EK ←HTML CSS XML

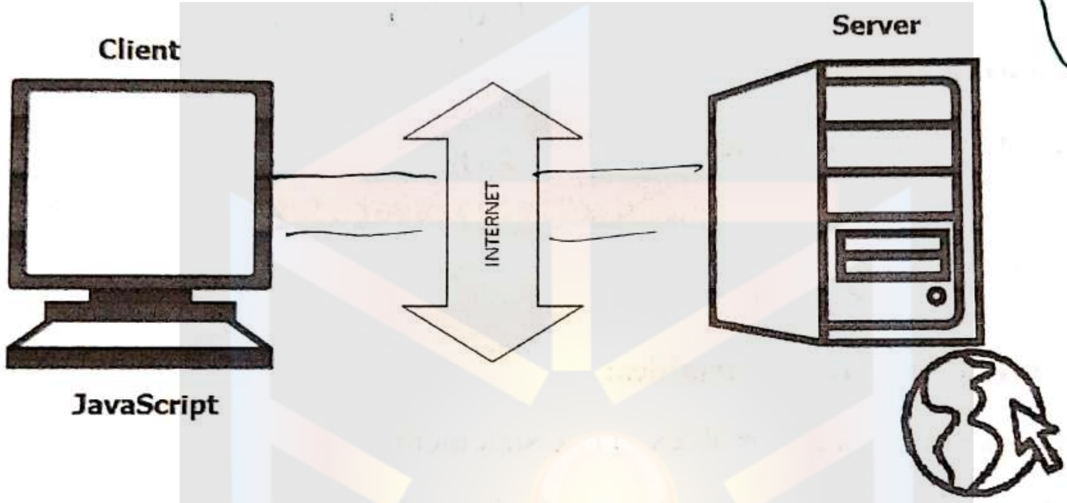
→JS
→Aj
→Py.

UNIT III

What is JavaScript? ①

JavaScript is a very powerful client-side scripting language. JavaScript is used mainly for enhancing the interaction of a user with the webpage. In other words, you can make your webpage more lively and interactive, with the help of JavaScript. JavaScript is also being used widely in game development and Mobile application development.

T.J.



You should place all your JavaScript code within **<script> tags**(<script> and </script>) if you are keeping your JavaScript code within the HTML document itself. You have to use the type attribute within the <script> tag and set its value to text/javascript like this:

```
<script type="text/javascript">
```

Hello World Example:

```
<html>
<head>
    <title>My First JavaScript
    code!!!</title> <script
    type="text/javascript">
        alert("Hello World!");
```

Hello World

<script type="text/javascript>

</script>

```
    </script>
</head>
<body>
</body>
</html>
```

JavaScript Variables

Variables

var

Variables are used to **store values** (name = "John") or **expressions** (sum = x + y).

Declaration of variable:

var name = "John"

```
var name;
```

Assigning value to variable:

```
var name;
name = "John";
```

One Statement, Many Variables:

You can declare many variables in one statement.

Start the statement with var and separate the variables by **comma**:

```
var person = "John Doe", carName = "Volvo", price = 200;
```

A declaration can span multiple lines:

int x=10, y=2;

```
var person="John Doe",
carName="Volvo",
price=200;
```

int x=10;
int x;

Re-Declaring JavaScript Variables

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

The variable carName will still have the value "Volvo" after the execution of these statements:

Example

```
var carName = "Volvo";
var carName;
```

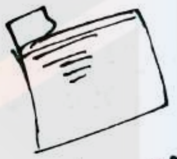
Naming Variables: Though you can name the variables as you like, it is a good programming practice to give descriptive and meaningful names to the variables. Moreover, variable names should start with a letter and they are case sensitive. Hence the variables student name and studentName are different because the letter n in a name is different (n and N).

Consider following example for variables:-

```
<html>
<head>
<title>Variables!!!</title>
<script type="text/javascript">
var one = 22;
var two = 3;
var add = one + two;
var minus = one - two;
var multiply = one * two;
var divide = one/two;
    document.write("First No: = " + one + "<br />Second
No: = " + two + " <br />");
    document.write(one + " + " + two + " = " + add +
    "<br/>");document.write(one + " - " + two + " = "
    + minus + "<br/>");document.write(one + " * " +
    two + " = " + multiply + "<br/>");
    document.write(one + " / " + two + " = " + divide
    + "<br/>");
</script>
</head>
<body>
</body>
</html>
```

~~alret~~ alert

+ - * / ' /n



document.write(" ⟷);

<br/>

First no: = 22
second no: = 3

set.

JAVASCRIPT FUNCTIONS

JavaScript Function:- ② ✓ I {

What is Function in JavaScript?

Functions are very important and useful in any programming language as they make the code reusable. A function is a block of code which will be executed only if it is called. If you have a few lines of code that needs to be used several times, you can create a function including the repeating lines of code and then call the function wherever you want.

f()

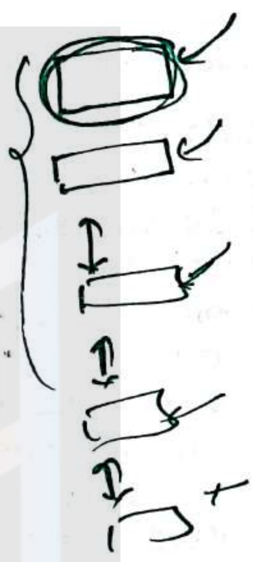
How to Create a Function in JavaScript

Use the keyword **function** followed by the name of the function.

After the function name, open and close parentheses.

After parenthesis, open and close curly braces.

Within curly braces, write your lines of code.



Syntax: $() \{ \longleftrightarrow \}$

```
function functionname() {
    lines of code to be executed
}
```

Consider the following example:-

```
<html>
<head>
    <title>Functions!!!</title>
    <script type="text/javascript">
    function myFunction()
   {
        document.write("This is a simple function.<br
        />");
   }
        myFunction();
    </script>
</head>
<body>
</body>
</html>
```

✓3

my Function ( )

Function with Arguments

You can create functions with arguments as well. Arguments should be specified within parenthesis

Syntax:

```
function functionname(arg1, arg2)
{
  lines of code to be executed
}
```

no. of vowels

Consider the following example:-

```
<html>
<head>
    <script type="text/javascript">
        var count = 0;
        function countVowels(name)
        {
            for (var i=0;i<name.length;i++)
            {
          if(name[i] == "a" || name[i] == "e" ||
name[i] == "i" || name[i] == "o" || name[i] == "u")
          count = count + 1;
            }
        document.write("Hello " + name + "!!! Your
name has " + count + " vowels.");
        }
        var myName = prompt("Please enter your name");
    countVowels(myName);
    </script>
</head>
<body>
</body>
</html>
```

prompt

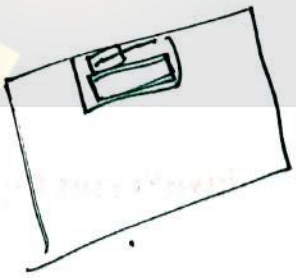
ramu

a, e, i, o, u.

name.length

4

0, 1, 2, 3

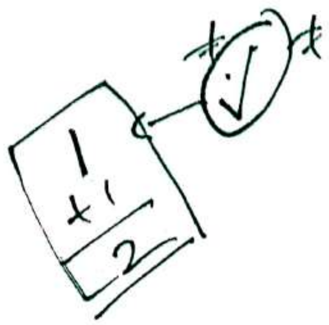


For

faju

a, e, i, o, u.

For (int i = 0;
i < n; i++)
{
=
}



JavaScript Return Value

You can also create JS functions that return values. Inside the function, you need to use the keyword **return** followed by the value to be returned.

Syntax:

```
function functionname(arg1, arg2)
{
  lines of code to be executed
  return val1;
}
```

return

Example:-

```
<html>
<head>
    <script type="text/javascript">
        function returnSum(first,
        second)
      {
        var sum = first + second;
        return sum;
      }
    var firstNo = 78;
    var secondNo = 22;
    document.write(firstNo + " + " + secondNo
+ " = " + returnSum(firstNo,secondNo));
    </script>
</head>
<body>
</body>
</html>
```

+ sum

JavaScript Operators

The Assignment Operator

JavaScript Operators ④ ③ > □ ✓ (==) (<= >=)

In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator. This is different from algebra. The following does not make sense in algebra:

x = x + 5

In JavaScript, however, it makes perfect sense: it assigns the value of x + 5 to x.

(It calculates the value of x + 5 and puts the result into x. The value of x is incremented by 5.)

The "equal to" operator is written like == in JavaScript.

The **assignment** operator (=) assigns a value to a variable.

Assignment:-

```
var x = 10;
```

The **addition** operator (+) adds numbers:

Adding

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

The **multiplication** operator (*) multiplies numbers.

Multiplying

```
  var x = 5;
  var y = 2;
  var z = x * y;
```

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic on numbers:

Operator	Description	
+	Addition	a + b
-	Subtractio n	a - b

? : *

uni

a^b ||

a^b a ** b

*	Multiplicati	a * b
**	Exponentiation	a ** b ~ a ^ b
/	Division	a/b → Q
%	Modulus	a % b → Re
++	Increment	a++ (+1) = a=a+1
--	Decrement	a -- (-1) = a= a-1

a-1 +

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

Operator	Example	Same As
=	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 += y = x = x+y

x += y

x = x+y

JavaScript String Operators

The + operator can also be used to add (concatenate) strings.

~~var~~ "Hello" + 5 || Hello 5

Example

```
var txt1 = "John";
var txt2 = "Doe";
var txt3 = txt1 + " " + txt2;
```

The result of txt3 will be:

John Doe

The += assignment operator can also be used to add (concatenate) strings:

Example

```
var txt1 = "What a very ";
txt1 += "nice day";
```

The result of txt1 will be:

What a very nice day

Adding Strings and Numbers

Adding two numbers, will return the sum, but adding a number and a string will return a string:

Example

```
var x = 5 + 5;
var y = "5" + 5;
var z = "Hello" + 5;
```

The result of *x*, *y*, and *z* will be:

```
10
55
Hello5
```

JavaScript Logical Operators

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

&& and — — T

|| or — = T

JavaScript 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

a == b

>=

JavaScript Type Operators

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

var ~ " "

typeof (nam

var name = "ramu".

typeof (nam ")

Operators and Operands

The numbers (in an arithmetic operation) are called **operands.**

The operation (to be performed between the two operands) is defined by an **operator.**

Operand	Operator	Operand
100	+	50

Operator Precedence

Operator precedence describes the order in which operations are performed in an arithmetic expression.

$\boxed{BODMAS}$

Example

100+500 * 2

```
var x = 100 + 50 * 3;
```

Is the result of example above the same as 150 * 3, or is it the same as 100 + 150?

Is the addition or the multiplication done first?

As in traditional school mathematics, the multiplication is done first.

Multiplication (*) and division (/) have higher **precedence** than addition (+) and subtraction (-).

IS ( ) + + − / * % → + −

And (as in school mathematics) the precedence can be changed by using parentheses:

Example

```
var x = (100 + 50) * 3;
```

When using parentheses, the operations inside the parentheses are computed first.

x : 100 * 10 * 10 % 10
0
: 10000 % 10

When many operations have the same precedence (like addition and subtraction), they are computed from left to right:

Example

```
var x = 100 + 50 - 3;
```

IF, IFeld
el-
such

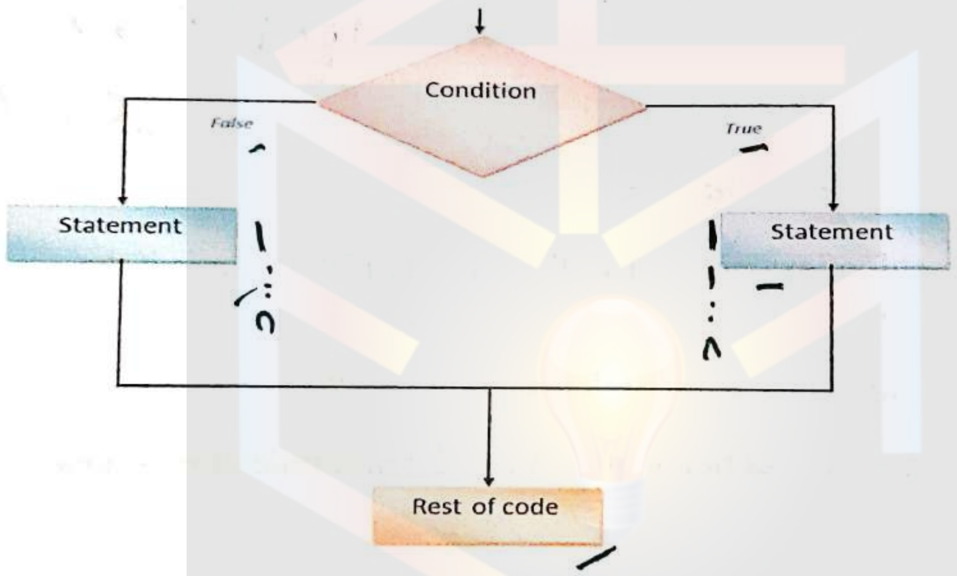
JavaScript Conditional Statements: IF, Else, Else IF

Loop

4

How to use Conditional Statements

Conditional statements are used to decide the flow of execution based on different conditions. If a condition is true, you can perform one action and if the condition is false, you can perform another action.



Different Types of Conditional Statements

There are mainly three types of conditional statements in JavaScript.

If statement

If…Else statement

If…Else If…Else statement

If statement

Syntax:

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

You can use If statement if you want to check only a specific condition.

Example:-

```
<html>
<head>
    <title>IF Statments!!!</title>
    <script type="text/javascript">
        var age = prompt("Please enter
        your age");if(age>=18)
        document.write("You are an adult
        <br />");if(age<18)
        document.write("You are NOT an adult
    <br />"); </script>
</head>
<body>
</body>
</html>
```

If…Else statement

Syntax:

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

You can use If….Else statement if you have to check two conditions and execute a different set
of codes.

Example:-

```
<html>
<head>
     <title>If...Else
     Statments!!!</title> <script
     type="text/javascript">
          // Get the current hours
          var hours = new Date().getHours();
          if(hours<12)
          document.write("Good
          Morning!!!<br />"); else
          document.write("Good
    Afternoon!!!<br />"); </script>
</head>
<body>
</body>
</html>
```

19:48:00 1-1-1999

12

If…Else If…Else statement

Syntax:

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

You can use If….Else If….Else statement if you want to check more than two conditions.

Example:-

```
<html>
<head>
    <script type="text/javascript">
        var one = prompt("Enter the first number");
        var two = prompt("Enter the second number");
        one = parseInt(one);
        two = parseInt(two);
        if (one == two)
            document.write(one + " is equal to " +
        two + ".");else if (one<two)
            document.write(one + " is less than " + two
            + ".");
        else
  document.write(one + " is greater than " + two + ".");
    </script>
</head>
<body>
</body>
</html>
```

JavaScript 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.

Syntax

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

This is how it works:

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.

Example

The getDay() method returns the weekday as a number between 0 and 6.

(Sunday=0, Monday=1, Tuesday=2 ..)

This example uses the weekday number to calculate the weekday name:

19:45:45 T

$\boxed{1}$-1-1995

```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
     day =  "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
```

1 - 17

0 → 6

5

```
    day = "Saturday";
}
```
The result of day will be:

Monday

JavaScript For Loop (5)

Loops can execute a block of code a number of times. Loops are handy, if you want to run the same code over and over again, each time with a different value.

Often this is the case when working with arrays:

Instead of writing:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

cars[0], {}
cars[1]

You can write:

```
var i;
for (i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}
```

Different Kinds of Loops

For each

JavaScript supports different kinds of loops:

for - loops through a block of code a number of times for/in - loops through the properties of an object 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

The For Loop

a=1 initiali
a=n
for(; ; (i++))
{
}
check
a<10
a<n
a>=1

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.

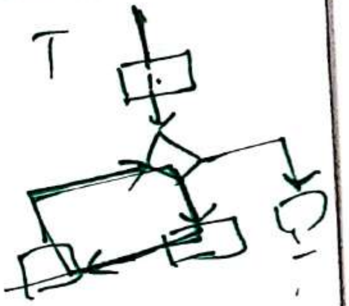
Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

Example

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

0 -14
0
T



From the example above, you can read:

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.

The For/In Loop

The JavaScript for/in statement loops through the properties of an object:

Example

```
var person = {fname:"John", lname:"Doe", age:25};

var text = "";
var x;
for (x in person) {
  text += person[x];
}
```

[m]

x

The While Loop

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

Syntax

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

Example

In the following example, the code in the loop will run, over and over again, as long as a variable (i) is less than 10:

i = 0

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

0 —> 9

The 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.

do
{
} ~~while~~ }

Syntax

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

Example

The example below uses a 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:

Example

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

0 — 9

JavaScript Break and Continue

Break & con

The break statement "jumps out" of a loop.

The continue statement "jumps over" one iteration in the loop.

The Break Statement

You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch() statement.

The break statement can also be used to jump out of a loop.

The break statement breaks the loop and continues executing the code after the loop (if any):

Example

```
for (i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
}
```

The Continue Statement

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 3:

Example

```
for (i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
}
```

JavaScript Arrays

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

Fname[ ]

0 - (n-1)

Javascript array

Example

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

cars[0]

What is an Array?

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

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
var car1 = "Saab";
var car2 = "Volvo";
var car3 = "BMW";
```

car1

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

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

Creating an Array

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

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

0 . . . . n-1

3-1

Example

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

0 1 2

3

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

1 2 3 x

Example

```
var cars = [
  "Saab",
  "Volvo",

  "BMW"
];
```

Access the Elements of an Array

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

This statement accesses the value of the first element in cars:

```
var name = cars[0];
```

Example

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

cars[0]

Changing an Array Element

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

```
cars[0] = "Opel";
```

cars[0] = ;

Example

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

Access the Full Array

With JavaScript, the full array can be accessed by referring to the array name:

0
1
2
3

Example

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

Array Properties and Methods

The real strength of JavaScript arrays are the built-in array properties and methods:

Examples

```
var x = cars.length; // The length property returns the
number of
elements
var y = cars.sort(); // The sort() method sorts arrays
```

Accessing the First Array Element

Example

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

1

$\boxed{f}[0]$

$\downarrow$

Accessing the Last Array Element

Example

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

f[f.l -1]

0 1 $\boxed{2}$

$\boxed{2}$

$l = 3 - 1$
$= \underline{2}$

Adding Array Elements

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

Example

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

$\leftarrow$ .push
("")

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

Example

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

JavaScript Array Methods

Converting Arrays to Strings

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

Example

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

toString()

Result:

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

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

Result:

Banana * Orange * Apple * Mango

Popping and Pushing

When you work with arrays, it is easy to remove elements and add new elements.

This is what popping and pushing is:

Popping items **out** of an array, or pushing items **into** an array.

Popping

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

Example

```
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":

Example

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

Pushing

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

Example

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

The push() method returns the new array length:

Example

```
va fruits = ["Banana",          "Apple",
r  "Orange",                   "Mango"];
   x =
va fruits.push("Kiwi");  the value of x
r  //                                is5
```

JavaScript Strings 7

Method	Description
charAt()	Returns the character at the specified index (position)
charCodeAt()	Returns the Unicode of the character at the specified index
concat()	Joins two or more strings, and returns a new joined strings

('' ; ') hello 11
01234 ①

endsWith()	Checks whether a string ends with specified string/characters
fromCharCode()	Converts Unicode values to characters
includes()	Checks whether a string contains the specified string/characters
indexOf()	Returns the position of the first found occurrence of a specified value in a string
lastIndexOf()	Returns the position of the last found occurrence of a specified value in a string
localeCompare()	Compares two strings in the current locale
match()	Searches a string for a match against a regular expression, and returns the matches
repeat()	Returns a new string with a specified number of copies of an existing string
replace()	Searches a string for a specified value, or a regular expression, and returns a new string where the specified values are replaced
search()	Searches a string for a specified value, or regular expression, and returns the position of the match
slice ()	Extracts a part of a string and returns a new string

==

()	
split()	Splits a string into an array of substrings
startsWith()	Checks whether a string begins with specified characters
substr()	Extracts the characters from a string, beginning at a specified start position, and through the specified number of character
substring()	Extracts the characters from a string, between two specified indices
toLocaleLowerCase()	Converts a string to lowercase letters, according to the host's locale
toLocaleUpperCase()	Converts a string to uppercase letters, according to the host's locale

toLowerCase()	Converts a string to lowercase letters
toString()	Returns the value of a String object
toUpperCase()	Converts a string to uppercase letters
trim()	Removes whitespace from both ends of a string
valueOf()	Returns the primitive value of a String object

D Hello(.....)

String Properties

Property	Description
constructor	Returns the string's constructor function

length	Returns the length of a string
prototype	Allows you to add properties and methods to an object

String Methods

All string methods return a new value. They do not change the original variable.

String Length

The length property returns the length of a string:

Example

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

Finding a String in a String

The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string:

Example

7

```
var str = "Please locate
where 'locate' occurs!";
var pos =
str.indexOf("locate");
```

JavaScript counts positions from zero.
0 is the first position in a string, 1 is the second, 2 is the third ...

The lastIndexOf() method returns the index of the **last** occurrence of a specified text in a string:

Example

```
var str = "Please locate
where 'locate' occurs!";
var pos =
str.lastIndexOf("locate");
```

Both indexOf(), and lastIndexOf() return -1 if the text is not found.
Example

```
var str = "Please locate where
'locate' occurs!"; var pos =
str.lastIndexOf("John");
```

Both methods accept a second parameter as the starting position for the search:

Example

```
var str = "Please locate where
'locate' occurs!"; var pos =
str.indexOf("locate", 15);
```

The lastIndexOf() methods searches backwards, meaning: if the second parameter is 15, the search starts at position 15, counting from the end, and searches to the beginning of the string.

Example

```
var str = "Please locate where 'locate'
occurs!"; var pos =
str.lastIndexOf("locate", 15);
```

Searching for a String in a String

The search() method searches a string for a specified value and returns the position of the match:

Example

```
var str = "Please locate where
'locate' occurs!"; var pos =
str.search("locate");
```

Extracting String Parts

There are 3 methods for extracting a part of a string:

slice(*start*, *end*)

substring(*start*, *end*)

substr(*start*, *length*)

The slice() Method

slice() extracts a part of a string and returns the extracted part in a new string.

The method takes 2 parameters: the start position, and the end position (end not included).

This example slices out a portion of a string from position 7 to position 12 (13-1):

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(7, 13);
```

The result of res will be:

Banana

The substring() Method

substring() is similar to slice().

The difference is that substring() cannot accept negative indexes.

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.substring(7, 13);
```

The result of *res* will be:

Banana

The substr() Method

substr() is similar to slice().

The difference is that the second parameter specifies the **length**

of the extracted part. Example

```
var str = "Apple, Banana, Kiwi";
var res = str.substr(7, 6);
```

The result of res will be:

Banana

Replacing String Content

The replace() method replaces a specified value with another value in a string:

Example

```
str = "Please visit Microsoft!";
var n = str.replace("Microsoft", "W3Schools");
```

Converting to Upper and Lower Case

A string is converted to upper case with toUpperCase():

Example

```
var text1 = "Hello World!";  // String
var text2 = text1.toUpperCase(); // text2 is text1
converted to upper
```

The concat() Method

concat() joins two or more strings:

Example

Hello w.

```
var text1 = "Hello";
var text2 = "World";
var text3 = text1.concat(" ", text2);
```

String.trim()

The trim() method removes whitespace from both sides of a string:

Example

```
var str = "   Hello World!    ";
alert(str.trim());
```

Extracting String Characters

There are 3 methods for extracting string characters:

charAt(*position*)

charCodeAt(*position*)

Property access [ ]

The charAt() Method

The charAt() method returns the character at a specified index (position) in a string:

Example

```
var str = "HELLO WORLD";
str.charAt(0);        // returns H
```

The charCodeAt() Method

The charCodeAt() method returns the unicode of the character at a specified index in a string:

The method returns a UTF-16 code (an integer between 0 and 65535).

Example

```
var str = "HELLO WORLD";
str.charCodeAt(0);  // returns 72
```

Property Access

ECMAScript 5 (2009) allows property access [ ] on strings:

Example

```
var str = "HELLO WORLD";
str[0];                  // returns H
```

Converting a String to an Array

A string can be converted to an array with the split() method:

Example

```
var txt =                 //String
"a,b,c,d,e";              //Split on commas
txt.split(",");           //Split on spaces
txt.split(" ");           //Split on pip
txt.split("|");
```

What is the DOM?

→ DOM ⑧ DOM Pr

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents: → HTML XML

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

The W3C DOM standard is separated into 3 different parts:

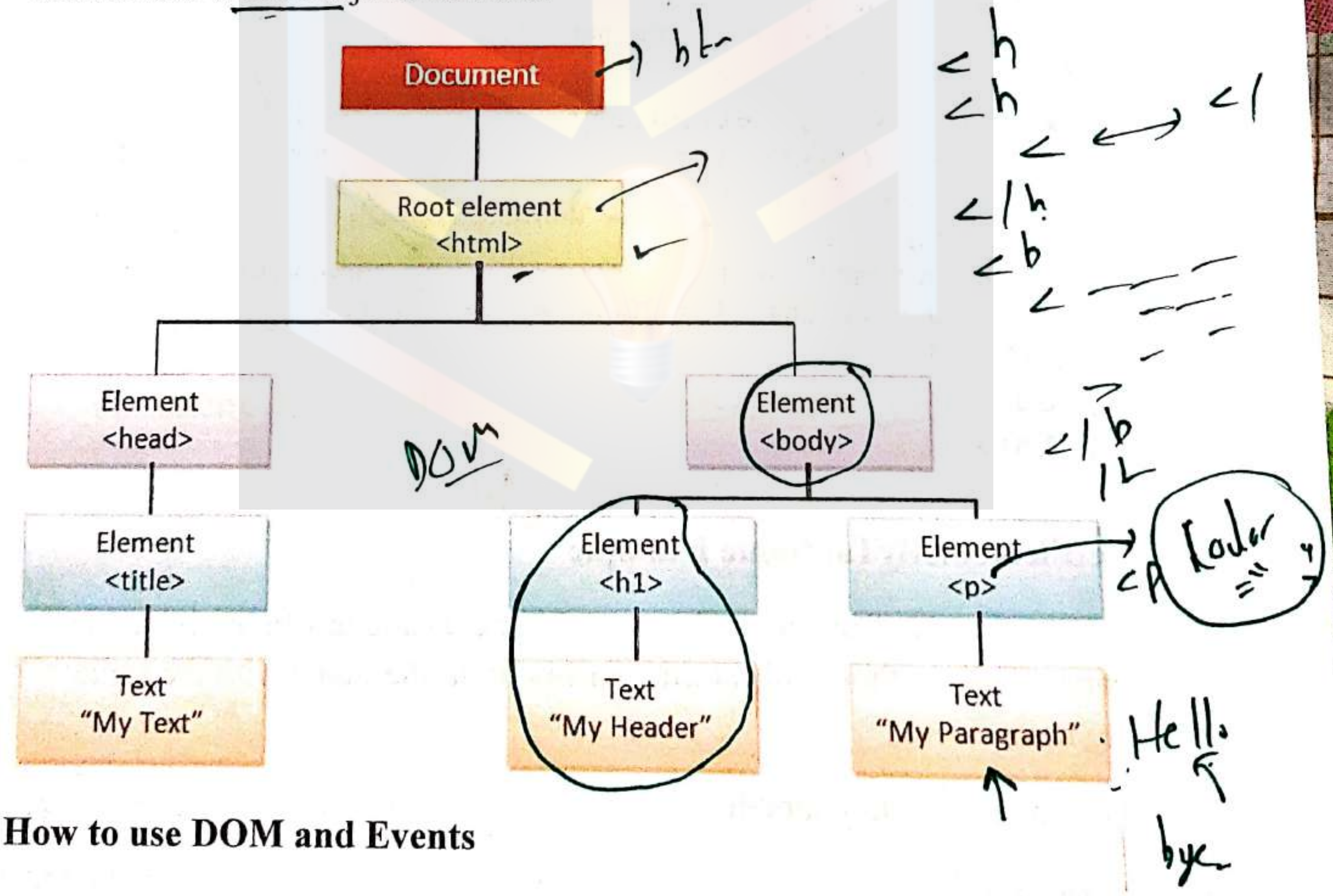
Core DOM - standard model for all document types

XML DOM - standard model for XML documents

HTML DOM - standard model for HTML documents

What is DOM in JavaScript?

JavaScript can access all the elements in a webpage making use of Document Object Model (DOM). In fact, the web browser creates a DOM of the webpage when the page is loaded. The DOM model is created as a tree of objects like this:



How to use DOM and Events

Using DOM, JavaScript can perform multiple tasks. It can create new elements and attributes, change the existing elements and attributes and even remove existing elements and attributes. JavaScript can also react to existing events and create new events in the page.

getElementById, innerHTML Example

getElementById: To access elements and attributes whose id is set. //

innerHTML: To access the content of an element.

Try this Example yourself:

Top of Form

each

getEd Id

Tag

```
<html>
<head>
    <title>DOM!!!</title>
</head>
<body>
 <h1 id="one">Welcome</h1>
 <p>This is the welcome message.</p>
 <h2>Technology</h2>
 <p>This is the technology
 section.</p><script
 type="text/javascript">
        var text =
        document.getElementById("one").innerHTML;
        alert("The first heading is " + text);
 </script>
</body>
</html>
```

(1)

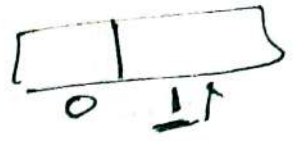
getElementsByTagName Example

getElementsByTagName: To access elements and attributes using tag name. This method will return an array of all the items with the same tag name.

Try this Example yourself:

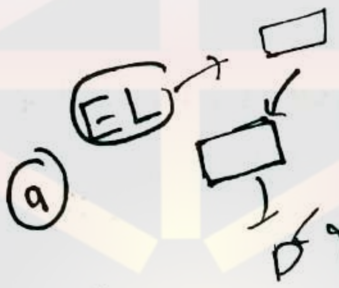
```
<html>
```

```
 <head>
     <title>DOM!!!</title>
 </head>
 <body>
  <h1>Welcome</h1>
  <p>This is the welcome message.</p>
  <h2>Technology</h2>
  <p id="second">This is the technology
  section.</p><script
  type="text/javascript">
     var paragraphs =
    document.getElementsByTagName("p");
    alert("Content in the second paragraph
    is " +
paragraphs[1].innerHTML);
    document.getElementById("second").innerHTML = "The
    orginal message
is changed.";
  </script>
</body>
</html>
```



[↗, ↗, ↗]

[0] . . . [n-1]



onchang →
onclick →
onmousehover →
onmouseout
on Keydown
onload.

Event handler Example

createElement: To create new element
removeChild: Remove an element

You can add an **event handler** to a particular element like this:

```
    document.getElementById(id).onclick=function()
    {
       lines of code to be executed
    }
```

OR

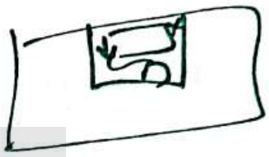
```
document.getElementById(id).addEventListener("click",
functionname)
```

Try this Example yourself:

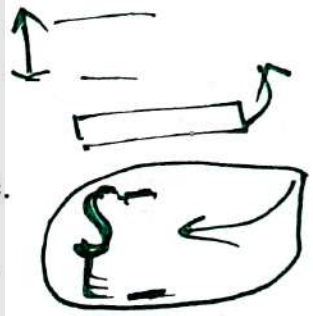
```
<html>
```

```
<head>
    <title>DOM!!!</title>
</head>
<body>
 <input type="button" id="btnClick"
 value="Click Me!!" /><script
 type="text/javascript">
   document.getElementById("btnClick").addEventL
istener("click", clicked);
  function clicked()
  {
        alert("You clicked me!!!");
  }
 </script>
</body>
</html>
```

Click!!



JavaScript Closures (10)



JavaScript variables can belong to the **local** or **global** scope.

Global variables can be made local (private) with **closures**.

Global Variables

A function can access all variables defined **inside** the function, like this:

Example
```
function myFunction() {
 var a = 4;
 return a * a;
}
```
local

But a function can also access variables defined **outside** the function, like this:

Example
```
var a = 4;
function myFunction() {
 return a * a;
```
global

```
}
```

In the last example, **a** is a **global** variable.

In a web page, global variables belong to the window object.

Global variables can be used (and changed) by all scripts in the page (and in the window).

In the first example, **a** is a **local** variable.

A local variable can only be used inside the function where it is defined. It is hidden from other functions and other scripting code.

Global and local variables with the same name are different variables. Modifying one, does not modify the other.

Variables created **without** the keyword **var**, are always global, even if they are created inside a function.

Variable Lifetime

Global variables live as long as your application (your window / your web page) lives.

Local variables have short lives. They are created when the function is invoked, and deleted when the function is finished.

JavaScript Closures

Remember self-invoking functions? What does this function do?

Example

```
var add = (function () {
  var counter = 0;
  return function () {counter += 1; return counter}
})();

add();
add();
add();

// the counter is now 3
```

Example Explained

The variable add is assigned the return value of a self-invoking function.

The self-invoking function only runs once. It sets the counter to zero (0), and returns a function expression.

This way add becomes a function. The "wonderful" part is that it can access the counter in the parent scope.

This is called a JavaScript **closure.** It makes it possible for a function to have "**private**" variables.

The counter is protected by the scope of the anonymous function, and can only be changed using the add function.

A closure is a function having access to the parent scope, even after the parent function has closed.

AJAX: (11) → It is a technique in which we request data from a remote server without reloading the entire page

AJAX stands for **A**synchronous **Ja**vaScript and **X**ML.
(not a programming language)

Just a technique to make websites behave as desktop applications

Through AJAX you can:

- Update a web page without reloading the page
- Request data from a server - after the page has loaded
- Receive data from a server - after the page has loaded
- Send data to a server - in the background
- Create better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

AJAX stands for **A**synchronous **Ja**vaScript and **X**ML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

- Ajax uses ~~X~~HTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display. DOM)
- Conventional web applications transmit information to and from the sever using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server.
- With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.
- XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.
- AJAX is a web browser technology independent of web server software.
- A user can continue to use the application while the client program requests information from the server in the background.
- Intuitive and natural user interaction. Clicking is not required, mouse movement is a sufficient event trigger.
- Data-driven as opposed to page-driven.

  Ex:Google Maps, Google Suggest ,Gmail, Yahoo Maps (new)

Rich Internet Application Technology:

AJAX is the most viable Rich Internet Application (RIA) technology so far. It is getting tremendous industry momentum and several tool kit and frameworks are emerging. But at the same time, AJAX has browser incompatibility and it is supported by JavaScript, which is hard to maintain and debug.

AJAX is Based on Open Standards:

AJAX is based on the following open standards –

Browser-based presentation using HTML and Cascading Style Sheets (CSS).

Data is stored in XML format and fetched from the server.

JAPw

Behind-the-scenes data fetches using XMLHttpRequest objects in the browser.

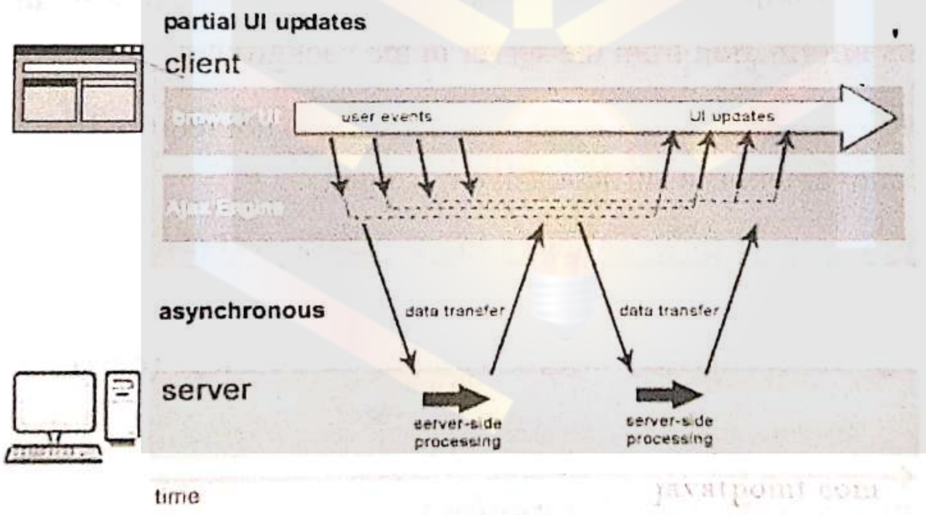
JavaScript to make everything happen.

There are too many web applications running on the web that are using ajax technology like **gmail**, **facebook**, **twitter**, **google map**, **youtube** etc.

Asychronous means that we are exchanging data to/from the server in the back ground without having to refresh the page

Asynchronous (AJAX Web-Application Model):

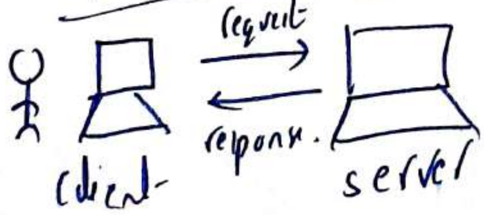
An asynchronous request doesn't block the client i.e. browser is responsive. At that time, user can perform another operations also. In such case, javascript engine of the browser is not blocked.



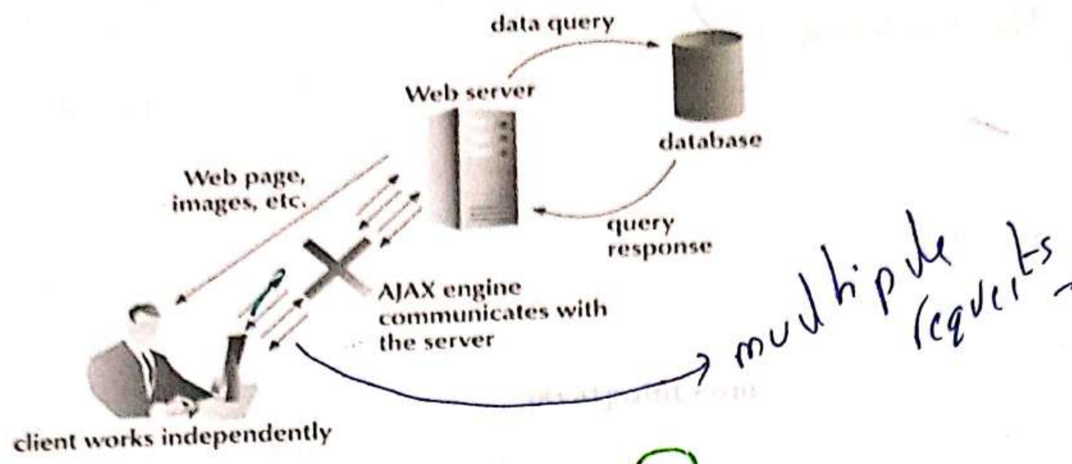
As you can see in the above image, full page is not refreshed at request time and user gets response from the ajax engine.

Let's try to understand asynchronous communication by the image given below.

Normal client server



In standard client server application where client has to wait for the response from the server to ~~exchan~~ execute the task on web page



Ajax Communication Techniques: 12

AJAX Technologies:

As describe earlier, ajax is not a technology but group of inter-related technologies. AJAX technologies includes: → Technique

HTML/XHTML and CSS

DOM

XML or JSON

XMLHttpRequest

JavaScript

HTML/XHTML and CSS:

These technologies are used for displaying content and style. It is mainly used for presentation.

DOM:

It is used for dynamic display and interaction with data.

XML or JSON:

For carrying data to and from server. JSON (Javascript Object Notation) is like XML but short and faster than XML.

XMLHttpRequest:
For asynchronous communication between client and server. For more visit next page.

JavaScript:
It is used to bring above technologies together.

Independently, it is used mainly for client-side validation.

The XMLHttpRequest Object: 13
The XMLHttpRequest object can be used to exchange data with a web server behind the scenes.
This means that it is possible to update parts of a web page, without reloading the whole page.

This helps to update parts of the web page without reloading whole page

Create an XMLHttpRequest Object:
All modern browsers (Chrome, Firefox, IE7+, Edge, Safari, Opera) have a built-in XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:

Step 1

variable = new XMLHttpRequest();

Example

var xhttp = new XMLHttpRequest();

The XMLHttpRequest object is the key to AJAX. It has been available ever since Internet Explorer 5.5 was released in July 2000, but was not fully discovered until AJAX and Web 2.0 in 2005 became popular.

XMLHttpRequest (XHR) is an API that can be used by JavaScript, JScript, VBScript, and other web browser scripting languages to transfer and manipulate XML data to and from a webserver using HTTP, establishing an independent connection channel between a webpage's Client-Side and Server-Side.

The data returned from XMLHttpRequest calls will often be provided by back-end databases. Besides XML, XMLHttpRequest can be used to fetch data in other formats, e.g. JSON or even plain text.

You already have seen a couple of examples on how to create an XMLHttpRequest object.

Listed below are some of the methods and properties that you have to get familiar with.

These are methods used to make request to the server (Note XMLHttp request is used for exchange of data with server)

XMLHttpRequest Methods:

abort()

Cancels the current request.

getAllResponseHeaders()

Returns the complete set of HTTP headers as a string.

getResponseHeader(headerName)

Returns the value of the specified HTTP header.

open(method, URL)

open(method, URL, async)

open(method, URL, async, userName)

open(method, URL, async, userName, password)

Specifies the method, URL, and other optional attributes of a request.

The method parameter can have a value of "GET", "POST", or "HEAD". Other HTTP methods such as "PUT" and "DELETE" (primarily used in REST applications) may be possible.

The "async" parameter specifies whether the request should be handled asynchronously or not. "true" means that the script processing carries on after the send() method without waiting for a response, and "false" means that the script waits for a response before continuing script processing.

step2 :-

ex xhttp.open("GET", "URL", true);

xyz.php

method (get/post) — url (of the file that we want on website) — informing using AJAX

→ It holds the status of XMLHttp request.

→ It also defines the function to be executed when ready state changes based on the status

send(content)

Sends the request.

setRequestHeader(label, value)

Adds a label/value pair to the HTTP header to be sent.

XMLHttpRequest Properties:

step3 onreadystatechange

An event handler for an event that fires at every state change.

readyState

The readyState property defines the current state of the XMLHttpRequest object.

The following table provides a list of the possible values for the readyState property –

State	Description
0	The request is not initialized.
1	The request has been set up.
2	The request has been sent.
3	The request is in process.
4	The request is completed.

readyState = 0 After you have created the XMLHttpRequest object, but before you have called the open() method.

readyState = 1 After you have called the open() method, but before you have called send().

readyState = 2 After you have called send().

step 3
ex:

```
xhttp.onreadystatechange = Function(){
    if(this.readyState ==4 && this.status
          == 200){
       document.getElementById("demo").inner
    HTML = this.reponse Text;
    }

};
```

200 ok

Step4: send the request

```
xhttp.send();
```

ex: 4 steps

```
var xhttp = new XMLHttpRequest();              → ①
                  → obj
(  xhttp.onreadystatechange = Function(){
      if(this.readyState == 4 && this.status
              == 200) {
         document.getElementById("Demo").
③        innerHTML = this.reponse Text;

      }
   };
③ ←—— xhttp.open("GET","ajax_info.txt", true);
④ ←—— xhttp.send();
```

readyState = 3 After the browser has established a communication with the server, but before the server has completed the response.

readyState = 4 After the request has been completed, and the response data has been completely received from the server.

responseText

Returns the response as a string.

responseXML

Returns the response as XML. This property returns an XML document object, which can be examined and parsed using the W3C DOM node tree methods and properties.

status

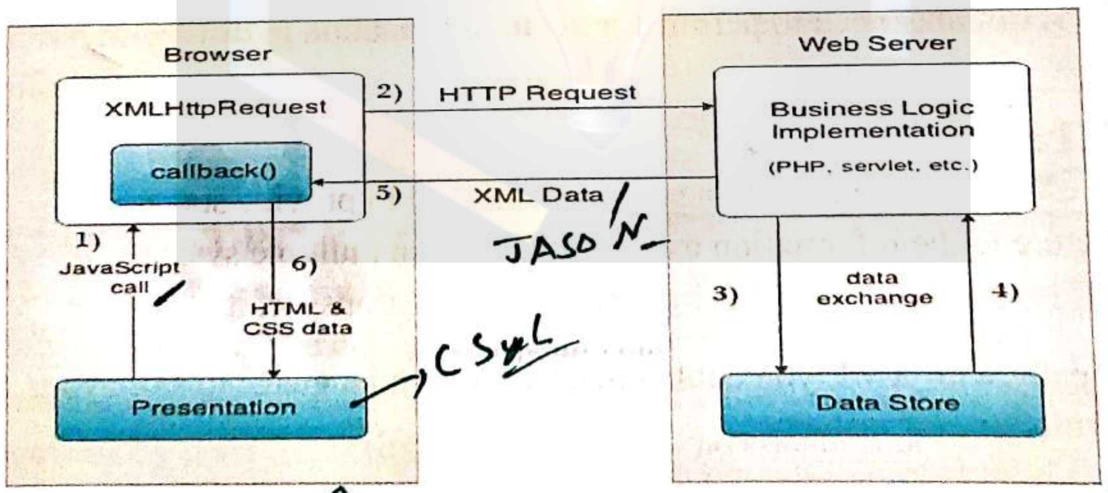
Returns the status as a number (e.g., 404 for "Not Found" and 200 for "OK").

statusText

Returns the status as a string (e.g., "Not Found" or "OK").

How AJAX works?:

AJAX communicates with the server using XMLHttpRequest object. Let's try to understand the flow of ajax or how ajax works by the image displayed below.



As you can see in the above example, XMLHttpRequest object plays a important role.

User sends a request from the UI and a javascript call goes to XMLHttpRequest object.

HTTP Request is sent to the server by XMLHttpRequest object.

Server interacts with the database using JSP, PHP, Servlet, ASP.net etc.

Data is retrieved.

Server sends XML data or JSON data to the XMLHttpRequest callback function.

HTML and CSS data is displayed on the browser.

AJAX data formats: 14

Upon sending an XMLHttpRequest request to the server, we will have a response. This response could be in two different formats: plain text or in XML.

-Plain text: we could receive just a text, a word, a sentence (JSON is actually a sentence)

-XML: the response we received will be XML encoded, so our JavaScript may need to perform some transformation to display it.

XML:

The eXtensible Markup Language was created to provide some structure to the information exchanged between multiple systems. It provides a way to store this information and to transport it.

Basically with XML you could create your own language to store your information, for instance:

?

```
<record>
  <from>John Doe</from>
  <to>Everyone</to>
  <title>First XML Record</title>
```

```
5    <description>This is the first of many XML records</description>
6  </record>
```

To access an XML response, the XMLHttpRequest object has a property called: responseXML.

Once you have the response in a JS variable, you can access the different nodes of the XML with standard DOM functions: i.e: getElementsByTagName.

String:
As mentioned before, an XMLHttpRequest could receive plain text as a response: a word, a letter, a complete sence, and text with special encoding. To access this response you have to use a property called: responseText.

JSON:
JSON stands for JavaScript Object Notation which is a data interchange format. It is now widely adopted as a way to handle responses from server, since it is a simple encoding that allows human and machines to read it easily.

In our tutorial, as I briefly mentioned before, a JSON response is actually a text response, so you should use the responseText property to access it. Once you have it available, you could use multiple JSON libraries to parse it, and perform any operation needed based on the information received.

AJAX SECURITY CONCERNS: (15)

The Ajax calls are sent in plain text format, this might lead to insecure database access. The data gets stored on the clients browser, thus making the data available to anyone. It also allows monitoring browsing sessions by inserting scripts.

AJAX function calls are sent in plain text to server. These calls may easily reveal database details, variable names etc

For same example jason file will be
{"record":[{"John Doe","Everyone","First ~~XML~~ JSON record","This is the first of many JSON records"}]} //.

User's browsing session can be monitored my maliciously inserting scripts

Ajax may encourage developers to use multiple server side pages thereby introducing multiple entry points for attackers

- A JavaScript can not access the local file system without the user's permission.
- An AJAX interaction can only be made with the servers-side component from which the page was loaded.
- A proxy pattern could be used for AJAX interactions with external services.
- The application model should not be exposed as some user might be able to reverse engineer the application.
- HTTPS can be used to secure the connection when confidential information is being exchanged

-

- AJAX Security: Server Side:

AJAX-based Web applications use the same server-side security schemes of regular Web applications.

You specify authentication, authorization, and data protection requirements in your web.xml file (declarative) or in your program (programmatic).

AJAX-based Web applications are subject to the same security threats as regular Web applications.

AJAX Security: Client Side:

JavaScript code is visible to a user/hacker. Hacker can use JavaScript code for inferring server-side weaknesses.

JavaScript code is downloaded from the server and executed ("eval") at the client and can compromise the client by mal-intended code.

Downloaded JavaScript code is constrained by the sand-box security model and can be relaxed for signed JavaScript.

USER INTERFACE DESIGN FOR AJAX: (16)

The user interface for ajax is designed using html,xml,javascript,css

File directory:

ajax_example.html

surprise.html

css (folder) containing styles.css file

you'll have a main HTML file, CSS stylesheet, JavaScript file, and an **additional** HTML file.

We're going to grab the contents of the **second** HTML file and insert it into the first page.

Here's the main HTML page.

```
<!DOCTYPE html>

<html>     <head>

    <meta charset="utf-8">

    <link rel="stylesheet" href="css/styles.css">

    <title>AJAX practice</title>

    <script>

    </script>

 </head>
```

issues

→ Complexity is increased

→ Ajax based application can be difficult to debug, test & maintain

→ Javascript code is visible to hacker

→ Not supported in old browser.

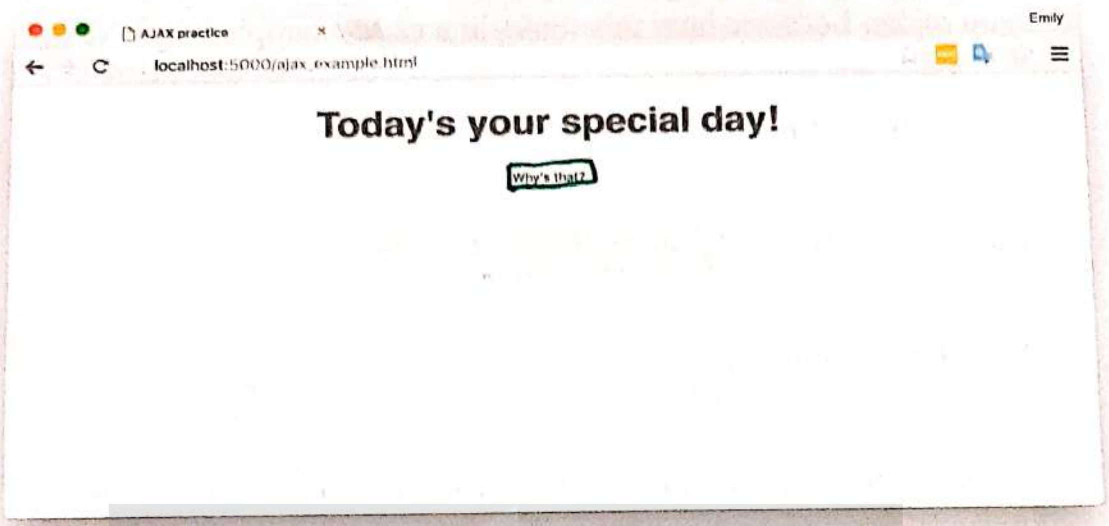
→ tool kits / Frameworks are not matured. yet

```
<body>

  <h1>Today's your special day!</h1>

  <button id="reveal">Why's that?</button>

  <div id="ajax-content">

  </div>

</body>

</html>
```

Here's the content you'll put in surprise.html (if on your computer)

```
<h1 id="birthday-greeting">It's your birthday!</h1>
```

surprise.html is the file we'll load from within ajax_example.html - via AJAX!



Today's your special day!

Main HTML page

Introduction to XMLHttpRequest:

XMLHttpRequest is a mouthful. It's a system that lets data be transferred between a client and a server. As you learned, this normally happens via request and response. The same is true with XMLHttpRequest, except you can grab data from a URL without the page refreshing!

Think to a time you've used Facebook or Gmail. You've performed actions without reloading an entire page. You've left comments that post instantly while you're on the same page, for example. That's what AJAX allows!

Your first AJAX call:

1.First, you'll create an XMLHttpRequest object.
2.Open your request with the open method.
3.Send the request with the send method.

You need to create an XMLHttpRequest object for every AJAX request you make. Let's see how this looks in a code example.

//1. create a new XMLHttpRequest object -- an

object like any other! var myRequest = new

XMLHttpRequest();

//2. open the request and pass the HTTP method name and the

resource as parameters myRequest.open('GET', 'surprise.html');

//3. write a function that runs anytime the state of the

AJAX request changes myRequest.onreadystatechange =

function () {

 //4. check if the request has a readyState of 4, which indicates the server has responded (complete) && [illegible] = 200

 if (myRequest.readyState === 4) {

  //5. insert the text sent by the server into the HTML of the 'ajax-

  content' document.getElementById('ajax-content').innerHTML =

  myRequest.responseText;

```
    }

}
```

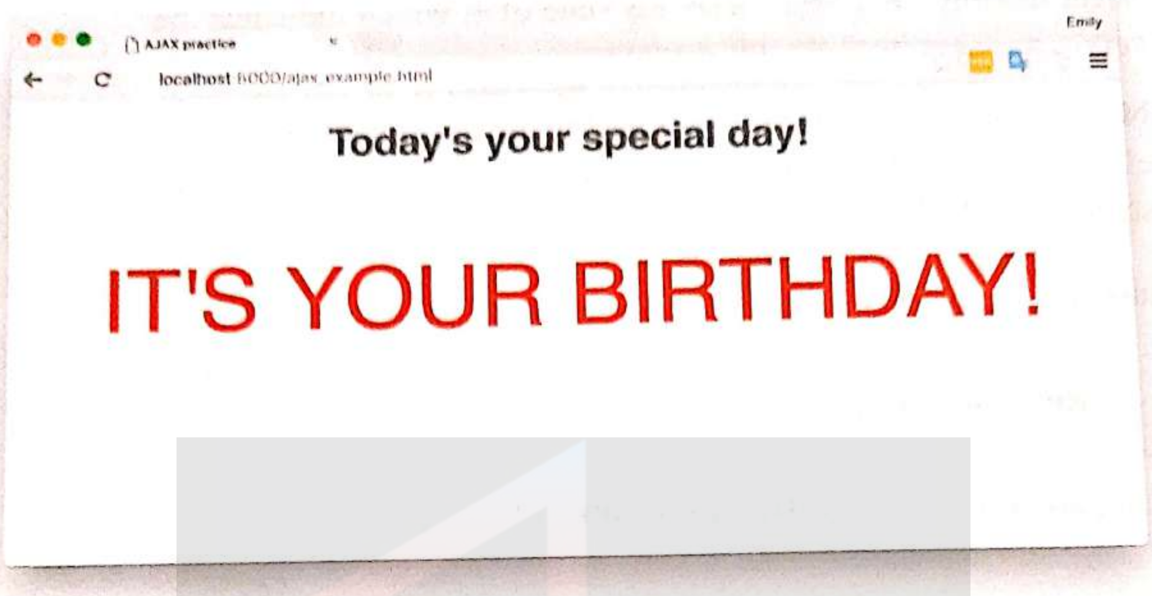
readyState can have a value between 0 and 4. You'll probably never use anything besides areadyState of 4, which indicates the server has sent back its full response.

Now, we need a function to call within the page when the button is clicked. Since a button click will send the AJAX, why not name it sendTheAJAX?

```
function sendTheAJAX() {

  myRequest.send();

  document.getElementById('reveal').style.display = 'none';

}
```

This function also hides the original button, leaving only the newly revealed text via setting display to'none'. Now, add this new function back to your HTML in this line

```
<button id="reveal" onclick="sendTheAJAX()" class="button">Why's
that?</button>
```



Today's your special day!

IT'S YOUR BIRTHDAY!

17

Python Introduction:

Python is a general purpose, dynamic, high level and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures.

Python is *easy to learn* yet powerful and versatile scripting language which makes it attractive for Application Development.

Python's syntax and *dynamic typing* with its interpreted nature, makes it an ideal language for scripting and rapid application development.

Python supports *multiple programming pattern,* including object oriented, imperative and functional or procedural programming styles.

Python is not intended to work on special area such as web programming. That is why it is known as *multipurpose* because it can be used with web, enterprise, 3D CAD etc.

We don't need to use data types to declare variable because it is *dynamically typed* so we can write a=10 to assign an integer value in an integer variable.

Python makes the development and debugging *fast* because there is no compilation step included in python development and edit-test-debug cycle is very fast.

Python Features:

Python provides lots of features that are listed below.

1) Easy to Learn and Use

Python is easy to learn and use. It is developer-friendly and high level programming language.

2) Expressive Language

Python language is more expressive means that it is more understandable and readable.

3) Interpreted Language

Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

4) Cross-platform Language

IDLE

Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc.
So, we can say that Python is a portable language.

5) Free and Open Source

Python language is freely available at <u>offical web address</u>.The source-code is also available.
Therefore it is open source.

6) Object-Oriented Language

Python supports object oriented language and concepts of classes and objects come into existence.

7) Extensible

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our python code.

8) Large Standard Library

Python has a large and broad library and prvides rich set of module and functions for rapid application development.

9) GUI Programming Support

Graphical user interfaces can be developed using Python.

10) Integrated

It can be easily integrated with languages like C, C++, JAVA etc.

Python Applications:

Python is known for its general purpose nature that makes it applicable in almost each domain of software development. Python as a whole can be used in any sphere of development.

Here, we are specifing applications areas where python can be applied.

1) Web Applications:

We can use Python to develop web applications. It provides libraries to handle internet protocols such as HTML and XML, JSON, Email processing, request, beautifulSoup, Feedparser etc. It also provides Frameworks such as Django, Pyramid, Flask etc to design and delelop web based applications. Some important developments are: PythonWikiEngines, Pocoo, PythonBlogSoftware etc.

2) Desktop GUI Applications: ✓

Python provides Tk GUI library to develop user interface in python based application. Some other useful toolkits wxWidgets, Kivy, pyqt that are useable on several platforms. The Kivy is popular for writing multitouch applications.

3) Software Development: ✓

Python is helpful for software development process. It works as a support language and can be used for build control and management, testing etc.

4) Scientific and Numeric: ✓

Python is popular and widely used in scientific and numeric computing. Some useful library and package are SciPy, Pandas, IPython etc. SciPy is group of packages of engineering, science and mathematics.

5) Business Applications: ✓

Python is used to build Bussiness applications like ERP and e-commerce systems. Tryton is a high level application platform.

6) Console Based Application: ✓

We can use Python to develop console based applications. For example: IPython.

7) Audio or Video based Applications: ✓ FJ B

Python is awesome to perform multiple tasks and can be used to develop multimedia applications. Some of real applications are: TimPlayer, cplay etc.

8) 3D CAD Applications:

To create CAD application Fandango is a real application which provides full features of CAD.

9) Enterprise Applications:

Python can be used to create applications which can be used within an Enterprise or an Organization. Some real time applications are: OpenErp, Tryton, Picalo etc.

10) Applications for Images:

Using Python several application can be developed for image. Applications developed are:
VPython, Gogh, imgSeek etc.

There are several such applications which can be developed using Python

Python Data Types: (18) Var

Variables can hold values of different data types. Python is a dynamically typed language hence we need not define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

Python enables us to check the type of the variable used in the program. Python provides us the type() function which returns the type of the variable passed.

Consider the following example to define the values of different data types and checking its type.

A=10

a=10 int . a=10.0 Fl a="" str a=[ ]

```
b="Hi Python"
c = 10.5
print(type(a));
print(type(b));
print(type(c));
```

(10)

Output:

```
<type 'int'>
<type 'str'>
<type 'float'>
```

<type 'int'>

a =

Standard data types:

A variable can hold different types of values. For example, a person?s name must be stored as a string whereas its id must be stored as an integer.

Python provides various standard data types that define the storage method on each of them. The data types defined in Python are given below.

- ☐ Numbers
- ☐ String
- ☐ List
- ☐ Tuple
- ☐ Dictionary

In this section of the tutorial, we will give a brief introduction of the above data types. We will discuss each one of them in detail later in this tutorial.

Numbers: (int)

Number stores numeric values. Python creates Number objects when a number is assigned to a variable. For example;

1. a = 3 , b = 5 #a and b are number objects Python supports 4 types of numeric data.

// <!-- -->

1. int (signed integers like 10, 2, 29, etc.)
2. long (long integers used for a higher range of values like 908090800L, -0x1929292L, etc.)
3. float (float is used to store floating point numbers like 1.9, 9.902, 15.2, etc.)
4. complex (complex numbers like 2.14j, 2.0 + 2.3j, etc.)

Python allows us to use a lower-case L to be used with long integers. However, we must always use an upper-case L to avoid confusion.

A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts respectively).

String:

The string can be defined as the sequence of characters represented in the quotation marks. In python, we can use single, double, or triple quotes to define a string.

String handling in python is a straightforward task since there are various inbuilt functions and operators provided.

In the case of string handling, the operator + is used to concatenate two strings as the operation *"hello"+" python"* returns *"hello python"*.

The operator * is known as repetition operator as the operation "Python " *2 returns "Python Python ".

```
1. str1 = 'hello javatpoint' #string str1
2. str2 = ' how are you' #string str2
3. print (str1[0:2]) #printing first two character using slice operator
4. print (str1[4]) #printing 4th character of the string
5. print (str1*2) #printing the string twice
6. print (str1 + str2) #printing the concatenation of str1 and str2
```

str1[0:2]

0 : 2

0 1

Output:

```
he
hello javatpointhello javatpoint
hello javatpoint how are you
```

List

Lists are similar to arrays in C. However; the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].

We can use slice [:] operators to access the data of the list. The concatenation operator (+) and repetition operator (*) works with the list in the same way as they were working with the strings.

Consider the following example.

```python
l = [1, "hi", "python", 2]
print (l[3:]);
print (l[0:2]);
print (l);
print (l + l);
print (l * 3);
```

Output:

```
[2]
[1, 'hi']
[1, 'hi', 'python', 2]
[1, 'hi', 'python', 2, 1, 'hi', 'python', 2]
[1, 'hi', 'python', 2, 1, 'hi', 'python', 2, 1, 'hi', 'python', 2]
```

Tuple:

A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types. The items of the tuple are separated with a comma (,) and enclosed in parentheses ().

A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.

Let's see a simple example of the tuple.

```
t = ("hi", "python", 2)
print (t[1:]);
print (t[0:1]);
print (t);
print (t + t);
print (t * 3);
print (type(t))
t[2] = "hi";
```

Output:

```
('python',
2) ('hi',)
('hi', 'python', 2)
('hi', 'python', 2, 'hi', 'python', 2)
('hi', 'python', 2, 'hi', 'python', 2, 'hi',
'python', 2) <type 'tuple'>
Traceback (most recent call
 last): File "main.py", line 8,
 in <module>
   t[2] = "hi";
TypeError: 'tuple' object does not support item assignment
```

Dictionary:

KVP

K : V

Dictionary is an ordered set of a key-value pair of items. It is like an associative array or a hash table where each key stores a specific value. Key can hold any primitive data type whereas value is an arbitrary Python object.

The items in the dictionary are separated with the comma and enclosed in the curly braces {}.

Consider the following example.d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'};

d[1

```
print("1st name is "+d[1]);
print("2nd name is "+ d[4]);
print (d);
print (d.keys());
print (d.values());
```

Output:

```
1st name is
Jimmy 2nd
name is mike
{1: 'Jimmy', 2: 'Alex', 3: 'john', 4:
'mike'} [1, 2, 3, 4]
```