DOM

(Document Object Model)

.NET

*The **Document Object Model** (**DOM**) is the data representation of the objects that comprise the structure and content of a document on the web. The DOM represents an HTML or XML document in memory.*

https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

DOM (Document Object Model)

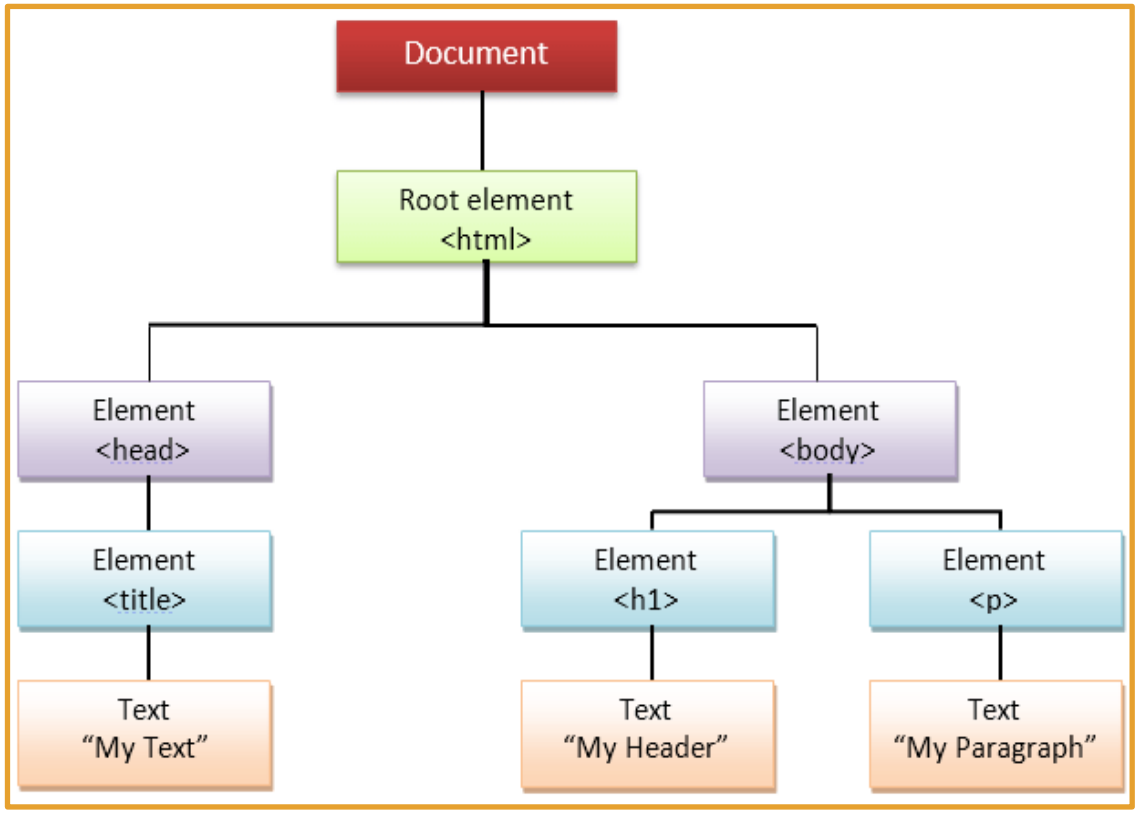
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

The ***Document Object Model (DOM)*** is a programming interface for HTML and XML documents. It represents the page as nodes and objects. The DOM allows programs to change the documents' structure, style, and content.

A Web page is one document that can be

- displayed in the browser window,
- viewed as HTML, and
- represented by the DOM.

The DOM can be manipulated by scripting languages like JavaScript.



DOM in action

https://javascript.info/

All the properties, methods, and events available for manipulating and creating web pages are organized into objects.

For example, the “document object” represents the document itself and the ***<table>*** object implements the ***HTMLTableElement*** DOM interface for accessing HTML tables.

getElementsByTagName(“p”) returns a list of all the ***<p>*** elements in the document.

```
1  const paragraphs = document.getElementsByTagName("p");
2  // paragraphs[0] is the first <p> element
3  // paragraphs[1] is the second <p> element, etc.
4  alert(paragraphs[0].nodeName);
```

DOM – How to Access the DOM

Within the <head> of your .html file or at the bottom of the <body>, include a <script> tag which contains the ***.js*** file you want to use for the .html page. You can then access the ***document*** in the .js file using dot notation on the keyword document as in:

document.getElementById("#IdName")

```
<head>
        <!--metadata tags....-->
</head>
<body>
        <!—body content–>
        <script type="text/javascript" src="jsfile. js"></script>
</body>
```

DOM – Selectors

https://blog.bitsrc.io/dom-selectors-explained-70260049aaf0

JS DOM ***Selectors*** are used to select HTML ***elements*** within a ***document***. There are 5 ***selectors***.

Selector Name	Purpose
document.getElementsByTagName("myTag")	Returns a collection (array) of Items matching the tag name.
document.getElementsByClassName("myClass")	Returns a collection (array) of Items matching the class name. The '.' is only needed for .querySelector(.className)
document.getElementById("myId")	Returns the first matched id name. Id's are supposed to be unique in the .HTML file.
document.querySelector("li")	Returns the first element that matches the specified CSS selector.
document.querySelectorAll("ol")	Returns all the elements that match the specified CSS selector

Walking the DOM - Basics

https://javascript.info/dom-navigation

We can perform many actions with ***elements*** and their contents after accessing the correct DOM element.

The topmost tree nodes are available directly as document properties:

To get the <html> element, use:

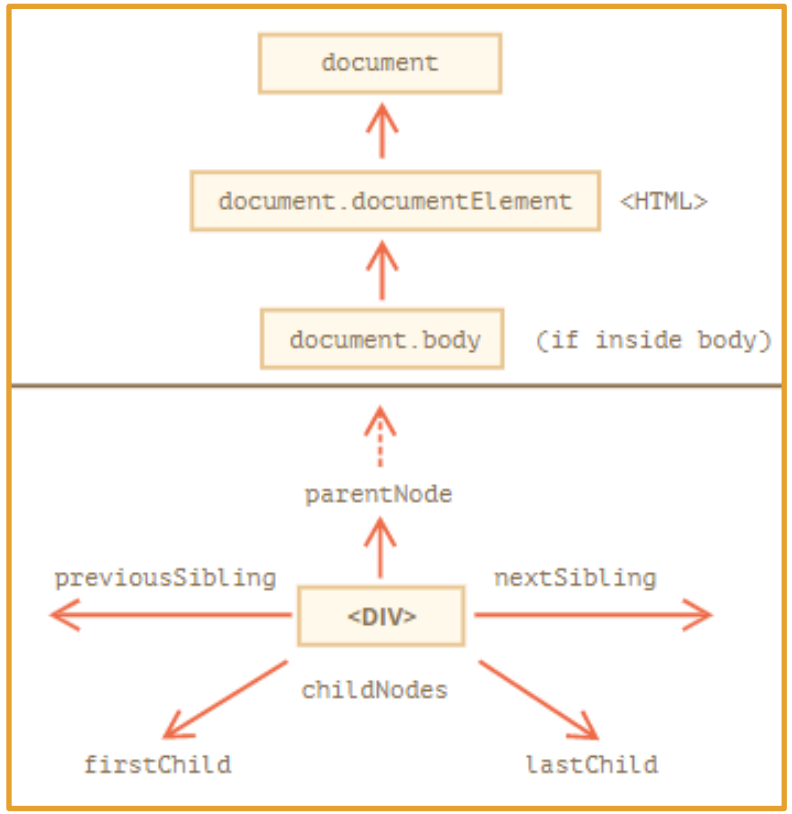
- let html = document.documentElement;

To get the <body> element, use:

- let body = document.body;

To get the <head> element, use

- let head = document.head;



Walking the DOM - Basics

https://javascript.info/dom-navigation

- Child nodes – Elements that are nested in the given element. <head> and <body> are both children of <html>.
- Siblings - nodes that are children of the same parent.
- Descendants – all elements nested in the given element. This includes children, their children, etc.

In this example, <body> has two children, and <ul>. and <ul> are siblings

Among the descendants of <body> are the direct children and <ul> and more deeply nested elements, like <li> (child of <ul>) and <b> (child of <li>).

```
1  <html>
2  <body>
3    <div>Begin</div>
4
5    <ul>
6      <li>
7        <b>Information</b>
8      </li>
9    </ul>
10 </body>
11 </html>
```

Walking the nodes of the DOM

https://javascript.info/dom-navigation

Method	Explanation/Example
document.body.childNodes[n]	.childNodes lists all child nodes as an htmlcollection, including text nodes.
.firstChild	Gives access to the first child. elem.firstChild
.lastChild	Gives access to the last child. elem.lastChild
.nextSibling	Access the following or “right” sibling going down the page.
.previousSibling	Access the prior or “left” sibling going up the page.
.parentNode	Access the parent of the current node.
document.createElement(‘div’);	Create a new element in the document object.

Walking the elements of the DOM

https://javascript.info/dom-navigation#children-childnodes-firstchild-lastchild

Method	Explanation/Example
.firstElementChild	Gives access to the first child.
.lastElementChild	Gives access to the last child.
.nextElementSibling	Access the next (“right”) sibling element going down the page.
.previousElementSibling	Access the prior (“left”) sibling element going up the page.
.parentElement	Access the parent of the current node if it’s an element. Returns ***null*** if not an element
.children	Returns an array of all children elements.

DOM – Events Overview

https://developer.mozilla.org/en-US/docs/Web/Events
https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers/onclick

---

DOM ***Events*** are sent when things happen on the HTML page, such as when a ***button*** is clicked, or an object is 'moused' over.

Each ***event*** is represented by an ***object*** which is based on the ***Event*** interface that can have fields and/or functions used to get additional information about what happened.

The two most common ***events*** are '***clicks'*** and form submissions.

Event Listeners and Event Handlers

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/A_first_splash

---

The “construct” that listens for an event to happen is called an ***event listener.*** The block of code that runs when the event fires is called an ***event handler***.

The below code creates an object that represents an HTML element with id=‘button’.

let button = document.getElementById(“button”);

guessSubmit holds all the data from an element.

It uses a built-in JS helper function called addEventListener() which takes two arguments.

1. The type of event we are listening for (***click***), and
2. A ***callback*** to the code we want to run when the event occurs. Because checkSubmission() is a callback, you don’t need to use the ( ).

button.addEventListener('click’, callbackFunction);

Bubbling and Capture

https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Examples#Example_5:_Event_Propagation
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events

---

Event ***bubbling*** and event ***capture*** are two mechanisms that describe what happens when two ***handlers*** of the same ***event type*** are activated on one ***element***.

When an event is fired (a '***click***') on an element that has parent elements, modern browsers run two different phases — the ***capturing*** phase and the ***bubbling*** phase.

Capture Phase	Bubble Phase
The browser checks to see if the element's outer-most ancestor (<html>) has an 'onclick' event handler registered on it and, if so, runs it. Then it moves on to the next element inside <html> and does the same thing, until it reaches the element that was actually clicked.	The browser checks to see if the element that was actually clicked on has an 'onclick' event handler registered on it for the bubbling phase and runs it if so. Then it moves on to the next immediate ancestor element and does the same thing until it reaches the <html> element.

Bubbling and Capture

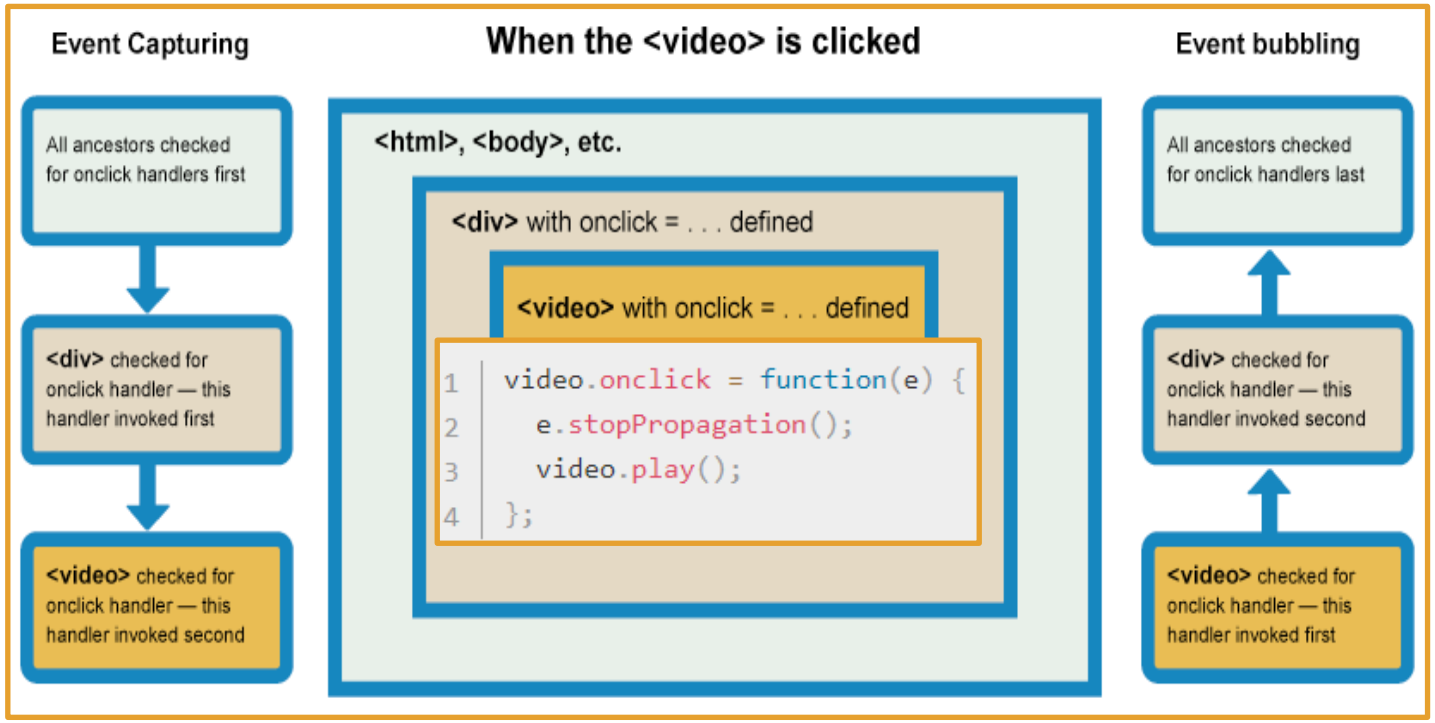
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Examples#Example_5:_Event_Propagation
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events

---

Browsers automatically register event handlers for the ***bubbling*** phase.

When the video is clicked, the 'click' event bubbles outward from the ***<video>*** element outwards to its parent *** ***, to the ***<html>***. If any of these elements has an '***on-click***' event handler, they will fire.

.stopPropagation() and ***.stopImmediatePropagation()*** are provided to stop further bubbling propagation.

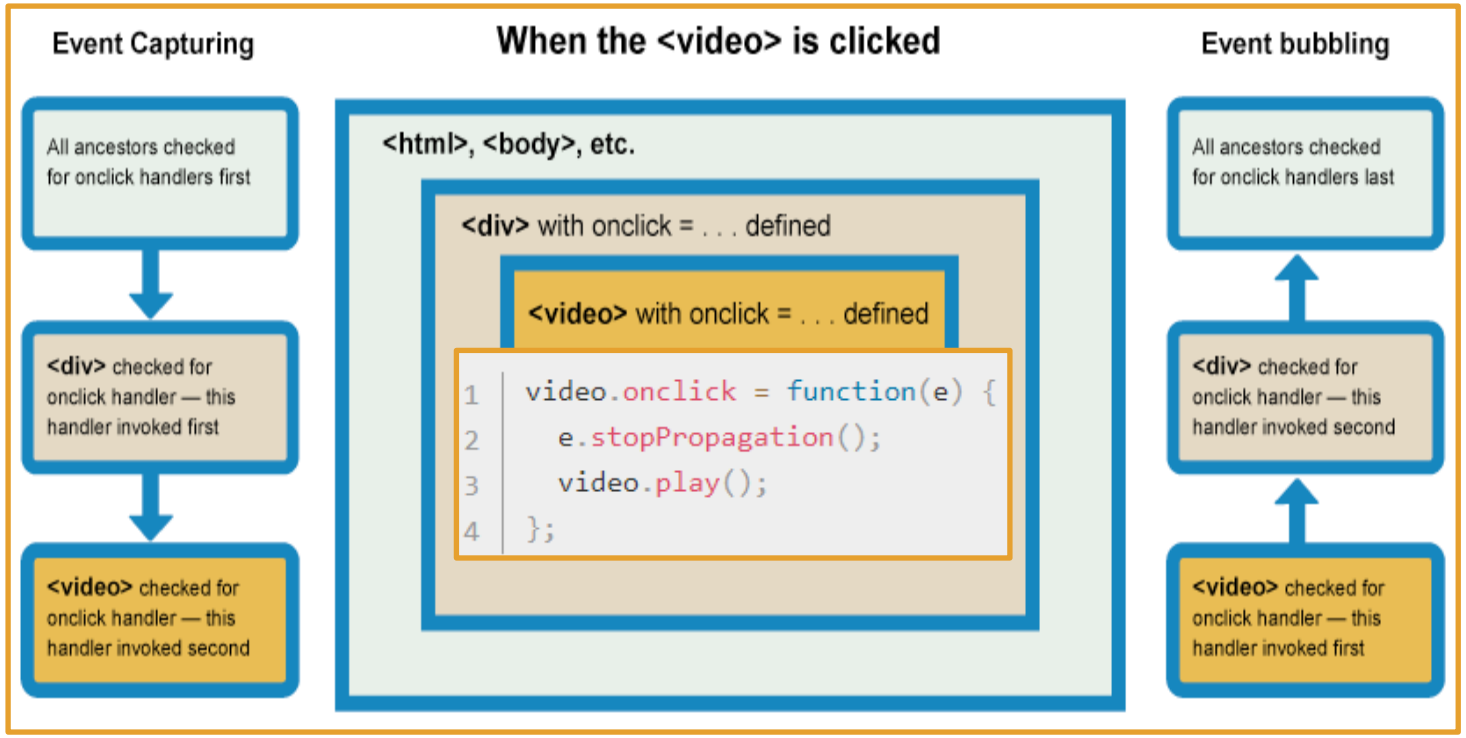


Bubbling and Capture

https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Examples#Example_5:_Event_Propagation
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events
https://www.carlrippon.com/stoppropagation-v-stopimmediatepropagation/

Browsers automatically register event handlers for the ***bubbling*** phase.

- stopPropagation() prevents other listeners above the triggered listener from being triggered.
- If several listeners are on the same element for the same event type, they are called in sequential order.
- stopImmediatePropagation() prevents remaining listeners from being called on the same element.



DOM Events Order

https://developer.mozilla.org/en-US/docs/Web/API/Document/DOMContentLoaded_event

---

The ***DOMContentLoaded*** event fires when the initial HTML document has been completely loaded and parsed, without waiting for stylesheets, images, and subframes to finish loading.

A different event, ***load***, should be used only to detect a fully-loaded page. It is a common mistake to use ***load*** where ***DOMContentLoaded*** would be more appropriate.

Synchronous JavaScript pauses when parsing the DOM. To parse the DOM as fast as possible after the user has requested the page, make your JavaScript asynchronous to optimize the loading of stylesheets.

If loaded as usual, stylesheets slow down ***DOM*** parsing as they're loaded in parallel. This "steals" traffic from the main HTML document.

```
<script>
  document.addEventListener('DOMContentLoaded', (event) => {
    console.log('DOM fully loaded and parsed');
  });

for( let i = 0; i < 1000000000; i++)
{} // This synchronous script is going to delay parsing of the DOM,
   // so the DOMContentLoaded event is going to launch later.
</script>
```

DOM Events Order

https://developer.mozilla.org/en-US/docs/Web/API/Document/DOMContentLoaded_event

DOMContentLoaded may fire before your script has a chance to run, so it is wise to check before adding a listener.

JS

```js
const log = document.querySelector('.event-log-contents');
const reload = document.querySelector('#reload');

reload.addEventListener('click', () => {
  log.textContent ='';
  window.setTimeout(() => {
      window.location.reload(true);
  }, 200);
});

window.addEventListener('load', (event) => {
    log.textContent = log.textContent + 'load\n';
});

document.addEventListener('readystatechange', (event) => {
    log.textContent = log.textContent + `readystate: ${document.readyState}\n`;
});

document.addEventListener('DOMContentLoaded', (event) => {
    log.textContent = log.textContent + `DOMContentLoaded\n`;
});
```

HTML

```html
<div class="controls">
  <button id="reload" type="button">Reload</button>
</div>

<div class="event-log">
  <label>Event log:</label>
  <textarea readonly class="event-log-contents" rows="8" cols="30"></textarea>
</div>
```

Result of the above

Reload

Event log:

```
readystate: interactive
DOMContentLoaded
readystate: complete
load
```

GuessingGame Tutorial

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/A_first_splash

---

1. Complete the guessingGame Tutorial.
2. Change guessingGame from using ***events*** to using a ***form*** to get the number.
3. Use https://javascript.info/ui for independent study.