Javascript

Obectives

Understand JS
Uses of JS
Programming Using JS

Whats is JS

High level interpreted scripting language
Along with HTML & CSS JS is one of the core wen technology
Event driven, functional and imperative programming styles
API’s for working with DateTime,Arrays,DOM
initially was a client side scripting language

Features

Universal Support
Imperative and structured
Dynamically typed
Can create fully functional web application

Writing JS

```
<script>
 Js code

</script>

<script>
      document.write(“My first javascript...”)...
</script>
```

Writing JS

Can be written in the head tag
Can be written in the body tag
Can also be written in an external .js file

JS Basics

Comments

```
<script>

    // single line comment

    /*
    Multi line comment
    */
</script>
```

JS Output

JS can display data in

1. Writing into an element using innerHTML
2. Writing into the HTML page using document.write()
3. Writing into and alert box using alert() method
4. Wriring into the browser console using console.log()

JS Basics - Variables

Variables are place holders for storing data
All JavaScript variables must be identified with unique names.
These unique names are called identifier
The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with $ and _
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as name

JS Basics - Variables

Decalred using the **var** keyword

var name = “Ashish”

If you re-declare a JavaScript variable, it will not lose its value.

Arithematic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Comparison Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Logical Operators

Operator	Description
&&	logical and
||	logical or
!	logical not

Type Operators

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

Arithematic Assignment Operators

Operator	Example	Means
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

Datatypes

Defines the data held by the variable

JS Types are dynamic

```
var x;          // Now x is undefined
x = 5;          // Now x is a Number
x = "John";     // Now x is a String
```

JS Strings

A string is a series of characters like "Ashish".

Strings are written with quotes. You can use single or double quotes:

Datatypes

JS Numbers
JS Has only two types of numbers, whole numbers and decimals

JS Booleans
Variables with true or false values

JS Arrays
Declared with square brackets [ ]
Example var names = [“Ashish”, “Rahul”]
Collection or list of values represented with single name

Datatypes

JS Objects
Are writtent using curly braces { }
Properties are written as *name* **:** *value* pairs separated by ,

Example
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

null and **undefined**
Null and undefined are equal in value but different in type

Datatypes

Primitive Data
string
number
boolean
undefined

Complex Data
function
object

Functions

Function is a block of code that is designed to perform some action
Function is executed when something invokes/calls it

Syntax
Function is defined by ***function*** keyword followed by **name** and **( )**

```
function  abc( ) {

}

function abc( parameter1,paramater2){
}
```

Functions

```
function abc( parameter1,paramater2){
}
```

Function **parameters** are listed inside the ( )

Function **arguments** are the value received by the function when invoked

The code inside the function will execute when "something" invokes (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

Function return

When JavaScript reaches a return statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a return value. The return value is "returned" back to the "caller":

Function return

```
var x = myFunction(4, 3);   // Function is called, return value will end up in x

function myFunction(a, b) {
  return a * b;           // Function returns the product of a and b
}
```

Why Functions

You can reuse code: Define the code once, and use it many times.

You can use the same code many times with different arguments, to produce different results.

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius(77);
```

( ) operator

( ) operator invokes the function

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius;
```

Using the example above, toCelsius refers to the function object, and toCelsius() refers to the function result.

Accessing a function without () will return the function definition instead of the function result:

Functions

Functions used as local variables

```
var x = toCelsius(77);
var text = "The temperature is " + x + " Celsius";
```

Can be written as

```
var text = "The temperature is " + toCelsius(77) + " Celsius";
```

Conditions

In JavaScript we have the following conditional statements:

- Use if to specify a block of code to be executed, if a specified condition is true
- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false
- Use switch to specify many alternative blocks of code to be execute

The if statement

```
if (hour < 18) {
  greeting = "Good day";
}
```

Use the if statement to specify a block of JavaScript code to be executed if a condition is true.

The else statement

```
if (condition) {
  //  block of code to be executed if the condition is true
} else {
  //  block of code to be executed if the condition is false
}
```

If the hour is less than 18, create a "Good day" greeting, otherwise "Good evening":

```
if (hour < 18) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

The else if statement

Use the else if statement to specify a new condition if the first condition is false.

```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and condition2 is true
} else {
  //  block of code to be executed if the condition1 is false and condition2 is false
}
```

The else if statement

If time is less than 10:00, create a "Good morning" greeting, if not, but time is less than 20:00, create a "Good day" greeting, otherwise a "Good evening":

```
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

Switch statement

The switch statement is used to perform different actions based on different conditions.

Use the switch statement to select one of many code blocks to be executed.

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

Switch statement

Working

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.

When JavaScript reaches a break keyword, it breaks out of the switch block.

This will stop the execution of inside the block.

It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.

Switch statement

```
switch (d) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
       //other
    case 6:
    day = "Saturday";
}
```

Switch statement

The default keyword specifies the code to run if there is no case match:

```
switch (d) {
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
      text = "Today is Sunday";
    break;
  default:
      text = "Looking forward to the Weekend";
}
```

Switch statement

Sometimes you will want different switch cases to use the same code.

```
switch (d) {
  case 4:
  case 5:
       text = "Soon it is Weekend";
    break;
  case 0:
  case 6:
       text = "It is Weekend";
    break;
  default:
    text = "Looking forward to the Weekend";
}
```

Switch statement

Switch cases use strict comparison (===).

The values must be of the same type to match.

A strict comparison can only be true if the operands are of the same type.

Switch statement

In this example there will be no match for x:

```
var x = "0";
switch (x) {
  case 0:
    text = "Off";
    break;
  case 1:
      text = "On";
    break;
  default:
    text = "No value found";
}
```

For Loop

Loops can execute a block of code a number of times.

JavaScript supports different kinds of loops:

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

For Loop

The for loop has the following syntax:

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

Statement 1 is executed (one time) before the execution of the code block.
Statement 2 defines the condition for executing the code block.
Statement 3 is executed (every time) after the code block has been executed.

For Loop

```
for (i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
}
```

Statement 1 sets a variable before the loop starts (var i = 0).
Statement 2 defines the condition for the loop to run (i must be less than 5).
Statement 3 increases a value (i++) each time the code block in the loop has been executed.

For Loop

```
<script>
var cars = ["BMW", "Volvo", "Saab", "Ford"];
var i, len, text;
for (i = 0, len = cars.length, text = ""; i < len; i++) {
  text += cars[i] + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
```

While Loop

The while loop loops through a block of code as long as a specified condition is true.

```
while (condition) {
  // code block to be executed
}
```

```
while (i < 10) {
  text += "The number is " + i;
  i++;
}
```

Do while Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
do {
  // code block to be executed
}
while (condition);
```

Do while Loop

The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

```
do {
  text += "The number is " + i;
  i++;
}
while (i < 10);
```

Arrays

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

```
var cars = ["Saab", "Volvo", "BMW"];
```

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

Create a JavaScript Array.

```
var array_name = [item1, item2, ...];
```

```
var cars = ["Saab", "Volvo", "BMW"];
```

Arrays

Spaces and line breaks are not important. A declaration can span multiple lines:

```
var cars = [
  "Saab",
  "Volvo",
  "BMW"
];
```

The following example also creates an Array, and assigns values to it:

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

Access array Elements

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

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
```

Changing the value in an array

```
var cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
document.getElementById("demo").innerHTML = cars[0];
```

Access array Elements

The full array can be accessed by referring to the array name:

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

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

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

Array Properties

The length Property

The length property of an array returns the length of an array (the number of array elements).

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.length;   // the length of fruits is 4
```

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
var first = fruits[0];
```

Accessing array

Array.forEach()

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

text = "<ul>";
fruits.forEach(myFunction);
text += "</ul>";

function myFunction(value) {
  text += "<li>" + value + "</li>";
}
```

Adding elements

The easiest way to add a new element to an array is using the push() method:

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

New element can also be added to an array using the **length** property:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Lemon";    // adds a new element (Lemon) to fruits
```

Array Methods

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

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

Banana,Orange,Apple,Mango

Array Methods

The join() method also joins all array elements into a string.
It behaves just like toString(), but in addition you can specify the separator:

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

Banana * Orange * Apple * Mango

Array Methods

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

```
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":

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.pop();      // the value of x is "Mango"
```

Array Methods

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:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.push("Kiwi");   //  the value of x is 5
```

Array Methods

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 fruit
```

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

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.shift();    // the value of x is "Banana"
```

Array Methods

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

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

The unshift() method returns the new array length.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");    // Returns 5
```

Array Methods

Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator delete:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];           // Changes the first element in fruits to undefined
```

Array Methods

The sort() method sorts an array alphabetically:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();        // Sorts the elements of fruits
```

The reverse() method reverses the elements in an array.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();        // First sort the elements of fruits
fruits.reverse();     // Then reverse the order of the elements
```

Array Methods

By default, the sort() function sorts values as strings.

However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1".

Because of this, the sort() method will produce incorrect result when sorting numbers.
You can fix this by providing a compare function:

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

Array Methods

The forEach() method calls a function (a callback function) once for each array element.

```
var txt = "";
var numbers = [45, 4, 9, 16, 25];
numbers.forEach(myFunction);

function myFunction(value) {
  txt = txt + value + "<br>";
}
```

Array Methods

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.

```
var numbers1 = [45, 4, 9, 16, 25];
var numbers2 = numbers1.map(myFunction);

function myFunction(value) {
  return value * 2;
}
```

Array Methods

The filter() method creates a new array with array elements that passes a test.
This example creates a new array from elements with a value larger than 18:

```
var numbers = [45, 4, 9, 16, 25];

var over18 = numbers.filter(myFunction);

function myFunction(value) {
  return value > 18;
}
```

Dates

var d = new Date();

By default, JavaScript will use the browser's time zone and display a date as a full text string:

Date objects are created with the new Date() constructor.
There are 4 ways to create a new date object:

new Date()
new Date(year, month, day, hours, minutes, seconds, milliseconds)
new Date(milliseconds)
new Date(date string)

Dates

new Date() creates a new date object with the current date and time:

var d = new Date();

new Date(year, month, ...) creates a new date object with a specified date and time.

7 numbers specify year, month, day, hour, minute, second, and millisecond
var d = new Date(2018, 11, 24, 10, 33, 30, 0);

Dates

5 numbers specify year, month, day, hour, and minute:

```
var d = new Date(2018, 11, 24, 10, 33);
```

4 numbers specify year, month, day, and hour:

```
var d = new Date(2018, 11, 24, 10);
```

3 numbers specify year, month, and day:

```
var d = new Date(2018, 11, 24);
```

Dates

One and two digit years will be interpreted as 19th century

var d = new Date(99, 11, 24);

new Date(dateString) creates a new date object from a date string:

var d = new Date("October 13, 2014 11:13:00");

Dates

d = new Date();
document.getElementById("demo").innerHTML = d;

document.getElementById("demo").innerHTML = d.toString();

The toUTCString() method converts a date to a UTC string (a date display standard).
document.getElementById("demo").innerHTML = d.toUTCString();

The toDateString() method converts a date to a more readable format:
document.getElementById("demo").innerHTML = d.toDateString();

Dates

getFullYear()	Get the year as a four digit number (yyyy)
getMonth()	Get the month as a number (0-11)
getDate()	Get the day as a number (1-31)
getHours()	Get the hour (0-23)
getMinutes()	Get the minute (0-59)
getSeconds()	Get the second (0-59)
getMilliseconds()	Get the millisecond (0-999)
getTime()	Get the time (milliseconds since January 1, 1970)
getDay()	Get the weekday as a number (0-6)
Date.now()	Get the time. ECMAScript 5.

Date methods

The getTime() method returns the number of milliseconds since January 1, 1970:

```
var d = new Date();
document.getElementById("demo").innerHTML = d.getTime();
```

The getFullYear() method returns the year of a date as a four digit number:

```
document.getElementById("demo").innerHTML = d.getFullYear();
```

The getMonth() method returns the month of a date as a number (0-11):

```
document.getElementById("demo").innerHTML = d.getMonth();
```

Date methods

The getDate() method returns the day of a date as a number (1-31):

document.getElementById("demo").innerHTML = d.getDate();

The getHours() method returns the hours of a date as a number (0-23):

document.getElementById("demo").innerHTML = d.getHours();

The getDay() method returns the weekday of a date as a number (0-6):

document.getElementById("demo").innerHTML = d.getDay();

Date methods

The getDate() method returns the day of a date as a number (1-31):

document.getElementById("demo").innerHTML = d.getDate();

The getHours() method returns the hours of a date as a number (0-23):

document.getElementById("demo").innerHTML = d.getHours();

The getDay() method returns the weekday of a date as a number (0-6):

document.getElementById("demo").innerHTML = d.getDay();

Date methods

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

Date methods

The setFullYear() method sets the year of a date object. In this example to 2020:

```
<script>
var d = new Date();
d.setFullYear(2020);
document.getElementById("demo").innerHTML = d;
</script>
```

Date methods

The setMonth() method sets the month of a date object (0-11):

```
<script>
var d = new Date();
d.setMonth(11);
document.getElementById("demo").innerHTML = d;
</script>
```

Date comparison

```
var today, someday, text;
today = new Date();
someday = new Date();
someday.setFullYear(2100, 0, 14);

if (someday > today) {
  text = "Today is before January 14, 2100.";
} else {
  text = "Today is after January 14, 2100.";
}
document.getElementById("demo").innerHTML = text;
```