***OBJECTS:*** 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.

For eg :

var person = { firstName: "Anil", lastName: "Lakhani", age: 25, eyeColor: "brown"};

The **name:values** pairs in JavaScript objects are called **properties**:

**Property** **Property** **Value**

firstName John

lastName Doe

age 50

eyeColor blue

**Accessing Object Properties** : objectName.propertyName or objectName["propertyName"]

For eg : person.firstName

***Object Methods :*** Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

For eg:

var person = {  
  firstName: "Anil",  
  lastName : "lakhani",  
  id       : 1589172,  
  fullName : function() {  
    return this.firstName + " " + this.lastName;  
  }  
};

## *Accessing Object Methods : objectName.methodName()*

## For eg:

## person.fullname()

Note: If you access a method **without** the () parentheses, it will return the **function definition**.

When a JavaScript variable is declared with the keyword "new", the variable is created as an object.

***Array:*** An array is a special variable, which can hold more than one value at a time.

JavaScript arrays are used to store multiple values in a single variable.

An array can hold many values under a single name, and you can access the values by referring to an index number.

Syntax: var array\_name = [item1, item2, ...];

Eg: var cars = ["Saab", "Volvo", "BMW"];

Using new keyword:

var cars = new Array("Saab", "Volvo", "BMW");

You access an array element by referring to the **index number**.

For eg :

document.getElementById("demo").innerHTML = cars[0];

gets first element from the array.

Cars[0] = “Hyundai”

This statement changes the value of the first element in cars.

## *Arrays are Objects*

## Arrays are a special type of objects. The *typeof* operator in JavaScript returns "*object*" for arrays.

## Basic difference is:

## Arrays use **numbers** to access its "elements" while Objects use **names** to access its "members".

You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

For eg:

myArray[0] = Date.now;  
myArray[1] = myFunction;  
myArray[2] = myCars;

## Array Properties

var x = cars.length;   // The length property returns the number of elements

var y = cars.sort();   // The sort() method sorts arrays

eg:

fruits = ["Banana", "Orange", "Apple", "Mango"];  
var last = fruits[fruits.length - 1];

We can access last element of the array here.

## Associative Arrays

Many programming languages support arrays with named indexes.

***Arrays with named indexes are called associative arrays (or hashes).***

JavaScript does **not** support arrays with named indexes.

In JavaScript, **arrays** always use **numbered indexes**.

For Eg

var person = [];  
person[0] = "John";  
person[1] = "Doe";  
person[2] = 46;  
var x = person.length;     // person.length will return 3  
var y = person[0];         // person[0] will return "John"5

If you use named indexes, JavaScript will redefine the array to a standard object.  
After that, some array methods and properties will produce **incorrect results**.

## The Difference Between Arrays and Objects

In JavaScript, **arrays** use **numbered indexes**.

In JavaScript, **objects** use **named indexes**

* JavaScript does not support associative arrays.
* You should use **objects** when you want the element names to be **strings (text)**.
* You should use **arrays** when you want the element names to be **numbers**.

Note: Array.isArray(array\_name);   // returns true

When you want to check if it is Array or not.

# **Array Methods:**

1. The JavaScript method toString() converts an array to a string of (comma separated) array values.

Eg:

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

result: Banana,Orange,Apple,Mango

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

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

Eg: var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.join(" \* ");

Result: Banana \* Orange \* Apple \* Mango

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

Eg:

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
 fruits.pop();     // Removes the last element ("Mango") from fruits

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

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

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
 fruits.push("Kiwi");       //  Adds a new element ("Kiwi") to fruits

The push() method returns the new array length.

5. Shifting is equivalent to popping, working on the first element instead of the last.

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

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.shift();  // Removes the first element "Banana" from fruits.

The shift() method returns the string that was "shifted out".

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

The unshift() method returns the new array length.

## Splicing an Array

## The splice() method can be used to add new items to an array.

Example:

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.splice(2, 0, "Lemon", "Kiwi");

The first parameter (2) defines the position **where** new elements should be **added** (spliced in).

The second parameter (0) defines **how many** elements should be **removed**.

The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**.

The splice() method returns an array with the deleted items.

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.splice(0, 1);        // Removes the first element of fruits

## *Merging (Concatenating) Arrays*

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

var myGirls = ["Cecilie", "Lone"];  
var myBoys = ["Emil", "Tobias", "Linus"];  
var myChildren = myGirls.concat(myBoys);   // Concatenates (joins) myGirls and myBoys

The concat() method does not change the existing arrays. It always returns a new array.

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

## Slicing an 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").

var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
var citrus = fruits.slice(1);

The slice() method creates a new array. It does not remove any elements from the source array.

The slice() method can take two arguments like slice(1, 3).

The method then selects elements from the start argument, and up to (but not including) the end argument.