JavaScript in the Browser

Handling web document structure

Credit

- This slide is copied and modified from Fulvio Corno, Luigi De Russis @

Politecnico di Torino

Goal

- Loading JavaScript in the browser
- Browser object model
- Document object model
- DOM Manipulation
- DOM Styling
- Event Handling
- Forms

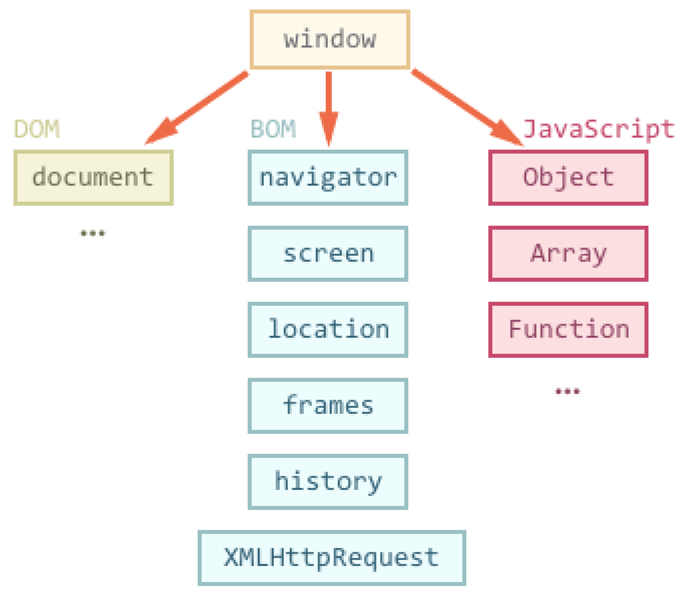
Mozilla Developer Network: The Script element
https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script

JS in the browser

LOADING JS IN THE BROWSER

Where Does The Code Run?

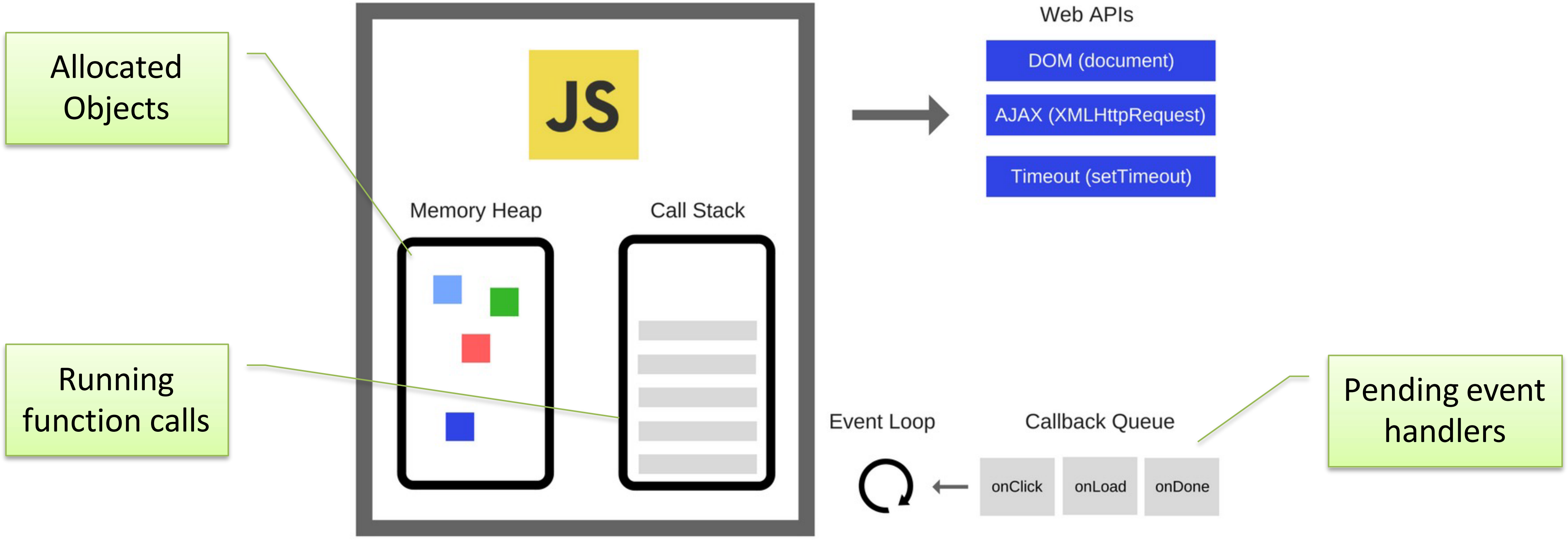
- Loaded and run in the browser *sandbox*
- Attached to a *global context:* the `window` object
- May access only a limited set of APIs
  - JS Standard Library
  - Browser objects (BOM)
  - Document objects (DOM)
- Multiple <`script`>s are independent
  - They all access the same global scope
  - To have structured collaboration, *modules* are needed



Events and Event Loop

- Most phases of processing and interaction with a web document will generate Asynchronous *Events* (100's of different types)
- Generated events may be handled by:
  - Pre-defined behaviors (by the browser)
  - User-defined *event handlers* (in your JS)
  - Or just ignored, if no event handler is defined
- But JavaScript is single-threaded
  - Event handling is *synchronous* and is based on an *event loop*
  - Event handlers are queued on a *Message Queue*
  - The Message Queue is polled when the main thread is idle

Execution Environment



Event Loop

- During code execution you may
  - Call functions ➔ the function call is pushed to the call stack
  - Schedule events ➔ the call to the event handler is put in the Message Queue
    - Events may be scheduled also by external events (user actions, I/O, network, timers, …)
- At any step, the JS interpreter:
  - If the call stack is not empty, pop the top of the call stack and executes it
  - If the call stack is empty, pick the head of the Message Queue and executes it
- A function call / event handler is **never** interrupted
  - Avoid blocking code!

https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop

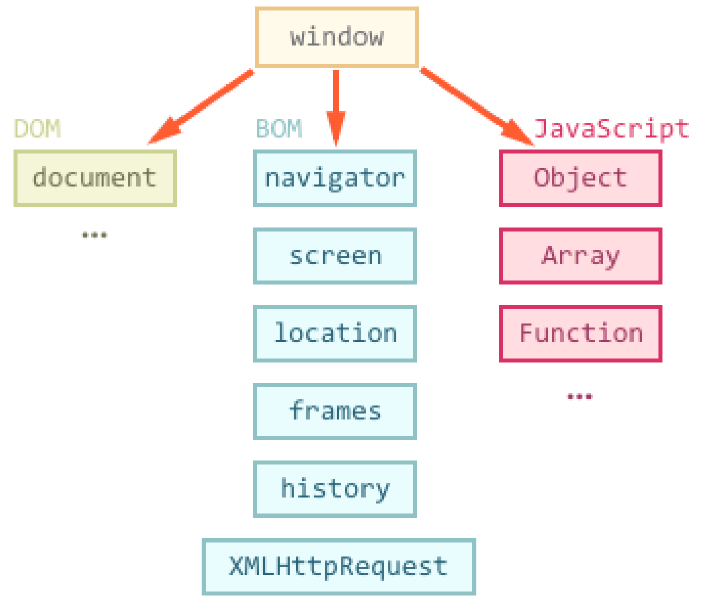
https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/#what-is-the-event-loop

JS in the browser

BROWSER OBJECT MODEL

Browser Main Objects

- `window` represents the window that contains the DOM document
  - allows to interact with the browser via the BOM: browser object model (not standardized)
  - global object, contains all JS global variables
    - can be omitted when writing JS code in the page
- `document`
  - represents the DOM tree loaded in a window
  - accessible via a `window` property: `window.document`



https://medium.com/@fknussel/dom-bom-revisited-cf6124e2a816

Browser Object Model

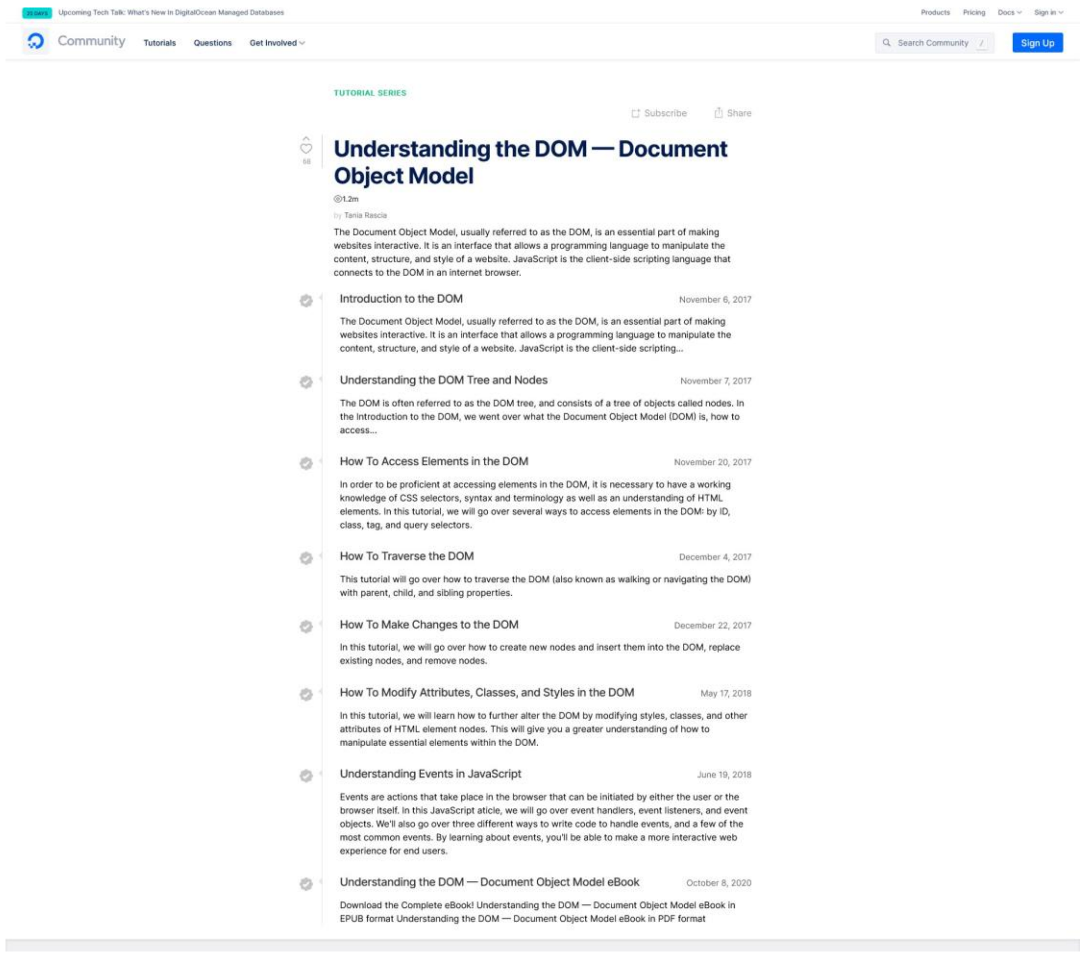
- `window` properties
  - `console`: browser debug console (visible via developer tools)
  - `document`: the document object
  - `history`: allows access to History API (history of URLs)
  - `location`: allows access to Location API (current URL, protocol, etc.). Read/write property, i.e., can be set to load a new page
  - `localStorage` and `sessionStorage`: allows access to the two objects via the Web Storage API, to store (small) info locally in the browser

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

Essential part of making websites interactive

DOCUMENT OBJECT MODEL

Suggested Reading



Understanding the DOM — Document Object Model

1.2m

by Tania Rascia

The Document Object Model, usually referred to as the DOM, is an essential part of making websites interactive. It is an interface that allows a programming language to manipulate the content, structure, and style of a website. JavaScript is the client-side scripting language that connects to the DOM in an internet browser.

Introduction to the DOM — November 6, 2017

The Document Object Model, usually referred to as the DOM, is an essential part of making websites interactive. It is an interface that allows a programming language to manipulate the content, structure, and style of a website. JavaScript is the client-side scripting...

Understanding the DOM Tree and Nodes — November 7, 2017

The DOM is often referred to as the DOM tree, and consists of a tree of objects called nodes. In the Introduction to the DOM, we went over what the Document Object Model (DOM) is, how to access...

How To Access Elements in the DOM — November 20, 2017

In order to be proficient at accessing elements in the DOM, it is necessary to have a working knowledge of CSS selectors, syntax and terminology as well as an understanding of HTML elements. In this tutorial, we will go over several ways to access elements in the DOM: by ID, class, tag, and query selectors.

How To Traverse the DOM — December 4, 2017

This tutorial will go over how to traverse the DOM (also known as walking or navigating the DOM) with parent, child, and sibling properties.

How To Make Changes to the DOM — December 22, 2017

In this tutorial, we will go over how to create new nodes and insert them into the DOM, replace existing nodes, and remove nodes.

How To Modify Attributes, Classes, and Styles in the DOM — May 17, 2018

In this tutorial, we will learn how to further alter the DOM by modifying styles, classes, and other attributes of HTML element nodes. This will give you a greater understanding of how to manipulate essential elements within the DOM.

Understanding Events in JavaScript — June 19, 2018

Events are actions that take place in the browser that can be initiated by either the user or the browser itself. In this JavaScript aticle, we will go over event handlers, event listeners, and event objects. We'll also go over three different ways to write code to handle events, and a few of the most common events. By learning about events, you'll be able to make a more interactive web experience for end users.

Understanding the DOM — Document Object Model eBook — October 8, 2020

Download the Complete eBook! Understanding the DOM — Document Object Model eBook in EPUB format Understanding the DOM — Document Object Model eBook in PDF format

- https://www.digitalocean.com/community/tutorial_series/understanding-the-dom-document-object-model
- Complete and detailed tutorial
- Here, we will *focus* on the core concepts and techniques

What is DOM?

- To parsing the style and structure of the HTML and CSS, the browser creates a representation of the document
- The document object is a built-in object that has many **`properties`** and **`methods`** that we can use to access and modify websites
- DOM is often referred to as the DOM tree, and consists of a tree of objects called nodes
- Simplest way to access an element with JavaScript is by the **`id`** attribute

```
...
<body>
  <h1>Document Object Model</h1>
  <a id="nav" href="index.html">Home</a>
</body>
...
```

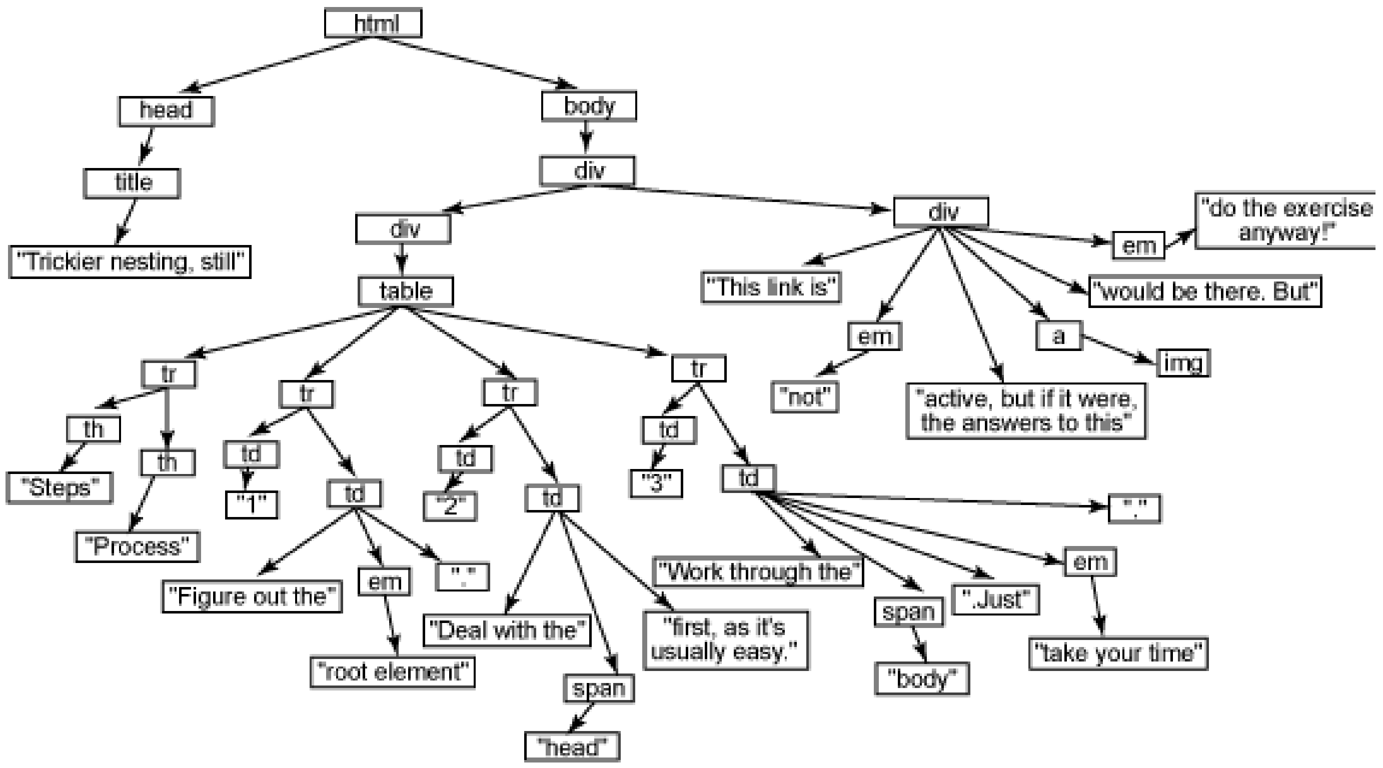
```
document.getElementById('nav’);
```

Output

```
<a id="nav" href="index.html">Home</a>
```

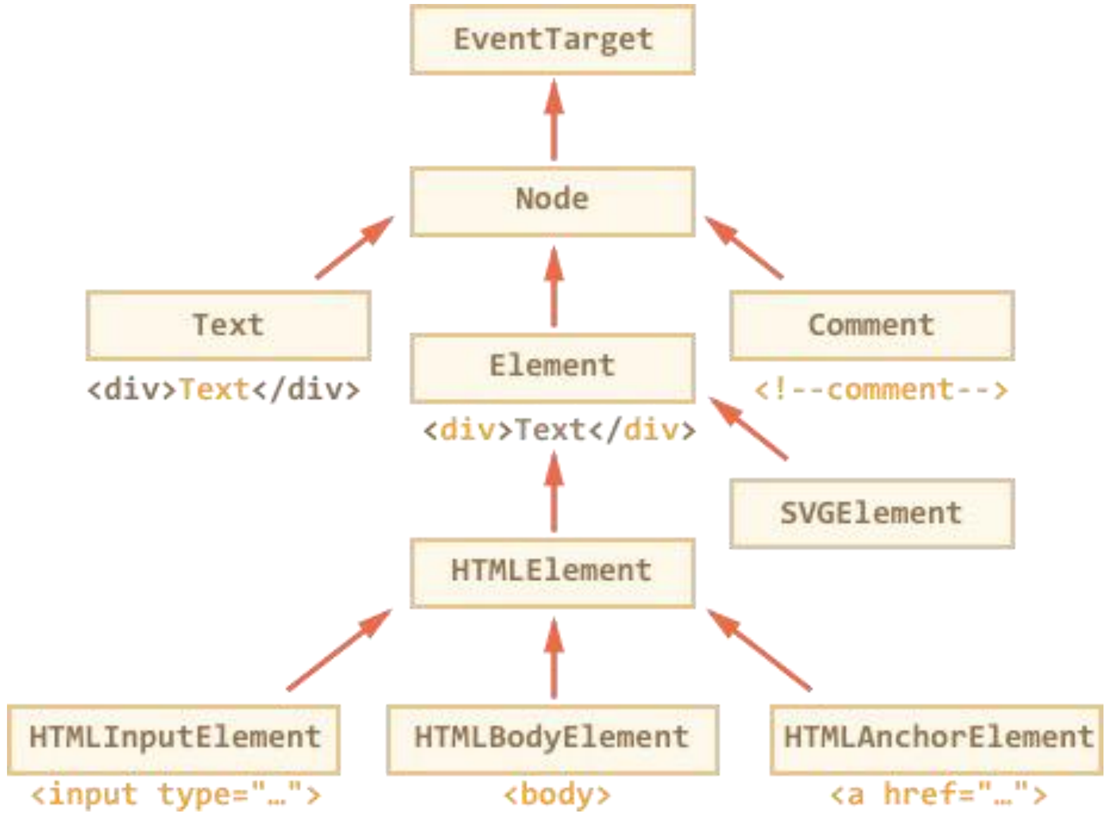
DOM

- Browser's internal representation of a web page
  - Obtained through parsing HTML
- Browsers expose an API that you can use interact with the DOM
  - Access the page metadata and headers
  - Inspect the page structure
  - Edit any node in the page
  - Change any node attribute
  - Create/delete nodes in the page
  - Edit the CSS styling and classes
  - Attach or remove event listeners



Types Of Nodes (classes)

- All items in the DOM are defined as nodes
- Document
  - The document Node, the root of the tree
- 3 main types of nodes
  - Element (1), Text (3) and Comment (8)
- Element: an HTML tag
- Text: text content of Element
- Comment: an HTML comment



Exercise

```html
<!DOCTYPE html>
<html>
    <head>
    <title>Learning About Nodes</title>
    </head>
    <body>
        <h1>An element node</h1>
        <!-- a comment node -->
        A text node.
   </body>
</html>
```

- **html** element node is the parent node
  - **head** and **body** are siblings, children of **html**
  - **body** contains three child nodes, which are all siblings
- Identifying Node Type
  - Every node in a document has a node type, which is accessed through the **nodeType** property
  - $0 indicate body

JS in the browser

DOM MANIPULATION

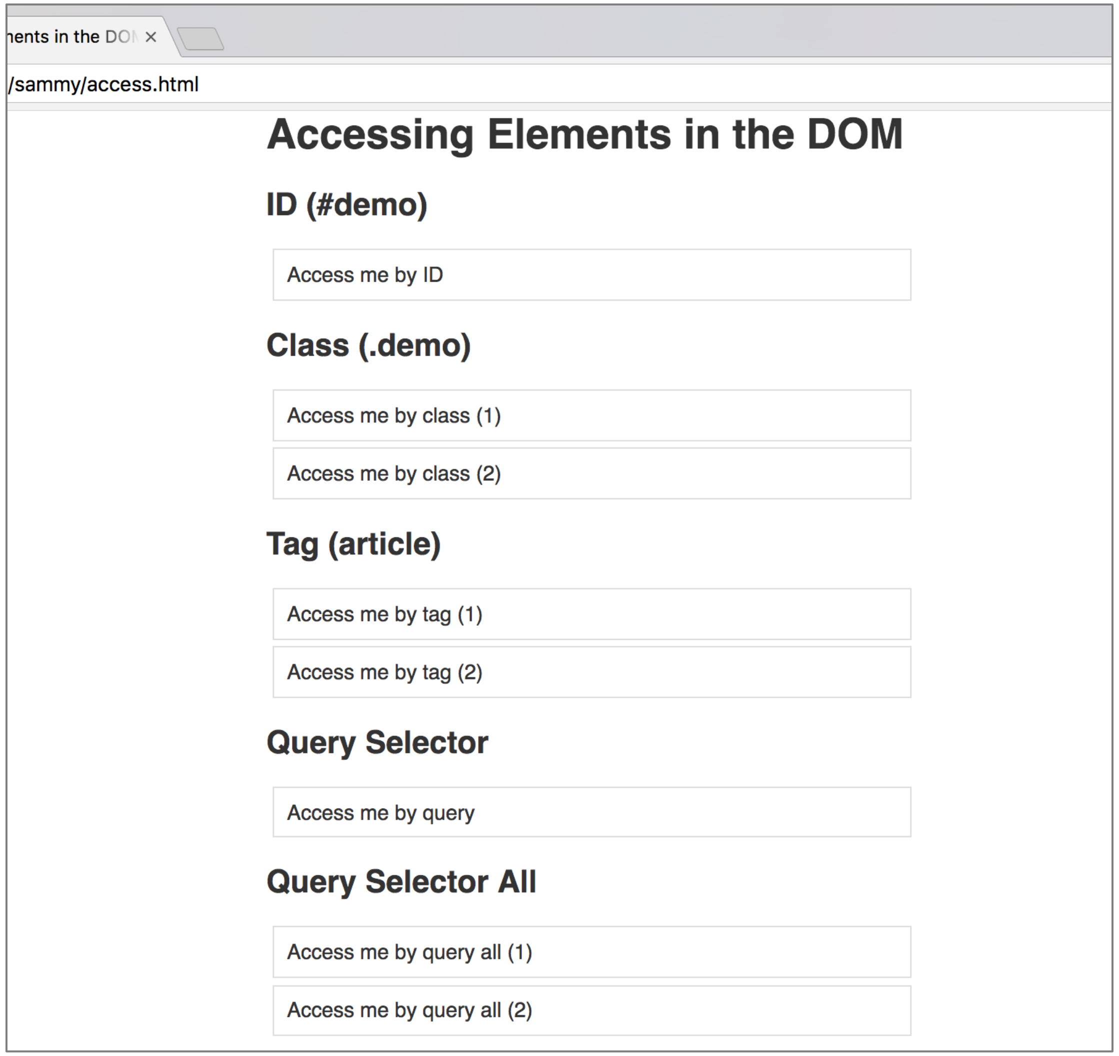
How To Access Elements in the DOM

Gets	**Selector Syntax**	**Method**
ID	#demo	**`getElementById()`**
Class	.demo	**`getElementsByClassName()`**
Tag	demo	**`getElementsByTagName()`**
Selector (single)		**`querySelector()`**
Selector (all)		**`querySelectorAll()`**

Finding DOM elements

- **`document.getElementById(value)`**
  - Returns the **`Node`** with the attribute **`id=value`**
- **`document.getElementsByTagName(value)`**
  - Returns the **`NodeList`** of all elements with the specified tag name (e.g., ‘**`div`**’)
- **`document.getElementsByClassName(value)`**
  - Returns the **`NodeList`** of all elements with attribute class=value (e.g., ‘col-8’)
- **`document.querySelector(css)`**
  - Returns the first **`Node`** element that matches the CSS selector syntax
- **`document.querySelectorAll(css)`**
  - Returns the **`NodeList`** of all elements that match the CSS selector syntax

```html
<style>
    html { font-family: sans-serif; color: #333; }
   body { max-width: 500px; margin: 0 auto; padding: 0 15px; }
    div, article { padding: 10px; margin: 5px; border: 1px
solid #dedede; }
</style>
</head>
<body>
    <h1>Accessing Elements in the DOM</h1>
    <h2>ID (#demo)</h2>
    <div id="demo">Access me by ID</div>
    <h2>Class (.demo)</h2>
    <div class="demo">Access me by class (1)</div>
    <div class="demo">Access me by class (2)</div>
    <h2>Tag (article)</h2>
    <article>Access me by tag (1)</article>
    <article>Access me by tag (2)</article>
    <h2>Query Selector</h2>
    <div id="demo-query">Access me by query</div>
    <h2>Query Selector All</h2>
  <div class="demo-query-all">Access me by query all (1)</div>
  <div class="demo-query-all">Access me by query all (2)</div>
</body>
```



Accessing Elements in the DOM

ID (#demo)

Access me by ID

Class (.demo)

Access me by class (1)

Access me by class (2)

Tag (article)

Access me by tag (1)

Access me by tag (2)

Query Selector

Access me by query

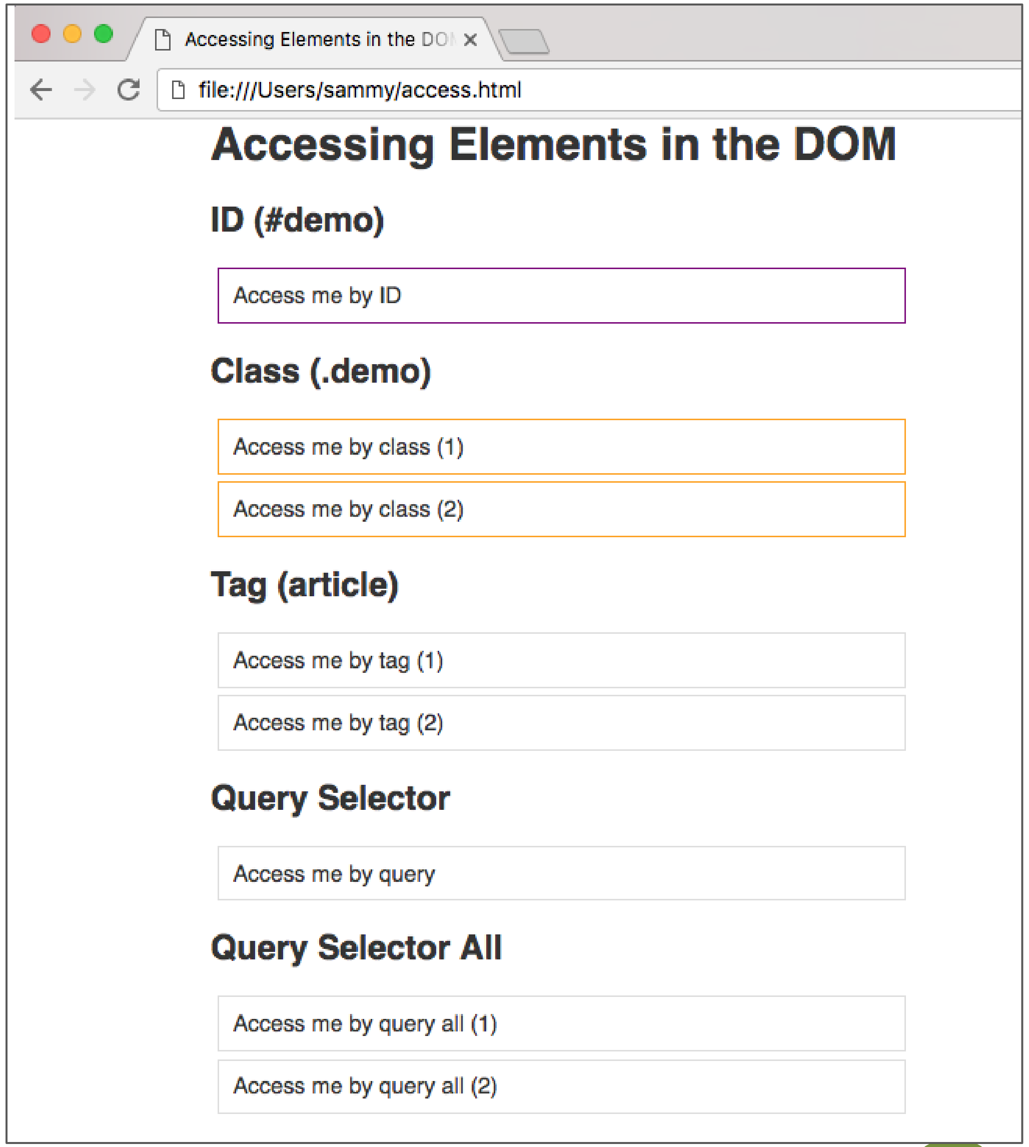
Query Selector All

Access me by query all (1)

Access me by query all (2)

Accessing Elements by Class

```
document.getElementsByClassName();

<div class="demo">Access me by class (1)</div>
<div class="demo">Access me by class (2)</div>

> const demoClass = document.getElementsByClassName('demo’);
> demoClass.style.border = '1px solid orange’;
Output
Uncaught TypeError: Cannot set property 'border' of undefined

> console.log(demoClass);
Output
(2) [div.demo, div.demo]

> demoClass[0].style.border = '1px solid orange’;

> for (i = 0; i < demoClass.length; i++) {
>  demoClass[i].style.border = '1px solid orange’;
> }
```



Accessing Elements in the DOM

ID (#demo)

Access me by ID

Class (.demo)

Access me by class (1)

Access me by class (2)

Tag (article)

Access me by tag (1)

Access me by tag (2)

Query Selector

Access me by query

Query Selector All

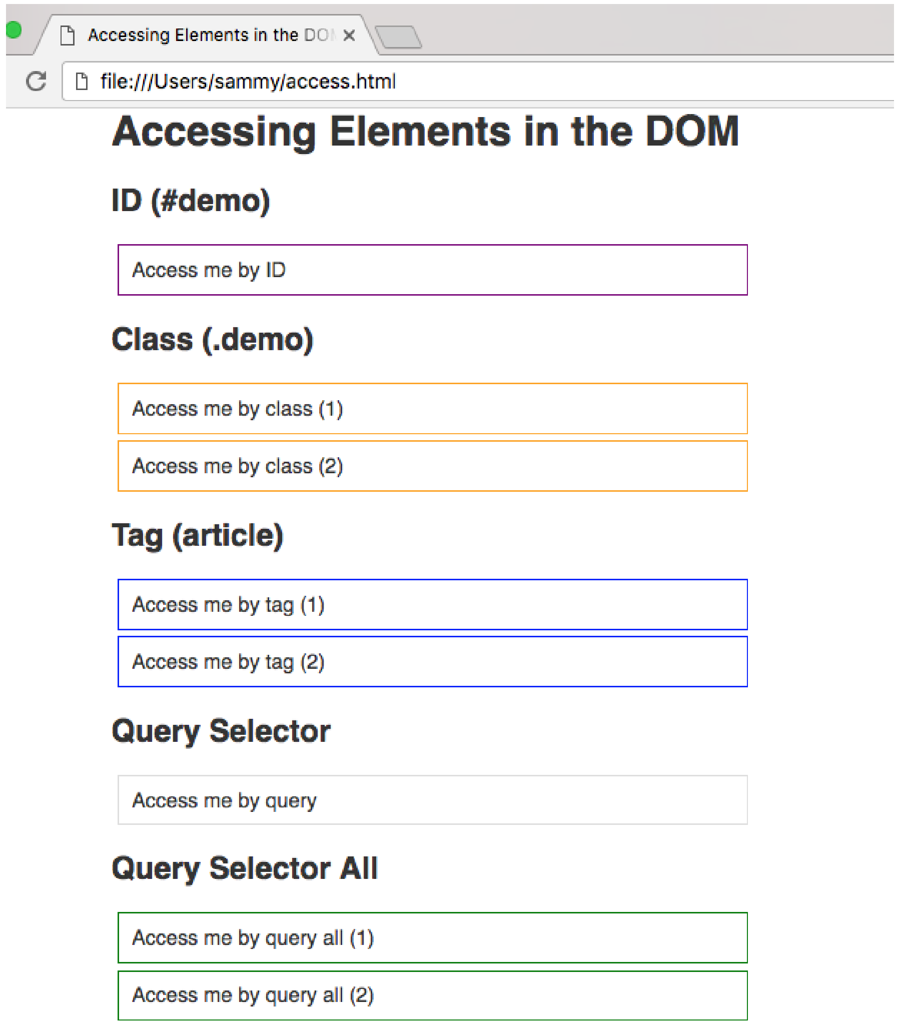
Access me by query all (1)

Access me by query all (2)

Query Selectors

- jQuery

```
$('#demo'); // returns the demo ID element in jQuery

document.querySelector();
document.querySelectorAll();

<div id="demo-query">Access me by query</div>
> const demoQuery = document.querySelector('#demo-query');
<div class="demo-query-all">Access me by query all (1)</div>
<div class="demo-query-all">Access me by query all (2)</div>
> const demoQueryAll = document.querySelectorAll('.demo-query-all');
> demoQueryAll.forEach(query => {
> query.style.border = '1px solid green';
> });
```


Accessing Elements in the DO ×

file:///Users/sammy/access.html

Accessing Elements in the DOM

ID (#demo)

Access me by ID

Class (.demo)

Access me by class (1)

Access me by class (2)

Tag (article)

Access me by tag (1)

Access me by tag (2)

Query Selector

Access me by query

Query Selector All

Access me by query all (1)

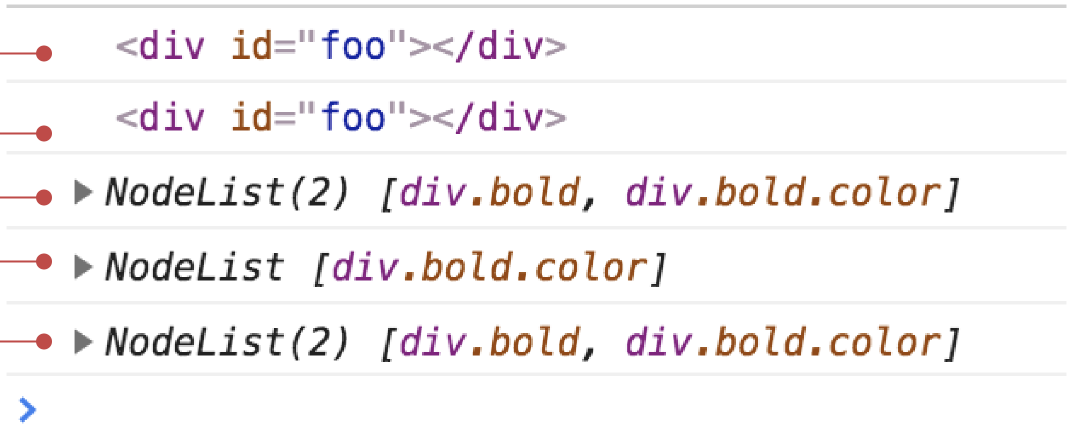
Access me by query all (2)

Node Lists

- The DOM API may manipulate sets/lists of nodes
- The `NodeList` type is an array-like sequence of Nodes
- May be accessed as a JS Array
  - `.length` property
  - `.item(i)` , equivalent to `list[i]`
  - `.entries()`, `.keys()`, `.values()` iterators
  - `.forEach()` functional iteration
  - `for…of` classical iteration

Accessing DOM Elements

```
<!DOCTYPE html>
<html>
<head></head>
<body>
<div id="foo"></div>
<div class="bold"></div>
<div class="bold color"></div>
<script>
 document.getElementById('foo');
 document.querySelector('#foo');
 document.querySelectorAll('.bold');
 document.querySelectorAll('.color');
 document.querySelectorAll('.bold, .color');
</script>
</body>
</html>
```

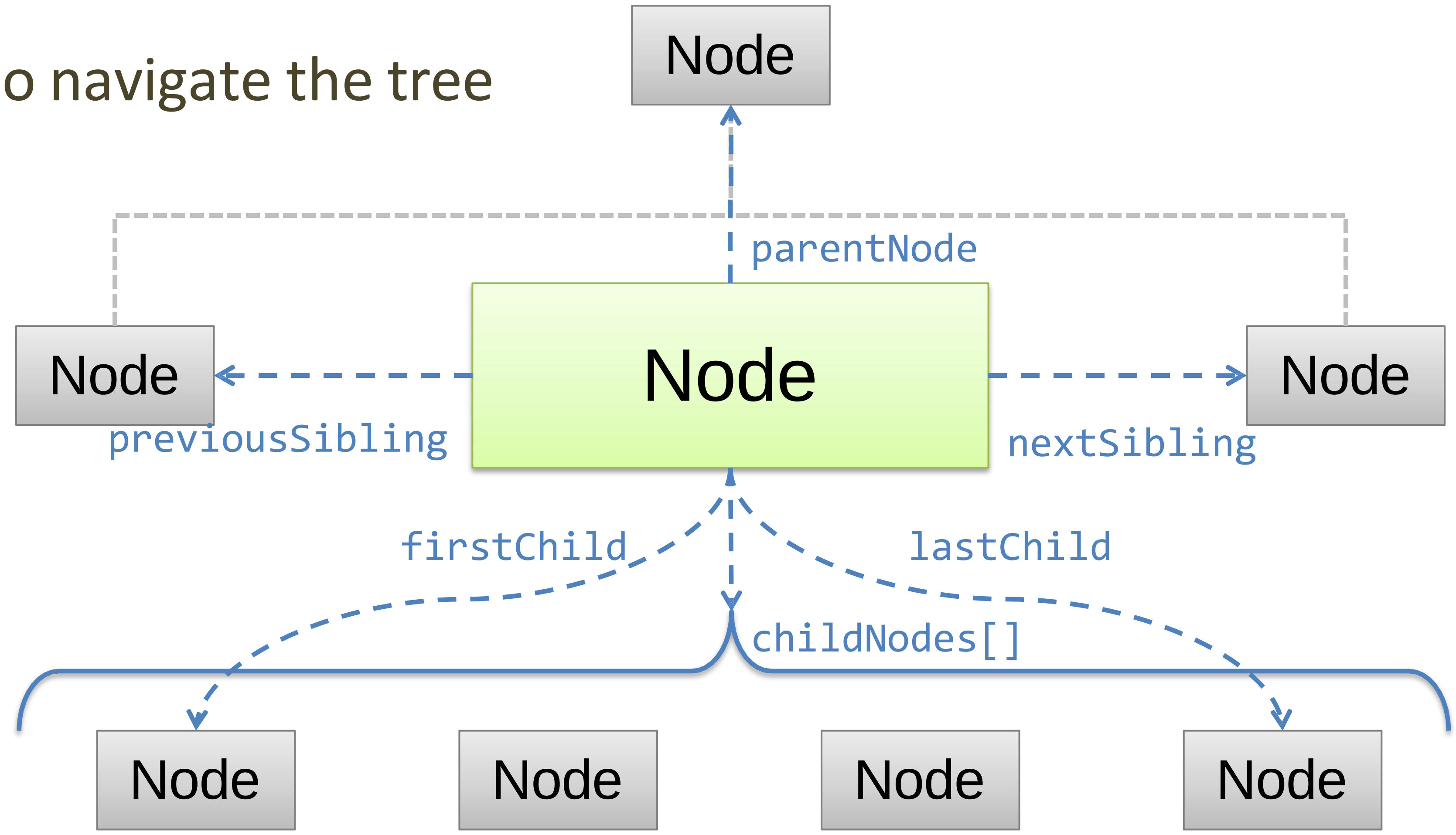


```
<div id="foo"></div>
<div id="foo"></div>
▶ NodeList(2) [div.bold, div.bold.color]
▶ NodeList [div.bold.color]
▶ NodeList(2) [div.bold, div.bold.color]
>
```

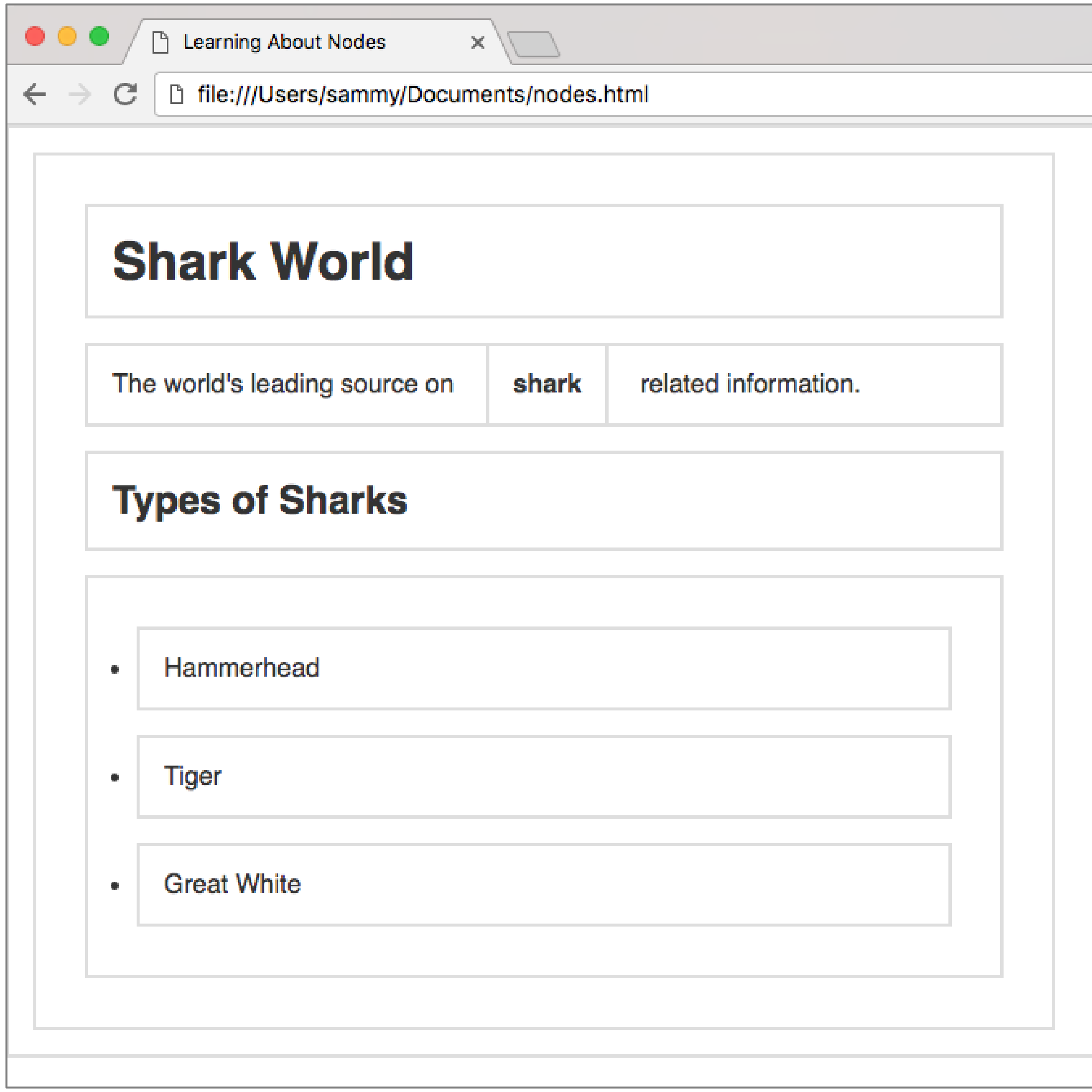
How To Traverse the DOM

Navigating The Tree

- Properties to navigate the tree



```html
<html>
<head>
<title>Learning About Nodes</title>
<style>
    * { border: 2px solid #dedede; padding: 15px; margin: 15px; }
    html { margin: 0; padding: 0; }
    body { max-width: 600px; font-family: sans-serif; color: #333; }
</style>
</head>
<body>
    <h1>Shark World</h1>
    <p>The world's leading source on <strong>shark</strong>
    related information.</p>
    <h2>Types of Sharks</h2>
    <ul>
        <li>Hammerhead</li>
        <li>Tiger</li>
        <li>Great White</li>
    </ul>
    </body>
<script>
    const h1 = document.getElementsByTagName('h1')[0];
    const p = document.getElementsByTagName('p')[0];
    const ul = document.getElementsByTagName('ul')[0];
</script>
```



Shark World

The world's leading source on **shark** related information.

Types of Sharks

- Hammerhead
- Tiger
- Great White

```html
<html>
<head>
<title>Learning About Nodes</title>
<style>
    * { border: 2px solid #dedede; padding: 15px; margin: 15px; }
    html { margin: 0; padding: 0; }
    body { max-width: 600px; font-family: sans-serif; color: #333; }
</style>
</head>
<body>
    <h1>Shark World</h1>
    <p>The world's leading source on <strong>shark</strong>
    related information.</p>
    <h2>Types of Sharks</h2>
    <ul>
        <li>Hammerhead</li>
        <li>Tiger</li>
        <li>Great White</li>
    </ul>
</body>
<script>
    const h1 = document.getElementsByTagName('h1')[0];
    const p = document.getElementsByTagName('p')[0];
    const ul = document.getElementsByTagName('ul')[0];
</script>
</html>
```

- **html** is the parent of **head**, **body**, and **script**
- **body** is the parent of **h1**, **h2**, **p** and **ul**
  - Not **li**, since **li** is two levels down from **body**

```
1.  > p.parentNode
2.  Output
3.  ▶ <body>...</body>
4.  > ul.childNode;
5.  Output
6.  ▶ (7) [text, li, text, li, text, li, text]
7.  ul.firstChild.style.background = 'yellow';
8.  ul.firstElementChild.style.background = 'yellow';
```

- Line 7, **firstChild** is text (error)
- **firstElementChild** returns only **element** node

Tag Attributes Exposed As Properties

- *Attributes* of the HTML elements become *object properties* of the DOM objects
- Example
  - `<body id="page">`
  - DOM object: `document.body.id="page"`
  - Also: `document["body"]["id"]`

  - `<input id="input" type="checkbox" checked />`
  - DOM object: `input.checked   // boolean`

Creating New Nodes

Property/Method	Description
createElement()	Create a new element node
createTextNode()	Create a new text node
node.textContent	Get or set the text content of an element node
node.innerHTML	Get or set the HTML content o f an element

- In a dynamic web app, elements and text are often added with JavaScript
- `createElement`() and `createTextNode`() methods are used to create new nodes in the DOM

Creating Elements

- Use document methods:
  - `document.createElement(tag)` to create an element with a chosen tag
  - `document.createTextNode(text)` to create a text node with the given text
- Example: div with class and content

```
let div = document.createElement('div');
div.className = "alert alert-success";
div.innerText = "Hi there! You've read an important message.";
```

```
<div class="alert alert-success">
Hi there! You've read an important message.
</div>
```

Inserting Elements In The DOM Tree

- If not inserted, they will not appear

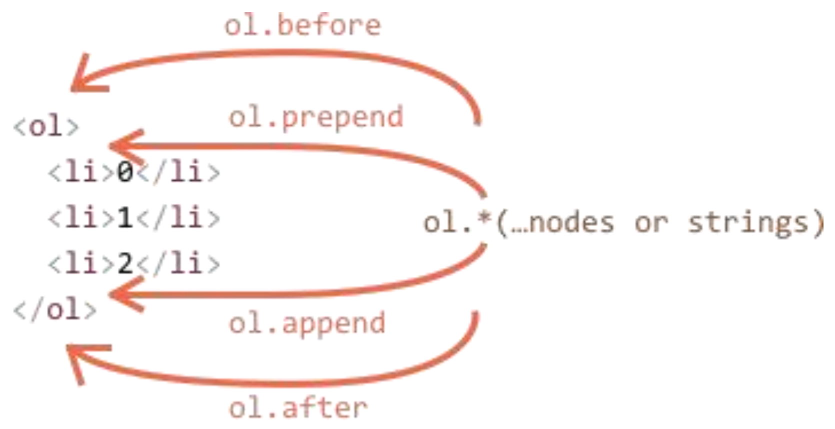
```
document.body.appendChild(div)
```

```
...
<body>
 <div class="alert alert-success">
 <strong>Hi there!</strong> You've read an important message.
 </div>
<body>
```

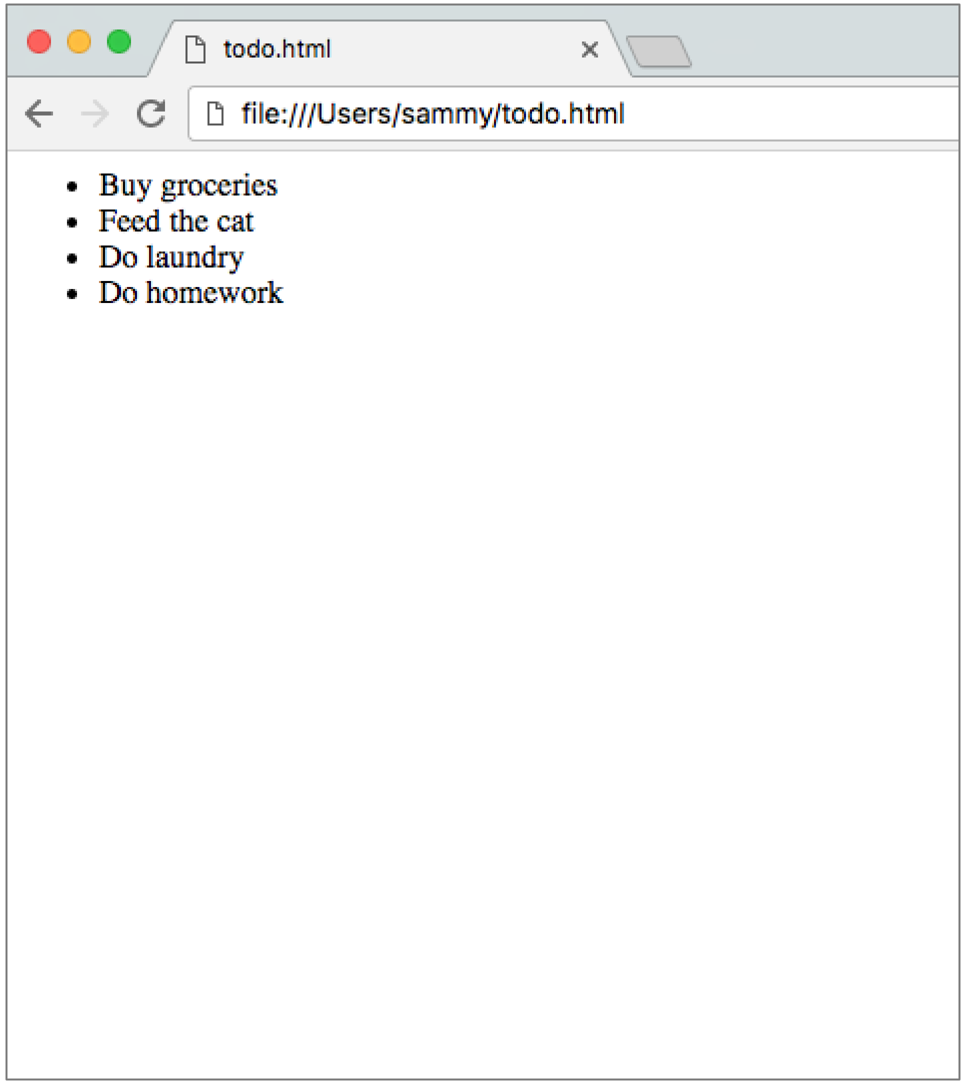
Inserting Children

- **parentElem.appendChild(node)**
- parentElem.insertBefore(node, nextSibling)
- parentElem.replaceChild(node, oldChild)

- node.append(…nodes or strings)
- node.prepend(…nodes or strings)
- node.before(…nodes or strings)
- node.after(…nodes or strings)
- node.replaceWith(…nodes or strings)



```
<ul>
  <li>Buy groceries</li>
  <li>Feed the cat</li>
  <li>Do laundry</li>
</ul>

// To-do list ul element
const todoList = document.querySelector('ul');

// Create new to-do
const newTodo = document.createElement('li');
newTodo.textContent = 'Do homework';
// Add new todo to the end of the list
todoList.appendChild(newTodo);
```



- Buy groceries
- Feed the cat
- Do laundry
- Do homework

Handling Tag Content

- `.innerHTML` to get/set element content in textual form
- The browser will parse the content and convert it into DOM Nodes and Attrs

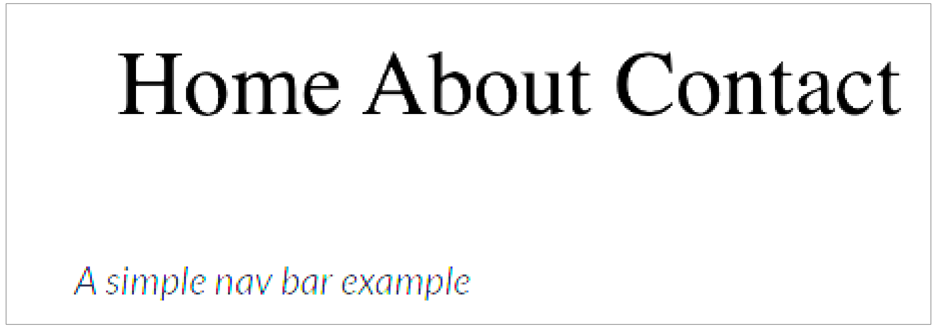
```
<div class="alert alert-success">
    <strong>Hi there!</strong> You've read an important message.
</div>
```

```
div.innerHTML  // "<strong>Hi there!</strong> You've read an important message."
```

innerHTML vs innerText vs textContent

- **`innerHTML`** reads both the HTML markup and the text content of the element
  - Cautious inserting content from user input or any untrusted source with **`innerHTML`**
  - Attackers can use <script> tag to insert and run malicious code in your app
- **`innerText`** returns text as it appears on screen
  - Ignores HTML tags. And it also does not include text that is hidden with CSS styles
  - When you need to account for styles, you should consider using **`innerText`**
- **`textContent`** also ignores all HTML tags and returns only the text
  - Only deals with the raw text and doesn't account for styles
- Whiles **`innerText`** reads text as it is rendered on screen, **`textContent`** reads text as it is in the markup

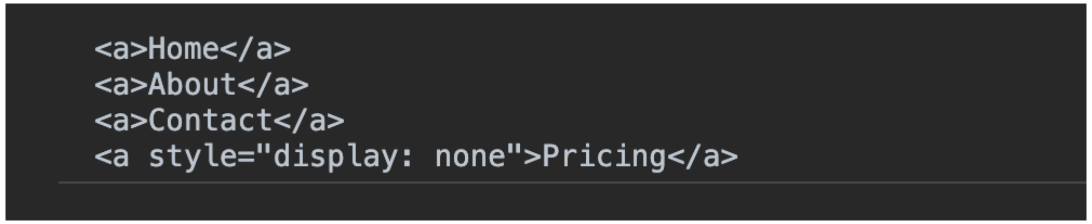
```html
<nav>
  <a>Home</a>
  <a>About</a>
  <a>Contact</a>
  <a style="display: none">Pricing</a>
</nav>
```


Home About Contact

A simple nav bar example

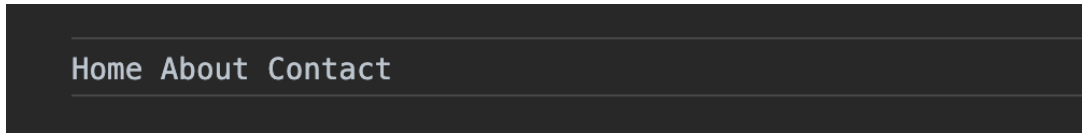
- Getting content with innerHTML

```javascript
const navElement = document.querySelector('nav')
console.log(navElement.innerHTML)
```

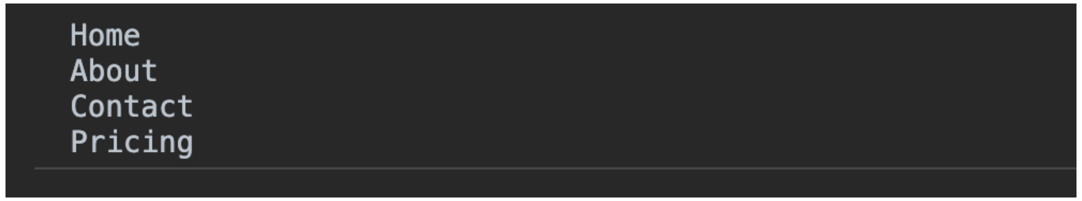


```
  <a>Home</a>
  <a>About</a>
  <a>Contact</a>
  <a style="display: none">Pricing</a>
```

- Getting content with innerText
- Getting content with textContent



```
Home About Contact
```



```
Home
About
Contact
Pricing
```

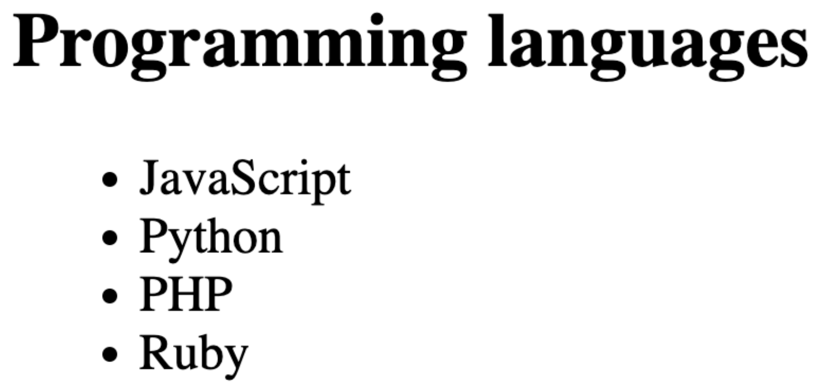
How to Update Content

```
<h2>Programming languages</h2>
<ul class="ll"></ul>
```

- Setting content with innerHTML

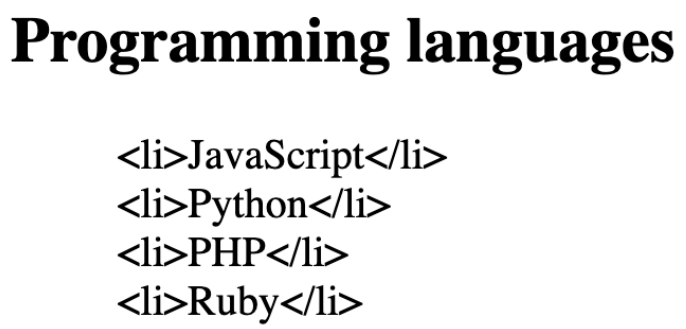
```
const aa = document.querySelector(‘.ll')
aa.innerHTML = `
  <li>JavaScript</li>
  <li>Python</li>
  <li>PHP</li>
  <li>Ruby</li>
`
```

- Setting content with innerText
- Setting content with textContent



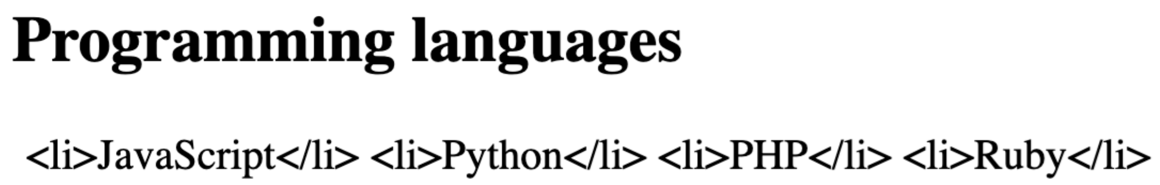
Programming languages

- JavaScript
- Python
- PHP
- Ruby



Programming languages

<li>JavaScript</li>
<li>Python</li>
<li>PHP</li>
<li>Ruby</li>



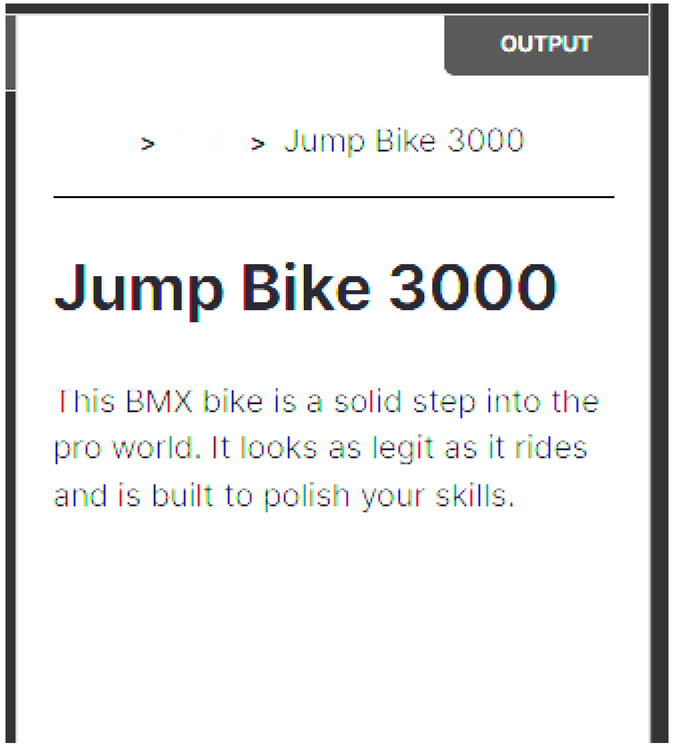
Programming languages

<li>JavaScript</li> <li>Python</li> <li>PHP</li> <li>Ruby</li>

<nav> Element

- Represents a section of a page whose purpose is to provide **navigation** links

```
<nav class="crumbs">
 <ol>
  <li class="crumb"><a href="#">Bikes</a></li>
  <li class="crumb"><a href="#">BMX</a></li>
  <li class="crumb">Jump Bike 3000</li>
 </ol>
</nav>
<h1>Jump Bike 3000</h1>
<p>
 This BMX bike is a solid step into the pro
world. It looks as legit as it rides and is
built to polish your skills.
</p>
```



> > Jump Bike 3000

Jump Bike 3000

This BMX bike is a solid step into the pro world. It looks as legit as it rides and is built to polish your skills.

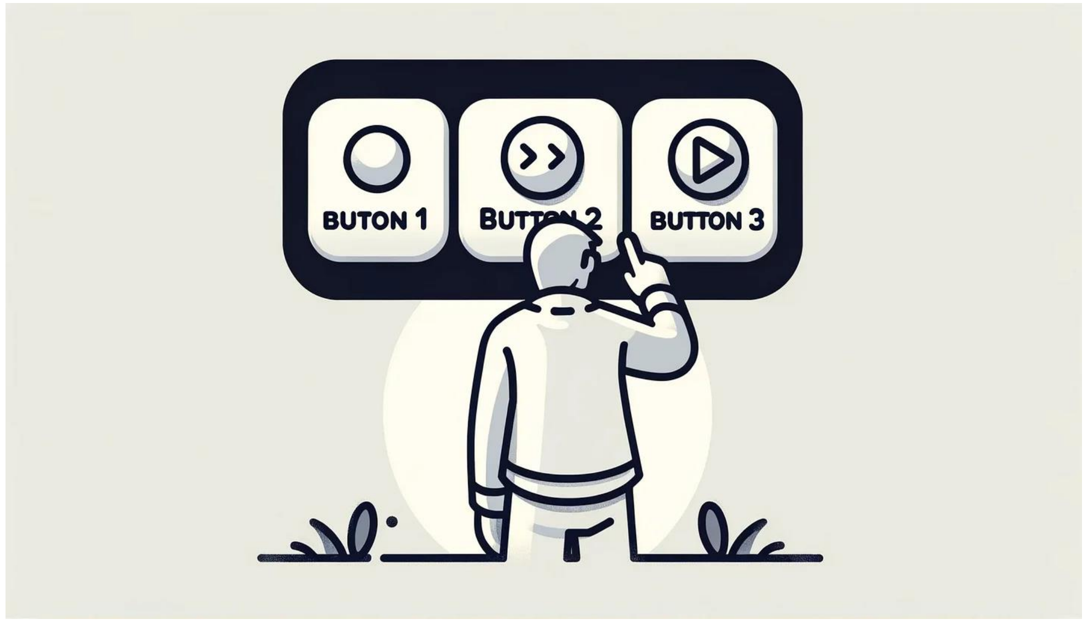
Mozilla Developer Network: Event Reference
https://developer.mozilla.org/en-US/docs/Web/Events

JS in the browser

EVENT HANDLING

Events

- They take place in browser that can be initiated by either user or browser itself
  - The page finishes loading
  - The user clicks a button
  - The user hovers over a dropdown
  - The user submits a form
  - The user presses a key on their keyboard
- By coding JavaScript responses that execute upon an event,
  - Developers can display messages to users, validate data, react to a button click



Event Handler and Event Listener

- An event handler is a JavaScript function that runs when an event fires
- There are three ways to assign events to elements:
  - Inline event handlers
    - Set property of the attribute in the HTML
  - Event handler properties
    - Set property of an element in JavaScript
  - Event listeners

- An event listener attaches a responsive interface to an element
- It allows that particular element to wait and "listen" for the given event to fire

Inline Event Handlers, Event handler properties

```html
<!DOCTYPE html>
<html lang="en-US">
<head>
    <title>Events</title>
</head>
<body>
    <button onclick="changeText()">Click me</button>
    <p>Try to change me.</p>
</body>
<script src="js/events.js"></script>
</html>
```

```javascript
const changeText = () => {
    const p = document.querySelector('p');
    p.textContent = "I changed because of an inline event handler.";
}
```

```html
...
<body>
<button>Click me</button>
<p>I will change.</p>
</body>
...
```

```javascript
const changeText = () => {
    const p = document.querySelector('p');
    p.textContent = "I changed because of an event handler property.";
}
const button = document.querySelector('button');
button.onclick = changeText;
```

Event Listeners

```
...
<body>
<button>Click me</button>
<p>I will change.</p>
</body>
...
```

```
// Function to modify the text content of the paragraph
const changeText = () => {
   const p = document.querySelector('p');
   p.textContent = "I changed because of an event listener.";
}

// Listen for click event
const button = document.querySelector('button');
button.addEventListener('click', changeText);
```

- Instead of assigning the event directly to a property on the element, we will use the `addEventListener()` method to listen for the event

addEventListener()

- Can add as many listeners as desired, even to the same node
- Callback receives as first parameter an Event object

```
window.addEventListener('load', (event) => {
  //window loaded
})
```

```
const link = document.getElementById('my-link')
link.addEventListener('mousedown', event => {
  // mouse button pressed
  console.log(event.button) //0=left, 2=right
})
```

https://flaviocopes.com/javascript-events/

Event

```html
<!DOCTYPE html>
<html lang="en">
   <head>
   <title>Learning the DOM</title>
   </head>
   <body>
      <h1>Document Object Model</h1>
      <button id="changeBackground">
      Change Background Color</button>

      <script src="scripts.js"></script>
   </body>
</html>
```

- An event in JavaScript is an action the user has taken
  - e.g, when the user hovers their mouse over an element, or clicks on an element
- Want our button to listen and be ready to perform an action when the user clicks
  - We can do this by adding an **event listener** to our button by using `addEventListener()`

```javascript
let button = document.getElementById('changeBackground');

button.addEventListener('click', () => {
  document.body.style.backgroundColor = 'fuchsia';
});
```

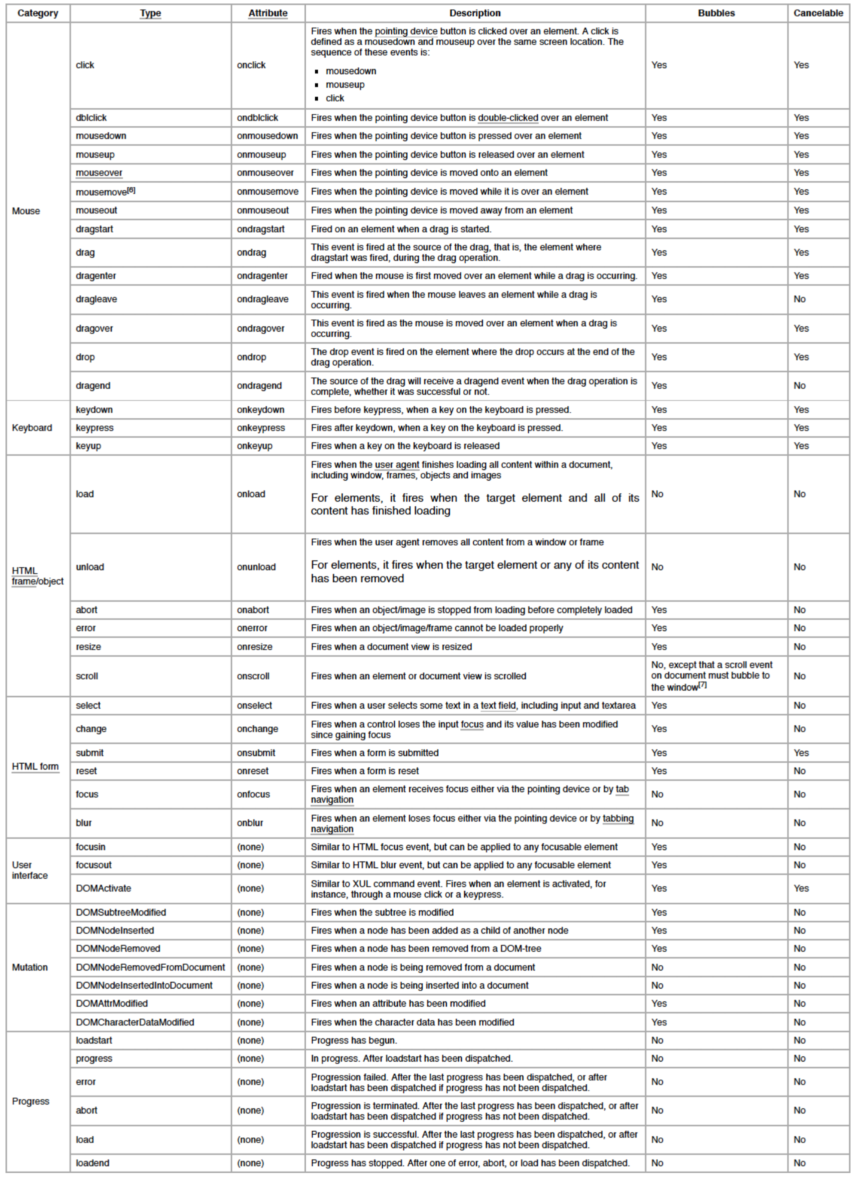
Event Object

- Main properties:
  - `target`, the DOM element that originated the event
  - `type`, the type of event

https://developer.mozilla.org/en-US/docs/Web/API/Event/type

Event Categories

- User Interface events (load, resize, scroll, etc.)
- Focus/blur events
- Mouse events (click, dblclick, mouseover, drag,
- Keyboard events (keyup, etc.)
- Form events (submit, change, input)
- Mutation events (DOMContentLoaded, etc.)
- HTML5 events (invalid, loadeddata, etc.)
- CSS events (animations etc.)



Category	Type	Attribute	Description	Bubbles	Cancelable
Mouse	click	onclick	Fires when the pointing device button is clicked over an element. A click is defined as a mousedown and mouseup over the same screen location. The sequence of these events is: ▪ mousedown ▪ mouseup ▪ click	Yes	Yes
	dblclick	ondblclick	Fires when the pointing device button is double-clicked over an element	Yes	Yes
	mousedown	onmousedown	Fires when the pointing device button is pressed over an element	Yes	Yes
	mouseup	onmouseup	Fires when the pointing device button is released over an element	Yes	Yes
	mouseover	onmouseover	Fires when the pointing device is moved onto an element	Yes	Yes
	mousemove[6]	onmousemove	Fires when the pointing device is moved while it is over an element	Yes	Yes
	mouseout	onmouseout	Fires when the pointing device is moved away from an element	Yes	Yes
	dragstart	ondragstart	Fired on an element when a drag is started.	Yes	Yes
	drag	ondrag	This event is fired at the source of the drag, that is, the element where dragstart was fired, during the drag operation.	Yes	Yes
	dragenter	ondragenter	Fired when the mouse is first moved over an element while a drag is occurring.	Yes	Yes
	dragleave	ondragleave	This event is fired when the mouse leaves an element while a drag is occurring.	Yes	No
	dragover	ondragover	This event is fired as the mouse is moved over an element when a drag is occurring.	Yes	Yes
	drop	ondrop	The drop event is fired on the element where the drop occurs at the end of the drag operation.	Yes	Yes
	dragend	ondragend	The source of the drag will receive a dragend event when the drag operation is complete, whether it was successful or not.	Yes	No
Keyboard	keydown	onkeydown	Fires before keypress, when a key on the keyboard is pressed.	Yes	Yes
	keypress	onkeypress	Fires after keydown, when a key on the keyboard is pressed.	Yes	Yes
	keyup	onkeyup	Fires when a key on the keyboard is released	Yes	Yes
HTML frame/object	load	onload	Fires when the user agent finishes loading all content within a document, including window, frames, objects and images For elements, it fires when the target element and all of its content has finished loading	No	No
	unload	onunload	Fires when the user agent removes all content from a window or frame For elements, it fires when the target element or any of its content has been removed	No	No
	abort	onabort	Fires when an object/image is stopped from loading before completely loaded	Yes	No
	error	onerror	Fires when an object/image/frame cannot be loaded properly	Yes	No
	resize	onresize	Fires when a document view is resized	Yes	No
	scroll	onscroll	Fires when an element or document view is scrolled	No, except that a scroll event on document must bubble to the window[7]	No
HTML form	select	onselect	Fires when a user selects some text in a text field, including input and textarea	Yes	No
	change	onchange	Fires when a control loses the input focus and its value has been modified since gaining focus	Yes	No
	submit	onsubmit	Fires when a form is submitted	Yes	Yes
	reset	onreset	Fires when a form is reset	Yes	No
	focus	onfocus	Fires when an element receives focus either via the pointing device or by tab navigation	No	No
	blur	onblur	Fires when an element loses focus either via the pointing device or by tabbing navigation	No	No
User interface	focusin	(none)	Similar to HTML focus event, but can be applied to any focusable element	Yes	No
	focusout	(none)	Similar to HTML blur event, but can be applied to any focusable element	Yes	No
	DOMActivate	(none)	Similar to XUL command event. Fires when an element is activated, for instance, through a mouse click or a keypress.	Yes	Yes
Mutation	DOMSubtreeModified	(none)	Fires when the subtree is modified	Yes	No
	DOMNodeInserted	(none)	Fires when a node has been added as a child of another node	Yes	No
	DOMNodeRemoved	(none)	Fires when a node has been removed from a DOM-tree	Yes	No
	DOMNodeRemovedFromDocument	(none)	Fires when a node is being removed from a document	No	No
	DOMNodeInsertedIntoDocument	(none)	Fires when a node is being inserted into a document	No	No
	DOMAttrModified	(none)	Fires when an attribute has been modified	Yes	No
	DOMCharacterDataModified	(none)	Fires when the character data has been modified	Yes	No
Progress	loadstart	(none)	Progress has begun.	No	No
	progress	(none)	In progress. After loadstart has been dispatched.	No	No
	error	(none)	Progression failed. After the last progress has been dispatched, or after loadstart has been dispatched if progress has not been dispatched.	No	No
	abort	(none)	Progression is terminated. After the last progress has been dispatched, or after loadstart has been dispatched if progress has not been dispatched.	No	No
	load	(none)	Progression is successful. After the last progress has been dispatched, or after loadstart has been dispatched if progress has not been dispatched.	No	No
	loadend	(none)	Progress has stopped. After one of error, abort, or load has been dispatched.	No	No

https://en.wikipedia.org/wiki/DOM_events

Preventing Default Behavior

- Many events cause a default behavior
  - Click on link: go to URL
  - Click on submit button: form is sent
- Can be prevented by `event.preventDefault()`

HTML Page Lifecycle: Events

- `DOMContentLoaded` (defined on `document`)
  - The browser loaded all HTML, and the DOM tree is ready
  - External resources are not loaded, yet
- `load` (defined on `window`)
  - The browser finished loading all external resources
- `beforeunload/unload`
  - The user is about to leave the page / has just left the page
  - Not recommended (not totally reliable)

```
document.addEventListener("DOMContentLoaded", ready);
```

Mozilla Developer Network:
Web forms — Collecting data from users
https://developer.mozilla.org/en-US/docs/Learn/Forms

Handling user input

FORM CONTROLS

Form Declaration

- `<form>` tag
- Specifies URL to be used for submission (attribute `action`)
- Specifies HTTP method (attribute `method`, default GET)

```
...
<form action="/new-task" method="POST" id="userdata">
       ...normal HTML content...
              and
       ...FORM Controls...
</form>
...
```

Form Controls

- A set of HTML elements allowing different types of user input/interaction
  - Each element should be uniquely identified by the value of the name attribute
- Several control categories
  - Input
  - Selection
  - Button
- Support elements
  - Label
  - Datalist

Input Control

- `<input>` tag
- Text input example
- The `value` attribute will hold user-provided text

```
...
<input type="text" name="firstname" placeholder="Your username"></input>
...
```

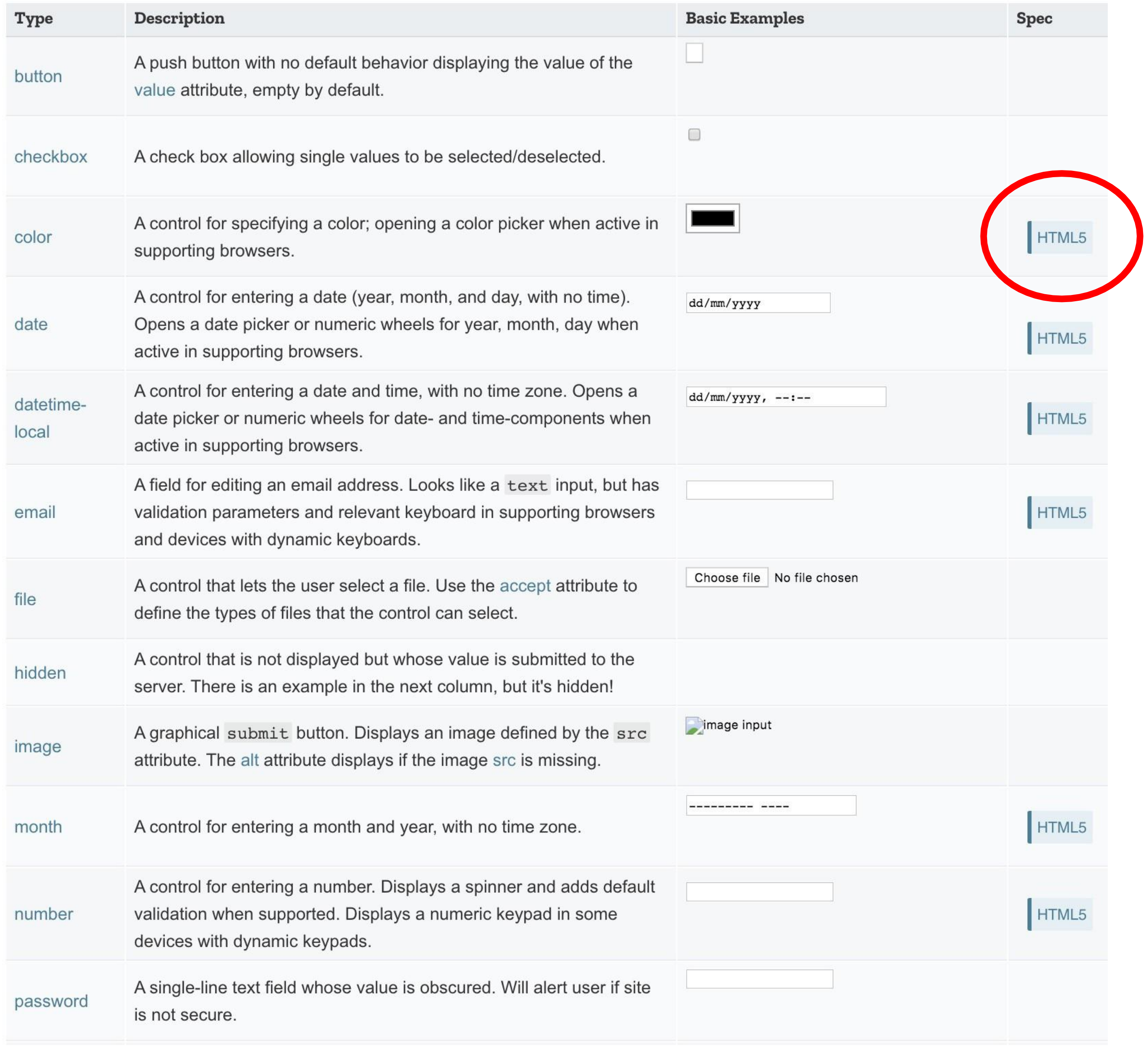
Locating a Form In The DOM

- document.forms is a collection of all forms in the page
  - const myForm = document.forms['form ID’]
- Each form node has an elements properties, that collects all data-containing inner elements
  - const myElement = myForm.elements['element ID']

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

Input Control (1)

- type attribute
  - button
  - checkbox
  - color
  - date
  - email
  - file
  - hidden
  - month
  - number
  - password

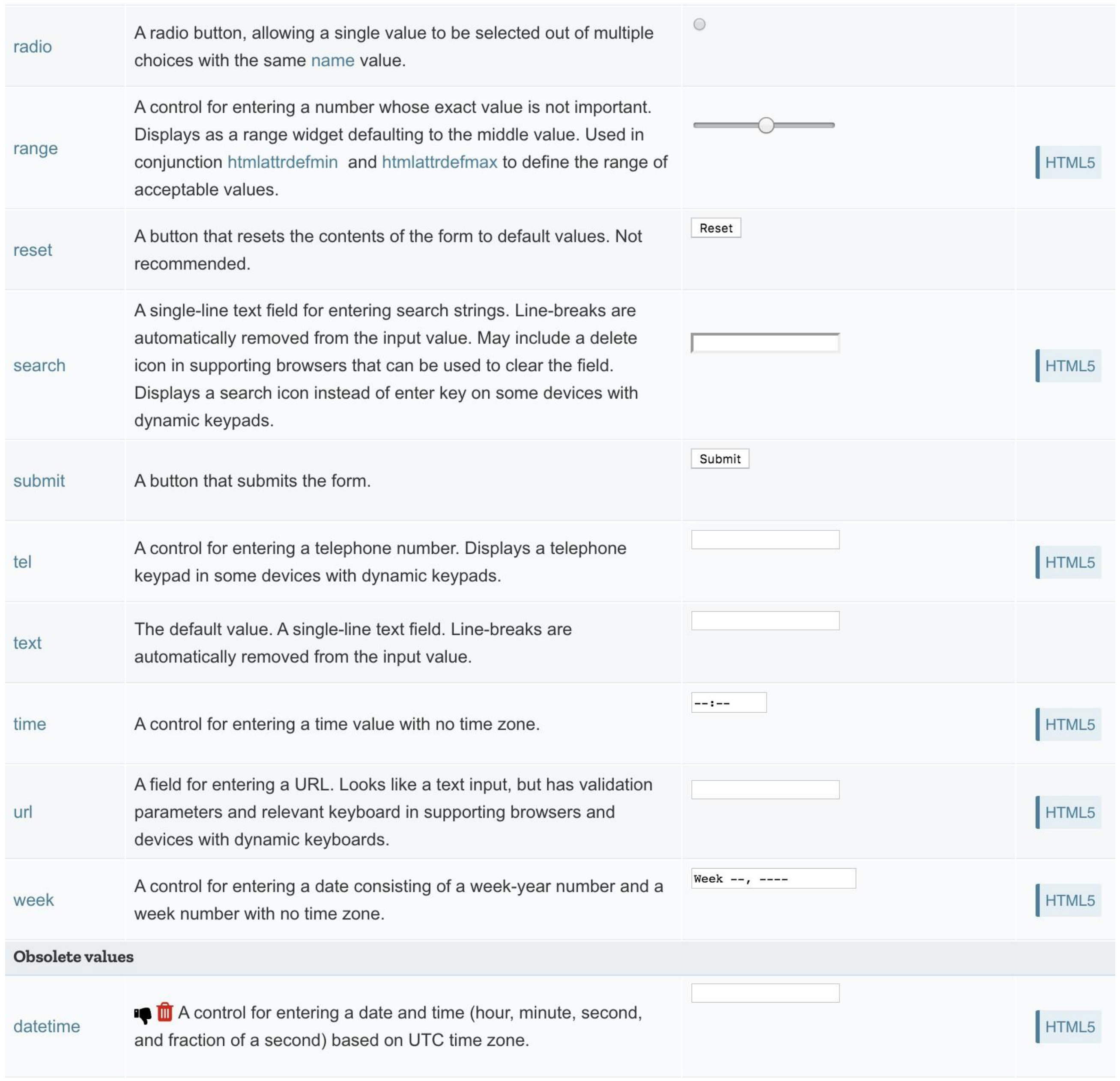


Type	Description	Basic Examples	Spec
button	A push button with no default behavior displaying the value of the value attribute, empty by default.		
checkbox	A check box allowing single values to be selected/deselected.	☐	
color	A control for specifying a color; opening a color picker when active in supporting browsers.	■	HTML5
date	A control for entering a date (year, month, and day, with no time). Opens a date picker or numeric wheels for year, month, day when active in supporting browsers.	dd/mm/yyyy	HTML5
datetime-local	A control for entering a date and time, with no time zone. Opens a date picker or numeric wheels for date- and time-components when active in supporting browsers.	dd/mm/yyyy, --:--	HTML5
email	A field for editing an email address. Looks like a `text` input, but has validation parameters and relevant keyboard in supporting browsers and devices with dynamic keyboards.		HTML5
file	A control that lets the user select a file. Use the accept attribute to define the types of files that the control can select.	Choose file No file chosen	
hidden	A control that is not displayed but whose value is submitted to the server. There is an example in the next column, but it's hidden!		
image	A graphical `submit` button. Displays an image defined by the `src` attribute. The alt attribute displays if the image src is missing.	image input	
month	A control for entering a month and year, with no time zone.	--------- ----	HTML5
number	A control for entering a number. Displays a spinner and adds default validation when supported. Displays a numeric keypad in some devices with dynamic keypads.		HTML5
password	A single-line text field whose value is obscured. Will alert user if site is not secure.		

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input

Input Control (2)

- type attribute
  - radio (button)
  - range
  - submit/reset (button)
  - search
  - tel
  - text
  - url
  - week



radio	A radio button, allowing a single value to be selected out of multiple choices with the same name value.	○	
range	A control for entering a number whose exact value is not important. Displays as a range widget defaulting to the middle value. Used in conjunction htmlattrdefmin and htmlattrdefmax to define the range of acceptable values.		HTML5
reset	A button that resets the contents of the form to default values. Not recommended.	Reset	
search	A single-line text field for entering search strings. Line-breaks are automatically removed from the input value. May include a delete icon in supporting browsers that can be used to clear the field. Displays a search icon instead of enter key on some devices with dynamic keypads.		HTML5
submit	A button that submits the form.	Submit	
tel	A control for entering a telephone number. Displays a telephone keypad in some devices with dynamic keypads.		HTML5
text	The default value. A single-line text field. Line-breaks are automatically removed from the input value.		
time	A control for entering a time value with no time zone.	--:--	HTML5
url	A field for entering a URL. Looks like a text input, but has validation parameters and relevant keyboard in supporting browsers and devices with dynamic keyboards.		HTML5
week	A control for entering a date consisting of a week-year number and a week number with no time zone.	Week --, ----	HTML5
Obsolete values			
datetime	A control for entering a date and time (hour, minute, second, and fraction of a second) based on UTC time zone.		HTML5

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input

Input Control: Commonly Used Attributes

Attribute	Meaning
`checked`	radio/checkbox is selected
`disabled`	control is disabled
`readonly`	value cannot be edited
`required`	need a valid input to allow form submission
`size`	the size of the control (pixels or characters)
`value`	the value inserted by the user
`autocomplete`	hint for form autofill feature of the browser

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#Attributes

Input Control: Other Attributes

- Depends on the control

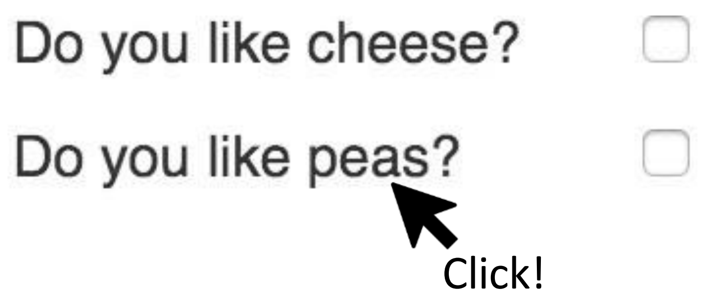
```
<input type="number" name="age" placeholder="Your age" min="18" max="110" />

<input type="text" name="username" pattern="[a-zA-Z]{8}" />

<input type="file" name="docs" accept=".jpg, .jpeg, .png" />
```

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#Attributes

Label Tag

- The HTML `<label>` element represents a caption for an item in a user interface. Associated with `for` attribute and `id` on input
- Important for accessibility purposes (e.g. screenreader etc.), clicking the label activates the control (larger activation area e.g. in touch screens)

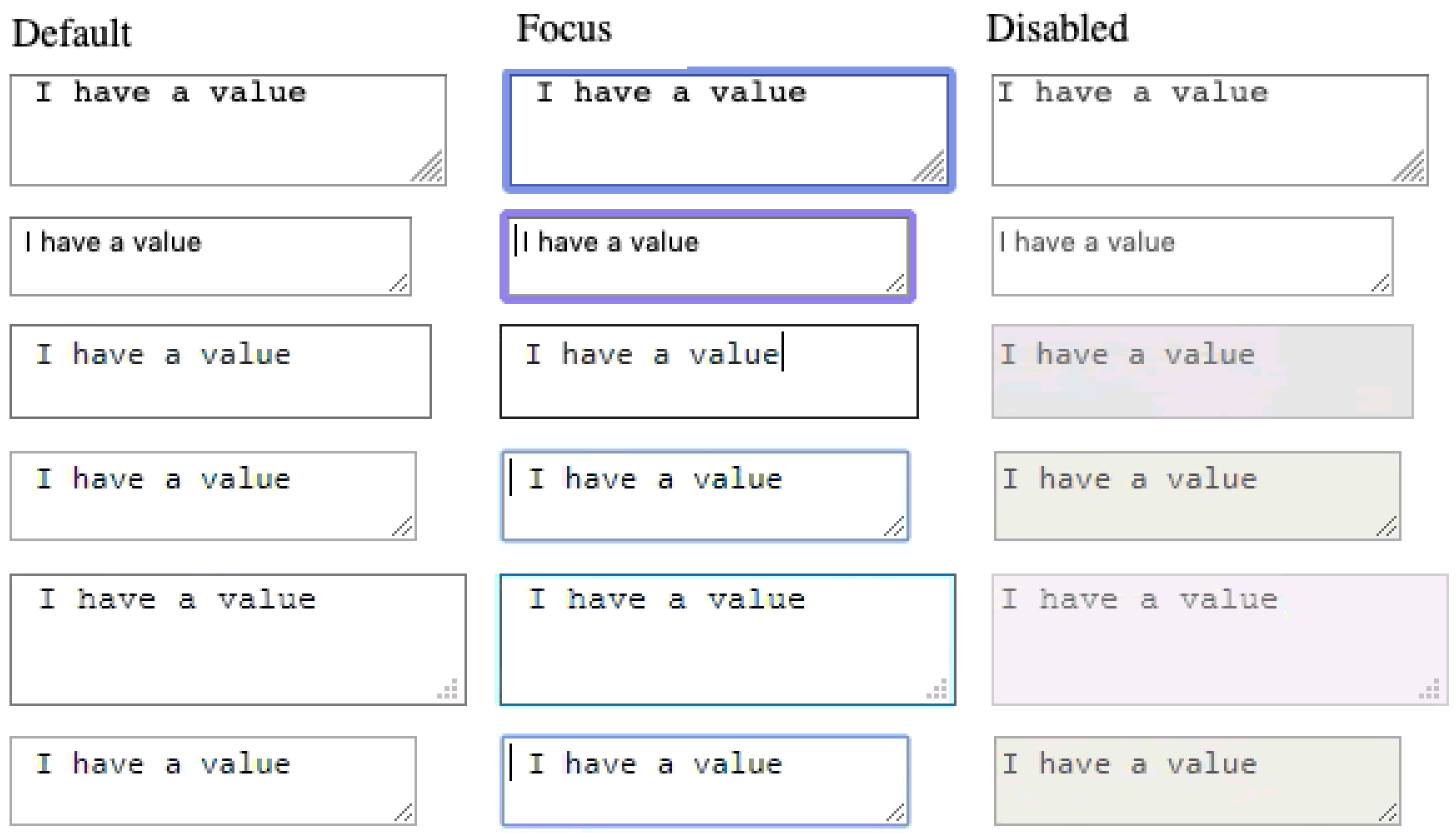
```
<div class="preference">
    <label for="cheese">Do you like cheese?</label>
    <input type="checkbox" name="cheese" id="cheese">
</div>
<div class="preference">
    <label for="peas">Do you like peas?</label>
    <input type="checkbox" name="peas" id="peas">
</div>
```



https://developer.mozilla.org/en-US/docs/Web/HTML/Element/label

Other Form Controls

`<textarea>`:
a multi-line text field

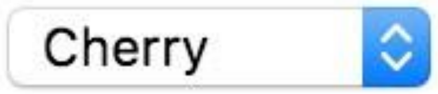


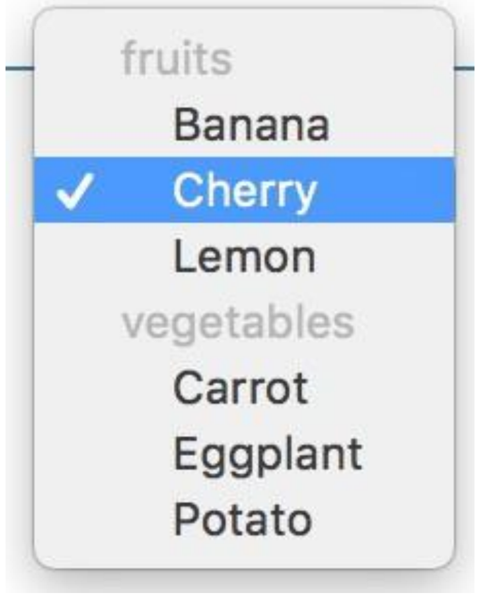
https://developer.mozilla.org/en-US/docs/Learn/Forms/Other_form_controls

Other Form Controls

Drop-down controls

```
<select id="groups" name="groups">
   <optgroup label="fruits">
     <option>Banana</option>
    <option selected>Cherry</option>
    <option>Lemon</option>
  </optgroup>
  <optgroup label="vegetables">
    <option>Carrot</option>
    <option>Eggplant</option>
    <option>Potato</option>
  </optgroup>
</select>
```





https://developer.mozilla.org/en-US/docs/Learn/Forms/Other_form_controls

Button Control

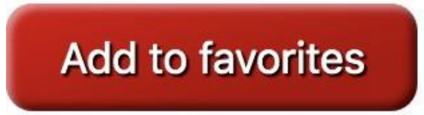
- `<button>` tag
- Three types of buttons
  - `submit`: submits the form to the server
  - `reset`: reset the content of the form to the initial value
  - `button`: just a button, whose behavior needs to be specified by JavaScript

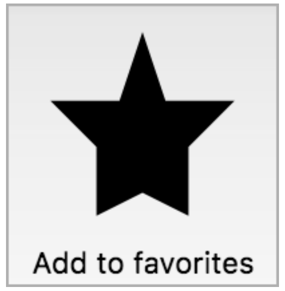
```
...
<button type="submit" value="Send data" />
...
```

Button vs. input type=button

More flexible, can have content (markup, images, etc.)

```
...
<button class="favorite styled"
        type="button">
    Add to favorites
</button>
...
<button name="favorite">
  <svg aria-hidden="true" viewBox="0 0 10 10"><path
d="M7 9L5 8 3 9V6L1 4h3l1-3 1 3h3L7 6z"/></svg>
  Add to favorites
</button>
...
```

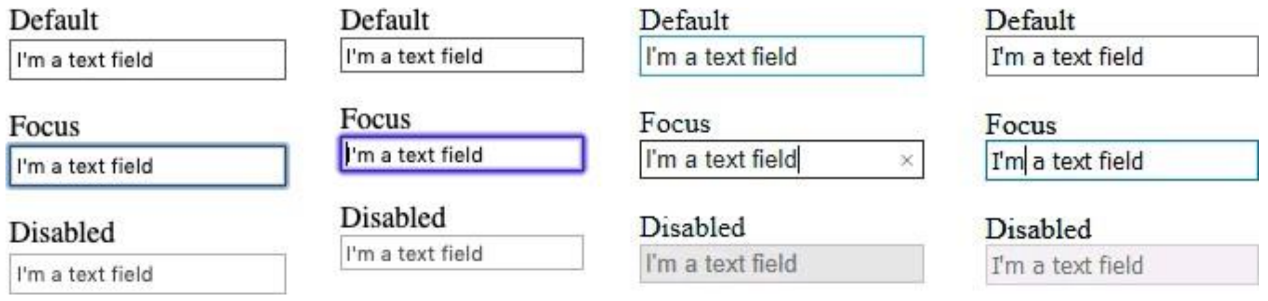




https://developer.mozilla.org/en-US/docs/Web/HTML/Element/button

Default Appearance May Vary

- Solve with CSS, but
- Some problems still remain
  - See: "Styling web forms" in MDN
  - Examples of controls difficult to manage:
    - Bad: Checkboxes, …
    - Ugly: Color, Range, File: cannot be styled via CSS



https://developer.mozilla.org/en-US/docs/Learn/Forms/Styling_web_forms

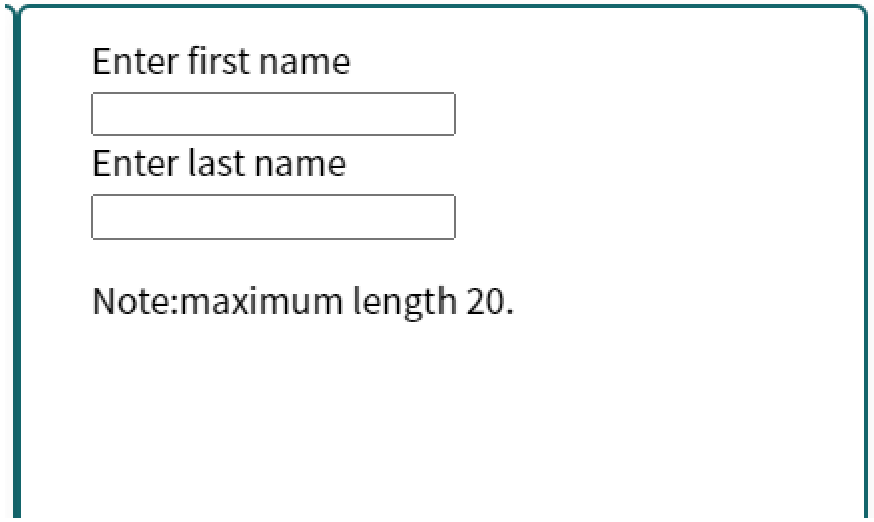
The Road to Nicer Forms

- Useful libraries (frameworks) and polyfills
  - Especially for controls difficult to handle via CSS
  - Rely on JavaScript
- Suggestions
  - Bootstrap
  - Using libraries may improve accessibility

https://developer.mozilla.org/en-US/docs/Learn/Forms/Advanced_form_styling

Example

```
<form>
    <label>Enter first name</label><br>
    <input type="text" name="firstname"><br>
    <label>Enter last name</label><br>
    <input type="text" name="lastname"><br>
    <p>Note:maximum length 20.</p>
</form>
```


Enter first name

Enter last name

Note:maximum length 20.

Input Outside Form

- HTML spec mandates that input have to be inside a form element
- You can have a valid input without a form

Mozilla Developer Network:
Web forms — Form Validation
https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation

Handling user input

FORM EVENTS

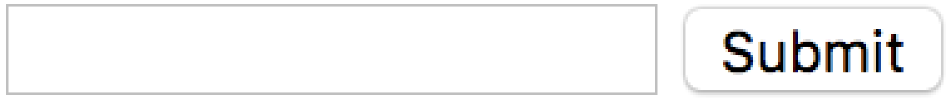
Events On Input Elements

Event	Meaning
`input`	the value of the element is changed (even a single character)
`change`	when something changed in the element (for text elements, it is fired only once when the element loses focus)
`cut copy paste`	when the user does the corresponding action
`focus`	when the element gains focus
`blur`	when the element loses focus
`invalid`	when the form is submitted, fires for each element which is invalid, and for the form itself

https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation

Example

```
...
<form action="/add" method="POST">
  <input type="text">
  <input type="submit">
</form>
...
```



```
const inputField = document.querySelector('input[type="text"]')

inputField.addEventListener('input', event => {
  console.log(`The current entered value is: ${inputField.value}`);
})

inputField.addEventListener('change', event => {
  console.log(`The value has changed since last time: ${inputField.value}`);
})
```

Form Submission

- Can be intercepted with the `submit` event
- If required, default action can be prevented in eventListener with the `preventDefault()` method
  - A new page is NOT loaded, everything is handled in the JavaScript: single page application

```
document.querySelector('form').addEventListener('submit', event => {
    event.preventDefault();
    console.log('submit');
})
```

References

- Web forms — Collecting data from users
  - https://developer.mozilla.org/en-US/docs/Learn/Forms
- Basic native form controls
  - https://developer.mozilla.org/en-US/docs/Learn/Forms/Basic_native_form_controls
- The HTML5 input types
  - https://developer.mozilla.org/en-US/docs/Learn/Forms/HTML5_input_types

References

- Web Engineering SS20 - TU Wien, prof. Jürgen Cito, https://web-engineering-tuwien.github.io/
- Async and defer
  - Efficiently load JavaScript with defer and async, Flavio Copes, https://flaviocopes.com/javascript-async-defer/
  - https://hacks.mozilla.org/2017/09/building-the-dom-faster-speculative-parsing-async-defer-and-preload/

License

- These slides are distributed under a Creative Commons license "**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**"
- **You are free to:**
  - **Share** — copy and redistribute the material in any medium or format
  - **Adapt** — remix, transform, and build upon the material
  - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
  - **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - **NonCommercial** — You may not use the material for commercial purposes.
  - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
  - **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
- https://creativecommons.org/licenses/by-nc-sa/4.0/