e.g let x = 1 && 2 ?? 3; // Syntax error.

let x = (1 && 2) ?? 3;

alert(x); // 2

(loops like while and for and switch would be prepared by students themselves)

→ functions

- functions allow code to be called many times without repetition.
- we have seen alert, prompt, confirm but we can create function of our own

- Declaration

name of the function

function showMessage() {
  alert("Hello");
}

keyword

or

function name(parameter1, parameter2 ... parameterN) {
  // body
}

- Local Variable

09

variable declared inside a function is
10 only visible inside that function

11 e.g
```
function showMessage() {
    let message = "Hello";
    alert(message);
}

showMessage(); // Hello

alert(message); // error
```

- Outer Variable

04

- function can also access outer variable

05

e.g
```
let userName = 'John';

function showMessage() {
    let message = 'Hello' + userName;
    alert(message);
}

showMessage(); // Hello John
```

06

07

28 Sun

- function has full access to the outer variable it can modify it as well.

e.g let userName = 'John';

```
function showMessage() {
    userName = "Bob"; // Changing Outer variable
    let message = 'Hello' + userName;
    alert(message);
}
alert(userName); // John before the function call.
showMessage();
alert(userName); // Bob, the modified variable
```

- Outer variable is used when there is no local variable.
- The same name variable ~~could be~~ declared inside the function then it shadows the outer one.

Global variable

Variables declared outside of any function such as the outer userName in the code can are called global

Global variables are visible from any function unless shadowed by locals.

We should minimize the use of global variable, modern code has few or no global variable.

- Parameters

09 These are arbitrary data passed to the
10 functions

11 e.g.

```
function showMessage(from, text); {
alert (from + ': ' + text);
}
```

01

```
showMessage('Adam', 'Hi');
showMessage('Adam', "How are you?");
```

02

03

```
// Adam: Hi
// Adam: How are you?
```

04

In the example below from is changed by the function but the change is not seen outside.

06

```
function showMessage(from, text) {
```

07

```
from = '*' + from + '*'; // *from*

alert (from + ': ' + text);
}
let from = 'Adam';
showMessage (from, "Hello"); // *Adam
alert (from); // Adam
// the value of "from" is the same, the function
```

modified a local copy

- When a value is passed a function parameter it's called an argument
  - A parameter is the variable listed inside the parantheses in the function declaration (it's declaration time term)
  - An argument is the value that is passed to the function when it is called (it's call time term)

we declare functions listing their parameters, then call them passing argument

previously the function showMessage is declared with two parameters, then called with two aregements : from and "hello" ;

- Default values

```
function showMessage (from, text = "no text given") {
    alert (from + ": " + text)
}
showMessage ("Adam");
// Adam : no text given
```

if a function ~~text~~ is called, but an argument is not provided then the corresponding value becomes undefined.

*: Hello

in JS a default parameter is evaluated every time the function is called without respective parameter.

function showMessage(from, text = anotherFunction()){
// anotherFunction() only executed if no text given
// its result becomes the value of text.
}

In this example anotherFunction() is only called at all if the text parameter is provided

It is independently called every time when text is missing.

Alternative default parameter:

we can check parameter passing during the function execution, by comparing it with undefined

```
function showMessage(text) {

  if (text === undefined) {
    text = 'empty message';
  }
  alert(text);
}
```

showMessage(); // empty message

or

function showMessage(text) {
  text = text || 'empty';
}

Returning a value

a function can return a value back into the calling code.

```
function sum(a, b){
  return a+b;
}

let result = sum(1, 2);
alert(result); // 3
```

Function Expression

```
function sayHello(){
  alert("Hello");
}  // function declaration
```

or

```
let sayHello = function() {
    alert("Hello");
}; // function Expression
```

Callback functions

passing functions as values and using function expressions.

```
function ask(question, yes, no) {
    if (confirm(question)) yes()
    else no();
}

function showOK() {
    alert("You agreed");
}

function showCancel() {
    alert("You cancelled exec");
}

// showOk, showCancel are passed as arguments to ask
ask("Do you agree?", showOk, showCancel);
```

→ Objects in JS

Objects are used to store keyed collections of various data and more complex entities

— or we can say an object is a collection of real entities

or we can imagine it as a cabinet to store data

— literal

① let user = { }; // object literal syntax

② let input = 'post';
let user = {

post : 'AP',
time : 3,
teach : 'javascript', 'give name'
}

Console.log(user); // { post: 'AP', time: 3 ...
Console.log(user.post); // AP
Console.log(user['post']); // AP
Console.log(user['give name']); //
Console.log(user[input]); //
alert(user.time);
delete user.time;

- Complex object

```
let user = {
    name: 'Yuvraj',
    post: 'AP',
    time: 3
    expertise: {
        coding: 'OOPs',
        scripting: 'JS'
    }
}

console.log(user.expertise.scripting); // JS
console.log(user.expertise.scripting.length); // 2
```

- for...in loop

Syntax:
```
for (key in object) {
}
```

Example:
```
let user = {
    name: 'Yuvraj',
    age: '29',
    isAdmin: true
};

for (let key in user) {
```

```
alert (key); // name, age, isAdmin
alert (user[key]); // Yuvraj, 29, true
}
```

Adding functions to object (method)

```
let expertise = {

    coding: 'OOPs',
    scripting: 'JS',

    getValue: function() {
        alert("sayHello");
    }
}

expertise.getValue(); // sayHello
```

this keyword

this represents the current object and if we want to access any property of an object we will use this keyword.

```
let user = {
    name: "Adam",
    age: 30,
    sayHello() {
        alert(this.name);
    }
}; user.sayHello(); // Adam
```

- Constructor, operator "new"

The regular syntax allows us to create one object. but often we need to create many similar objects, like multiple users or menu items

this can be done using constructor functions and the "new" operator.

Constructor fn

- Constructor functn are regular functions.
  - ① they are named with capital letter first
  - ② They should be executed only with "new" operator

Example

```
function User(name) {
    this.name = name;
    this.isAdmin = false;
}

let user = new User("James");

alert(user.name); // James
alert(user.isAdmin); // false
```

When a function is executed with new, it does the following steps:

1. A new empty object is created and assigned to this.
2. The funcn. body executes, Usually it modifies this, adds new properties to it.
3. The value of this is returned.

```
function User(name) {
    // this = {};  (implicitly)
    // add properties to this
    this.name = name;
    this.isAdmin = false;
    // return this;  (implicitly)
}
```

So if we have

```
let user = new User("Jack")
```

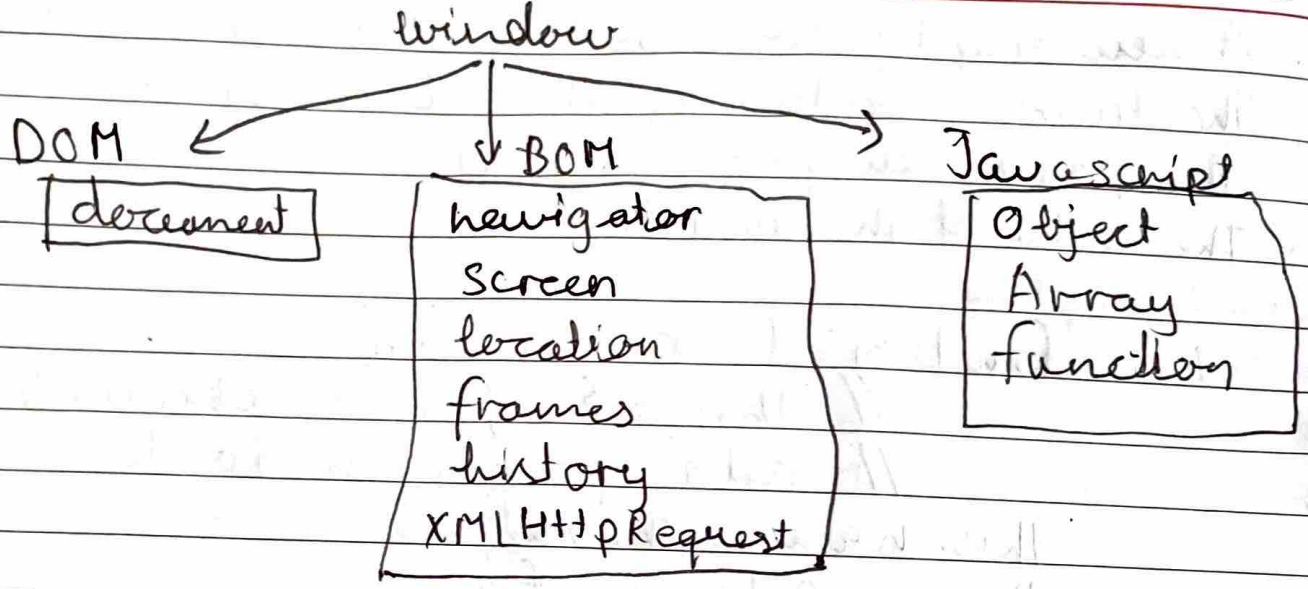
Or

```
let user = {
    name: "Jack",
    isAdmin: false
};
```

— Browser Environment

JS is a language with many uses and platforms. Browser and webserver are the platforms where JS could run. Each of these provides platform specific functionality, this called host environment by JS specification

A host environment provides its own objects and functions in addition to the language, here web browser provides a means to control webpages.



There is a root object called window. It has two roles

1.) first, it is a global object for JS code
2.) It ~~provides~~ represents the "browser window" and provides methods to control it.

fo e.g we can use window as a global object:

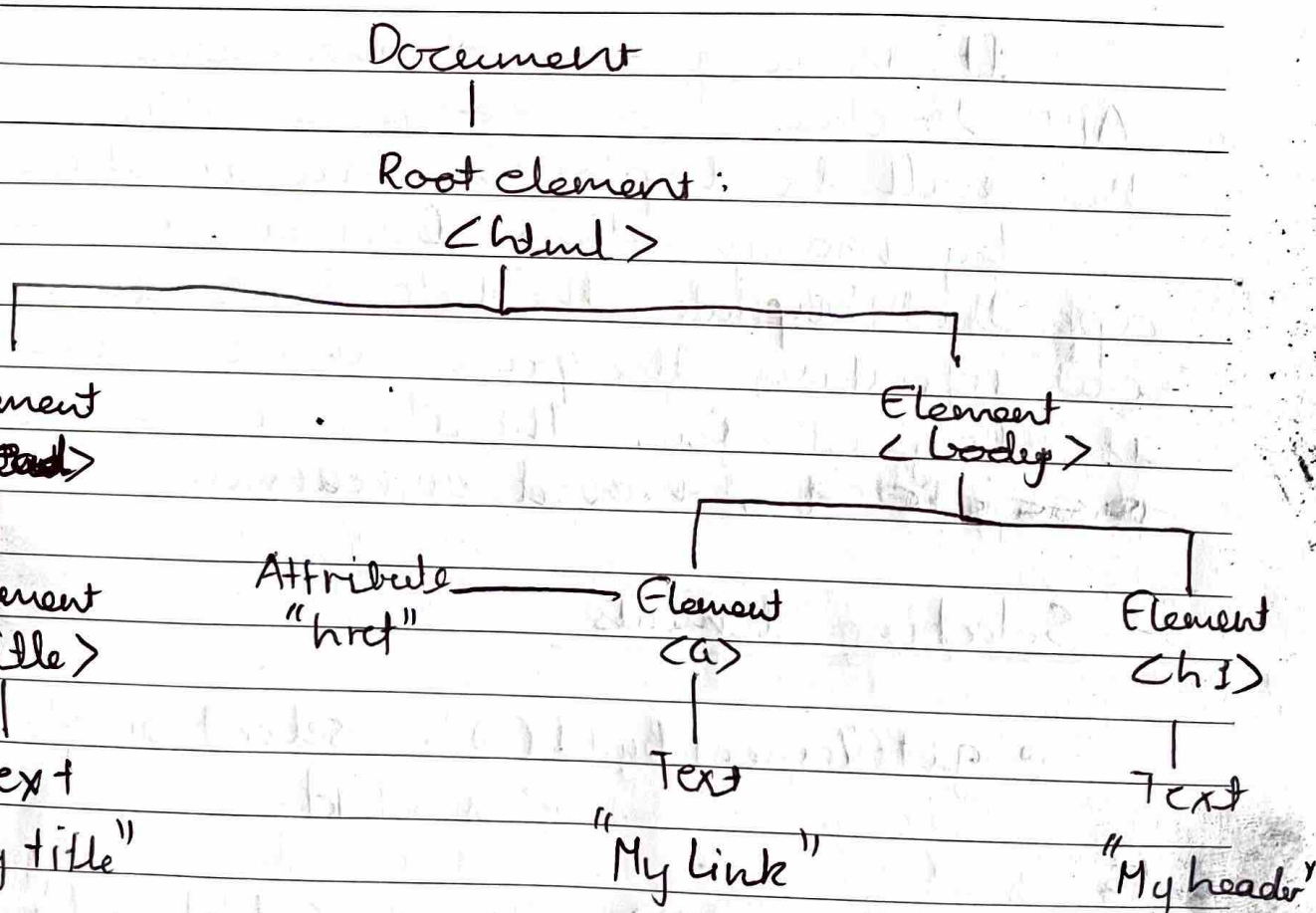
```
function sayHello() {
    alert("Hello");
}
window.sayHello();
```

// global functions are methods of the global object.

12 Sun

DOM in JS

DOM stands for Document Object Model. It is a tree like structure illustrating the hierarchical relationship between various HTML elements.
It can be easily explained as a tree of nodes generated by the browser. Each node has unique properties and methods that can be changed using JS



① Document – is the foundation of DOM
② HTML root element is the child of documentobject
③ Body and Head elements are children of the HTML elements and siblings to each other.
④ Title element is the parent to the text node.

"my text" and the child of the head element.

⑤ a and h1 are child of at body and are siblings

⑥ href attr. is child of <a> tag

DOM is referred to as a programming API for XML and HTML Documents.

- Dom manipulation

It is a process of interacting with DOM API to change or modify an HTML document that will be displayed in a webbrowser.

by manipulating DOM we can create web app that update the data in a webpage without refreshing the page and can change its layout also. The items can be deleted, ~~manipulated~~, moved or rearranged.

- Selecting elements

- getElementById() - select an element by id
- getElementByName() - Selects element by name
- getElementByTagName() - selects element by Tagname.

- getElementByClassName() - selects element by one or more class name

- querySelector() - selects elements by CSS Selectors

- Traversing in DOM

- Get the Parent Element : to get the parent node of a particular node in DOM tree.

e.g let parent = node.parentNode;

ParentNode is a readonly objed. there is no parent node for Document and Document fragments as result ParentNode is Empty.

- Get Child Elements :

① firstChild : this property returns the first child element of a specific element

e.g. let firstChild = parentElement.firstChild;

② lastChild : returns the first element node, text node, or comment node.

e.g. let lastChild = parentElement.lastChild;

③ child Nodes: to get all child nodes this property returns a live Node list of child element of a specific element.

e.g. let children = parentElement.childNodes;

- Get Siblings of an Element:

① nextElementSibling:

let nextSibling = currentNode.nextElementSibling;

② previousElementSibling:

let current = document.querySelector('.current');
let prevSibling = currentNode.previousElementSibling;

- Manipulate Elements in the DOM

① createElement() - create a new element
② appendChild() - append a node to a list of child nodes of a specified parent node.
③ textContent - get and set the text content of an element.
④ innerHTML - get and set the HTML content of an element
⑤ innerHTML vs. createElement — the difference between two while creating new elements.

⑥ DocumentFragment - learn how to compose DOM nodes and insert them the active DOM tree

⑦ after() - insert a node after an element.

⑧ append() - insert a node after the last child node of a parent node.

⑨ prepend() - insert a node before the first child node of a parent node.

⑩ insertAdjacentHTML() - parse a text as HTML and insert the resulting nodes to the document at a specified position.

⑪ replaceChild() - replace a child element by a new element

⑫ cloneNode() - clone an element and all of its descendants.

⑬ removeChild() - remove child elements of a node

⑭ insertBefore() - insert a new node before an existing node as a child node of a specified parent node.

⑮ insertAfter() helper function - insert a new node after an existing node as a child node of a specified parent node.

- Working with Events

① Handling Events: includes HTML event handler attribute, element's event handler property, and addEventListener()

② Page load Events: includes DOM Content Loaded, load, beforeunload, and unload

③ load event: includes dependent resources like JS files, CSS files, and images.

④ Unload event: The unload event is fired after befor unload event and pagehide event.

⑤ Mouse events: like click, scroll, up down

⑥ Keyboard events:

⑦ Scroll events: scrollX, scrollY properties that returns the number of pixels that the document is currently scrolled horizontally and vertically.

Manipulating Element's Styles

① style property: element.style.color = 'red';

② getComputedStyle(): this method returns the object that contains the computed style of an element.

19 Sun

③ className Property: element.className

④ classList Property: const classes = element.classList;

Event Handling in the DOM

Events

1. Mous events - click
   contextmenu
   mouseover / mouseout
   mousedown / mouseup
   mousemove
2. Keyboard events - Keydown
   Key up
3. Form element events :- submit
   focus
4. Document events - DOM content loaded
5. CSS events - transitioned

Event handlers

① HTML Attb.

```
<input value="Clickme" onclick="alert('Click')" type="button">
```

② DOM Property

on<event> (handler)

```
<input id="elem" type="button" value="Clickme">
<Script>
```

```
elem.onclick = function() {
    alert('Thanks');
};
</script>
```

- addEventListener

we cannot assign multiple handlers to one event

for e.g.

```
input.onclick = function() {alert(1);}
input.onclick = function() {alert(2);}
    // replaces the previous handler.
```

an alternate way of doing this with the special methods

addEventListener and removeEventListener

Syntax

```
element.addEventListener(event, handler, [option]);
element.removeEventListener(event, handler, [option]);
```

for e.g.

```
<input id="elem" type="button" value="push"/>

<script>
    function handler1(){
        alert('Thanks');
    };

    function handler2(){
        alert('Thank you very much');
    }

    elem.onclick = () => alert("Hello");
    elem.addEventListener("click", handler1);
                          "        "  handler2);
</script>
```

- Scripting Web forms

1. JS form (form Validation)
2. Radio button
3. Checkbox
4. Select box
5. Handling Input Event

23 Thursday Jul
Wk 30 205-161

23 • 07 • 2020

June
Weeks	23	24	25	26	27
Monday	1	8	15	22	29
Tuesday	2	9	16	23	30
Wednesday	3	10	17	24	
Thursday	4	11	18	25	
Friday	5	12	19	26	
Saturday	6	13	20	27	
Sunday	7	14	21	28	

July
Weeks	27	28	29	30	31
Monday		6	13	20	27
Tuesday		7	14	21	28
Wednesday	1	8	15	22	29
Thursday	2	9	16	23	30
Friday	3	10	17	24	31
Saturday	4	11	18	25	
Sunday	5	12	19	26	

★ JQuery

09

- It is a JS library that provides an easy way for developers.
- It is an open source cross platform JS library.
- It helps developers perform document traversal and manipulation, event handling, animation, Ajax, applying styles, and more
- It was released in Jan 2006 by John Resig
- write less do more library

10 11 12 01 02 03

Benefits : ① Add collapsable / expandable content
② Add animation
③ Change CSS styles dynamically
④ Perform AJAX Requests
⑤ Create widgets
⑥ General HTML DOM manipulation

04 05 06

07 Syntax :

```
$(function(){
    $("p").text("Hello");
});
```

$ is used to indicate that it is JQuery code
to avoid conflict, put $ |

ready() method

```
$(document).ready(function() {
});
```

allows us to run code as soon as the page's Document Object Model (DOM) becomes safe to manipulate

Basic syntax

- $ sign to define jQuery
- A (selector) to query (or find) HTML element
- A jQuery action() to be performed on the elements

```
$(selector).action()
```

JQuery Selectors

① JQuery Element Selectors

```
$("p")
$("p.intro")
$("p#demo")
```

② JQuery Attb. Selectors

```
$("[href]")
$("[href$=#]")
```

$("[href != '#']")
$("[href $= '.jpg']")

③ jQuery CSS selectors

$("p").css("background-color", "yellow");

- jQuery Attributes:

① Read: $(#image).attr("src");
② Set: $(#image).attr("src", "images/jquery.jpg");
③ MultipleSet: $(#image).attr({src: "images/jquery.jpg", alt: "jQuery"});
④ SetClass: $(p:last).addClass("selected");
⑤ Read/SetHtml: $(#id).html() & $(#id).html("value");

- jQuery Events

26 Sun

<Script type="text/js"> ~~src="jquery.js"~~
$(document).ready(function(){
$("button").click(function(){
$("p").hide();
});

- jQuery Callback functions

Call back function is executed after the current animation (effect) is finished.

Syntax:

```
$(selector).hide(speed, callback)
```

- jQuery HTML Manipulation

- Changing HTML content

```
$(selector).html(content)
```

- Adding HTML content

```
$(selector).append(content)
      //        prepend   //
      //        after     //
      //        before    //
```

- jQuery CSS Manipulation

- jQuery css() method
  - css(name) - Return CSS property value
  - css(name, value) - Set CSS property and value
  - css({properties}) - Set multiple CSS properties and values.

Example

```
$(this).css("background-color");
$("p").css("background-color", "yellow");
$("p").css({"background-color": "yellow", "font-size": "200%"});
$("#div1").height("200px"); $("#div2").width("300px") // Size manipulation
```

- offset
  - $(this).offset().left
  - $(this).offset().top
- Position
  - $(this).position().left
  - " " .top

- CSS Manipulation Methods

$(selector).css(name) - get style prop. of first matched element

$(selector).css(name, value) - set the value of one style property for matched element

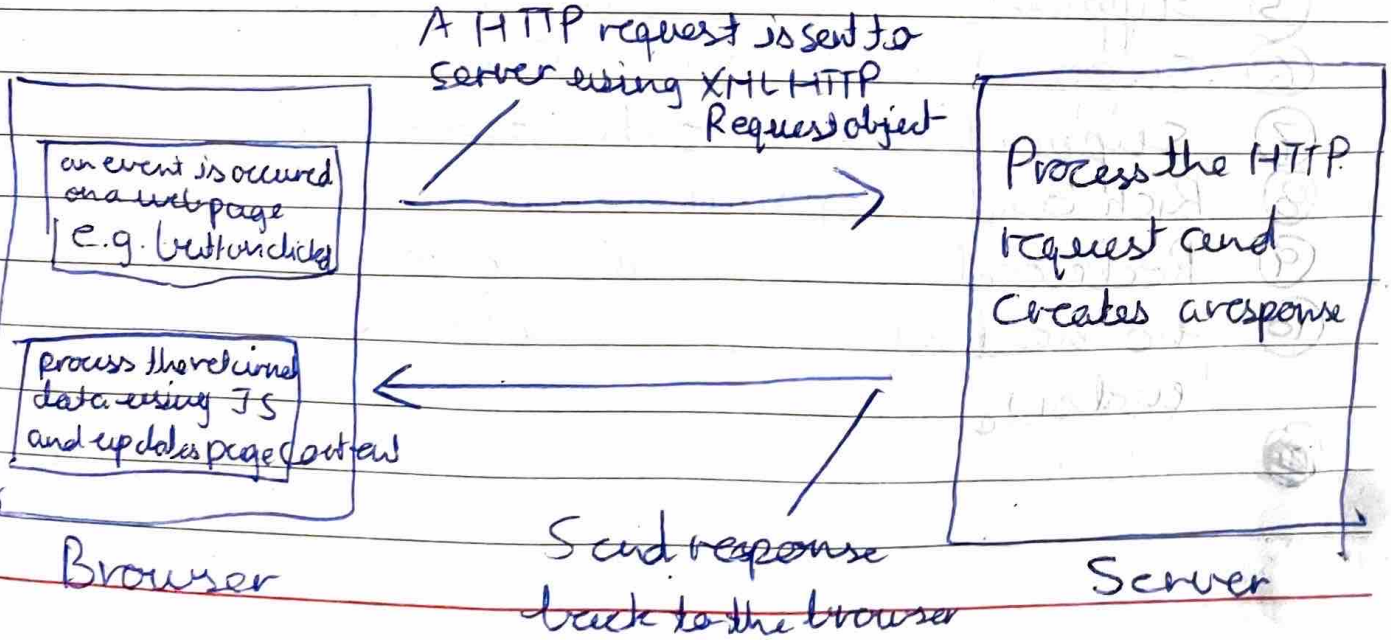
$(Selector).css({properties}) - Set multiple style properties

Saturday Sunday 2 9 16 23 30

$(selector).height(value) - height of matched
$(selector).width(value) - width " "

☆ Ajax

- stands for Asynchronous JaveScript and Xml.
- it is a means of loading data from the server and selectively updating parts of a webpage without reloading the whole page.
- basically Ajax makes ~~use~~ use of the browser built-in XML Http Request (XHR) object to send and recieve information to and from a web server asynchronously, in the background, without blocking the page or interfering with user experience
- Ajax driven online apps. Gmail, Google Maps, Google Docs, YouTube, Facebook, Flickr etc.
- Ajax is a term to describe the process of exchanging data from a webserver asynchronously through JS without refreshing the page



30 • 07 • 2020

- Applications of Ajax

1. Updating a webpage without reloading the page
2. Requesting data from the server after the page has been loaded.
3. Recieving data from the server after the page has been loaded.
4. Sending data to the server in the background without disturbing UI or other processes.

- Features

1. User friendly
2. Makes Webpag faster
3. Independant of server technology
4. Increases performance of a webpage
5. Supports live data binding
6. Support data view Control
7. supports client side ~~data~~ template rendering
8. Rich and responsive user interfaces
9. Reduced Consumption of server resources
10. no need of pushing button and reloading website