OBJECT™
TECHNOLOGIES

Arrays in Java script

What will be covered

OBJECT

- Built in objects
- Introduction to JavaScript arrays
- Creating array
- Accessing array elements
- Iterating through an array
- Basic operations on array
- Advanced Operations

Built in objects

OBJECT

- Objects, in JavaScript, is it's most important data-type and forms the building blocks for modern JavaScript. These objects are quite different from JavaScript's primitive data-types(Number, String, Boolean, null, undefined and symbol) in the sense that while these primitive data-types all store a single value each.
- Objects are more complex and each object may contain any combination of these primitive data-types as well as reference data-types.
- Loosely speaking, objects in JavaScript may be defined as an unordered collection of related data, of primitive or reference types, in the form of "key: value" pairs. e.g.

```
let center = {
    name : "Know-IT",
    location : "Pune",
    established : "1999"
}
```

Introduction to JavaScript arrays

OBJECT

- In JavaScript, an array is an ordered list of values. Each value is called an element specified by an index.
- An array in javascript can hold values of different types. For example, you can have an array that stores the number and string, and boolean values.
- The length of an array is dynamically sized and auto-growing. In other words, you don't need to specify the array size upfront.

Creating array

OBJECT

- JavaScript provides you with two ways to create an array. The first one is to use the Array constructor.

```
let scores = new Array(); //declares an empty array
let scores = new Array(10); declares  array with an initial size
let scores = new Array(9,10,8,7,6); //declares array with some elements
```

- When a value of another type like string is apssed into the Array() constructor, an array with an element of that value is created.

```
let signs = new Array('Red'); // creates an array with one element 'Red'
```

- JavaScript allows you to omit the new operator when you use the array constructor like : let artists = Array();
- This way is used very rarely to create array.
- The more preferred way to create an array is to use the array literal notation:

```
let arrayName = [element1, element2, element3, ...];
```

Creating array

- The array literal form uses the square brackets [] to wrap a comma-separated list of elements.

```
let colors = ['red', 'green', 'blue'];
let emptyArray = [];  //this creates empty array
```

Accessing array elements

OBJECT

- JavaScript arrays are zero-based indexed. In other words, the first element of an array starts at index 0, the second element starts at index 1, and so on.
- To access an element in an array, an index in the square brackets [] needs to be specified like arrayName[index].

```
let colors = ["Red","Blue","Green"];
console.log(colors[1])  //displays 'Blue'
```

- To change the value of an element, assign that value to the element using it's index

```
colors[1] = "Yellow"
```

- To access last element of an array, length property can be used to specify an index

```
console.log(colors[color.length - 1]
```

Iterating through an array

OBJECT TECHNOLOGIES

- With JavaScript, the full array can be accessed by referring to the array name:
- let colors = ["Red","Blue","Green"];
- document.write(colors);
- Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays.
- Using for loop, array elements can be iterated one by one.

```
<script>
array = [ 1, 2, 3, 4, 5, 6 ];
for (index = 0; index < array.length; index++) {
    console.log(array[index]);
}
</script>
```

- while loop even can be used to iterate through all array elements

Iterating through an array

OBJECT

- The forEach method calls the provided function once for every array element in the order.

```
<script>
index = 0;
array = [ 1, 2, 3, 4, 5, 6 ];

array.forEach(myFunction);
function myFunction(item, index)
{
    console.log(item);
}</script>
```

Basic operations on array

OBJECT

- Adding an element to the end of an array

```
let colors = ["Red","Blue","Green"];
colors.push("Magenta");   //add "Magenta" as last element
```

- Adding an element to the beginning of an array

```
colors.unshift("Yellow");  //add "Yellow" as first element
```

- Removing an element from the end of an array

```
colors.pop();     //removes "Magenta" element
```

- Removing an element from the beginning of an array

```
colors.shift();    //removes "Yellow" element
```

- Adding and removing from any position

JavaScript Array type provides a very powerful splice() method that allows you to insert new elements into the middle of an array and one or more elements can be inserted into an array by passing three or more arguments to the splice() method.

Basic operations on array

OBJECT TECHNOLOGIES

```
let colors = ["Red","Blue","Green"];
colors.splice(1,1,"Orange","Pink")
```

It deletes 1 element from index 1 and simultaneously add "Orange" and "Pink" at the same position. So now array elements will be Red, Orange, Pink, Green

- Concatenating array

The method arr.concat() creates a new array that includes values from other arrays and additional items.

The syntax is: arr.concat(arg1, arg2...)

It accepts any number of arguments – either arrays or values.

```
let arr = [1, 2];
console.log( arr.concat([3, 4]) ); // 1,2,3,4
```

- Slicing the array

It returns a new array copying to it all items from index start to end (not including end). Both start and end can be negative, in that case position from array end is assumed.

Basic operations on array

OBJECT

```
let arr = ["t", "e", "s", "t"];
console.log( arr.slice(1, 3) ); // e,s (copy from 1 to 3)
```

- Searching an element in the array

The methods arr.indexOf, arr.lastIndexOf and arr.includes have the same syntax and searches for an element in the array

arr.indexOf(item, from) – looks for item starting from index from, and returns the index where it was found, otherwise -1.

arr.lastIndexOf(item, from) – same, but looks for from right to left.

arr.includes(item, from) – looks for item starting from index from, returns true if found.

```
let arr = [1, 0, false]; alert( arr.indexOf(0) ); // 1
console.log( arr.indexOf(false) ); // 2
console.log( arr.indexOf(null) ); // -1
console.log( arr.includes(1) ); // true
```

Basic operations on array

- Reversing the array

  The JavaScript Array reverse method returns the array in reverse order. The syntax of the reverse() method is:

  arr.reverse()

  It does not take any parameter

Advanced operations

OBJECT

Sorting an Array

- The **sort()** method sorts the elements of an array *in place* and returns the sorted array. The default sort order is ascending, built upon converting the elements into strings. This function accepts one argument which specifies a function that defines the sort order but it is optional
- const points = [40, 100, 1, 5, 25, 10];
  points.sort(function(a, b){return a - b});
- The argument to sort() function can be provided as :
  - sort(compareFn) // Compare function
  - sort(function compareFn(firstEl, secondEl) { ... }) // Inline compare function
  - sort((firstEl, secondEl) => { ... }) // Arrow function
- The compare function should return a negative, zero, or positive value, depending on the arguments. When the sort() function compares two values, it sends the values to the compare function, and sorts the values according to the returned (negative, zero, positive) value.

Transforming array

- The map() method creates a new array by performing a function on each array element. The map() method does not execute the function for array elements without values. The map() method does not change the original array.

```
const numbers1 = [5, 4, 9, 8, 7];
const squares = numbers1.map(myFunction);
function myFunction(value, index, array) {
    return value * value;
}
```

- Note : myFunction takes 3 arguments: The item value, The item index, The array itself
- When a callback function uses only the value parameter, the index and array parameters can be omitted.

Advanced operations

OBJECT

Filtering array elements

- The filter() method creates a new array with array elements that passes a test.

```
const numbers = [45, 4, 9, 16, 25];
const over18 = numbers.filter(myFunction);    //45,25
function myFunction(value, index, array) {
    return value > 18;
}
```

- Note : In this case callback function executes a test for each element and it returns either true or false. If it returns true the element will be considered in the returned array.

Reducing array to single value

- The reduce() method runs a function on each array element to produce (reduce it to) a single value. The reduce() method works from left-to-right in the array by default. reduceRight() can be used if reductions to be done from right to left.

Advanced operations

OBJECT

```
const numbers = [45, 4, 9, 16, 25];
 let sum = numbers.reduce(myFunction);
 function myFunction(total, value, index, array) {
   return total + value;
 }
```

- Note : function takes 4 arguments: The total (the initial value / previously returned value), The item value, The item index, The array itself. Index and array can be omitted if not required.

Verifying array elements for a condition(test)

- The every() method check if all array values pass a test. It returns boolean value true only if all the elements pass a condition

```
const numbers = [45, 4, 9, 16, 25];
let allOver18 = numbers.every(myFunction);     //false
function myFunction(value, index, array) {
   return value > 18;
}
```

Arrays in Javascript

As we know, arrays can be used to store multiple values. Javascript makes use of arrays very frequently for client side scripting. In this chapter we will learn about basics of array like creating array, accessing elements from array and different array functionality. Some array functions use callback functions as arguments.

JS Objects

JavaScript object is a non-primitive data-type that allows you to store multiple collections of data.

JavaScript objects are a bit different. You do not need to create classes in order to create objects.

Here is an example of a JavaScript object.

```
// object
const student = {
   firstName: 'ram',
   class: 10
};
```

Here, student is an object that stores values such as strings and numbers.

The general syntax to declare an object is:

```
const object_name = {
  key1: value1,
  key2: value2
}
```

Here, an object object_name is defined. Each member of an object is a key: value pair separated by commas and enclosed in curly braces {}.

```
// object creation
const person = {
   name: 'John',
   age: 20
};
console.log(typeof person); // object
```

If we use typeof on the above person object, it will print type of person object as "object".

If we use typeof on the above person object, it will print type of person object as "object".

In the above example, name and age are keys, and John and 20 are values respectively.

In JavaScript, "key: value" pairs are called properties. For example,

```
let person = {
    name: 'John',
    age: 20
};
```

Here, name: 'John' and age: 20 are properties.

Accessing Object Properties

You can access the value of a property by using its key.

For example,

```
const person = {
   name: 'John',
   age: 20,
};
```

1. Using dot Notation

   Here's the syntax of the dot notation.

   ```
   objectName.key
   // accessing property
   console.log(person.name); // John
   ```

2. Using bracket Notation

   Here is the syntax of the bracket notation.

   ```
   objectName["propertyName"]
   // accessing property
   console.log(person["name"]); // John
   ```

Nested object

An object can also contain another object. e.g.

```
const student = {
```

Nested object

An object can also contain another object. e.g.

```
const student = {
   name: 'John',
   age: 20,
   marks: {
      science: 70,
      math: 75
   }
}
// accessing property of student object
console.log(student.marks); // {science: 70, math: 75}
// accessing property of marks object
console.log(student.marks.science); // 70
```

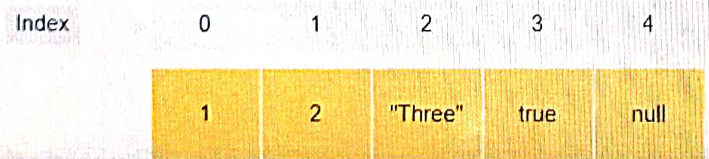
Object containing functions

In JavaScript, an object can also contain a function. For example,

```
const person = {
   name: 'Sam',
   age: 30,
   // using function as a value
   greet: function() { console.log('hello') }
}
person.greet(); // hello
```

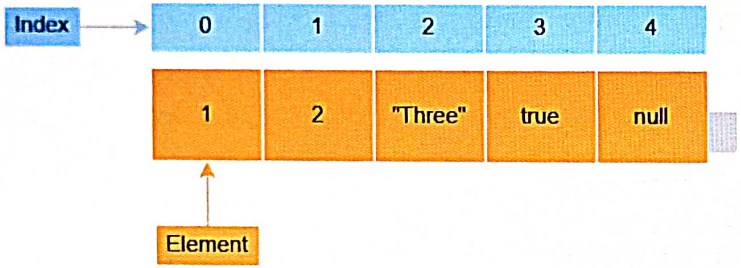
JS built in objects - array

In JavaScript, an array is an ordered list of values. Each value is called an element specified by an index:



JS built in objects - array

In JavaScript, an array is an ordered list of values. Each value is called an element specified by an index:



A JavaScript array has the following characteristics:

- First, an array can hold values of mixed types. For example, you can have an array that stores elements with the types number, string, boolean, and null.
- Second, the size of an array is dynamic and auto-growing. In other words, you don't need to specify the array size up front.

Creating JavaScript arrays

JavaScript provides you with two ways to create an array.

The first one is to use the Array constructor as follows:

```
let scores = new Array();
```

The scores array is empty, which does hold any elements.

If you know the number of elements that the array will hold, you can create an array with an initial size as shown in the following example:

```
let scores = Array(10);
```

To create an array and initialize it with some elements, you pass the elements as a comma-separated list into the Array() constructor.

For example, the following creates the scores array that has five elements (or numbers):

```
let scores = new Array(9,10,8,7,6);
let athletes = new Array(3); // creates an array with initial size 3
let scores = new Array(1, 2, 3); // create an array with three numbers 1,2 3
let signs = new Array('Red'); // creates an array with one element 'Red'
```

```
let scores = new Array(9,10,8,7,6);
let athletes = new Array(3); // creates an array with initial size 3
let scores = new Array(1, 2, 3); // create an array with three numbers 1,2 3
let signs = new Array('Red'); // creates an array with one element 'Red'
```

JavaScript allows you to omit the new operator when you use the Array() constructor. For example, the following statement creates the artists array.

```
let artists = Array();
```

The more preferred way to create an array is to use the array literal notation:

```
let arrayName = [element1, element2, element3, ...];
```

The array literal form uses the square brackets [] to wrap a comma-separated list of elements.

The following example creates the colors array that holds string elements:

```
let colors = ['red', 'green', 'blue'];
```

To create an empty array, you use square brackets without specifying any element like this:

```
let emptyArray = [];
```

Accessing JavaScript array elements

JavaScript arrays are zero-based indexed. In other words, the first element of an array starts at index 0, the second element starts at index 1, and so on.

To access an element in an array, you specify an index in the square brackets []:

```
arrayName[index]
```

The following shows how to access the elements of the mountains array:

```
let mountains = ['Everest', 'Fuji', 'Sahyadri];
console.log(mountains[0]); // 'Everest'
console.log(mountains[1]); // 'Fuji'
console.log(mountains[2]); // 'Sahyari'
```

Modify existing element

To change the value of an element, you assign that value to the element like this:

Modify existing element

To change the value of an element, you assign that value to the element like this:

```
let mountains = ['Everest', 'Fuji', 'Sahyari'];
mountains[2] = 'K2';
console.log(mountains);
```

Output:

['Everest', 'Fuji', 'K2']

Getting the array size

Typically, the length property of an array returns the number of elements. The following example shows how to use the length property:

```
let mountains = ['Everest', 'Fuji', 'Sahyari];
console.log(mountains.length); // 3
```

Iterating through arrays

Using the for loop – Instead on printing element by element, you can iterate the index using for loop starting from 0 to length of the array (ArrayName.length) and access elements at each index which means iterating over zero to the length of the array.

```
let arr = [1, 2, 3, 4, 5];
for (let i = 0; i < arr.length; i++)
   console.log(arr[i]);
```

The for...in loops through the properties of an object. It is among the most used and straightforward methods for iterating over objects. You define a variable like i in for (let i in arr), and in each iteration, the variable i will become equal to a key in arr

```
let names = ["John", "Jack", "Alice"];
for (let i in names)
   console.log(names[i]);
```

Using for..of loop If you want to access the values inside an array directly, and not the keys, then you should use for...of

```
let names = ["John", "Jack", "Alice"];
```

Using for..of loop If you want to access the values inside an array directly, and not the keys, then you should use for...of

```
let names = ["John", "Jack", "Alice"];
for (let i of names)
   console.log(i);
```

Basic array functionality

converting array to string

Method toString() converts an array to a string of (comma separated) array values.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

Ouput:

Banana,Orange,Apple,Mango

The join() method also joins all array elements into a string.

It behaves just like toString(), but in addition you can specify the separator:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
```

Output:

Banana * Orange * Apple * Mango

adding and removing from the end

The pop() method removes the last element from an array:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();
```

The pop() method returns the value that was "popped out":

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
```

The pop() method returns the value that was "popped out":

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.pop();
```

The push() method adds a new element to an array (at the end):

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");
```

The push() method returns the new array length:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.push("Kiwi");
```

adding and removing from the beginning

shift() method removes the first array element and "shifts" all other elements to a lower index.

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();
```

The shift() method returns the value that was "shifted out":

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.shift();
```

The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
```

The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");
```

The unshift() method returns the new array length:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");
```

Adding and removing from any position

The splice() method can be used to add new items to an array as well as used for deleting existing elements.

This method accepts 3 parameters(third is optional)

The first parameter defines the position where new elements should be added (spliced in).

The second parameter defines how many elements should be removed.

The rest of the parameters define the new elements to be added.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 2, "Lemon", "Kiwi");
console.log(fruits)
```

Above method removes 2 elements from index 2 and adds "Lemon" and "Kiwi" at that position

Banana,Orange,Lemon,Kiwi

slicing array

The slice() method slices out a piece of an array into a new array.

This example slices out a part of an array starting from array element 1 ("Orange"):

Example

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1);
console.log(citrus.toString);
```

slicing array

The slice() method slices out a piece of an array into a new array.

This example slices out a part of an array starting from array element 1 ("Orange"):

Example

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1);
console.log(citrus.toString);
```

Ouput:

Orange, Lemon, Apple, Mango

concatenating array

The concat() method creates a new array by merging (concatenating) existing arrays:

Example (Merging Two Arrays)

```
const myGirls = ["Cecilie", "Lone"];
const myBoys = ["Emil", "Tobias", "Linus"];
const myChildren = myGirls.concat(myBoys);
```

The concat() method can take any number of array arguments:

The concat() method can also take strings as arguments:

Advanced array functionality

Callback function

A function is a block of code that performs a certain task when called. In JavaScript, a function can be passed as an argument to a function. This function that is passed as an argument inside of another function is called a callback function. For example,

```
// callback function
function callMe() {
   console.log('I am callback function');
}
// function definition
//second argument is a function
function greet(name, callback) {
   console.log('Hi' + ' ' + name);
```

Advanced array functionality

Callback function

A function is a block of code that performs a certain task when called. In JavaScript, a function can be passed as an argument to a function. This function that is passed as an argument inside of another function is called a callback function. For example,

```
// callback function
function callMe() {
   console.log('I am callback function');
}
// function definition
//second argument is a function
function greet(name, callback) {
   console.log('Hi' + ' ' + name);
   callback();
}
// passing function as an argument
greet('Peter', callMe);
```

Arrays in javascript support some functionality which needs another function as a argument like forEach, map, filter, reduce, every etc.

Function which needs to be passed as argument to these functions can be defined separately or can be passed as anonymous funtion or can be passed as arrow function.

Sorting an array

The **sort()** method sorts the elements of an array in place and returns the reference to the same array which is now sorted.

The default sort order is ascending, built upon converting the elements into strings, then comparing their sequences of UTF-16 code units values.

The time and space complexity of the sort cannot be guaranteed as it depends on the implementation.

```
const months = ['March', 'Jan', 'Feb', 'Dec'];
months.sort();
console.log(months);
// Expected output: Array ["Dec", "Feb", "Jan", "March"]
const array1 = [1, 30, 4, 21, 100000];
array1.sort();
```

Sorting an array

The **sort()** method sorts the elements of an array in place and returns the reference to the same array which is now sorted. The default sort order is ascending, built upon converting the elements into strings, then comparing their sequences of UTF-16 code units values.

The time and space complexity of the sort cannot be guaranteed as it depends on the implementation.

```
const months = ['March', 'Jan', 'Feb', 'Dec'];
months.sort();
console.log(months);
// Expected output: Array ["Dec", "Feb", "Jan", "March"]
const array1 = [1, 30, 4, 21, 100000];
array1.sort();
console.log(array1);
// Expected output: Array [1, 100000, 21, 30, 4]
```

This function accepts optional argument in the form of callback function which accepts 2 arguments and returns difference between them

```
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a - b});
```

The argument to sort() function can be provided as :

```
sort(compareFn) // Compare function
sort(function compareFn(firstEl, secondEl) { ... }) // Inline compare function
sort((firstEl, secondEl) => { ... } ) // Arrow function
```

Iterating in the array

The **forEach()** method executes a provided function once for each array element.

```
function myiter(val,idx,arr) {
   console.log("Value present at "+idx+" is "+val);
}
const array1 = ['a', 'b', 'c'];
array1.forEach(myiter);                    //using named function
array1.forEach(function(val,idx,arr){
```

Iterating in the array

The **forEach()** method executes a provided function once for each array element.

```
function myiter(val,idx,arr) {
   console.log("Value present at "+idx+" is "+val);
}
const array1 = ['a', 'b', 'c'];
array1.forEach(myiter);                    //using named function
array1.forEach(function(val,idx,arr){
   console.log("Value present at "+idx+" is "+val);
});                                        //using anonymous function
array1.forEach(element => console.log(element));        //using arrow function
```

Note :

- Callback function required for forEach() function accepts maximum 3 arguments.
- Write only those arguments which are required for the implementation
- Names of the arguments can vary
- But the sequence is important : first argument represents element, second represents index and third represents rntire array

Transforming an array

The **map()** method creates a new array populated with the results of calling a provided function on every element in the calling array. In short every element in the array gets transformed into some different value as defined by the callback function

```
const array1 = [1, 4, 9, 16];
// Pass a function to map
const map1 = array1.map(x => x * 2);
console.log(map1);
// Expected output: Array [2, 8, 18, 32]
```

In the above example every element gets converted into element multiplied by 2. Here callback function is passed as an arraow function

Converting to single value

The **reduce()** method executes a user-supplied "reducer" callback function on each element of the array, in order, passing in the return value from the calculation on the preceding element. The final result of running the reducer across all elements of

Converting to single value

The **reduce()** method executes a user-supplied "reducer" callback function on each element of the array, in order, passing in the return value from the calculation on the preceding element. The final result of running the reducer across all elements of the array is a single value.

The first time that the callback is run there is no "return value of the previous calculation". If supplied, an initial value may be used in its place. Otherwise the array element at index 0 is used as the initial value and iteration starts from the next element (index 1 instead of index 0).

```
const array1 = [1, 2, 3, 4];
// 0 + 1 + 2 + 3 + 4
const initialValue = 0;
const sumWithInitial = array1.reduce(
  (accumulator, currentValue) => accumulator + currentValue,
  initialValue
);
console.log(sumWithInitial);
// Expected output: 10
```

Filtering an array

The **filter()** method creates a shallow copy of a portion of a given array, filtered down to just the elements from the given array that pass the test implemented by the provided function.

```
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];
const result = words.filter(word => word.length > 6);
console.log(result);
// Expected output: Array ["exuberant", "destruction", "present"]
```

Every element is tested with the condition whether the length of the string is above 6. If elements passes this test, then only it will be part of the filtered array in the outcome.

Testing array elements for some condition

The **every()** method tests whether all elements in the array pass the test implemented by the provided function. It returns a Boolean value.

Filtering an array

The **filter()** method creates a shallow copy of a portion of a given array, filtered down to just the elements from the given array that pass the test implemented by the provided function.

```
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];
const result = words.filter(word => word.length > 6);
console.log(result);
// Expected output: Array ["exuberant", "destruction", "present"]
```

Every element is tested with the condition whether the length of the string is above 6. If elements passes this test, then only it will be part of the filtered array in the outcome.

Testing array elements for some condition

The **every()** method tests whether all elements in the array pass the test implemented by the provided function. It returns a Boolean value.

The **some()** method tests whether at least one element in the array passes the test implemented by the provided function. It returns true if, in the array, it finds an element for which the provided function returns true; otherwise it returns false. It doesn't modify the array.

Example :

```
//checking whether every element is even
var alleven = arr.every(v => v % 2 == 0);
document.write("all even : "+alleven);
document.write("<br/>**********<br/>");
//checking atleast one element is even
var someeven = arr.some(v => v % 2 == 0);
document.write("some even : " + someeven);
document.write("<br/>**********<br/>");
```

Assignments

1. Write a program to create an array of integers. Accept the numbers of the array from user using prompt
   1) Display all array elements in ordered list using normal for loop and using for each loop
   2) Accept the element to be searched in the array from user. Check whether the element is present and if present at which index

Example :

```
//checking whether every element is even
var alleven = arr.every(v => v % 2 == 0);
document.write("all even : "+alleven);
document.write("<br/>**********<br/>");
//checking atleast one element is even
var someeven = arr.some(v => v % 2 == 0);
document.write("some even : " + someeven);
document.write("<br/>**********<br/>");
```

Assignments

1. Write a program to create an array of integers. Accept the numbers of the array from user using prompt
   1) Display all array elements in ordered list using normal for loop and using for each loop
   2) Accept the element to be searched in the array from user. Check whether the element is present and if present at which index
   3) Sort the numbers on numeric values
      a. using named function
      b. using anonymous function
      c. using arrow function
   4) Display all the elements as bulleted list using forEach
      a. using named function
      b. using anonymous function
      c. using arrow function
   5) display the sum, average, maximum and minimum number in that array.
   6) check whether all numbers are even
   7) check whether all numbers are odd
   8) check whether all numbers are prime
   9) create 2 arrays for even and odd numbers(use filter function)
2. Create a slide show of images. Web page should contain an area to display image and previous and next buttons. On click of these buttons display a slide show. (First gather few images and store them in array).

Example :

```
//checking whether every element is even
var alleven = arr.every(v => v % 2 == 0);
document.write("all even : "+alleven);
document.write("<br/>**********<br/>");
//checking atleast one element is even
var someeven = arr.some(v => v % 2 == 0);
document.write("some even : " + someeven);
document.write("<br/>**********<br/>");
```

Assignments

1. Write a program to create an array of integers. Accept the numbers of the array from user using prompt
   1) Display all array elements in ordered list using normal for loop and using for each loop
   2) Accept the element to be searched in the array from user. Check whether the element is present and if present at which index
   3) Sort the numbers on numeric values
      a. using named function
      b. using anonymous function
      c. using arrow function
   4) Display all the elements as bulleted list using forEach
      a. using named function
      b. using anonymous function
      c. using arrow function
   5) display the sum, average, maximum and minimum number in that array.
   6) check whether all numbers are even
   7) check whether all numbers are odd
   8) check whether all numbers are prime
   9) create 2 arrays for even and odd numbers(use filter function)
2. Create a slide show of images. Web page should contain an area to display image and previous and next buttons. On click of these buttons display a slide show. (First gather few images and store them in array).