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. You send data to APIs endpoints to create web content and applications.

HTTPS://DEVELOPER.MOZILLA.ORG/EN-US/DOCS/WEB/API/DOCUMENT_OBJECT_MODEL/INTRODUCTION

DOM (Document Object Model)

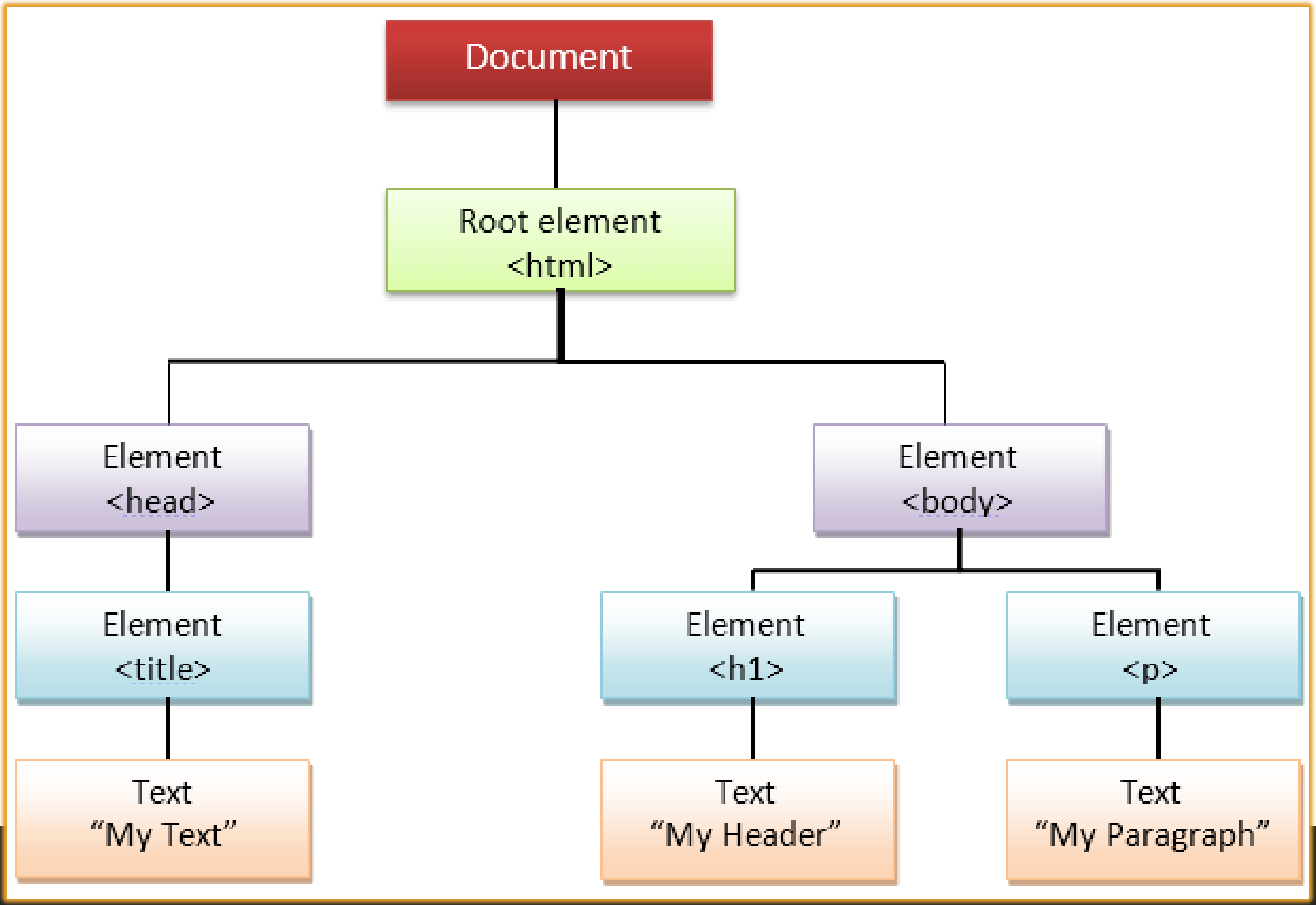
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

The ***Document Object Model (DOM)*** is a programming interface for HTML and XML documents. It represents the page (as nodes and objects) so that programs can change the document 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.

Only the DOM can be manipulated by scripting languages like JavaScript.

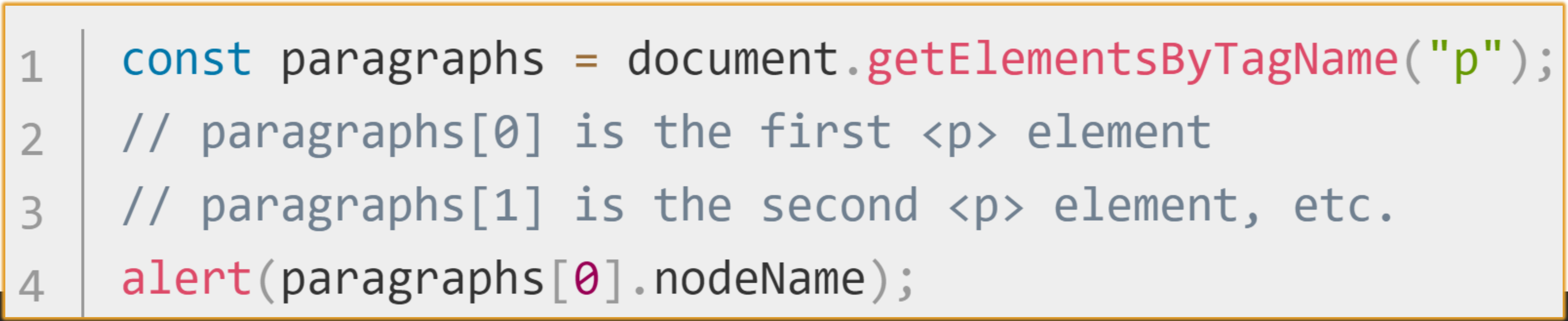


DOM in action

https://javascript.info/

All the properties, methods, and events available for manipulating and creating web pages are organized into objects. The document object represents the document itself and the *<table>* object implements the *HTMLTableElement* DOM interface for accessing HTML tables.

getElementsByTagName( ) must return a list of all the *<p>* elements in the document.



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

Dom – Gaining Access

Within the ***<head>*** or your .HTML file, include a ***<script>*** tag which contains the ***.js*** file you want to use for the ***.HTML*** page. You can then load the document in the ***.js*** file.

```
<head>
        // other tags, etc....
</head>
<body>
        //...
        <script type="text/javascript" src="jsfile. js"></script>
</body>
```

You an add JavaScript code directly in the .HTML in a <script> tag.

DOM – Selectors

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

DOM Selectors are used to select HTML elements within a document using JavaScript.

There are 5 selectors.

Selector Name	Purpose
document.getElementsByTagName("tagName")	Returns a collection (array) of Items matching the tag name.
document.getElementsByClassName("className")	Returns a collection (array) of Items matching the class name.
document.getElementById("idName")	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

DOM – Events

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

DOM *Events* are sent to notify code when things happen, such as when a *button* is clicked. Each event is represented by an object which is based on the Event interface and may have additional custom fields and/or functions used to get additional information about what happened.

The two most prevalent events are mouse 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 a event happen is called an ***event listener***, and the block of code that runs when the event fires is called an ***event handler***.

guessSubmit.addEventListener('click', checkGuess);

guessSubmit holds all the data from an element. It uses a built-in 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 (***checkGuess()***) Because checkGuess is a callback, you don’t need to use the ( ).

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 runs it if so. 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.

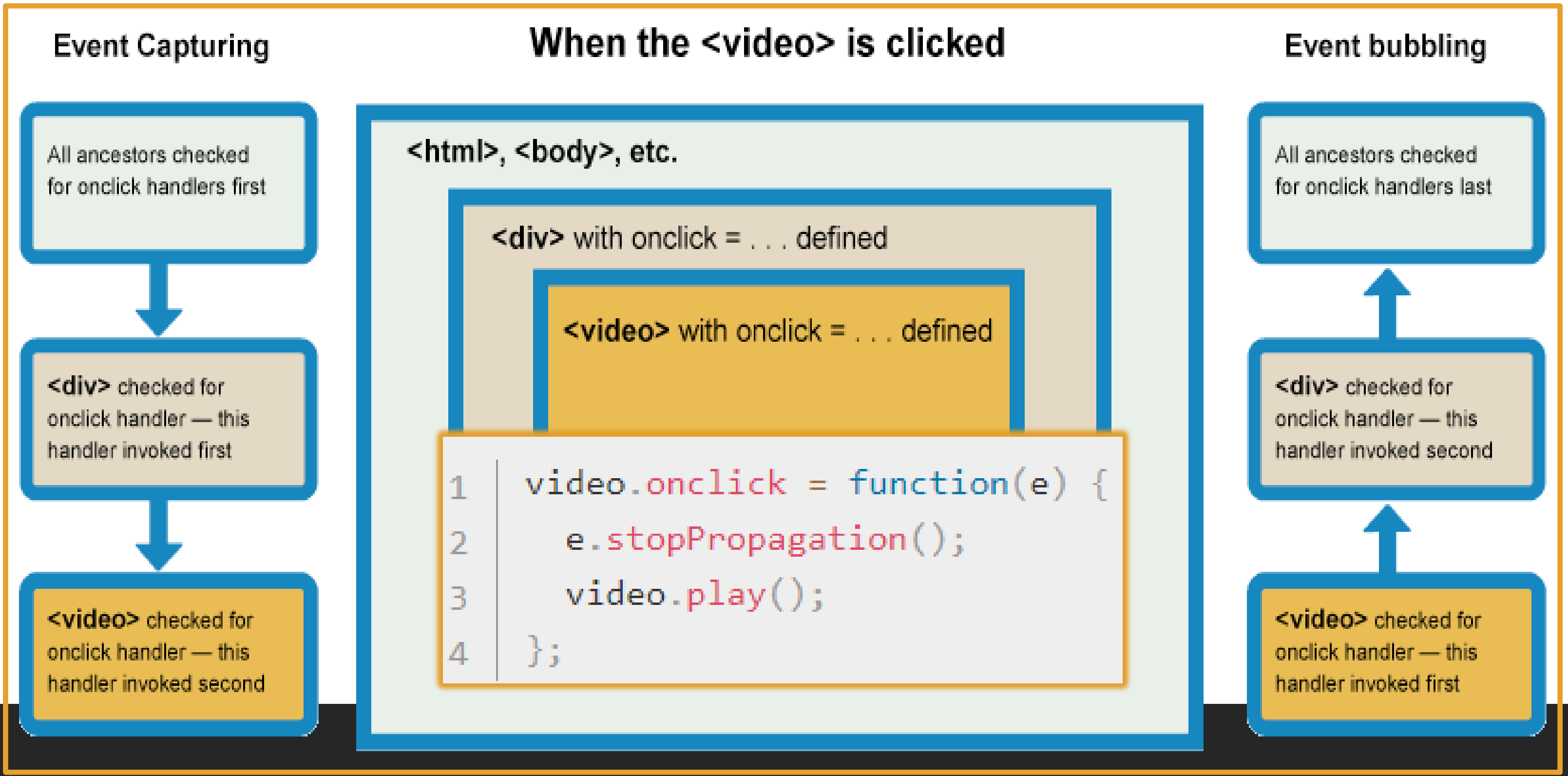
Bubbling and Capture

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 you click a video, the click event bubbles from the ***<video>*** element outwards to its parent ***<div>*** element, then to the ***<html>*** element. Along the way, if any of these elements has an '***on-click***' event handler, they will fire, too.

.stopPropagation() is provided to stop further propagation.


Event Capturing
All ancestors checked for onclick handlers first
<div> checked for onclick handler — this handler invoked first
<video> checked for onclick handler — this handler invoked second
When the <video> is clicked
<html>, <body>, etc.
<div> with onclick = . . . defined
<video> with onclick = . . . defined
Event bubbling
All ancestors checked for onclick handlers last
<div> checked for onclick handler — this handler invoked second
<video> checked for onclick handler — this handler invoked first

```
video.onclick = function(e) {
  e.stopPropagation();
  video.play();
};
```

Bubbling and Capture

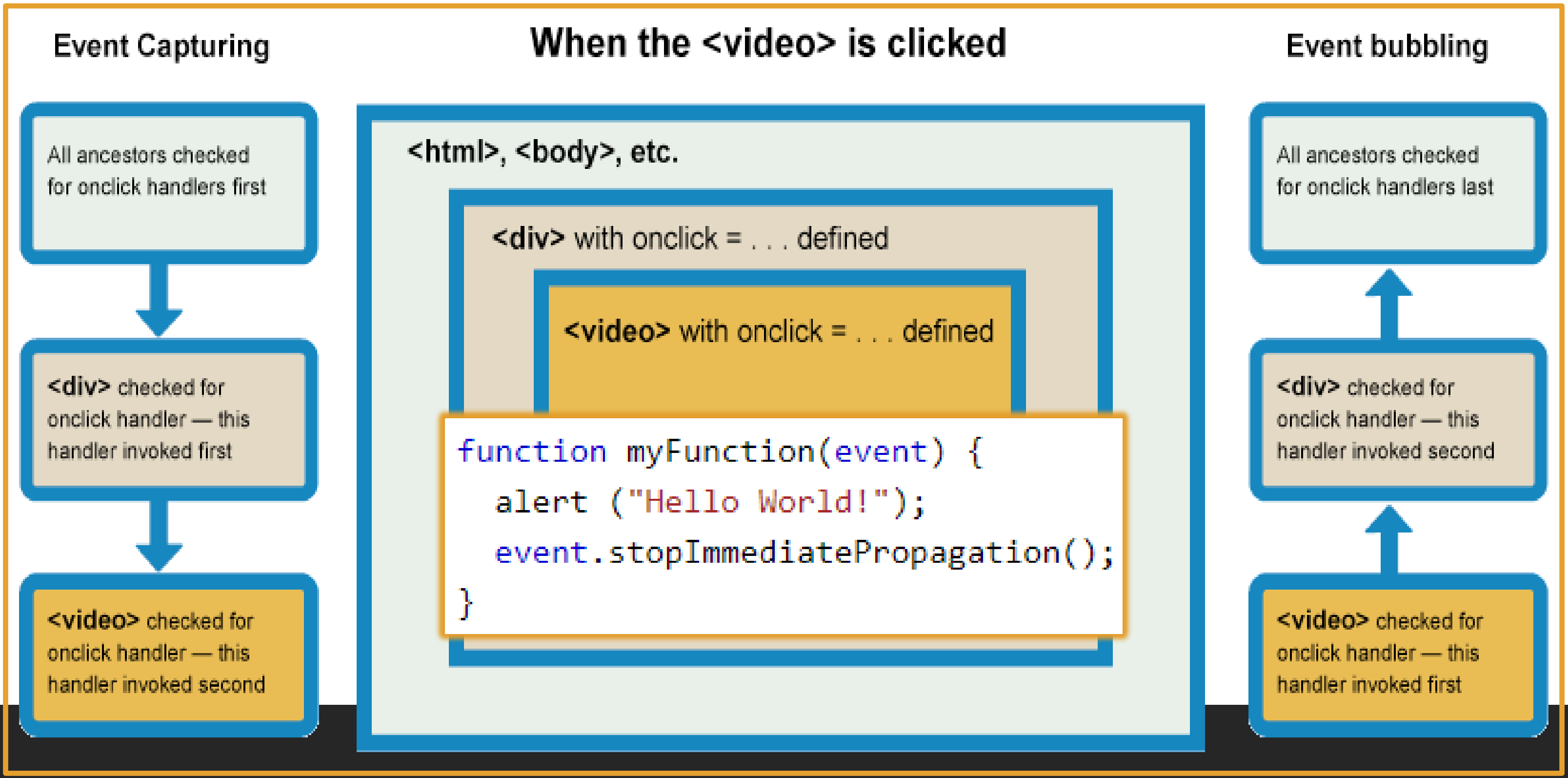
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Examples#Example_5:_Event_Propagation
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events

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

The ***stopImmediatePropagation()*** method prevents other listeners of the same event from being called.

If several listeners are attached to the same element for the same event type, they are called in the order in which they were added.

If ***stopImmediatePropagation()*** is invoked, no remaining listeners will be called.


Event Capturing
When the <video> is clicked
Event bubbling
All ancestors checked for onclick handlers first
<div> checked for onclick handler — this handler invoked first
<video> checked for onclick handler — this handler invoked second
<html>, <body>, etc.
<div> with onclick = . . . defined
<video> with onclick = . . . defined
All ancestors checked for onclick handlers last
<div> checked for onclick handler — this handler invoked second
<video> checked for onclick handler — this handler invoked first

```
function myFunction(event) {
  alert ("Hello World!");
  event.stopImmediatePropagation();
}
```

Walking the DOM - Basics

https://javascript.info/dom-navigation

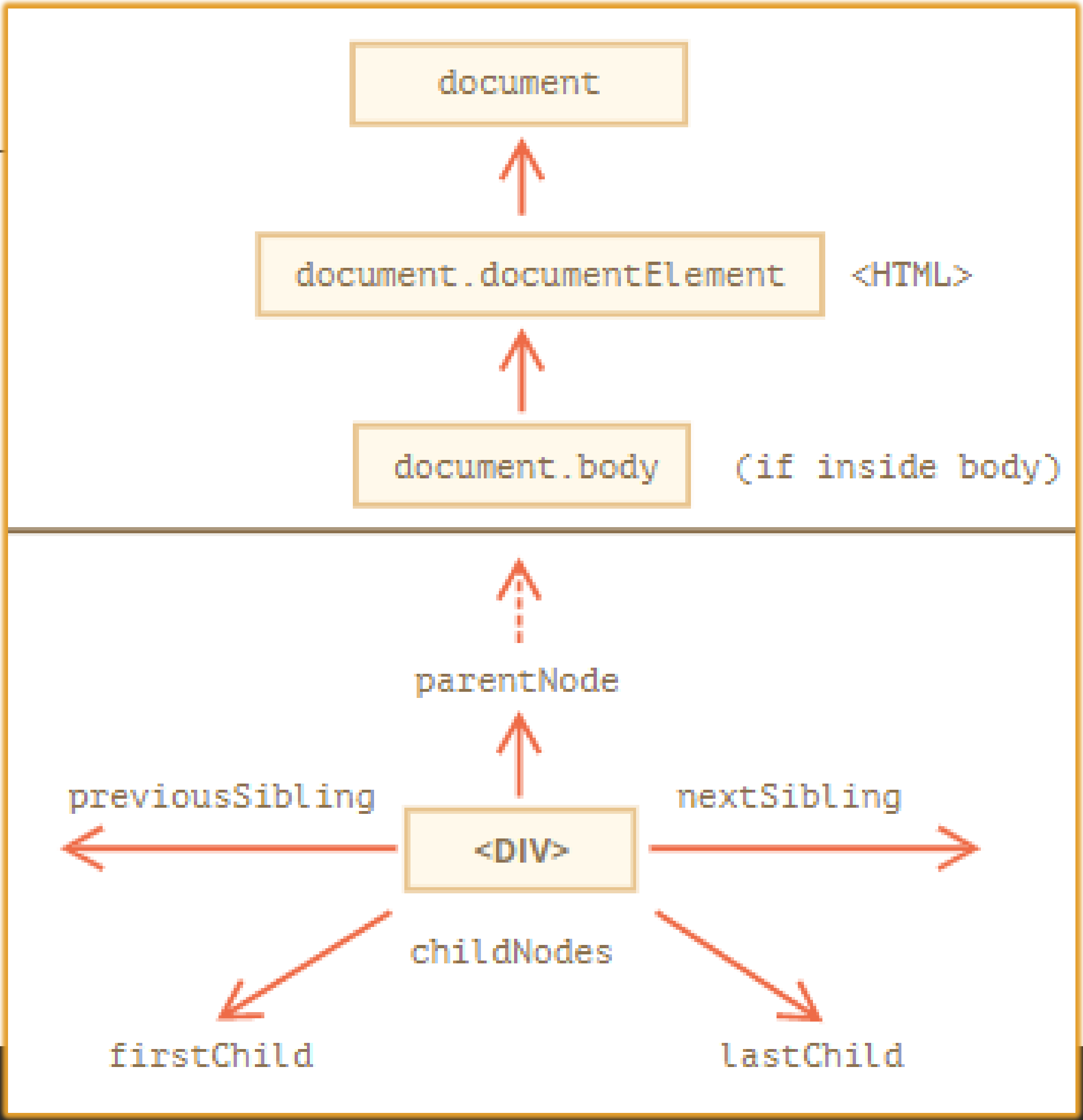
We can do anything with elements and their contents after reaching the correct DOM object.

The topmost tree nodes are available directly as document properties:

<html> = document.documentElement

<body> = document.body

<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 children of *<html>* element.

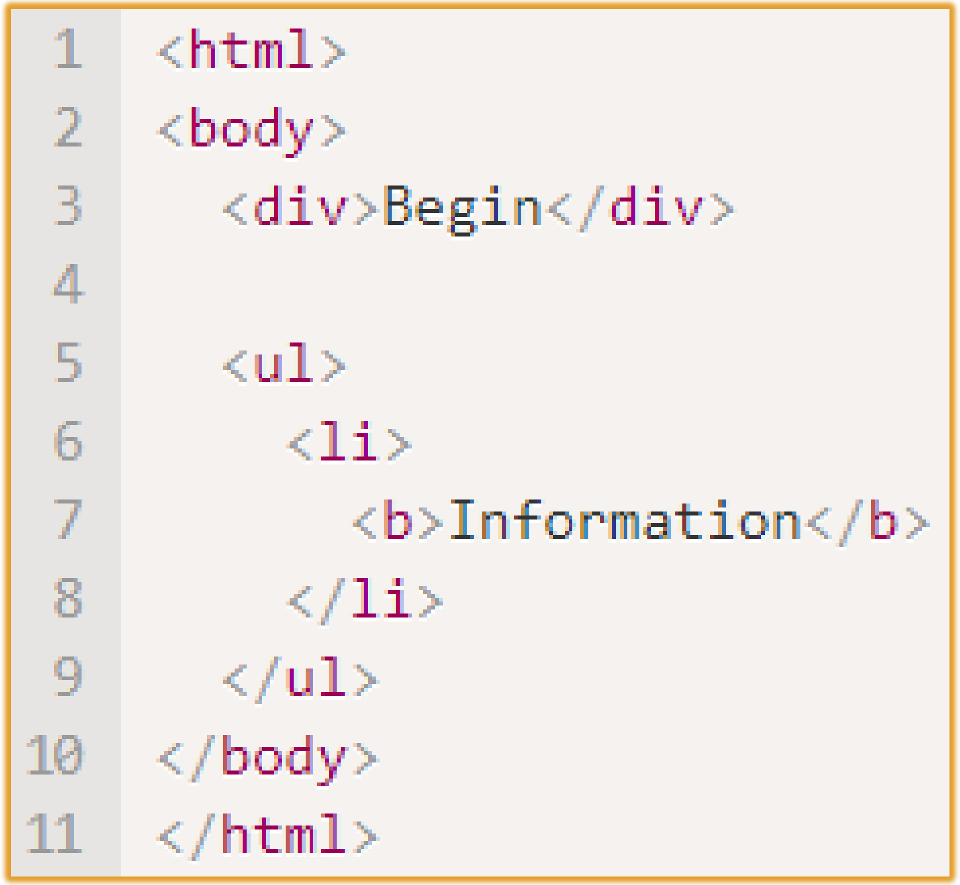
Siblings - nodes that are children of the same parent.

Descendants – all elements that are nested in the given one, including children, their children, etc.

Here *<body>* has children *<div>* and *<ul>.*

<div> and *<ul>* are siblings

Descendants of *<body>* are not only direct children *<div>, <ul>* but also more deeply nested elements, such as *<li>* (a child of *<ul>*) and *<b>* (a child of *<li>*) – the entire subtree.



```html
<html>
<body>
  <div>Begin</div>

  <ul>
    <li>
      <b>Information</b>
    </li>
  </ul>
</body>
</html>
```

Walking the nodes of the DOM

https://javascript.info/dom-navigation

Method	Explanation/Example
document.body.childNodes[i]	.childNodes lists all child nodes in a (read-only) collection, 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.

Walking the elements of the DOM

https://javascript.info/dom-navigation

Method	Explanation/Example
.firstElementChild	Gives access to the first child.
.lastElementChild	Gives access to the last child.
.nextElementSibling	Access the following or “right” sibling element going down the page.
.previousElementSibling	Access the prior or “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 all children elements.

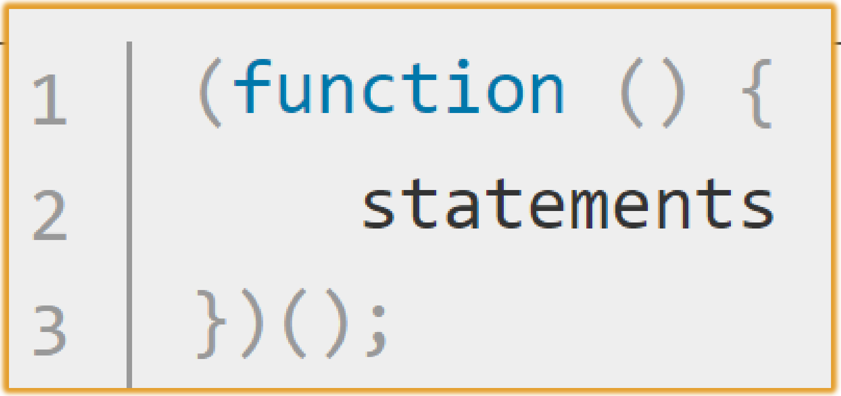
IIFE
Immediately Invoked Function Expression

https://developer.mozilla.org/en-US/docs/Glossary/IIFE

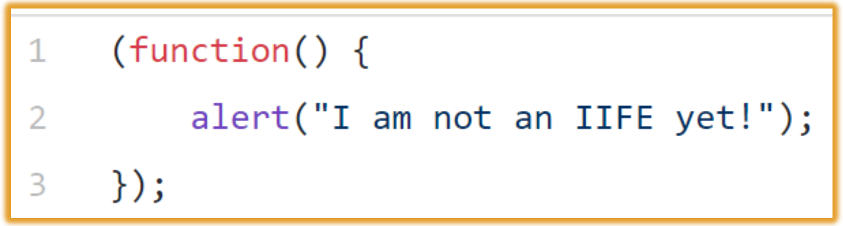
An **IIFE *(Immediately Invoked Function Expression)*** (pronounced “iffy”) is a JavaScript function that runs as soon as it is defined. It’s also known as a Self-Executing Anonymous Function

IIFE functions contain two major parts:

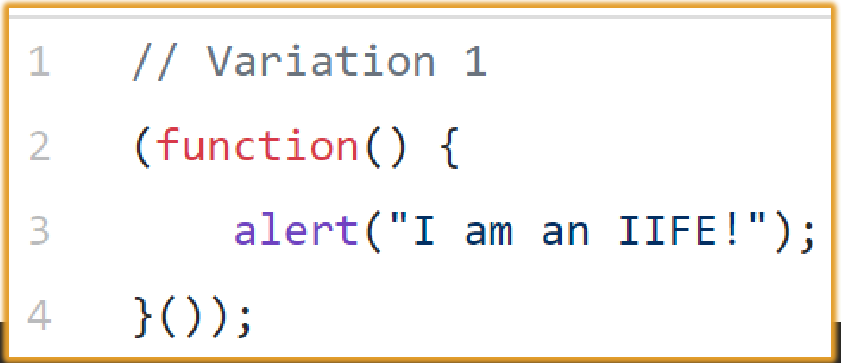
- The first is the anonymous function with lexical scope enclosed within the Grouping Operator (). This prevents accessing variables within the IIFE idiom as well as polluting the global scope.
- The second part creates the immediately invoked function expression () through which the JavaScript engine will directly interpret the function.



```
(function () {
    statements
})();
```



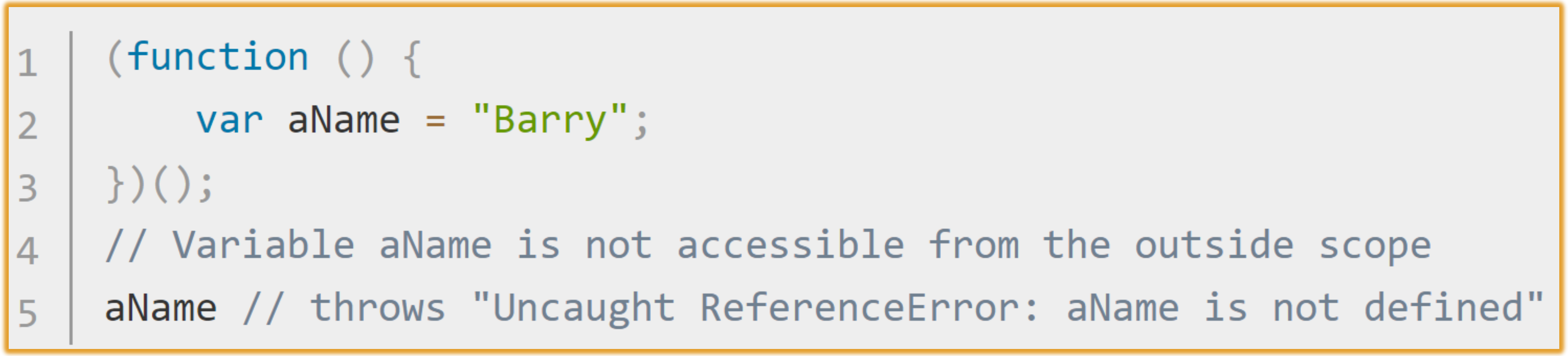
```
(function() {
    alert("I am not an IIFE yet!");
});
```



```
// Variation 1
(function() {
    alert("I am an IIFE!");
}());
```

IIFE
Immediately Invoked Function Expression

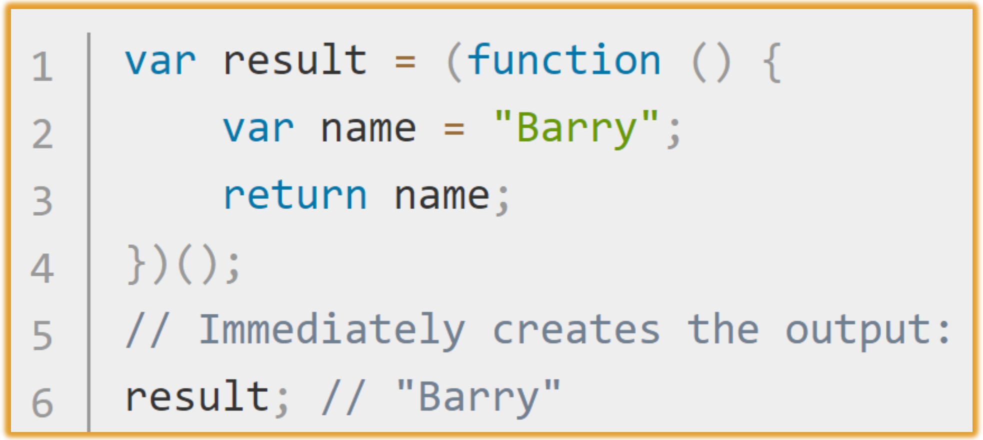
https://developer.mozilla.org/en-US/docs/Glossary/IIFE



```
(function () {
    var aName = "Barry";
})();
// Variable aName is not accessible from the outside scope
aName // throws "Uncaught ReferenceError: aName is not defined"
```

Any variable declared within the expression can not be accessed from outside it.

Assigning an *IIFE* to a variable stores the function's return value, not the function definition itself.



```
var result = (function () {
    var name = "Barry";
    return name;
})();
// Immediately creates the output:
result; // "Barry"
```

DOM Events Order

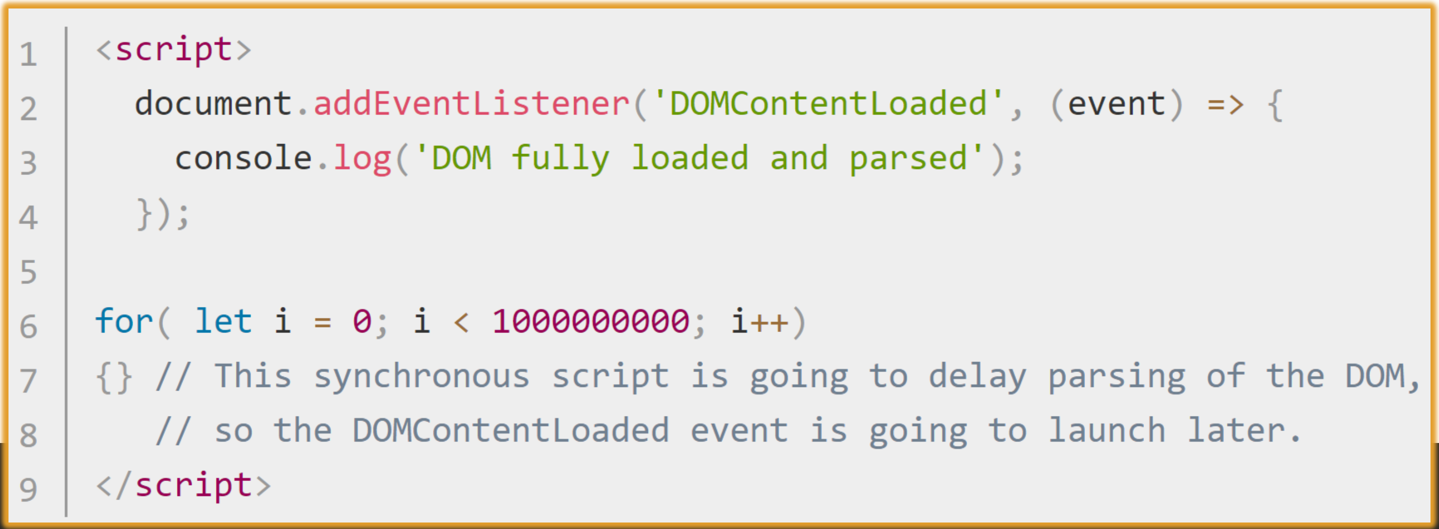
https://developer.mozilla.org/en-US/docs/Web/API/Document/DOMContentLoaded_event

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

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



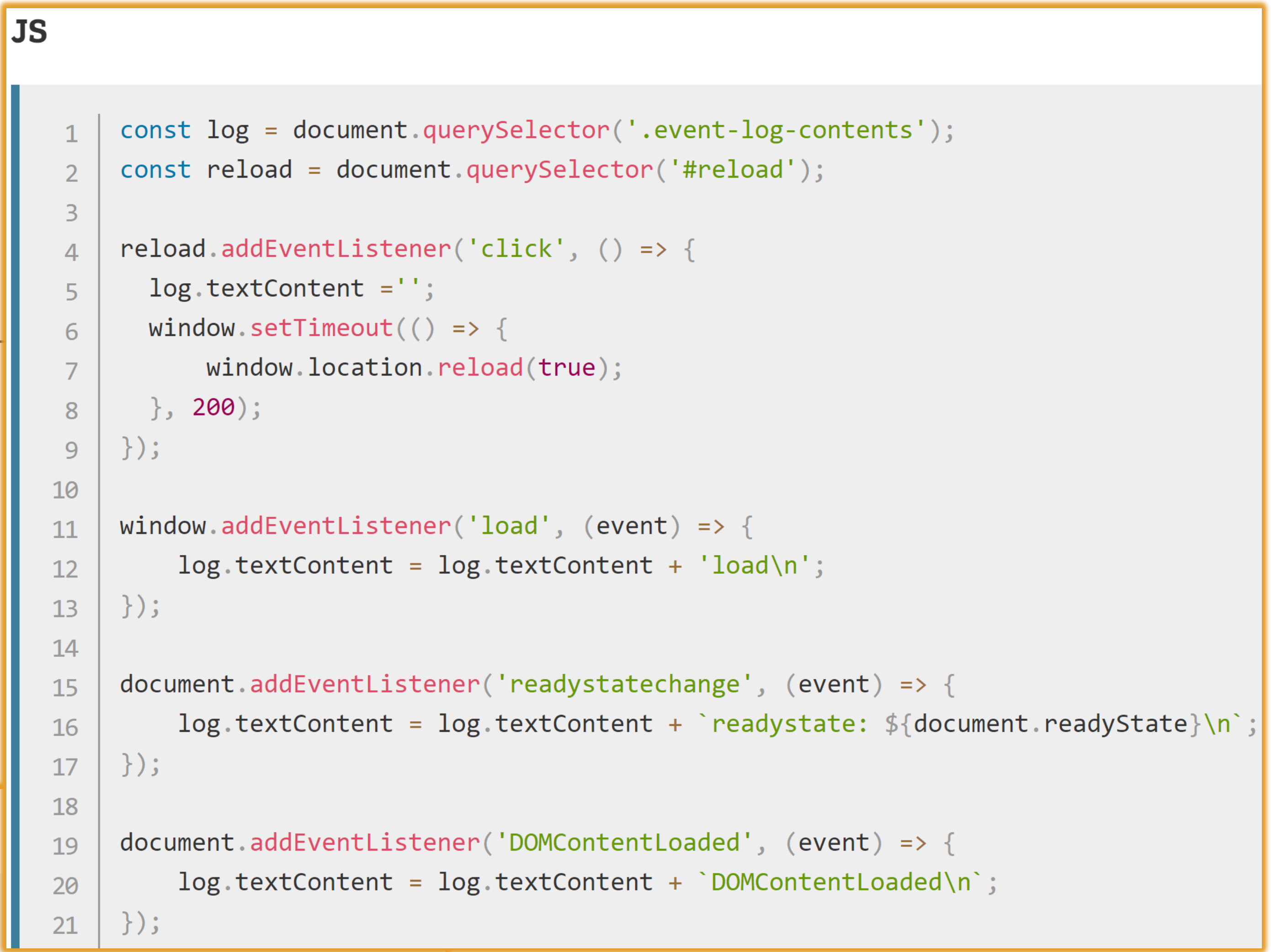
```
<script>
  document.addEventListener('DOMContentLoaded', (event) => {
    console.log('DOM fully loaded and parsed');
  });

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

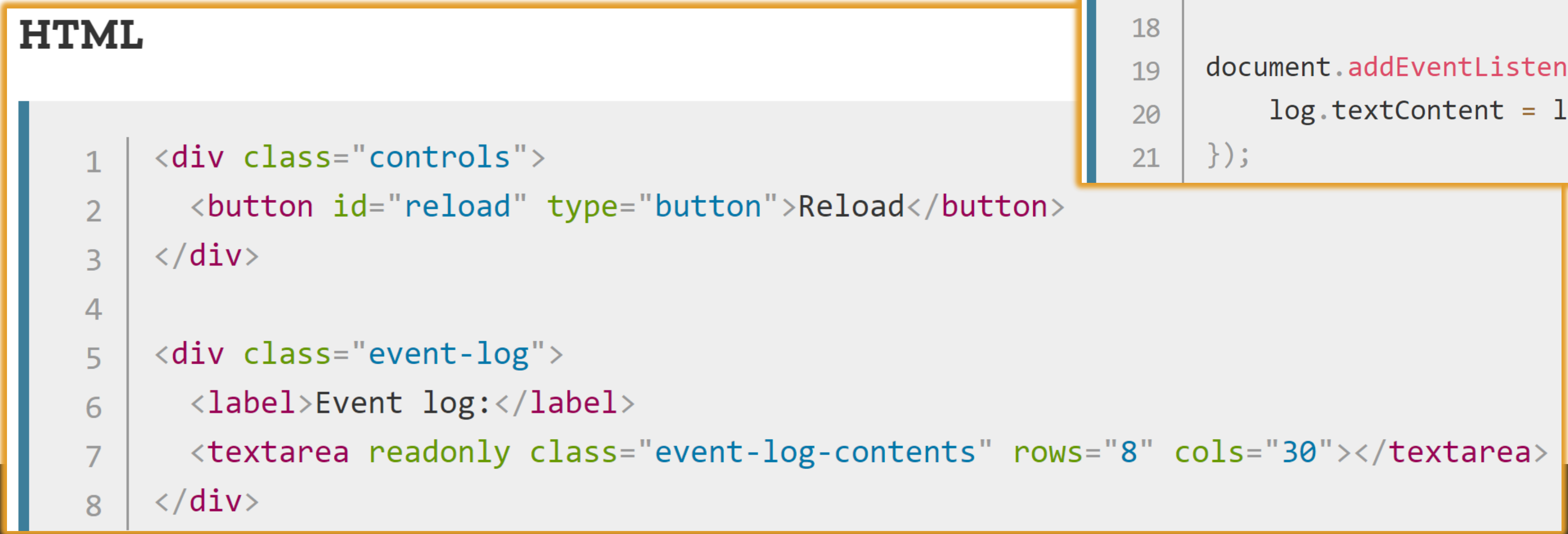
```
const log = document.querySelector('.event-log-contents');
const reload = document.querySelector('#reload');

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`;
});
```

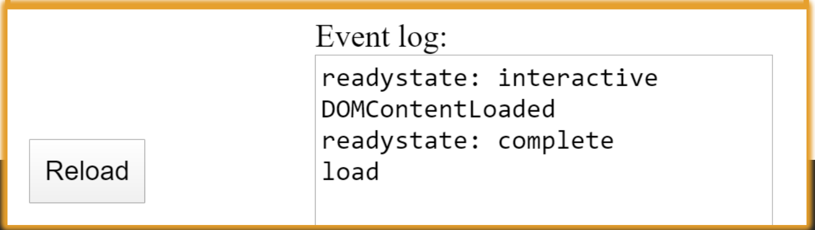


HTML

```
<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



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.