Document Object Model (DOM) interactions in JS

Basic Interaction

- `alert()`: pauses script until 'ok' is pressed(creates a modal)
- `prompt("text to show", <default input></default>)`: same as alert but *returns* the input value and allows extra text
- `confirm()`: same as alert *returns* a boolean (through 2 buttons of "ok" and "cancel") and allows text
- The styling and position of these modals are determined by browser

Document Object Model

- represents all page content as objects that can be modified
- `document` object
- each element inside a HTML page is an element
- `html` -> `body` -> `h1` etc
- *newline (\n) and space (_) are both valid characters*
- having space inside forms a `text` object with spaces as values
- comments will also be object properties
- **general rule: if something is in the html, it will be in the DOM tree**
- *exceptions:*
  - spaces and newlines before `<head>` are ignored for historical reasons
  - anything after `</body>` it will auto move to before the body (HTML spec requires all content be inside body)
- `null` in DOM world means does not exist

CSS Object Model

- used with DOM to modify style rules
- rarely used

Browser object model

- additional objects/methods provided by browser
- `navigator`: background info on browser and OS
- `location`: read current URL and can redirect
- **it is part of HTML specification**

DOM collections:

- **not arrays, but array-like**
- **read only**
- **represents current state (real-time)**
- they are iterables, so `for(let node of col)` works
- cannot use array methods, but *can* use `Array.from()` to transform into a variable

DOM traversal

- `<html>`: `document.docmentElement`
- `<body></body>`: `document.body`
- `<head></head>`: `document.head`
- `element.childNodes`: collection of all **child** nodes
- `element.firstChild`: `element.childNodes[0]`
- `element.lastChild`: `element.childNodes[element.childNodes.length - 1]`
- `element.hasChildNodes()`: does it have children? (bool)
- `nextSibling`: has the sibling to the right
- `previousSibling`: has the sibling to the left
- `parentNode`: parent... of the node

Element only traversal (excluding comments, text, etc)

- `children`: collection of all child *elements*
- `parentElement`: most likely same as `parentNode`
    - `document.body.parentNode`: document
    - `document.body.parentElement`: null
    - document is not an element
- `previousElementSibling`
- `nextElementSibling`
- `firstElementChild`
- `lastElementChild`

Element search

- `document.getElementById`: searches `document` regardless of where it is called (O(n) most likely)
    - *not live*
- all element ids become variables (or `window['id']`) in the global scope, unless a variable is defined with the same name (**not recommended**)
- `document.querySelectorAll(cssSelector)`
    - returns all **elements** matching the selector
    - *not live*
    - can use pseudo classes as well
- `elem.matches(css)`: sees if the element has the selector
- `elem.closest(css)`: finds the closest *ancestor* with matching selector
- `getElementsBy`: (PLURAL) - all recursively searches
    - `elem.getElementsByTagName(tag)`: return collection with the right tag e.g. `div`
    - `elem.getElementsByClassName(name)`: return collection
    - `document.getElementsByName(name)`: search `name` attr of elements, document wide
- `elementA.contains(elementB)`: check if elA == elB or if it is a descendant

Node properties

- all DOM nodes correspond to a specific built-in class, but they all inherit from the same class
- `EventTarget`: the root class that everyone inherits from
- `Node`: also "abstract" - provides core tree functionality to all nodes
    - inherits `EventTarget`: all DOM nodes can support "events"
- `Element`: base class for DOM elements - provides element-level navigation

- `HTMLElement`: basic class for all HTML element
- all DOM nodes are regular JS objects - use prototype-based classes for inheritance
- `nodeType`: check the type of node
    - `document.body.nodeType //element`
    - returns a number, each number corresponds to a element

Tag: nodeName and tagName

- use to read the tag name
- `nodeName`: works for any node (same result if used on elements as tagName)
- `tagName`: only Element nodes

innerHTML

- access HTML inside the element *as a string*
- **allowed to change using this property getter**
- `document.body.innerHTML = "replace everything"`
- can technically do `...innerHTML += "more"`
    - *this actually removes all previous contents, and rewrites the new content entirely*
    - not good if the innerHTML contains imports (everything reloads)

outerHTML

- `innerHTML` + the current element itself
- when setting `outerHTML:
    - the old element is removed
    - the new element is set in its place
    - *if the element was saved to a variable, it is **not** updated*

nodeValue/data

- `innerHTML` only valid for elements
- can use `nodeValue` or `data` to access textnode values
    - these two are the same

textContent

- returns the *text* inside elements (strip all tags)
- `element.textContent`
- rarely used for reading, much more common for writing (safer)
- setting `innerHTML` will parse the HTML as well
- setting `textContent` will treat it as string

hidden

- returns a boolean for visibility of element
- same as `style="display:none"`

Additional props

- `value`: for input, select, textarea etc
- `href`: for anchors
- `id`: value of id attribute

Properties vs Attributes

- all DOM objects are JS objects - have regular properties
- standard HTML tag attributes will be created as object properties
- can set custom tag attributes: attributes (these will not be created as objects, need special accessors)
    - `elem.hasAttribute(name)` – checks for existence.
    - `elem.getAttribute(name)` – gets the value.
    - `elem.setAttribute(name, value)` – sets the value.
    - `elem.removeAttribute(name)` – removes the attribute.
- attribute rules:
    - case-insensitive (id === ID)
    - value must be strings
- updating attributes that are regular DOM properties will work (and vice versa)
- attribute type *not* always the same as prop tyle
    - the `style` attribute is a string
    - the `style` prop is an object
    - the `href` attribute is relative
    - the `href` prop is full url

Reserved Attributes

- all attributes starting with `data-` are reserved for programmers and can be accessed as a **prop** `element.dataset.prop_name`
- any multiword attributes will be converted into camel case
- ```
  <body data-about="Elephants">
  <script>
    alert(document.body.dataset.about); // Elephants
  </script>
  ```

Document Modifications

- `document.createElement(tag)`: returns a new element node
- `document.createTextNode(text)`: returns a text node
- `document.node.append(el)`: appends an element node
- `node.append(...nodes or strings)`: append nodes or strings at the end of node
- `node.prepend(...nodes or strings)`: insert nodes or strings at the beginning of node
- `node.before(...nodes or strings)`: insert nodes or strings before node
- `node.after(...nodes or strings)`: insert nodes or strings after node
- `node.replaceWith(...nodes or strings)`: replaces node with the given nodes or strings

HTML lifecycle
  ```

- `DOMContentLoaded`: HTML is done loading, DOM tree is built (external src may *not* be loaded yet)
- `load`: **everything** is loaded (`window.onload`)
- `beforeunload/unload`: user leaving the page
- can add `document.addEventListener` to run things at different periods

HTML readyState

- if unsure whether the document is ready or not
- `document.readState`: 3 possible states
    - "loading": loading
    - "interactive": document was fully read
    - "complete": all resources are loaded as well

Definitions

- Modal: visitor cannot interact with the rest of the page unless this component is dealt with
- Host Environment: JS works with many things, the "thing" its working on is the host environment (the host provides additional objects and functions, e.g. browser)
- child nodes: direct children of depth 1
- descendants: all elements nested within one element
- ancestor: parent, and everything above parent
- sibling: children of the same parent(e.g. body <-> head)