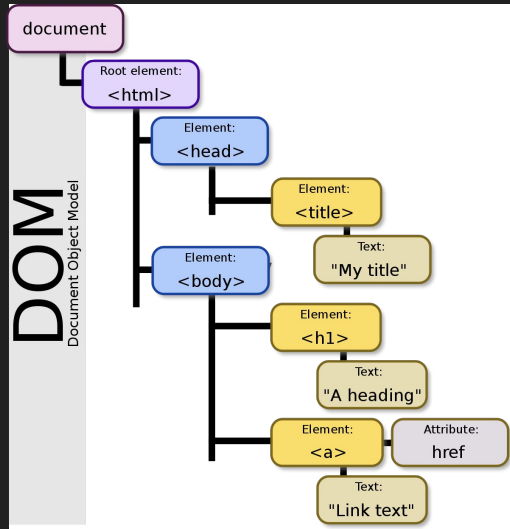


Document Object Model (DOM)

Document Object Model (DOM)



When a browser loads a web page, it creates a model of that page.

This is called a “DOM tree” and it is stored in the browser’s memory.

Every element, attribute, and piece of text in the HTML is represented by its own “DOM node.”

Types of DOM Nodes

There are four main types of nodes:

- The Document node, which represents the entire page
- Element nodes, which represent individual HTML tags
- Attribute nodes, which represent attributes of HTML tags, such as a class
- Text nodes, which represents the text within an element, such as the content of a `<p>` tag

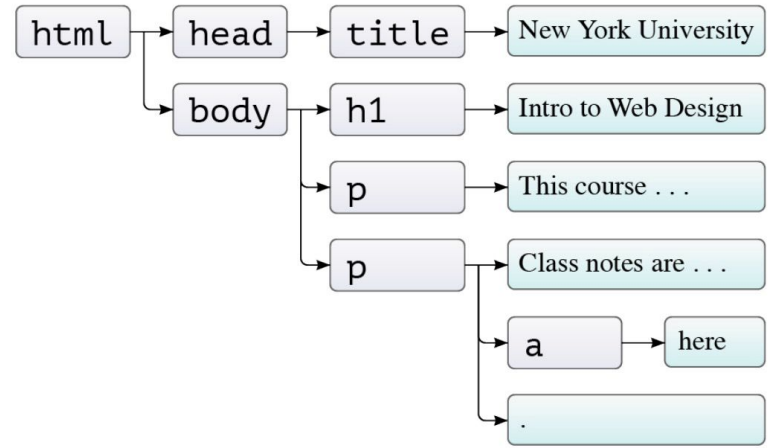
We talk about the relationship between element nodes as “parents,” “children,” and “siblings.”

```
<html>
  <head>
    <title>New York University</title>
  </head>

  <body>
    <h1>Intro to Web Design</h1>

    <p>In this course you will learn how to build websites.</p>

    <p>Class notes are available
    <a href="notes.html">here</a>.<p>
  </body>
</html>
```



DOM Queries

- JavaScript methods that find elements in the DOM tree are called “DOM queries.”
- DOM queries may return one element, or they may return a “node list.”
- Which DOM query you use depends on what you want to do and the scope of browser support required.

DOM Queries

Methods that return a single element node:

```
.getElementById()
```

```
.querySelector()
```

Methods that return one or more elements as a node list

```
.getElementsByClassName()
```

```
.getElementsByTagName()
```

```
.querySelectorAll()
```

What can we change?

.textContent

sets or returns the text content of the specified node

```
let element = document.getElementById("changeMe")
element.textContent = "I have changed!";
```

.innerHTML

sets or returns the HTML content (inner HTML) of an element

```
let element = document.getElementById("changeMe")
element.innerHTML = "<a href='#'>I have changed!</a>";
```

.style.CSSPropertyName

sets or returns the value of a given CSS property

```
let element = document.getElementById("changeMe")
element.style.color = "red";
```

DOM Events

DOM Events

As you navigate the web, your browser registers different types of events.

Common events include:

- Clicking or tapping on a link
- Hovering or swiping over an element
- Resizing the browser window
- A web page loading

JavaScript can be used to respond to the multitude of events that occur within the DOM.

Keyboard Events

- keydown
- keyup
- keypress

Mouse Events

- click
- dblclick
- mousedown
- mouseup
- mouseover
- mouseout

Focus Events

- focus
- blur

Touch Events

- touchstart
- touchmove
- touchend
- touchcancel

Pointer Events

- pointerover
- pointerenter
- pointerdown
- pointermove
- pointerup
- pointercancel
- pointerout
- pointerleave
- gotpointercapture
- lostpointercapture

User Interface (UI) Events

- load
- unload
- error
- resize
- scroll

Mutation Events

- DOMSubtreeModified
- DOMNodeInserted
- DOMNodeRemoved
- DOMNodeInsertedIntoDocument
- DOMNodeRemovedFromDocument

Form Events

- input
- change
- submit
- reset
- cut
- copy
- paste
- select

Binding

Specifying which event will trigger the response is also known as “binding”.

There are three different ways to bind an event to an element:

- HTML event handler
- DOM event handler
- DOM Event listener *

** preferred method*

HTML Event Handler

```
<button onclick="myFunction()">Click me</button>
```

Downsides: Mixing HTML markup with JavaScript can make the code less maintainable and harder to debug. It's generally considered a best practice to separate HTML and JavaScript code.

DOM Event Handler

```
let btn = document.querySelector("button");  
btn.onclick = myFunctionName;
```

Downsides: Assigning multiple event handlers to the same event on the same element will overwrite the previous assignment.

DOM Event Listener

```
let btn = document.querySelector('button');  
btn.addEventListener('click', myFunctionName);
```

Most recommended! They're most flexible and powerful. They allow you to attach multiple event handlers to a single event on a DOM element without overwriting existing ones.

Event Handling

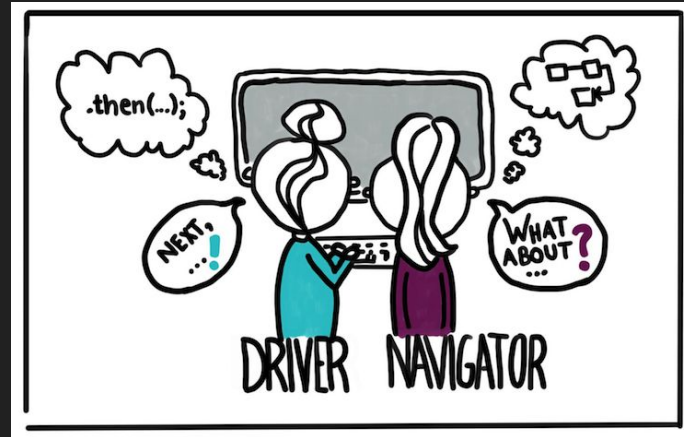
1. Select an element for the script to respond to
2. Specify which event will trigger the response
3. Run code specific to that event

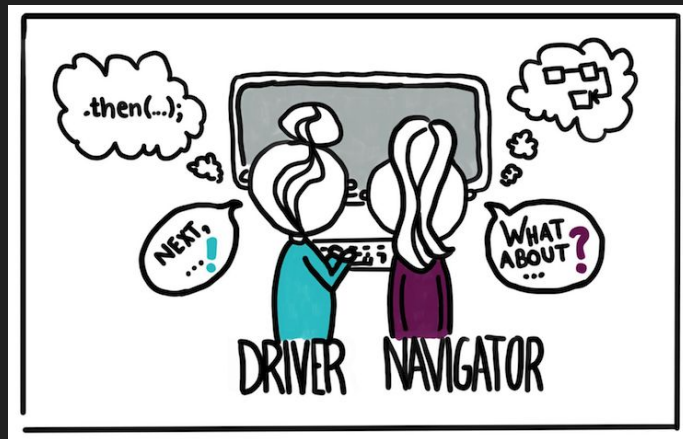
```
// Step 1: Select an element for the script to respond to
const button = document.getElementById('myButton');

// Step 2: Specify which event will trigger the response
button.addEventListener('click', myFunction);

// Step 3: Run code specific to that event
function myFunction() {
  alert('Button clicked! This is the response to the click event.');
```

Pair Programming





The **Driver** is the person at the wheel, i.e. the keyboard.

- focused on completing the tiny goal at hand, ignoring larger issues for the moment.
- A driver should always talk through what she is doing while doing it.

The **Navigator** is in the observer position, while the driver is typing.

- reviews the code on-the-go, gives directions and shares thoughts.
- The navigator also has an eye on the larger issues, bugs, and makes notes of potential next steps or obstacles.