Advanced CSS

Objectives

01

Introduction to CSS

Understanding the basics of CSS.

02

Box model

Understanding and Analyzing box model.

03

Position Property

Understanding and Analyzing position property of CSS.

04

Transform Property

Understanding and Analyzing Transform Property in CSS.

Objectives

05

Transition Property

Understanding and Analyzing Transition Property of CSS.

06

CSS Animation

Understanding and Analyzing Animation in CSS.

07

DOM

Understand DOM related operations.

08

Callbacks

Objective: Discover how to pass functions as arguments

Objectives

09

Promise

Develop an understanding of Promise

10

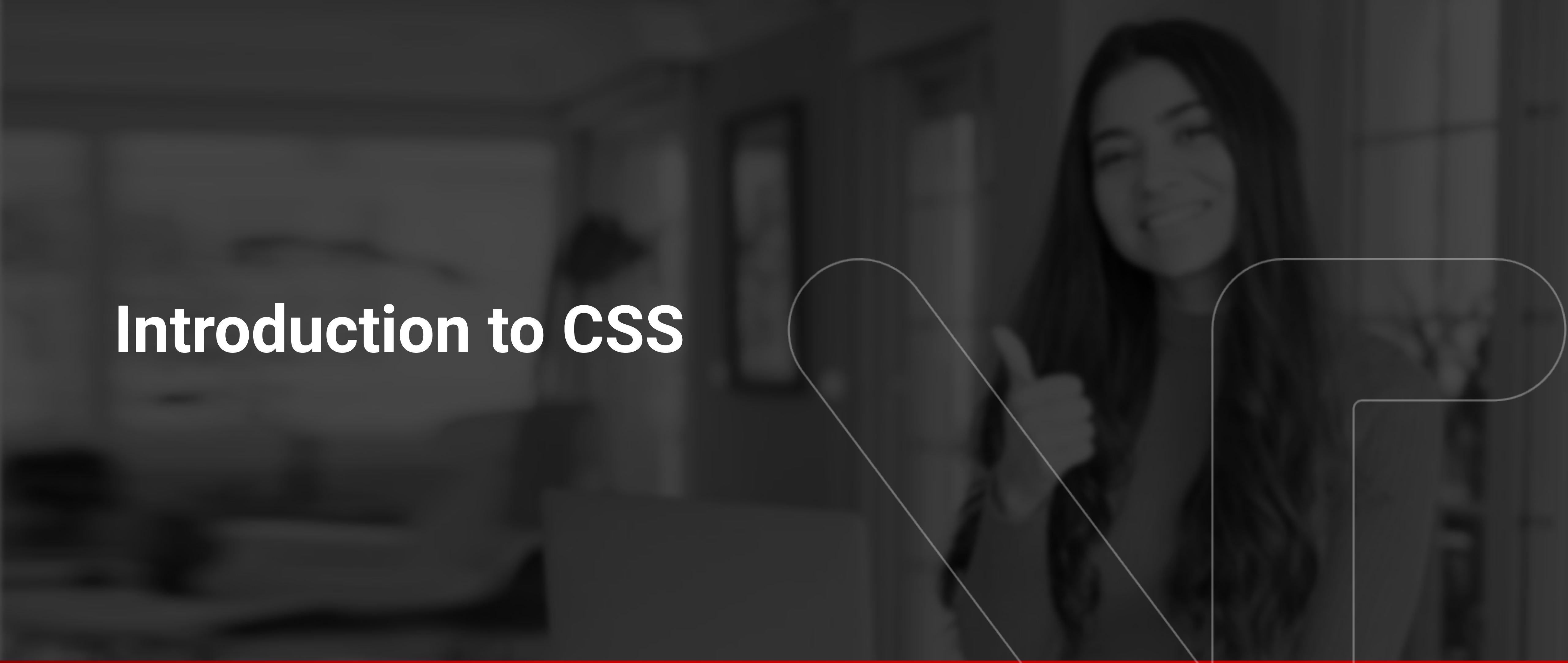
Asynchronous

Apply asynchronous code to handle AJAX requests

11

AJAX

Apply AJAX reque sts to load data between the client & the server



Introduction to CSS

Why Stylesheets?

- CSS is designed to enable the separation of presentation and content, including **layout, colors, and fonts**.[3]
- This separation can improve content **accessibility**;
- Provide more flexibility and control in the specification of presentation characteristics;
- Enable multiple **web pages** to share formatting by specifying the relevant CSS in a separate. css file, which reduces complexity and repetition in the structural content;
- And enable the .css file to be **cached** to improve the page load speed between the pages that share the file and its formatting.
- Separation of formatting and content also makes it feasible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (via speech-based browser or **screen reader**), and on **Braille-based** tactile devices. CSS also has rules for alternate formatting if the content is accessed on a mobile device.[4]

Ref: https://en.wikipedia.org/wiki/CSS

Introduction to CSS

- Cascading Style Sheets (CSS) are used to **style HTML elements in web pages**. HTML elements like headings and paragraphs can be styled using CSS. In addition, the **background colour, font size, font family, colour property, margins, padding and borders** to the HTML elements on a web page can be styled using CSS.

- There are **three types of CSS:**
  - Inline CSS
  - Internal or Embedded CSS
  - External CSS

Applying CSS

CSS can be applied in the following ways:

- **Inline** - by using the style attribute inside HTML elements
- **Internal** - by using a <style> element in the <head> section
- **External** - by using a <link> element to link to an external CSS file

The most common way to add CSS, is to keep the styles in external CSS files.

Box Model

CSS Box Model

In CSS, the term "box model" is used when talking about design and layout.

The CSS box model is essentially a box that wraps around every HTML element.

It consists of margins, borders, padding, as well as the actual content.

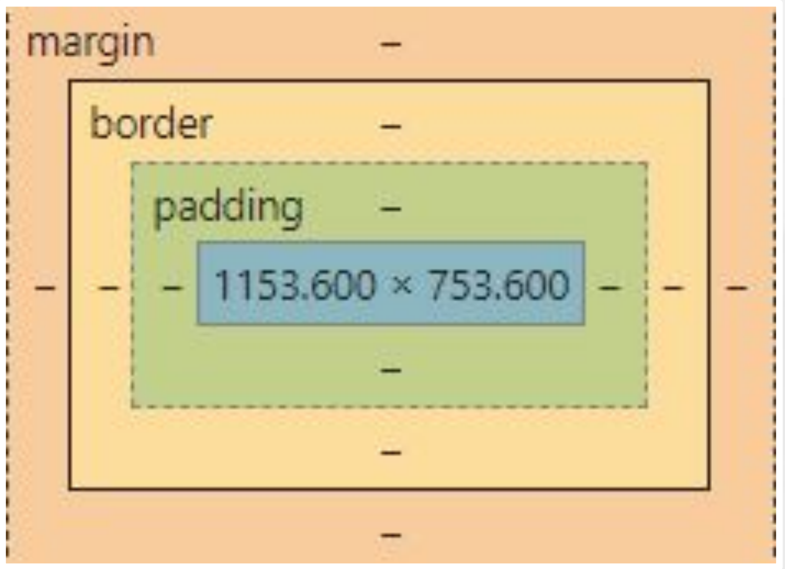
CSS Box Model

Content - This is the content of a box, where text and images appear.

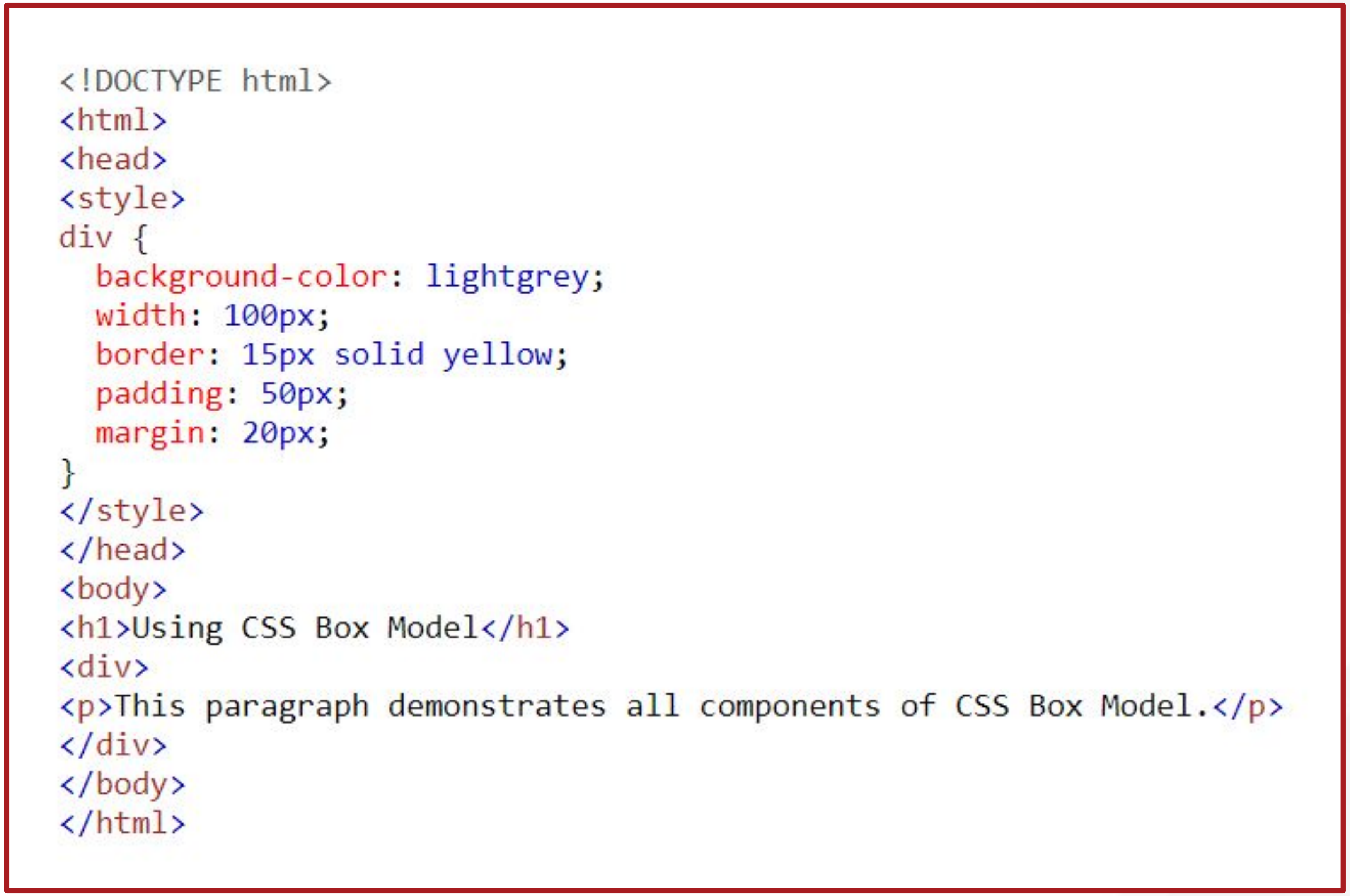
Padding - This clears an area around the content. The padding is transparent.

Border - This is a border that goes around the padding and content.

Margin - This clears an area outside the border. The margin is transparent.



Implementing Borders, Padding and Margins



```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  background-color: lightgrey;
  width: 100px;
  border: 15px solid yellow;
  padding: 50px;
  margin: 20px;
}
</style>
</head>
<body>
<h1>Using CSS Box Model</h1>
<div>
<p>This paragraph demonstrates all components of CSS Box Model.</p>
</div>
</body>
</html>
```

Position Property

CSS Position Property

Positioning Elements

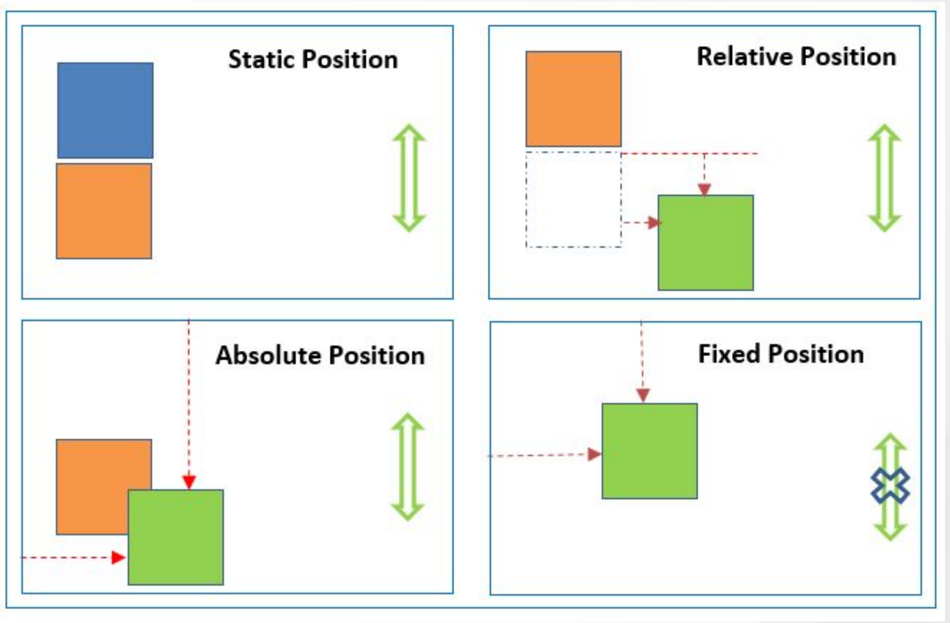
Position is set using TRBL (top, right, bottom & left)

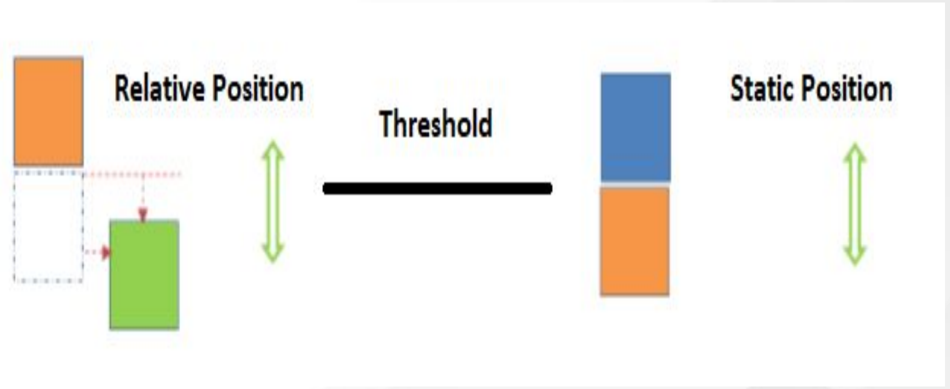
There are five different position values:

- static: default position (no effect of TRBL)
- relative: positioned relative to its normal position
- absolute: positioned relative to its relative or absolutely positioned parent, else document body
- fixed: positioned at a fixed place relative to the document body and no effect of page scrolling
- sticky: relative until a specified threshold, after that point it holds a static position

Syntax:

- position:static|relative|absolute|fixed|sticky
- z-index: order of overlapping elements (z-index:value)





Transform Property

CSS Transform Property (2D/3D)

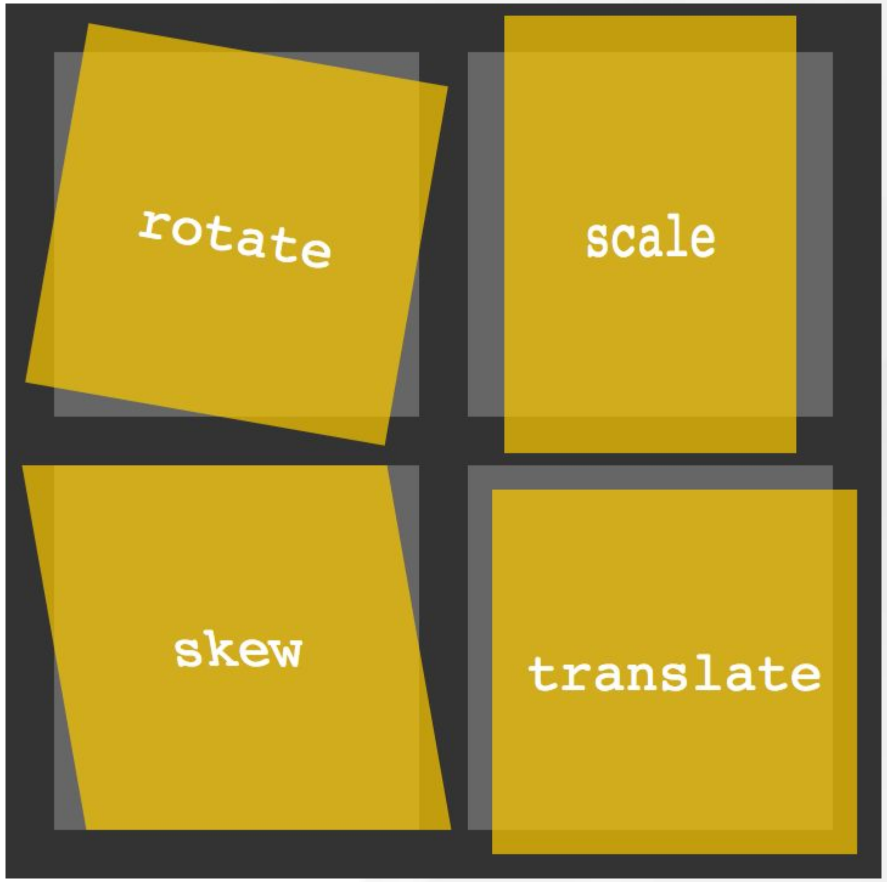
How to move, rotate, scale, and skew elements

- The transform property allows to move, rotate, scale, and skew elements.

Different transformation methods (2D):

- rotate(): transform:rotate(20deg)
- [rotateX(), rotateY(), rotateZ() (for 3D)]:
- translate(): transform:translate(50px,100px)
- scaleX(): transform:scaleX(2)
- scaleY(): transform:scaleY(3)
- scale(): transform:scale(2,3)
- skewX(): transform:skewX(20deg)
- skewY(): transform:skewY(40deg)
- skew(): transform:skew(20deg,40deg)
- matrix(): transform:matrix(1,-0.3,0,1,0,0)

matrix(scaleX,skewY,skewX,scaleY,translateX,translateY)



Transition Property

CSS Transition Property

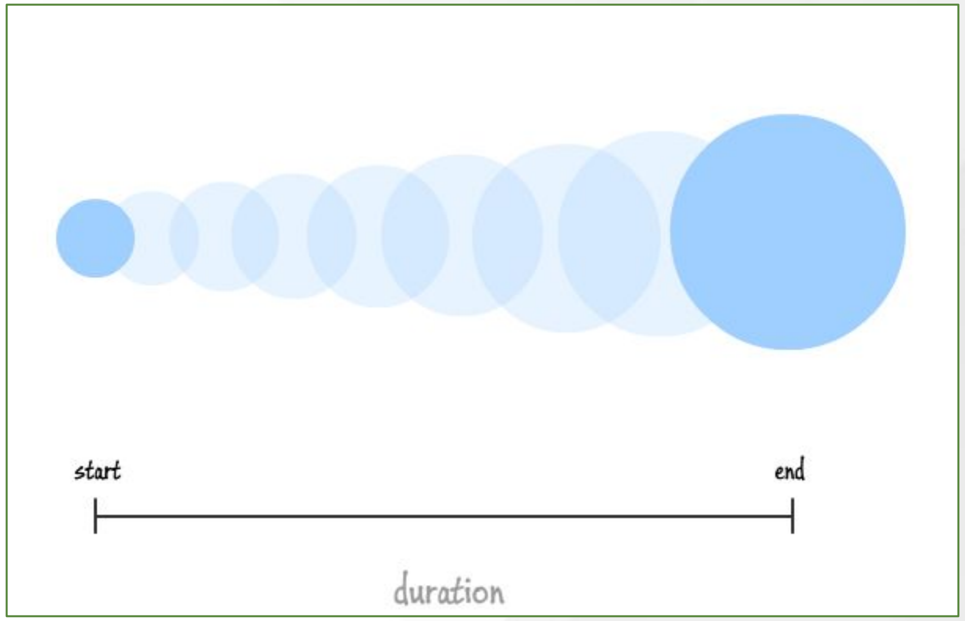
Smooth Changes in Property Values

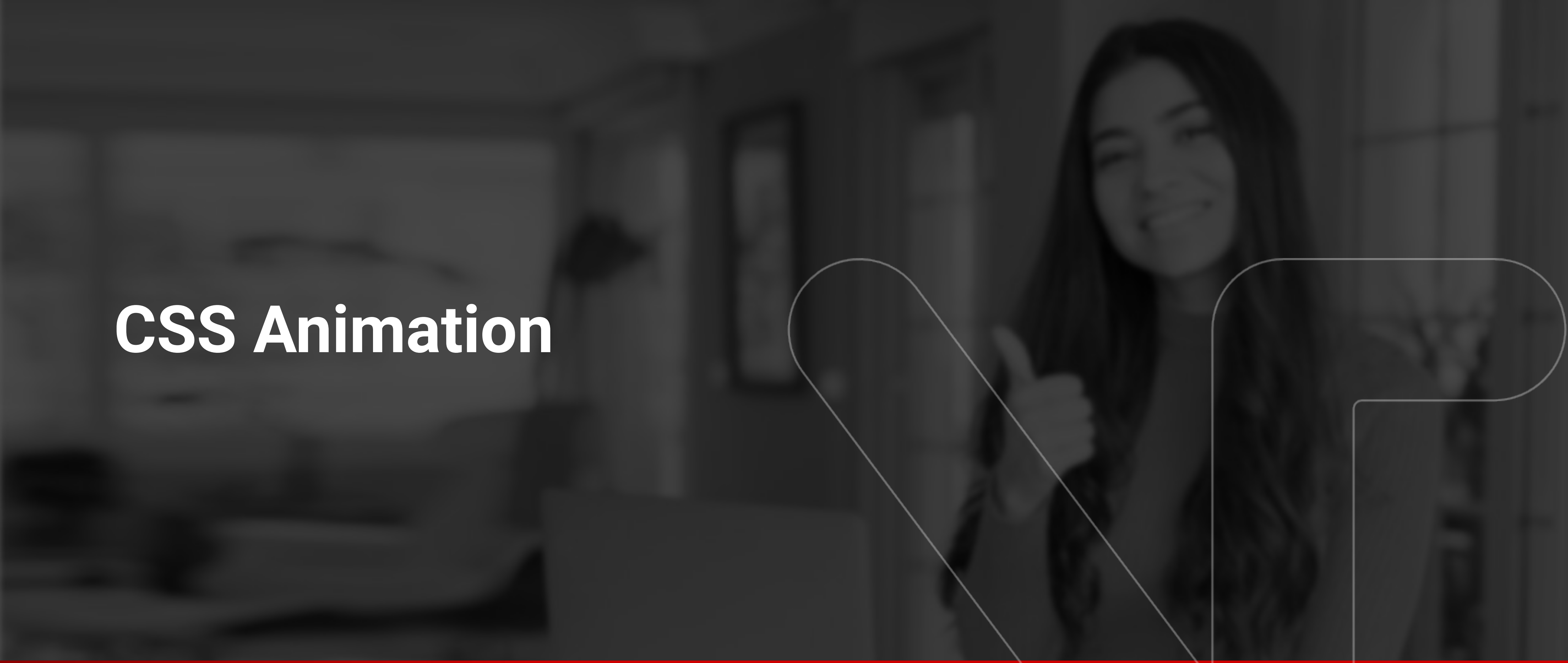
- transitions allows to change property values smoothly, over a given duration.

Different transition properties:

- transition-property:width|height|transform
- transition-duration:2s
- transition-timing-function: ease(default)|linear|ease-in|ease-out| ease-in-out|cubic-bezier(n,n,n,n)
- transition-delay:1s
- transition:width 2s linear 1s

  transition: transition-property transition-duration
  transition-timing-function transition-delay





CSS Animation

CSS Animation

Animating HTML Elements

- **animation is effectively transitions with key frames**
  @keyframes:
  either specify start state and end state (from to)
  or specify a set of various % values of duration

- **Different transition properties:**
  animation-name (keyframes name)
  animation-duration (total animation time)
  animation-delay (start lag, takes -ve values also)
  animation-iteration-count (num or infinite)
  animation-direction (reverse, alternate, etc.)
  animation-timing-function (speed curve)
  animation-fill-mode (state before & after anim.)
  animation **(one-liner of above property values)**

```
@keyframes anim1 {
 from {background-color: red;}
 to {background-color: yellow;}
}
@keyframes anim2 {
 0% {background-color:red; left:0px; top:0px;}
 25% {background-color:yellow; left:200px; top:0px;}
 50% {background-color:blue; left:200px; top:200px;}
 75% {background-color:green; left:0px; top:200px;}
 100% {background-color:red; left:0px; top:0px;}
}
div {
 width: 100px;
 height: 100px;
 position: relative;
 background-color: red;
 animation-name: anim2;
 animation-duration: 4s;
}
```

CSS Media Queries

Property to Gather Information Required for Responsive Design

- Media Queries makes a webpage adapt its layout to different screen sizes and media types. i.e. Design webpage based on Media Types & Media Features
- Media Queries can be used to check following things :
  - Width and height of the viewport
  - Width and height of the device
  - Device orientation (landscape or portrait mode)
  - Resolution
- Syntax :

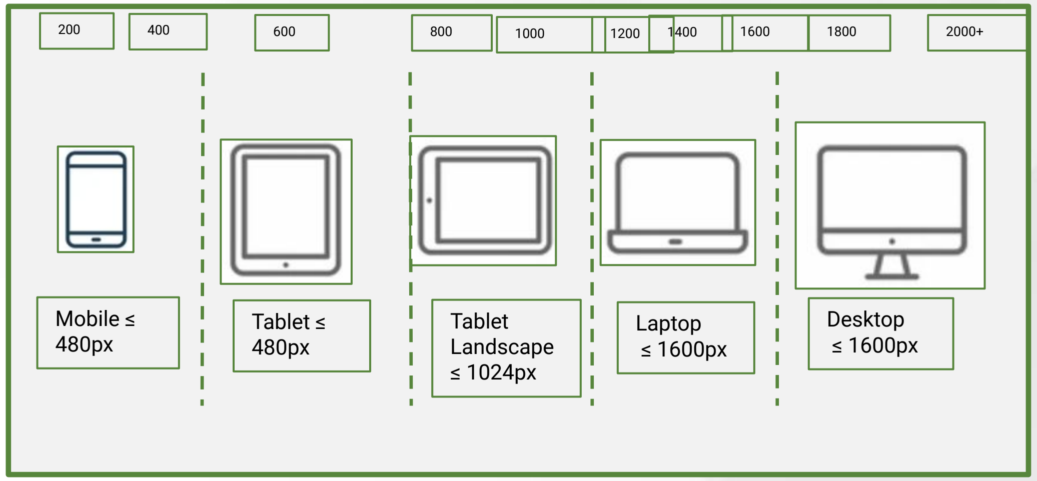
  @media not|only mediatype and (expressions){}

  mediatype values are all|print|screen|speech

  expressions min-width:80px, orientation:portrait

  Referring to different stylesheets for different media :

  <link rel="stylesheet" media="mediatype and|not|only (expressions)" href="media1.css">



DOM

The Document Object Model - What is?

- The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. - *W3C definition*

  - A document is the root node.
  - The root node has one child which is the <html> element. The <html> element is called the document element.
  - There are 12 type of nodes.
  - DOM elements are a type of node (written with HTML tags)

01. Node.ELEMENT_NODE
02. Node.ATTRIBUTE_NODE
03. Node.TEXT_NODE
04. Node.CDATA_SECTION_NODE
05. Node.ENTITY_REFERENCE_NODE
06. Node.ENTITY_REFERENCE_NODE
07. Node.PROCESSING_INSTRUCTION_NODE
08. Node.COMMENT_NODE
09. Node.DOCUMENT_NODE
10. Node.DOCUMENT_TYPE_NODE
11. Node.DOCUMENT_FRAGMENT_NODE
12. Node.NOTATION_NODE

HTML DOM

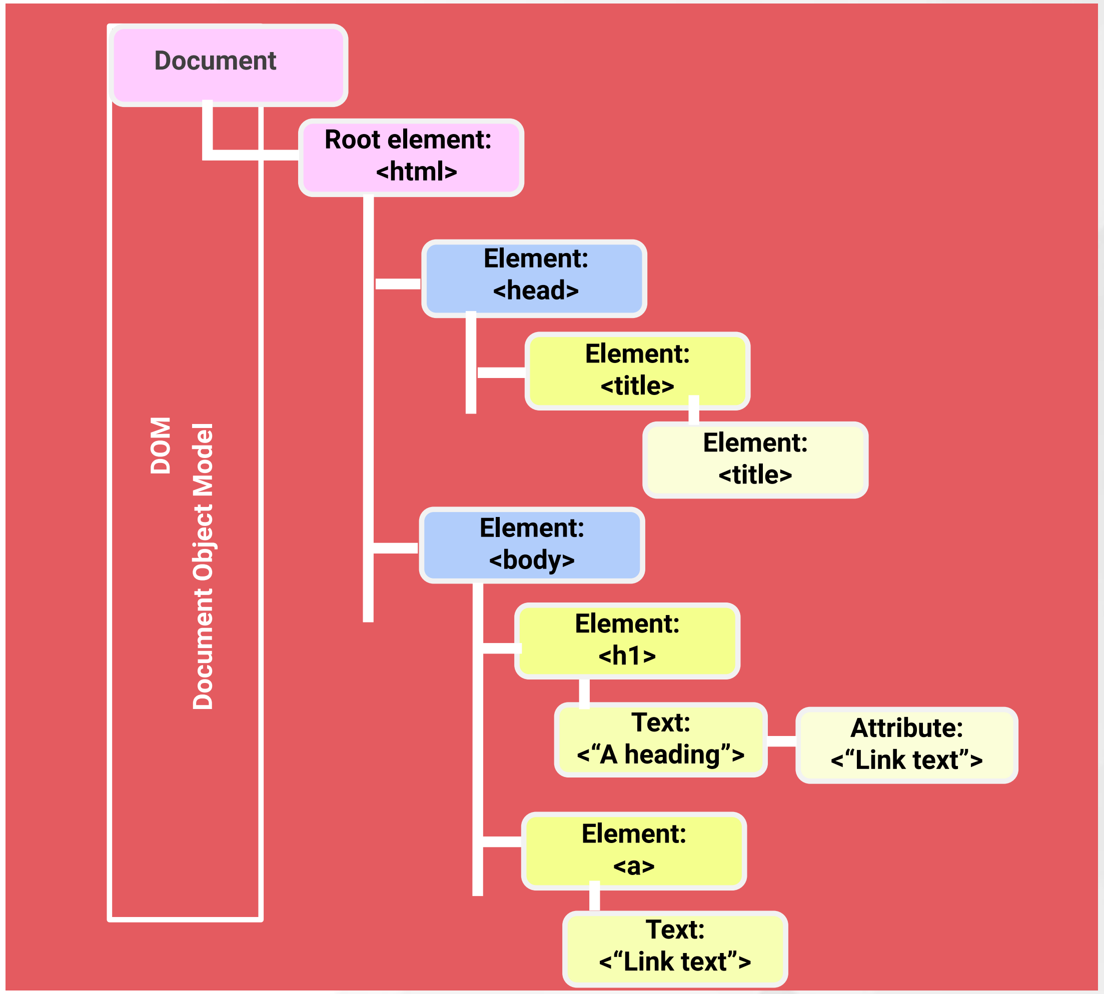
Standard object model and programming interface for HTML

The HTML DOM is a standard for how to:

- Get HTML elements
- Change HTML elements
- Add HTML elements
- Delete HTML elements

HTML DOM defines:

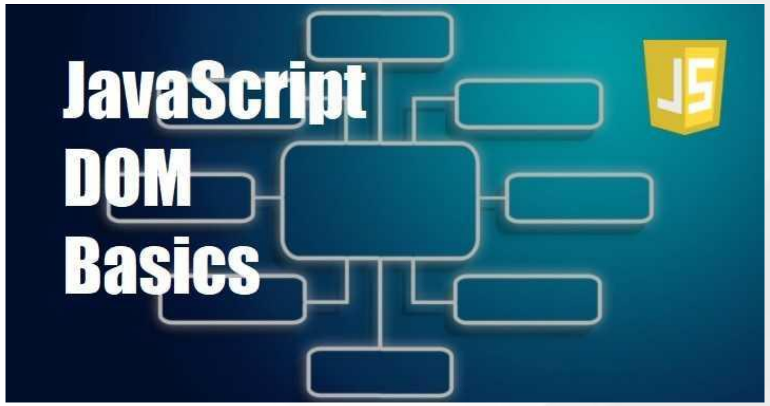
- The HTML elements as objects
- The properties of all HTML elements
- The methods to access all HTML elements
- The events for all HTML elementsv



HTML DOM & JavaScript

JavaScript can access and change all the elements of an HTML DOM

- **With DOM JavaScript can :**
  - Change all the HTML elements in the page
  - Change all the HTML attributes in the page
  - Change all the CSS styles in the page
  - Remove existing HTML elements and attributes
  - Add new HTML elements and attributes
  - React to all existing HTML events in the page
  - Create new HTML events in the page



HTML DOM Methods & Properties

JavaScript way to access DOM methods and properties

- HTML DOM methods are actions you can perform (on HTML Elements).
- HTML DOM properties are values (of HTML Elements) that you can set or change.

- **Methods to find HTML Elements :**

  getElementById()
  returns the element having the given id value.
  getElementsByName()
  returns all the elements having the given name value.
  getElementsByTagName()
  returns all the elements having the given tag name.
  getElementsByClassName()
  returns all the elements having the given class name.

```
document.getElementById(div).
else if (i==2)
{
    var atpos=inputs[i].indexOf("@");
    var dotpos=inputs[i].lastIndexOf(
    if (atpos<1 || dotpos<atpos+2 ||
    document.getElementById('errEmail').
    else
ment.getElementById(div).
```

HTML DOM

JavaScript way to access DOM methods and properties

Change properties of HTML Elements:

Change the innerHTML of an element

element.innerHTML = new html content

Change the attribute value of an HTML element

element.attribute = new value

Change the style of an HTML element

element.style.property = new style

Methods to change properties of HTML Elements:

Change the attribute value of an HTML element

element.setAttribute(attribute, value)

```
document.getElementById(div).
else if (i==2)
{
    var atpos=inputs[i].indexOf("@");
    var dotpos=inputs[i].lastIndexOf(".");
    if (atpos<1 || dotpos<atpos+2 ||
    document.getElementById('errEmail').
    else
    ment.getElementById(div).
```

Callbacks

Callbacks in Javascript

Callbacks are functions passed as arguments to another function

Callbacks: some useful examples of callbacks

- **setTimeout:** specifying a callback function to be executed on specified time-out parameter in millisec
  someVariableAsId = setTimeout(callback function, milliseconds);
- **clearTimeout:** prevent the function set with the setTimeout to execute
  clearTimeout(someVariableAsId);
- **setInterval:** specifying a callback function to be executed for each interval parameter in millisec
  someVariableAsId = setInterval(callback function, milliseconds);
- **clearInterval:** clears a timer set in setInterval method
  clearInterval(someVariableAsId);

```
function logData(data) {
  console.log(data);
}
function f(a, b, cbFn) {
  let sum = a + b;
  cbFn(sum);
}
f(2,3,logData);
setTimeout(function(){f(1,2,logData);}, 3000);
setInterval(clock, 1000);
function clock() {
  let d = new Date();
  document.getElementById("output").innerHTML=
  d.getHours() + ":" +
  d.getMinutes() + ":" +
  d.getSeconds();
}
```

Promise

Promises in Javascript

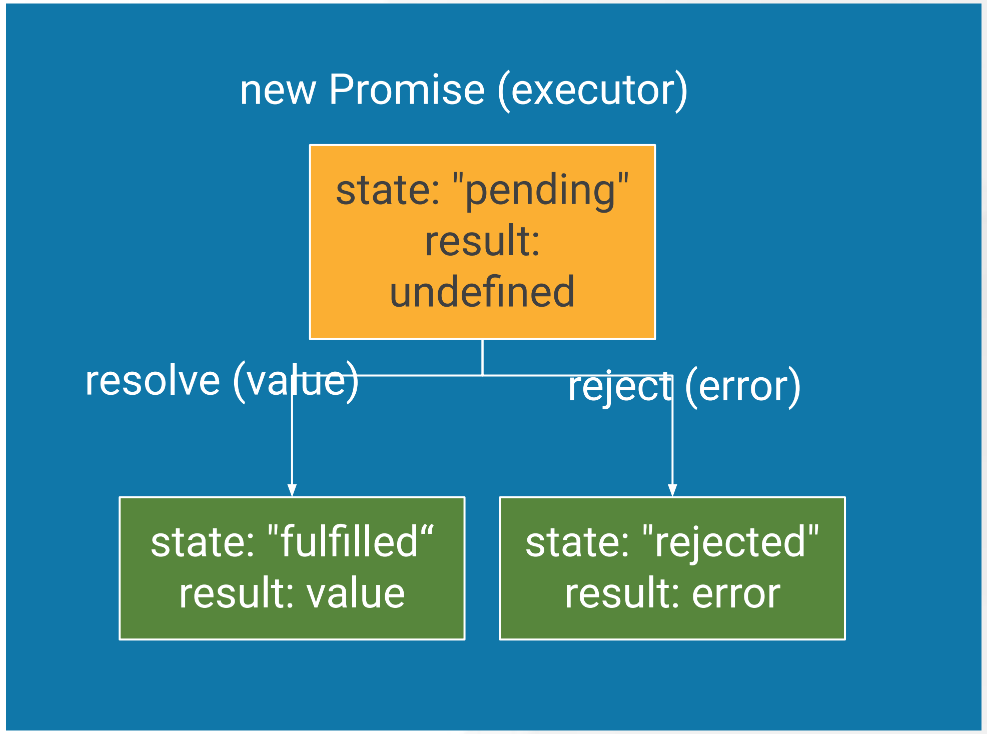
An object contains producing code and calls to consuming code

Promises: Some code may take time to execute and some other code must wait for the result. Promise links those

- What is a promise: A promise can be loosely defined as a ***proxy*** for a ***value*** that will ***eventually*** become available.

How it works (three stages):

1. A promise has been called: it will start in a ***pending state***
2. The calling function continues executing while the promise is ***pending***: until it ***resolves***
3. Giving the calling function whatever data was being requested.


new Promise (executor)
state: "pending" result: undefined
resolve (value)
reject (error)
state: "fulfilled“ result: value
state: "rejected" result: error

Promises in Javascript

An object contains producing code and calls to consuming code

Promises: Some code may take time to execute and some other code must wait for the result, promise links those

- **Concept of asynchronous functions:** functions which run in parallel with other functions
- **Producing code:** code that may take time to execute

```
let aPromise = new Promise(function(a, b) { a();
/*in success*/ b();/*in failure*/ });
```

- **Consuming code:** code that must wait for the result

```
aPromise.then(
  function(value) { /*success code*/ },
  function(error) { /*failure code*/ }
);
```

```
// Promise
function logData(data) {console.log(data);}
let prom = new Promise(function(succ, fail) {
  let x = 1;
  if (x == 0) {
     succ("OK");
  } else {
     fail("Error");
  }
});
prom.then(
  function(value) {logData(value);},
  function(error) {logData(error);}
);
```

Asynchronous

Promises in Javascript

An object contains producing code and calls to consuming code

async and await: Make promises easier to write

- **Concept of asynchronous functions:** functions which run in parallel with other functions
- **async:** makes a function return a Promise

```
async function f(){/* code; return; */}
```

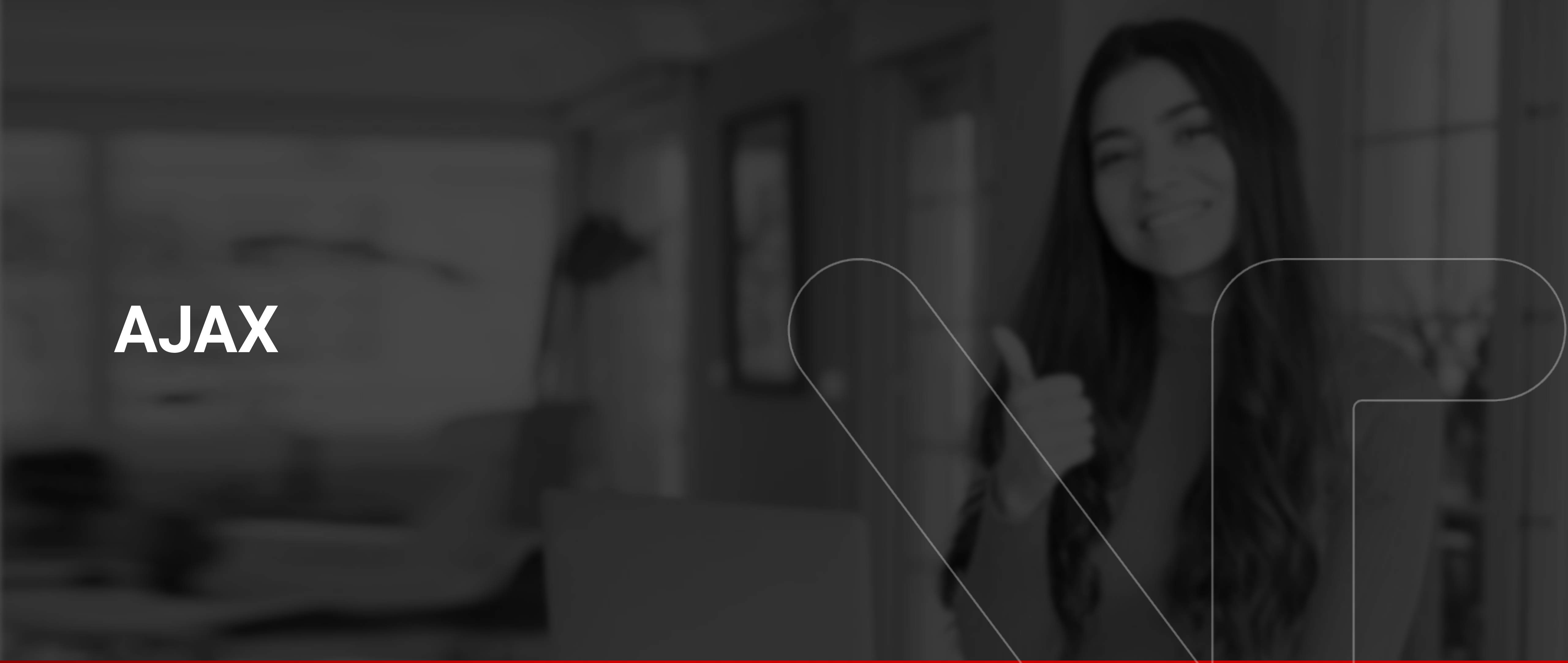
- **await:** makes a function within a async function wait for a Promise

```
async function f(){
  let aPromise = new
  Promise(function(a, b){a();b();});
  let c = await aPromise;
);
```

```
// async, await
async function fnAsync() {
 let prom = new
Promise(function(succ,fail) {
   succ("I Promised a return!");
 });

document.getElementById("output").inner
HTML=
 await prom;
}

fnAsync();
```



AJAX

AJAX

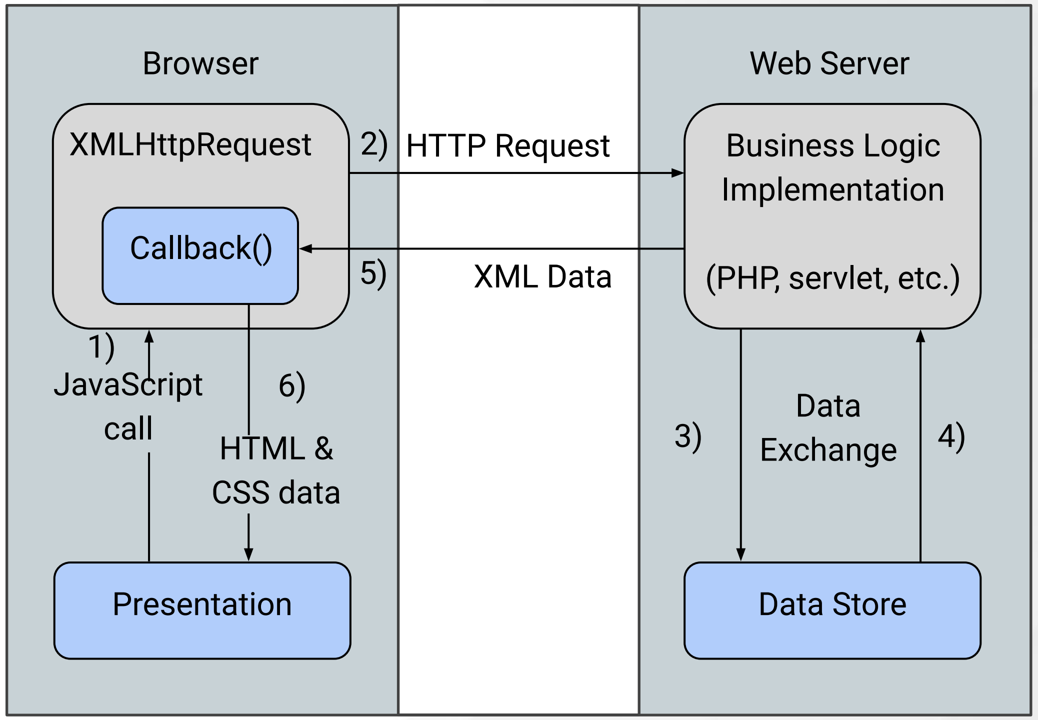
AJAX: Asynchronous JavaScript And XML

AJAX: Load data from server & display without reloading the client

- **What is XML**: Extensible Markup Language. Tag based data structure, easy to transfer.

```
<t1>
<t2>some data</t2>
<t3>some other data</t3>
</t1>
```

- **Request data from the server**: it can request data (XML or Text) from the server by optionally sending data (to specify the request) to the server using GET or POST method.
- **Receive the response from the server**: it can receive data (XML or Text) as a response from the server and allow JS to work with that data.



Http Request

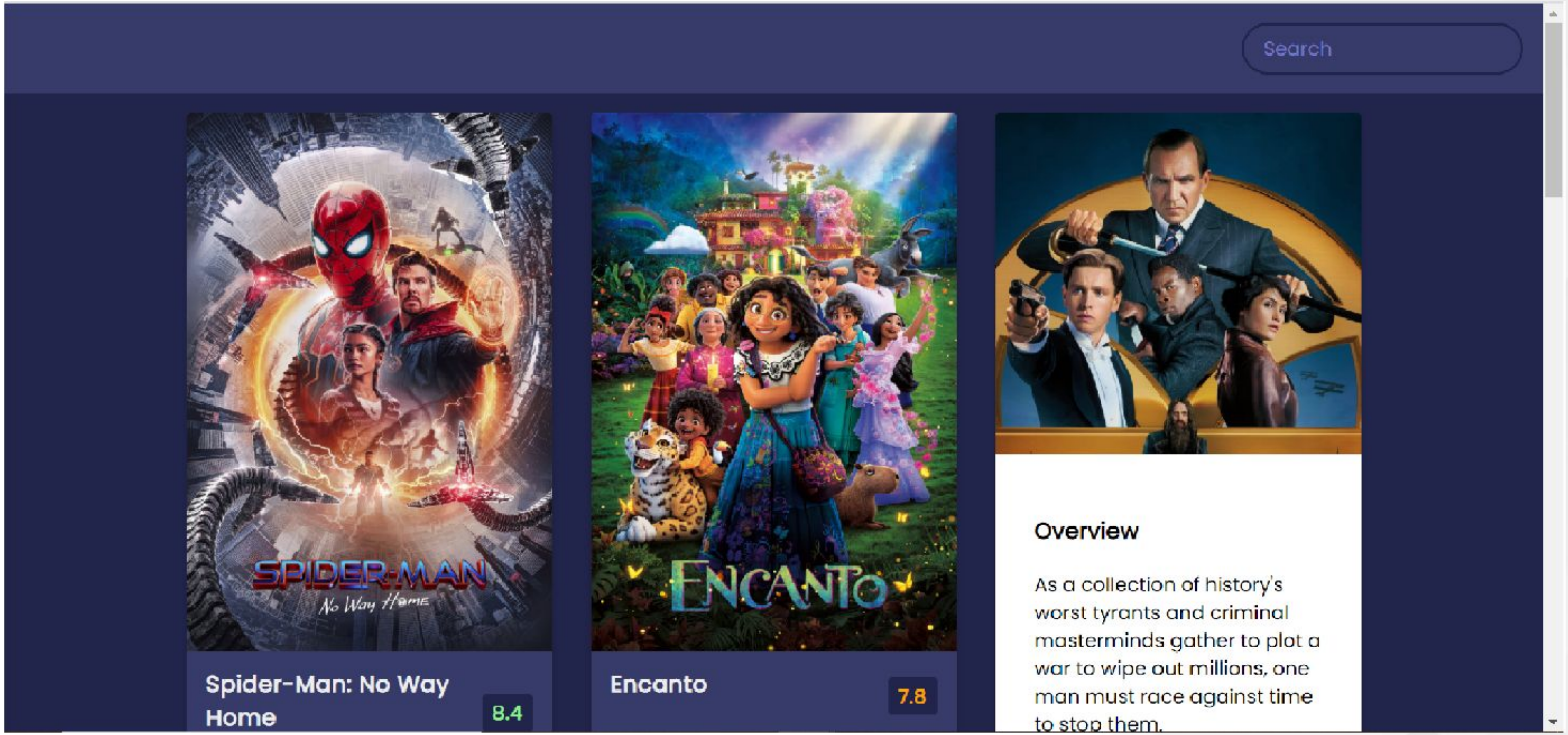
Http Request is for exchanging data with the server

XMLHttpRequest: built-in object of the browser

- **Create XMLHttpRequest object**:
  ```
  var req = new XMLHttpRequest();
  ```
- **Onreadystatechange**: ResponseText/XML, status,statusText
  ```
  req.onreadystatechange = function(){
  if(this.readyState == 4 // if < 4 not ready
    && this.status == 200 // 200 : status OK
   ){/* code with this.responseText */}
   };
  ```
- **Open Http request**: True - asynchronous, False - synchronous
  ```
  req.open("GET/POST", URL, true/false);
  ```
- **Send Http request**: Send request to server
  ```
  req.send();
  ```

```
// XMLHttpRequest
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
 if(this.readyState==4 &&
this.status==200){
   console.log(this.responseText;)
  }
};
xhttp.open("GET", "../some.txt", true);
xhttp.send();
```

Movie App: Let's Get Started



Spider-Man: No Way Home 8.4

Encanto 7.8

Overview

As a collection of history's worst tyrants and criminal masterminds gather to plot a war to wipe out millions, one man must race against time to stop them.

Hero Vired

Thank You!