Arrays

Overview

The array is an **ordered collection of data**(can be primitive or non-primitive) used to store multiple values. This helps in storing an indefinite number of values.
Each item/value has an **index** attached to it, using which we can access the values. In JavaScript index starts at 0.
The array also contains a property **length** that stores the number of elements present inside the array. It changes its value dynamically as the size of the array changes.

Creating an Array

There are two ways to create an array -

- **Using square brackets -**

  1. Create an empty array as

     ```
     var arrayName = [ ]   ;
     ```

  2. Create an array with initial values as

     ```
     var arrayName = [value1, value2, ..., valueN] ;
     ```

- **Using Array Object -**

  1. Create an empty array using a **new** keyword

     ```
     var arrayName = new Array( ) ;
     ```

  2. Create an array with some length as

     ```
     var arrayName = new Array(N) ;
     where 'N' is length of array.
     ```

3. Providing the values

```
var arrayName = new Array(value1, value2, ..., valueN) ;
```

Examples : var arr = [1 , 2 , 3 , 4] ;
var arr = new Array(1 ,2 , 3 ,4) ;

❖ Both the statements do the same work

- **Using Array indexes -** Create an empty array, and then provide the elements.

Example : var arr = [ ];
arr[0]= 1 ;
arr[1]= 2 ;
arr[2]= 3 ;

❖ Creates an array [1 , 2 , 3]

Placing Elements at Outside Array Range

When a value is assigned to a positive index outside the range of the array, then the array stores this value at the specified index and all other indices before it are filled with **undefined.** The length of the array also changes to **index+1.**

When a negative value is used, the array stores the element as a key-value pair, where the negative index is the key and the element to be inserted is the value.

Example : var arr = [1 , 2 , 3] ;
arr[5] = 10 ;

❖ Initially, the array has a size of 3; now, the 10 is pushed at the 5th index, size becomes 6, and the remaining index is filled with **undefined** values

arr becomes : [1 , 2 , 3 , undefined , undefined , 10] ;

Accessing Element in Array

You can access the individual elements of the array using the **square bracket notation.**

Example:
```
var arr = [ 1 , 2 , 3 ] ;
console.log( arr[1] ) ;  // Output 2
```

Modify the value

Example :
```
var arr = [ 1 , 2 , 3 ] ;
arr[1] = 20 ;
```
- ❖ array now becomes - [1 , 20 , 3]

NOTE : When using an array inside an array, you can access the value of the inner array directly

Example :
```
var arr = [ 1 , 2 , [ 3 , 4 ] ] ;

arr[2][1] ; //  return 3
arr[5]  ;     // return undefined
arr[-1]  ;   //return undefined
```

If you access the array outside its range, i.e., pass an invalid index(whether negative or greater than the array's length), then **undefined** is returned.

Heterogeneous in Nature

It can contain different types of values at the same time. Also, the array can store primitive and non-primitive values.

Examples :
```
var arr = [ 1 , 2 , "Coding Ninjas" ] ;
```
- ❖ arr contains both Number and String type values

```
var arr = [ 1 , "Coding Ninjas" , [3,4] ] ;
```
- ❖ arr contains Number, String and Array type values

Functions on Arrays

1. push Method :

The push( ) method **adds one or more elements to the end of the array and returns the new length of the array**. It uses the length property to add elements.
If the array is empty, the push( ) method will create the length property and add an element.
In case you are adding multiple elements, separate them using a comma(,).

Example : var arr = [1 , 2 , 3] ;
arr.**push**(4) ;

❖ arr becomes [1 , 2 , 3 , 4]

arr.**push**(5,6) ;

❖ arr becomes [1 , 2 , 3 , 4 , 5 , 6]

2. pop Method :

The pop() method is used to **remove the last element from the array and return that element**. It also decreases the length of the array by 1.
Using pop on an empty array returns 'undefined'.

Example : var arr = [1 , 2 , 3] ;
arr.**pop( )** ; // returns 3

3. shift Method :

The shift( ) method is used to **remove the first element from an array and return that element.**
If the length property is 0, i.e. empty array, then **undefined** is returned.

Example : var arr = [1, 2 , 3] ;
arr.**shift**( ) ; // returns 1

var arr = [ ] ;
arr.**shift**( ) ; // returns **undefined**

4. unshift Method :

The unshift( ) method is used to **add one or more elements to the beginning of the array and returns the new length of array.**
In case, you are adding multiple elements, separate them using a comma(,).

Example : var arr = [1 , 2 , 3] ;
arr.**unshift**(0) ;

- ❖ arr becomes [0 , 1 , 2 , 3]

arr.**unshift**(10) ;

- ❖ arr becomes [10 , 0 , 1 , 2 , 3]

5. indexOf Method :

The indexOf( ) method is used to **return the first index at which the given element is found in the array**. If the element is not found, then -1 is returned.

By default, the whole array is searched, but you can provide the start index from which the search should begin. It is optional. If the index provided is negative, the offset is set from the end of the array, and a search in the opposite direction is done.

Example : var arr = [1 , 2 , 3 , 2 , 5] ;
arr.**indexOf**(2); // return 1
arr.**indexOf**(2 , 2); // return 3

6. splice Method :

The splice( ) method is used to **remove or replace or add elements to an array**. If an element is removed, it returns the array of deleted elements. If no elements are removed, an empty array is returned.

Syntax : arr.**splice**(start , deleteCount , item1, ..., itemN) ;

Example : var arr = [1, 2 , 3 , 4] ;
arr.**splice**(1 , 1) ;

- ❖ arr becomes - [1 , 3 , 4]

7. reverse Method :

The reverse method is used to **reverse the content of the array** and return the new reversed array. This means that the first element becomes last and vice-versa.

Example : var arr = [1 , 2 , 3 , 4] ;
arr.**reverse**( ) ;

- ❖ arr becomes - [4 , 3 , 2 , 1]

8. sort Method :

The sort( ) method is used to **sort the elements of an array and return the sorted array**. The sort is done by converting the elements to string and then comparing them.

Example : var arr = [1 , -2 , 13 , 4] ;
arr.**sort**( ) ;

- ❖ arr becomes - [-2 , 1 , 4 , 13]

9. join Method

The join( ) method is used to **concatenate all the elements in an array and return a new string.**
They are separated by comma(,) by default, but you can also provide your separator as a string. It is optional to provide a separator.

Example : var arr = [1, 2 , 3 , “Coding Ninjas”] ;
arr.**join**(' ,') ; // returns **1,2,3,Coding Ninjas** as a String

var arr = [1, 2 , null , 3] ;
arr.**join**(' ,') ; // returns **1,2,,3** as a String

NOTE : If an element in the array is **undefined** or **null**, it is converted into an empty string.

10.toString Method

The toString( ) method is used to **return the array in the form of a string**. The string contains all the elements separated by a comma.
We do not need to provide the separator as a parameter as we did in the join method.

```
Example :    var arr =  [ 1, 2 , 3 , "Coding Ninjas"]  ;
             arr.toString(  ) ;  // returns 1,2,3,Coding Ninjas as a String
```

Iterating over arrays

Iterating over the array in accessing the values and manipulating each one of them individually, using a lesser line of code. We have used two methods to iterate over the arrays.

1. **for loop:** It is used commonly to iterate over all the values of the array.

```
Example :    var arr = [10, 20, 30] ;
             for(var i=0; i<arr; ++i) {
                 console.log( arr[i]*2 ) ;
             }
    ❖ Output : 20 40 60
```

2. **forEach Method :** The forEach( ) method calls a function once for each array element.

```
Syntax :     arr.forEach( function callback(currentValue, index, array) {
                               /* Function Statements */
                     }, thisArg);
```

You can either provide a function definition as shown in the syntax above. Or you can pass the function name to it.

Example :

```
var items = [ 1 , 2 , 3 ] ;
 items.forEach( function(item) {
       console.log ( item * 10  );
  }) ;
```

Output :

```
10
20
30
```