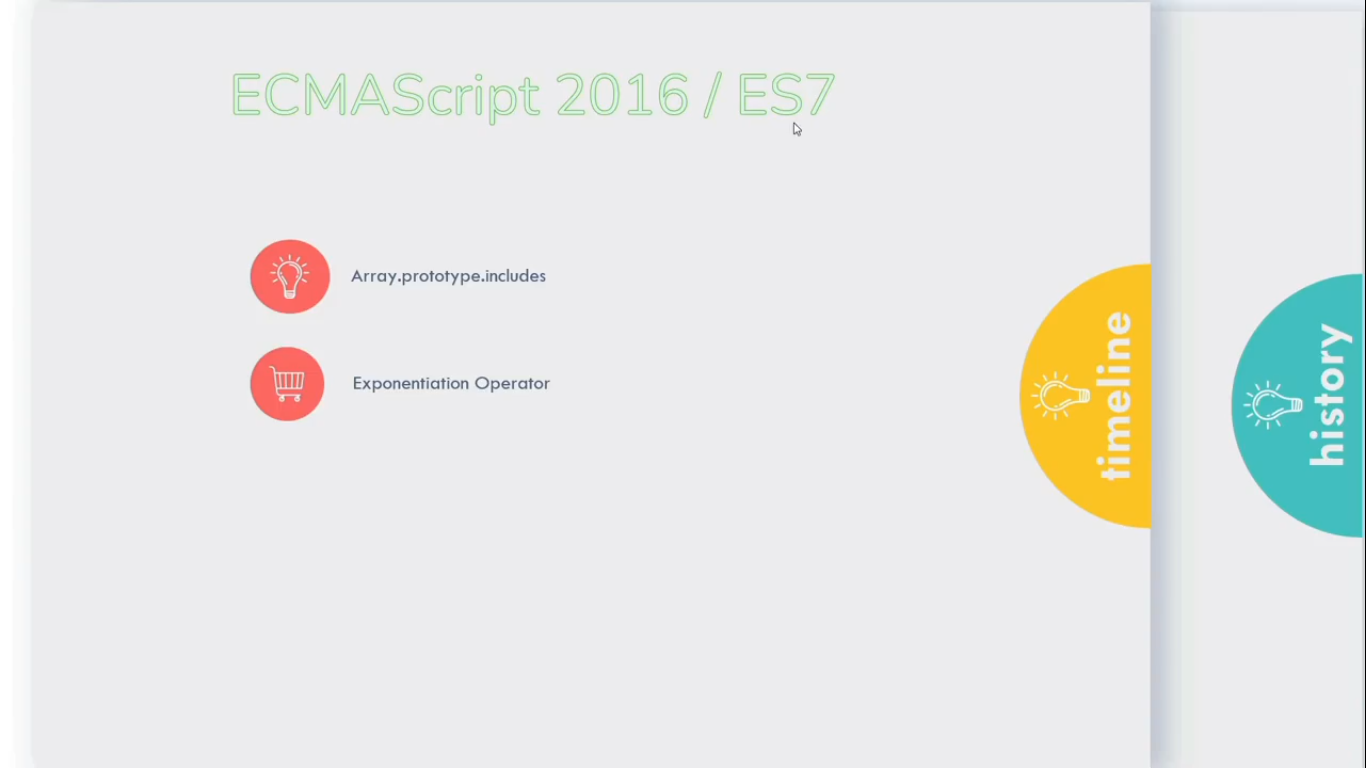
**New Features after ES6 (ES7, ES8 and so on)**

**ES7:-**Valuable Updates came in ECMAScript **2016 / ES7**

****

1. **Array.includes**

**Example:**

const colors = ['red', 'blue', 'black', 'green'];  
 console.log(colors.includes('red'));  
**//Check wheather ‘red’ value is included in the colors array or not**

**Output:**true

1. **Exponentiation Operator (\*\*)**

**Example:**

console.log(2\*\*3);  
**//2\*\*3 means 2^3 or 2x2x2 = 8**

**Output:**8

**ES8:-**Valuable Updates came in ECMAScript **2017 / ES8**

Few Important Features are:

1. String Padding
2. Object.values
3. Object.entries()
4. Spread Properties / Operator (Intorduced for **Objects**)
5. Async Function (**Most Important** discussed in Advance JS)
6. **String Padding**

**Example - 1: (Try this example at console of Chrome)**const name = 'Chandan'

name.padEnd(10)

**Output:**  
'Chandan ' **//10 spaces at the end of Chandan**

**Example- 2: (Try this example at console of Chrome)**const name = 'Chandan'

name.padStart(10)

**Outptu:**  
' Chandan' **//10 spaces at the beigning of Chandan**

1. **Object.values() //convert object’s property’s values into array**

**Example:**

const bio = {

name: `Chandan`,  
age: 20

};

console.log(Object.values(bio));  
console.log(Object.values(bio)[1]);

**Output:**['Chandan', 20]  
20

1. **Object.entries() //convert object’s property & values pair into array**

**Example:**

const bio = {

name: `Chandan`,  
age: 20

};

console.log(Object.entries(bio));  
console.log(Object.entries(bio)[1]);

**Output:  
//First Output***[Array(2), Array(2)]*  
**0**: (2) ['name', 'Chandan']  
**1**: (2) ['age', 20]

**//Second Output**  
['age', 20]

1. **Spread Properties / Operator**

**Eample:   
//Here, Spread Operator work with Objects exactly same like worked with Array**const bio = {

name: `Chandan`,

age: 20

};

const myBio = {...bio, Branch: 'CSFE'}

console.log(myBio);

**Output:**{name: 'Chandan', age: 20, Branch: 'CSFE'}

**ES9:-**Important Updates came in ECMAScript **2018 / ES9.**

1. Array.{flat, flatMap}

Previously, By default we can only flat multi-dimensional array to 1-Dimensional up to only one level using flat method.

But now we can flat it to any level up to Infinity by passing level (2 to Infinity ) as parameter via flat method.

**Example -1** **//without passing parameter i.e. By Default flat method**

const name = ['Chandan','Kumar','Abhishek', ['Dog', 'Cat', 'Lion', ['Delhi', 'Mumbai', 'Goa']]];

console.log(name.flat());

**Output:**['Chandan', 'Kumar', 'Abhishek', 'Dog', 'Cat', 'Lion', Array(3)]  
**//** Array(**3**) 🡪 3 indicating the size/length of the array

**Example -2** **//without passing parameter i.e. By Default flat method**

const name = ['Chandan','Kumar','Abhishek', ['Dog', 'Cat', 'Lion', ['Delhi', 'Mumbai', 'Goa']]];

console.log(name.flat(Infinity));

**Output:**['Chandan', 'Kumar', 'Abhishek', 'Dog', 'Cat', 'Lion', 'Delhi', 'Mumbai', 'Goa']

1. **Object.fromentries()**

As we discussed Object.entries() method in ES8, that helps in coverting an object to an array.

On the contrary, Object.fromentries() method in ES9, help s in getting back the converted object into an array.

**Example:**

const bio = {

name: `Chandan`,  
age: 20

};

const getObj = Object.entries(bio);

Object.fromEntries(getObj);

**Output:**{name: 'Chandan', age: 20}

**ES2020:-**

Important Updates came in ECMAScript **2020 / ES2020.**

Introduced Important Features:

* BIGINT

let maxJSnum= **Number.MAX\_SAFE\_INTEGER;**

console.log(maxJSnum);

**Output:**9007199254740991 **//Maximum JavaScript Number**

**Note:  
Number.MAX\_SAFE\_INTEGER** gives this numbers (**9007199254740991**), which is the maximum JavaScript Number on which any kind of further arithmatic operations start returning wrong result most of the time.   
**Example:**

console.log(9007199254740991 + 20); **//Addition**  
**Output**:  
9007199254741012  
**//The result is wrong and its datatype is ‘Number’**

But after ES2020, we can deal with such condition. Whenever we need arithmetic operations on huge numbers that would equal or Greater than this number (**9007199254740991).** We’ve to follow a method like the following example:

**Example:**

const total = 9007199254740991n + 20n; **//Addition**console.log(total);  
console.log(typeof(total));

**Output:**9007199254741011n  
‘bigint'  
**//The result is correct and its datatype is ‘BIGINT’**