console.log("element is : " + element );

var element = numbers.pop();
console.log("element is : " + element );


// out put :

// Output
// element is : 9
// element is : 4
```

## 9) Method push()



push() method appends the given element(s) in the last of the array and returns the length of the new array.

Syntax

array.push(element1, ..., elementN);

```
var numbers = new Array(1, 4, 9);
var length = numbers.push(10);
console.log("new numbers is : " + numbers );
length = numbers.push(20);
console.log("new numbers is : " + numbers );

// new numbers is : 1,4,9,10
// new numbers is : 1,4,9,10,20
```

## 10) Method reduce()

reduce() method applies a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.

Syntax

array.reduce(callback[, initialValue]);

Parameter Details:

callback : Function to execute on each value in the array.

initialValue : Object to use as the first argument to the first call of the callback.

## 10) reduceRight()

Applies a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.

Syntax:

array.reduceRight(callback[, initialValue]);

Parameter Details:

callback : Function to execute on each value in the array.

initialValue : Object to use as the first argument to the first call of the callback.

```
var total = [0, 1, 2, 3].reduceRight(function(a, b){
return a + b;
});
 console.log("total is : " + total );
```

## 11) Method reverse() :

reverse() method reverses the element of an array. The first array element becomes the last and the last becomes the first.

Syntax

array.reverse();

Return Value

Returns the reversed single value of the array.

```
var arr = [0, 1, 2, 3].reverse();
console.log("Reversed array is : " + arr );

//Reversed array is : 3,2,1,0
```

## 12) Method shift() :

shift()method removes the first element from an array and returns that element.

Syntax

array.shift();

Return Value

Returns the removed single value of the array.

```
var arr = [10, 1, 2, 3].shift();
console.log("Shifted value is : " + arr )


Output
Shifted value is : 10
```

## 13 Method slice():



slice() method extracts a section of an array and returns a new array.

Syntax

array.slice( begin [,end] );

Parameter Details

begin : Zero-based index at which to begin extraction. As a negative index, start indicates an offset from the end of the sequence.

end : Zero-based index at which to end extraction.

```
var arr = ["orange", "mango", "banana", "sugar", "tea"];
console.log("arr.slice( 1, 2) : " + arr.slice( 1, 2) );
console.log("arr.slice( 1, 3) : " + arr.slice( 1, 3) );

// out put :


// arr.slice( 1, 2) : mango
// arr.slice( 1, 3) : mango,banana
```

## 14 ) Method some() :

some() method tests whether some element in the array passes the test implemented by the provided function.

Syntax

array.some(callback[, thisObject]);

Parameter Details

callback : Function to test for each element.

This Object : Object to use as this when executing callback.

```
function isBigEnough(element, index, array) {
return (element >= 10);
}
var retval = [2, 5, 8, 1, 4].some(isBigEnough);
console.log("Returned value is : " + retval );

var retval = [12, 5, 8, 1, 4].some(isBigEnough);
console.log("Returned value is : " + retval );
// Output
// Returned value is : false
// Returned value is : true
```

## 15) Method sort()

sort() method sorts the elements of an array.

Syntax

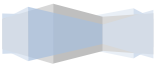
array.sort( compareFunction );

Parameter Details

compareFunction : Specifies a function that defines the sort order. If omitted, the array is sorted lexicographically.

Return Value





```
var arr = new Array("orange", "mango", "banana", "sugar");
var sorted = arr.sort();
console.log("Returned string is : " + sorted );
// Output
// Returned string is : banana,mango,orange,sugar
```

## 16) Method splice()

splice() method changes the content of an array, adding new elements while removing old elements.

Syntax

array.splice(index, howMany, [element1][, ..., elementN]);

Parameter Details

index : Index at which to start changing the array.

howMany : An integer indicating the number of old array elements to remove. If howMany is 0, no elements are removed.

element1, ..., elementN : The elements to add to the array. If you don't specify any elements, splice simply removes the elements from the array.



```
var arr = ["orange", "mango", "banana", "sugar", "tea"];
var removed = arr.splice(2, 0, "water");
```

```
console.log("After adding 1: " + arr );
console.log("removed is: " + removed);

removed = arr.splice(3, 1);
console.log("After adding 1: " + arr );
console.log("removed is: " + removed);

// output :

// After adding 1: orange,mango,water,banana,sugar,tea
// removed is:
// After adding 1: orange,mango,water,sugar,tea
// removed is: banana
```

## 17) toString();

toString() method returns a string representing the source code of the specified array and its elements.

Syntax

array.toString();

```
var arr = new Array("orange", "mango", "banana", "sugar");
var str = arr.toString();
console.log("Returned string is : " + str );
// Output
// Returned string is : orange,mango,banana,sugar
```

## 18) Method unshift()

unshift() method adds one or more elements to the beginning of an array and returns the new length of the array.

Syntax

array.unshift( element1, ..., elementN );

Paremeter Details

element1, ..., elementN :

The elements to add to the front of the array.

```
var arr = new Array("orange", "mango", "banana", "sugar");
var length = arr.unshift("water");
console.log("Returned array is : " + arr );
console.log("Length of the array is : " + length );

// Output
// Returned array is : water,orange,mango,banana,sugar
// Length of the array is : 5
```