UNIT-I

JAVASCRIPT

JavaScript is the scripting language of the Web.

JavaScript is used in millions of Web pages to add functionality, validate forms, detect browsers, and much more.

Introduction to JavaScript

JavaScript is used in millions of Web pages to improve the design, validate forms, detect browsers, create cookies, and much more.

JavaScript is the most popular scripting language on the Internet, and works in all major browsers, such as Internet Explorer, Mozilla Firefox, and Opera.

What is JavaScript?

- JavaScript was designed to add interactivity to HTML pages
- JavaScript is a scripting language
- A scripting language is a lightweight programming language
- JavaScript is usually embedded directly into HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
- Everyone can use JavaScript without purchasing a license

Java and JavaScript are two completely different languages in both concept and design!

Java (developed by Sun Microsystems) is a powerful and much more complex programming language - in the same category as C and C++.

What can a JavaScript Do?

- JavaScript gives HTML designers a programming tool - HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax! Almost anyone can put small "snippets" of code into their HTML pages
- JavaScript can put dynamic text into an HTML page - A JavaScript statement like this: document.write("<h1>" + name + "</h1>") can write a variable text into an HTML page

- JavaScript can react to events - A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element
- JavaScript can read and write HTML elements - A JavaScript can read and change the content of an HTML element
- JavaScript can be used to validate data - A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing
- JavaScript can be used to detect the visitor's browser - A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser
- JavaScript can be used to create cookies - A JavaScript can be used to store and retrieve information on the visitor's computer.

JavaScript advantages

(i) Fast speed: JavaScript is executed on the client side that’s why it is very fast.
(ii) Easy to learn: JavaScript is easy to learn. Anyone who has basic knowledge of programming can easily learn JavaScript.
(iii) Versatility: It refers to lots of skills. It can be used in a wide range of applications.
(iv) Browser Compatible: JavaScript supports all modern browsers. It can execute on any browser and produce the same result.
(v) Server Load: JavaScript reduces the server load as it executes on the client side.
(vi) Rich interfaces: JavaScript provides the drag and drop functionalities which can provide a rich look to the web pages.
(vii) Popularity: JavaScript is a very popular web language because it is used everywhere on the web.
(viii) Regular Updates: JavaScript is updated annually by ECMA.

JavaScript disadvantages

(i) Code Visibility: JavaScript code is visible to everyone and this is the biggest disadvantage of JavaScript.
(ii) Stop Render: One error in JavaScript code can stop the whole website from rendering.
(iii) No Multiple Inheritance: JavaScript only supports single inheritance.

JavaScript Variables

Variables are "containers" for storing information. JavaScript variables are used

to hold values expressions.

A variable can have a short name, like x, or a more descriptive name,

like carname. Rules for JavaScript variable names:

- Variable names are case sensitive (y and Y are two different variables)
- Variable names must begin with a letter or the underscore character

Note: Because JavaScript is case-sensitive, variable names are case-sensitive.

Example

A variable's value can change during the execution of a script. You can refer to a variable by its name to display or change its value.

```
<html>
<body>
<script
type="text/javascript"> var
firstname;
firstname="Welcome";
document.write(firstname);
document.write("<br />");
firstname="XYZ";
document.write(firstname);
</script>

<p>The script above declares a variable,
assigns a value to it, displays the value, change the
value, and displays the value again.</p>

</body>
</html>
```

Output :

```
Welco
me
XYZ
```

The script above declares a variable, assigns a value to it, displays the value, change the value, and displays the value again.

Declaring (Creating) JavaScript Variables

Creating variables in JavaScript is most often referred to as "declaring"

```
var x;
var carname;
```

variables. You can declare JavaScript variables with the var statement:

After the declaration shown above, the variables are empty (they have no values

yet). However, you can also assign values to the variables when you declare

them:

```
var x=5;
var carname="Scorpio";
```

After the execution of the statements above, the variable x will hold the value 5, and carname will hold the value Scorpio.

Note: When you assign a text value to a variable, use quotes around the value.

Assigning Values to Undeclared JavaScript Variables

If you assign values to variables that have not yet been declared, the variables will automatically be declared.

These statements:

```
x=5;
carname="Scorpio";
```

have the same effect as:

```
var x=5;
var carname="Scorpio";
```

Redeclaring JavaScript Variables

If you redeclare a JavaScript variable, it will not lose its original value.

```
var x=5;
var x;
```

After the execution of the statements above, the variable x will still have the value of 5. The value of x is not reset (or cleared) when you redeclare it.

DataTypes

- Numbers - are values that can be processed and calculated. You don't enclose them in quotation marks. The numbers can be either positive or negative.
- Strings - are a series of letters and numbers enclosed in quotation marks. JavaScript uses the string literally; it doesn't process it. You'll use strings for text you want displayed or values you want passed along.
- Boolean (true/false) - lets you evaluate whether a condition meets or does not meet specified criteria.
- Null - is an empty value. null is not the same as 0 -- 0 is a real, calculable number, whereas null is the absence of any value.

Data Types

TYPE	EXAMPLE
Numbers	Any number, such as 17, 21, or 54e7

Strings	"Greetings!" or "Fun"
Boolean	Either true or false
Null	A special keyword for exactly that – the null value (that is, nothing)

JavaScript Arithmetic

As with algebra, you can do arithmetic operations with JavaScript variables:

```
y=x-5;
z=y+5;
```

JavaScript Operators

The operator = is used to assign

values. The operator + is used to

add values.

The assignment operator = is used to assign values to JavaScript

variables. The arithmetic operator + is used to add values

together.

```
y=5;
z=2;
x=y+z;
```

The value of x, after the execution of the statements above is 7.

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic between variables

and/or values. Given that y=5, the table below explains the arithmetic

operators:

Operator	Description	Example	Result
+	Addition	x=y+2	x=7
-	Subtraction	x=y-2	x=3
*	Multiplication	x=y*2	x=10
/	Division	x=y/2	x=2.5
%	Modulus (division remainder)	x=y%2	x=1
++	Increment	x=++y	x=6
--	Decrement	x=--y	x=4

<u>JavaScript Assignment Operators</u>

Assignment operators are used to assign values to JavaScript variables. Given that x=10 and y=5, the table below explains the assignment operators:

Operator	Example	Same As	Result
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

<u>The + Operator Used on Strings</u>

The + operator can also be used to add string variables or text values together. To add two or more string variables together, use the + operator.

```
txt1="What a very";
txt2="nice day";
txt3=txt1+txt2;
```

After the execution of the statements above, the variable txt3 contains "What a

```
txt1="What a very ";
txt2="nice day";
txt3=txt1+txt2;
```

verynice day". To add a space between the two strings, insert a space into one of

the strings:

or insert a space into the expression:

```
txt1="What a very";
txt2="nice day";
txt3=txt1+" "+txt2;
```

After the execution of the statements above, the variable

txt3 contains: "What a very nice day"

Adding Strings and Numbers

Look at these examples:

```
x=5+5;
document.write(x);

x="5"+"5";
document.write(x);

x=5+"5";
document.write(x);

x="5"+5;
document.write(x);
```

The rule is:

If you add a number and a string, the result will be a string.

JavaScript Comparison and Logical Operators

Comparison and Logical operators are used to test for true or false.

<u>Comparison Operators</u>

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that x=5, the table below explains the comparison operators:

Operator	Description	Example
==	is equal to	x==8 is false
===	is exactly equal to (value and type)	x===5 is true x==="5" is false
!=	is not equal	x!=8 is true
>	is greater than	x>8 is false
<	is less than	x<8 is true
>=	is greater than or equal to	x>=8 is false
<=	is less than or equal to	x<=8 is true

How Can it be Used

Comparison operators can be used in conditional statements to compare values and take action depending on the result:

```
if (age<18) document.write("Too young");
```

You will learn more about the use of conditional statements in the next chapter of this tutorial.

<u>Logical Operators</u>

Logical operators are used to determine the logic between variables or values. Given that x=6 and y=3, the table below explains the

logical operators:

Operator	Description	Example
&&	and	(x < 10 && y > 1) is true
||	or	(x==5 || y==5) is false
!	not	!(x==y) is true

Conditional Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax

```
variablename=(condition)?value1:value2
```

Example

```
greeting=(visitor=="PRES")?"Dear President ":"Dear ";
```

If the variable visitor has the value of "PRES", then the variable greeting will be assigned the value "Dear President " else it will be assigned "Dear".

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- if statement - use this statement if you want to execute some code only if a specified condition is true
- if...else statement - use this statement if you want to execute some code if the condition is true and another code if the condition is false

- if...else if....else statement - use this statement if you want to select one of many blocks of code to
  be executed
- switch statement - use this statement if you want to select one of many blocks of code to be executed

If Statement

You should use the if statement if you want to execute some code only if a specified condition is true.

Syntax

```
if (condition)
{
code to be executed if condition is true
}
```

Note that if is written in lowercase letters. Using uppercase letters (IF) will generate a JavaScript error!

Example 1

```
<script type="text/javascript">
//Write a "Good morning" greeting if
//the time is less than 10
var d=new Date();
var time=d.getHours();
```

```
if (time<10)
{
document.write("<b>Good morning</b>");
}
</script>
```

Example 2

```
<script type="text/javascript">
//Write "Lunch-time!" if the time is 11
var d=new Date();
var time=d.getHours();

if (time==11)
{
document.write("<b>Lunch-time!</b>");
}
</script>
```

Note: When comparing variables you must always use two equals signs next to each other (==)!

Notice that there is no ..else.. in this syntax. You just tell the code to execute some code only if the specified condition is true.

If...else Statement

If you want to execute some code if a condition is true and another code if the condition is not true, use the if   else statement.

Syntax

```
if (condition)
{
code to be executed if condition is true
}
else
{
code to be executed if condition is not true
}
```

Example

```
<script type="text/javascript">
//If the time is less than 10,
//you will get a "Good morning" greeting.
//Otherwise you will get a "Good day" greeting.
var d = new Date();
var time = d.getHours();

if (time < 10)
{
document.write("Good morning!");
}
else
{
document.write("Good day!");
}
</script>
```

If...else if...else Statement

You should use the if....else if...else statement if you want to select one of many sets of lines to execute.

Syntax

```
if (condition1)
{
code to be executed if condition1 is true
}
else if (condition2)
{
code to be executed if condition2 is true
}
else
{
code to be executed if condition1 and
condition2 are not true
}
```

Example

```
<script type="text/javascript"> var
d = new Date()
var time = d.getHours() if
(time<10)
{
document.write("<b>Good morning</b>");
}
else if (time>10 && time<16)
{
document.write("<b>Good day</b>");
}
else
```

```
{
document.write("<b>Hello World!</b>");
}
</script>
```

The JavaScript Switch Statement

You should use the switch statement if you want to select one of many blocks of code to be executed.

Syntax

```
switch(n)
{
case 1:
  execute code block 1
  break;
case 2:
  execute code block 2
  break;
default:
  code to be executed if n is
  different from case 1 and 2
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use break to prevent the code from running into the next case automatically.

```
<script type="text/javascript">
//You will receive a different greeting based
//on what day it is. Note that Sunday=0,
//Monday=1, Tuesday=2, etc.
var d=new Date();
theDay=d.getDay();
switch (theDay)
{
case 5:
  document.write("Finally Friday");
  break;
case 6:
  document.write("Super Saturday");
  break;
case 0:
  document.write("Sleepy Sunday");
```

Example

```
  break;
default:
  document.write("I'm looking forward to this weekend!");
}
</script>
```

JavaScript Controlling(Looping) Statements

Loops in JavaScript are used to execute the same block of code a specified number of times or while a specified condition is true.

JavaScript Loops

Very often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.

In JavaScript there are two different kind of loops:

- for - loops through a block of code a specified number of times
- while - loops through a block of code while a specified condition is true

The for Loop

The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (var=startvalue;var<=endvalue;var=var+increment)
{
   code to be executed
}
```

Example

Explanation: The example below defines a loop that starts with i=0. The loop will continue to run as long as i is less than, or equal to 10. i will increase by 1 each time the loop runs.

Note: The increment parameter could also be negative, and the <= could be any comparing statement.

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=10;i++)
{
```

```
document.write("The number is " + i);
document.write("<br />");
}
</script>
</body>
</html>
```

Result

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
The number is 10
```

JavaScript While Loop

Loops in JavaScript are used to execute the same block of code a specified number of times or while a specified condition is true.

The while loop

The while loop is used when you want the loop to execute and continue executing while the specified condition is true.

```
while (var<=endvalue)
{
   code to be executed
}
```

Note: The <= could be any comparing statement.

Example

<html>

```
<body>
<script type="text/javascript">
var i=0;
while (i<=10)
{
document.write("The number is " + i);
document.write("<br />");
i=i+1;
}
</script>
</body>
</html>
```

Result

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
The number is 10
```

The do...while Loop

The do...while loop is a variant of the while loop. This loop will always execute a block of code ONCE, and then it will repeat the loop as long as the specified condition is true. This loop will always be executed at least once, even if the condition is false, because the code is executed before the condition is tested.

```
do
{
   code to be executed
}
while (var<=endvalue);
```

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
do
{
document.write("The number is " + i);
document.write("<br />");
i=i+1;
}
while (i<0);
```

Result

```
The number is 0
```

JavaScript Break and Continue

There are two special statements that can be used inside loops: break and continue.

JavaScript break and continue Statements

There are two special statements that can be used inside loops: break and continue.

<u>Break</u>

The break command will break the loop and continue executing the code that follows after the loop (if any).

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=10;i++)
{
if (i==3)
{
break;
}
document.write("The number is " + i);
document.write("<br />");
}
</script>
</body>
```

```
</html>
```

Result

```
The number is 0
The number is 1
The number is 2
```

Continue

The continue command will break the current loop and continue with the next value.

Example

```
<html>
<body>
<script type="text/javascript">
var i=0
for (i=0;i<=10;i++)
{
if (i==3)
{
continue;
}
document.write("The number is " + i);
document.write("<br />");
}
</script>
</body>
</html>
```

Result

```
The number is 0
The number is 1
The number is 2
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
The number is 10
```

Error Handling

Handling errors in JavaScript is crucial for writing robust and reliable code. There are several mechanisms and best practices for error handling in JavaScript:

- **Try...Catch Statement:**
  - Use the try...catch statement to catch and handle exceptions. Code inside the try block is executed, and if an exception occurs, it's caught and handled in the catch block.
- **Throw Statement:**
  - You can throw your own custom errors or exceptions using the throw statement.
- **Finally Block:**
  - The finally block is optional and is executed regardless of whether an exception is thrown or not.

Handling Errors Using try…catch

Syntax:

```
try{
//code that may throw an error}catch (error){
//code to handle the errror
}finally{
//optional: Code that runs regardless of success or error}
```

Example:

```
try {
  let result = 10 / 0;
  if (!isFinite(result)) {
    throw new Error("Cannot divide by zero");
  }
  console.log(result);
} catch (error) {
  console.error("Error occurred:", error.message);
} finally {
  console.log("Execution completed");
}
```

Output:

```
Error occurred: cannot divide by zero
Execution Completed
```

Throwing Custom Errors

```
function checkAge(age) {
  if (age < 18) {
    throw new Error("Age must be 18 or above");
  }
  console.log("Access granted");
}
try {
  checkAge(16);
} catch (error) {
  console.error(error.message);
  // Age must be 18 or above
}
```

Output:

```
Age must be 18 or above
```

Using finally

The finally block is executed regardless of whether an error occurred or not.

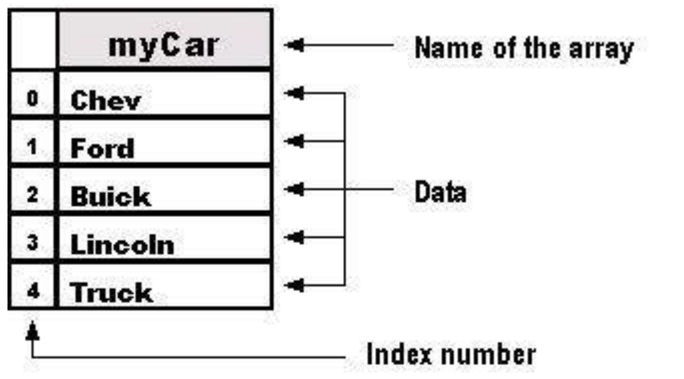
```
try {
  console.log("Trying to solve...");
  throw new Error("An error occurred");
} catch (error) {
  console.error(error.message);
} finally {
  console.log("Cleaning up...");}
```

Output:

```
Trying to solve
An error occurred
Cleaning up
```

JavaScript Arrays

An array object is used to create a database-like structure within a script. Grouping data points (*array elements*) together makes it easier to access and use the data in a script. There are methods of accessing actual databases (which are beyond the scope of this series) but here we're talking about small amounts of data.



Comparison of an array to a column of data

An array can be viewed like a column of data in a spreadsheet. The name of the array would be the same as the name of the column. Each piece of data (*element*) in the array is referred to by a number (*index*), just like a row number in a column.

An array is an *object*. Earlier, I said that an object is a thing, a collection of properties (*array elements*, in this case) grouped together.

You can name an array using the same format as a variable, a function or an object. Remember our basic rules: The first character cannot be a number, you cannot use a reserved word, and you cannot use spaces. Also, be sure to remember that the name of the array object is capitalized, e.g. Array.

The JavaScript interpreter uses numbers to access the collection of elements (i.e. the data) in an array. Each index number (as it is the number of the data in the array's index) refers to a specific piece of data in the array, similar to an ID number. It's important to remember that the index numbering of the data starts at "0." So, if you have 8 elements, the first element will be numbered "0" and the last one will be "7."

Elements can be of any type: character string, integer, Boolean, or even another array. An array can even have different types of elements within the same array. Each element in the array is accessed by placing its index number in brackets, i.e. myCar[4]. This would mean that we are looking for data located in the array myCar which has an index of "4." Since the numbering of an index starts at "0," this would actually be the fifth index. For instance, in the following array,

```
var myCar = new
Array("Chev","Ford","Buick","Lincoln","Truck"); alert(myCar[4])
```

the data point with an index of "4" would be Truck. In this example, the indexes are numbered as follows: 0=Chev, 1=Ford, 2=Buick, 3=Lincoln, and 4=Truck. When creating loops, it's much easier to refer to a number than to the actual data itself.

The Size of the Array

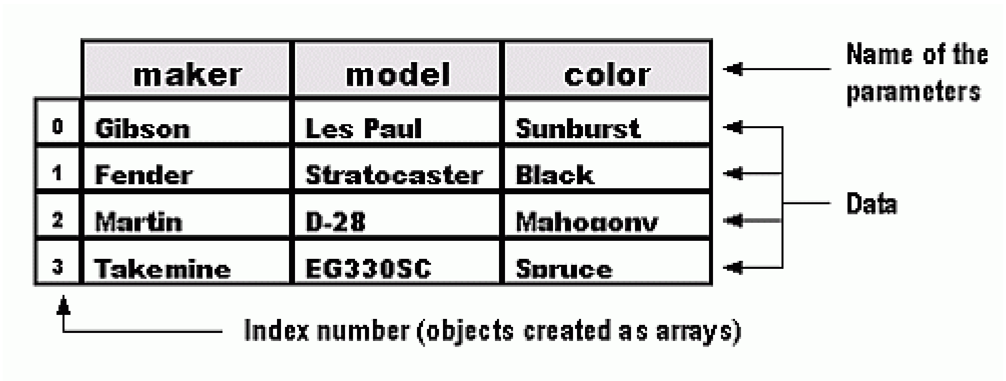
The size of an array is determined by either the actual number of elements it contains or by actually specifying a given size. You don't need to specify the size of the array. Sometimes, though, you may want to pre-set the size, e.g.:

```
var myCar = new Array(20);
```

That would pre-size the array with 20 elements. You might pre-size the array in order to set aside the space in memory.

Multidimensional Arrays

This type of an array is similar to parallel arrays. In a multidimensional array, instead of creating two or more arrays in tandem as we did with the parallel array, we create an array with several levels or "dimensions." Remember our example of a spreadsheet with rows and columns? This time, however, we have a couple more columns.



	maker	model	color
0	Gibson	Les Paul	Sunburst
1	Fender	Stratocaster	Black
2	Martin	D-28	Mahogony
3	Takemine	EG330SC	Spruce

Comparison of a multidimensional array to a column of data

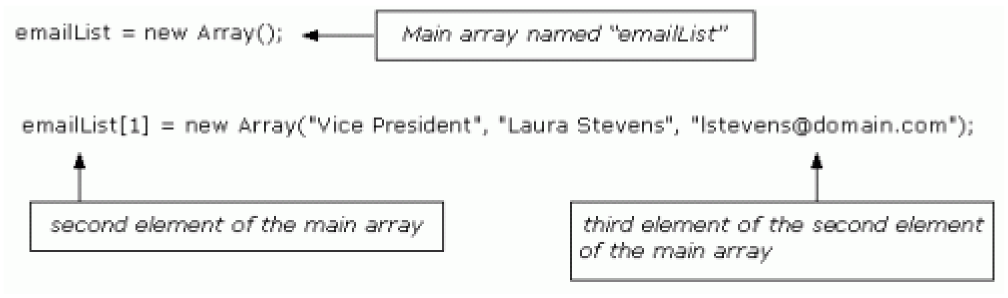
Multidimensional arrays can be created in different ways. Let's look at one of these method. First, we create the main array, which is similar to what we did with previous arrays.
var emailList = new Array();

Next, we create arrays for elements of the main array:

emailList[0] = new Array("President", "Paul Smith", "psmith@domain.com"); emailList[1] = new Array("Vice President", "Laura Stevens", "lstevens@domain.com"); emailList[2] = new Array("General Manager", "Mary Larsen", "mlarsen@domain.com"); emailList[3] = new Array("Sales Manager", "Bob Lark", "blark@domain.com");

In this script we created "sub arrays" or arrays from another level or "dimension." We used the name of the main array and gave it an index number (e.g., emailList[0]). Then we created a new instance of an array and gave it a value with three elements.

In order to access a single element, we need to use a double reference. For example, to get the e-mail address for the Vice President in our example above, access the third element "[2]" of the second element "[1]" of the array named emailList.


emailList = new Array();
Main array named "emailList"
emailList[1] = new Array("Vice President", "Laura Stevens", "lstevens@domain.com");
second element of the main array
third element of the second element of the main array

It would be written like this:

var vpEmail = emailList[1][2]
alert("The address is: "+
vpEmail)

1. We declared a variable, named it emailList, and initialized it with a value of a new instance of an array.
2. Next, we created an array for each of the elements within the original array. Each of the new arrays contained three elements.
3. Then we declared a variable named vpEmail and initialized it with the value of the third element (lstevens@domain.com) of the second element "[1]" of the array named emailList.

You could also retrieve the information using

something like: var title = emailList[1][0]
var email = emailList[1][2]
alert("The e-mail address for the " + title +" is: " + email)

1. Create Array using Literal

Creating an array using array literal involves using square brackets [] to define and initialize the array.

// Creating an Empty Array

```
let a = [];
console.log(a);
```

// Creating an Array and Initializing with Values

```
let b = [20,30,40];
console.log(b);
```

```
Output:[]
[20,30,40]
```

2. Create using new Keyword (Constructor)

The “Array Constructor” refers to a method of creating arrays by invoking the Array constructor function

// Creating and Initializing an array with values

```
let a = new Array(20,30,40);
console.log(a);
```

Output

```
[20,30,40]
```

Basic Operations on JavaScript Arrays

1. Accessing Elements of an Array

Any element in the array can be accessed using the index number. The index in the arrays starts with 0.

// Creating an Array and Initializing with Values

```
let a = ["C++”,"PHP”,"JS"];
```

// Accessing Array Elements

```
console.log(a[0]);
console.log(a[1]);
```

Output

```
C++
PHP
```

2. Accessing the First Element of an Array

The array indexing starts from 0, so we can access first element of array using the index number.

```
// Creating an Array and Initializing with Values
let a = ["C++”,"PHP”,"JS"];

// Accessing First Array Elements
let fst = a[0];

console.log("First Item: ", fst);
```

Output

```
First Item: C++
```

3. Accessing the Last Element of an Array

We can access the last array element using [array.length – 1] index number.

```
// Creating an Array and Initializing with Values
let a = ["C++”,"PHP”,"JS"];

// Accessing Last Array Elements
let lst = a[a.length - 1];

console.log(" Last Item: ", lst);
```

Output

```
Last Item: JS
```

4. Modifying the Array Elements

Elements in an array can be modified by assigning a new value to their corresponding index.

```
// Creating an Array and Initializing with Values
let a = ["C++”,"PHP”,"JS"];
console.log(a);
a[1]= "Bootstrap";
console.log(a);
```

Output

```
C++,PHP,JS
C++,Bootstrap,JS
```

5. Adding Elements to the Array

Elements can be added to the array using methods like push() and unshift().

- The push() method add the element to the end of the array.
- The unshift() method add the element to the starting of the array.

```
// Creating an Array and Initializing with Values
let a = ["HTML", "CSS", "JS"];

// Add Element to the end of Array
a.push("Node.js");

// Add Element to the beginning
a.unshift("Web Development");
console.log(a);
```

Output

```
[‘Web Development’,’HTML’,‘CSS’,’JS’,Node.js]
```

6. Removing Elements from an Array

To remove the elements from an array we have different methods like pop(), shift(), or splice().

- The pop() method removes an element from the last index of the array.
- The shift() method removes the element from the first index of the array.
- The splice() method removes or replaces the element from the array.

```
// Creating an Array and Initializing with Values
let a = ["C++”,"PHP”,"JS"];
console.log("Original Array: " + a);

// Removes and returns the last element
let lst = a.pop();
console.log("After Removing the last: " + a);

// Removes and returns the first element
let fst = a.shift();
console.log("After Removing the First: " + a);

// Removes 2 elements starting from index 1
a.splice(1, 2);
console.log("After Removing 2 elements starting from index 1: " + a);
```

Output

```
Original Array:C++,PHP,JS
After Removing the last:C++,PHP
After Removing the First:PHP,JS
After Removing 2 elements starting from index 1:PHP
```

7. Array Length

We can get the length of the array using the array length property.

```
// Creating an Array and Initializing with Values
let a = ["HTML", "CSS", "JS"];
let len = a.length;
console.log("Array Length: " + len);
```

Output

```
Array Length: 3
```

8. Increase and Decrease the Array Length

We can increase and decrease the array length using the JavaScript length property.

```
// Creating an Array and Initializing with Values
let a = ["C++","PHP","JS"]
// Increase the array length to 7
a.length = 7;
console.log("After Increasing Length: ", a);
// Decrease the array length to 2
a.length = 2;
console.log("After Decreasing Length: ", a)
```

Output

```
After Increasing Length: ['C++','PHP','JS',<4 empty items>]
After Decreasing Length:  ['C++','PHP']
```

9. Iterating Through Array Elements

We can iterate array and access array elements using for loop and forEach loop.

Example: It is an example of for loop.

```
// Creating an Array and Initializing with Values
let a = ["C++”,"PHP”,"JS"];

// Iterating through for loop
for (let i = 0; i < a.length; i++) {
    console.log(a[i])
}
```

Output

```
C++
PHP
JS
```

Example: It is the example of Array.forEach() loop.

```
// Creating an Array and Initializing with Values
let a = ["HTML", "CSS", "JS"];

// Iterating through forEach loop
a.forEach(function myfunc(x) {
    console.log(x);
});
```

Output

```
C++
PHP
JS
```

10. Array Concatenation

Combine two or more arrays using the concat() method. It returns new array containing joined arrays elements.

```
// Creating an Array and Initializing with Values
let a = ["C++”,"PHP”,"JS","React"];
let b = ["Node.js", "Expess.js"];

// Concatenate both arrays
let concateArray = a.concat(b);

console.log("Concatenated Array: ", concateArray);
```

Output

```
Concatenated Array:[‘C++.,’PHP’,’JS’,’React’,’Node.js’,‘Expess.js’]
```

11. Conversion of an Array to String

We have a builtin method toString() to converts an array to a string.

```
// Creating an Array and Initializing with Values
let a = ["C++”,"PHP”,"JS"];

// Convert array ot String
console.log(a.toString());
```

Output

```
C++,PHP,JS
```

12. Check the Type of an Arrays

The JavaScript typeof operator is used ot check the type of an array. It returns “object” for arrays.

```
// Creating an Array and Initializing with Values
let a = ["HTML", "CSS", "JS"];

// Check type of array
console.log(typeof a);
```

Output

```
object
```

JavaScript Array Methods

To help you perform common tasks efficiently, JavaScript provides a wide variety of array methods. These methods allow you to add, remove, find, and transform array elements with ease.

Types of methods

- JavaScript Array length
- JavaScript Array toString() Method
- JavaScript Array join() Method
- JavaScript Array delete Operator
- JavaScript Array concat() Method
- JavaScript Array flat() Method
- JavaScript Array.push() Method
- JavaScript Array.unshift() Method
- JavaScript Array.pop() Method
- JavaScript Array.shift() Method
- JavaScript Array.splice() Method
- JavaScript Array.slice() Method
- JavaScript Array some() Method
- JavaScript Array map() Method
- JavaScript Array filter() method
- JavaScript Array reduce() Method
- JavaScript Array reverse() method
- JavaScript Array values() method

- JavaScript Array Complete Reference

1. JavaScript Array length

The length property of an array returns the number of elements in the array. It automatically updates as elements are added or removed.

Example:

```
let a = ["JAVA", "HTML", "JS", "React"];
console.log(a.length);
```

Output

```
4
```

In this example

- The code defines an array 'a' containing the elements "["JAVA", "HTML", "JS", "React" a.length returns the number of elements in the array.

2. JavaScript Array toString() Method

The toString() method converts the given value into the string with each element separated by commas.

Example:

```
let a  = ["jJAVA", "C PROGRAM", "JS", "HTML"];
let s = a.toString();
console.log(s);
```

OUTPUT:

```
jJAVA ,C PROGRAM , JS, HTML
```

In this example

- The code defines an array "a" containing the elements ["jJAVA", "C PROGRAM", "JS", "HTML"
- The toString() method converts the array 'a' into a string.

3. JavaScript Array join() Method

This join() method creates and returns a new string by concatenating all elements of an array. It uses a specified separator between each element in the resulting string.

Example:

```
let a = ["jJAVA", "C PROGRAM", "JS", "HTML"];
console.log(a.join('|'));
```

Output

```
jJAVA ,C PROGRAM , JS, HTML
```

In this example

The code defines an array ‘a’ with the elements In this example

- The code defines an array ‘a’ with the elements "jJAVA", "C PROGRAM", "JS", "HTML"];
- The join(‘|’) method combines the array elements into a single string, with each element separated by a pipe (|) character.

4. JavaScript Array delete Operator

Th**e delete operator** is used to delete the given value which can be an object, array, or anything.

example

```
let emp = {
    firstName: "vijay",
    lastName: "ramya",
    salary: 40000
}

console.log(delete emp.salary);
console.log(emp);
```

Output

```
True
= {  firstName: "vijay", lastName: "ramya"}
```

In this example

- The delete emp.salary statement removes the salary property from the emp object and returns true if successful.
- After deletion, console.log(emp) prints the updated object without the salary property.

5. JavaScript Array concat() Method

The **concat() method** is used to concatenate two or more arrays and it gives the merged array.

example

```
let a1 = [11, 12, 13];
let a2 = [14, 15, 16];
let a3 = [17, 18, 19];

let newArr = a1.concat(a2, a3);
console.log(newArr);
```

Output

```
[11, 12, 13, [14, 15, 16, 17, 18, 19]
```

In this example

- The code defines three arrays, a1, a2, and a3, and uses the concat() method to merge them into a single array newArr.
- The resulting array [11, 12, 13, 14, 15, 16, 17, 18, 19] is logged to the console, preserving the order of elements from the original arrays.

6. JavaScript Array flat() Method

The **flat() method** is used to flatten the array i.e. it merges all the given array and reduces all the nesting present in it.
example

```
const a1 = [['1', '2'], ['3', '4', '5',['6'], '7']];
const a2 = a1.flat(Infinity);
console.log(a2);
```

Output

```
['1', '2', '3', '4', '5','6', '7']
```

In this example

- The code defines a multilevel (nested) array ‘a1’ and uses the flat(Infinity) method to flatten it completely into a single-level array.

7. JavaScript Array.push() Method

The **push() method** is used to add an element at the end of an Array. As arrays in JavaScript are mutable objects, we can easily add or remove elements from the Array.

Example:

```
let a = [10, 20, 30, 40, 50];
a.push(60);
a.push(70, 80, 90);
console.log(a);
```

output

```
[10, 20, 30, 40, 50,60,70, 80, 90];
```

8. JavaScript Array.unshift() Method

The **unshift() method** is used to add elements to the front of an Array.

Example:

```
let a = [20, 30, 40];

a.unshift(10, 20);

console.log(a);
```

Output

```
[10,20,20, 30, 40]
```

9. JavaScript Array.pop() Method

The **pop() method** is used to remove elements from the end of an array.

Example:

```
let a = [20, 30, 40, 50];
a.pop();
console.log(a);
```

Output

```
[20, 30, 40]
```

10. JavaScript Array.shift() Method
The **shift() method** is used to remove elements from the beginning of an array

Example:
```
let a = [20, 30, 40, 50];
a.shift();
console.log(a);
```

Output

```
[30, 40, 50]]
```

11. JavaScript Array.splice() Method
The **splice() method** is used to Insert and Remove elements in between the Array.

Example:
```
let a = [20, 30, 40, 50];
a.splice(1, 3);
a.splice(1, 0, 3, 4, 5);
console.log(a);
```

Output

```
[20,3,4,5]
```

12. JavaScript Array.slice() Method
The **slice() method** returns a new array containing a portion of the original array, based on the start and end index provided as arguments

Example:
```
const a = [1, 2, 3, 4, 5];
const res = a.slice(1, 4);
console.log(res);
console.log(a)
```

Output

```
[2,3,4]
[1,2,3,4,5]
```

In this example

- The slice() method creates a new array by extracting elements from index 1 to 3 (exclusive of 4) from the original array.
- The original array remains unchanged, and the result is [2, 3, 4].

13. JavaScript Array some() Method

The **some() method** checks whether at least one of the elements of the array satisfies the condition checked by the argument function.

Example:

```
const a = [1, 2, 3, 4, 5];
let res = a.some((val) => val > 4);
console.log(res);
```

Output

```
True
```

In this example

- The some() method checks if at least one element in the array satisfies the provided condition.
- (val) => val > 4: The condition checks if any element in the array is greater than 4.
- The method stops as soon as it finds the first matching element, making it efficient for large arrays.

14. JavaScript Array map() Method

The **map() method** creates an array by calling a specific function on each element present in the parent array. It is a non-mutating method.

Example:

```
let a = [4, 9, 16, 25];
let sub = a.map(great);

function great() {
   return a.map(Math.sqrt);
}
console.log(sub);
```

Output

```
[[2,3,4,5], [2,3,4,5], [2,3,4,5], [2,3,4,5]]
```

In this example

- The code defines a great function, but instead of operating on the individual array ‘a’ elements, it applies **Math.sqrt()** to the entire a array, resulting in a nested array of square roots.
- The output will be an array of the same length as a, but each element will be the result of applying arr.map(Math.sqrt).

15. JavaScript Array filter() method

The filter() method in JavaScript creates a new array with all elements that pass the test implemented by the provided function. It does not modify the original array.

Example:

```
let a1 = [1, 2, 3, 4, 5]
let a2 = a1.filter((num) => num > 1)
console.log(a2)
```

Output

```
{2, 3, 4, 5]
```

In this example

- The **filter() method** is used to create a new array (a2) that contains all elements of the original array that satisfy the condition (num > 1).
- The filter() method does not modify the original array. It returns a **new array**, leaving the original one unchanged.

16. JavaScript Array reduce() Method

The **reduce() method** is used to reduce the array to a single value and executes a provided function for each value of the array (from left to right) and the return value of the function is stored in an accumulator.

Example:

```
let a = [88, 50, 25, great);

Function great(tot, num) {
   return tot - num;
}
console.log(sub);
```

Output:

```
3
```

In this example

- The reduce() method iterates over the array ‘a’ and applies the great function, which subtracts each element (num) from the running total (tot).
- For the array [88, 50, 25, 10], the calculation proceeds as: 88 – 50 – 25 – 10.

17. JavaScript Array reverse() method

The **reverse() method** is used to reverse the order of elements in an array. It modifies the array in place and returns a reference to the same array with the reversed order.

Example:

```
let a = [1, 2, 3, 4, 5];
a.reverse();
console.log(a);
```

output

```
[5,4,3,2,1]
```

In this example

- The reverse() method reverses the order of elements in the array arr in place, modifying the original array.
- After applying reverse(), the array.

18. JavaScript Array values() method

The **values()** method returns a new Array Iterator object that contains the values for each index in the array.

Example:

```
const a = ["Apple", "orange", "grape"];
const res = a.values();

for (const value of res) {
    console.log(value);
}
```

Output:

```
Apple", "orange", "grape”
```

In this example

- The values() method returns an iterator object that allows iterating over the values of the ‘a’ array.
- The for…of loop is used to iterate over the iterator and log each fruit ("Apple", "orange", "grape”) to the console