General Assembly

DOM + jQuery

- Class Rules reminder
- Thanks for Exit Tickets
- [ask class]: Is whiteboard usage ok?
    - Joke: Am I the only one who thinks Whiteboards are truly Re-Markable!?

JS1

Objects + JSON Recap

- Access & Assign Object values
- Object iteration (`for...in` & `Object.keys`)
- OOP
- Data Privacy with closures
- JSON

JS1

Sticky Notes Homework

```
<input id="colour" placeholder="Sticky Colour" />
<input id="content" placeholder="Note Message" />
<button>Create</button>
```

```
var noteCount = 1;
document.querySelector('button').addEventListener('click', function() {

  var colour = document.getElementById('colour').value
  var content = noteCount + '. ' + document.getElementById('content').value

  var newNoteElement = document.createElement('div')
  newNoteElement.className = 'box'
  newNoteElement.style.backgroundColor = colour
  newNoteElement.innerHTML = content

  document.querySelector('.container').appendChild(newNoteElement)

  noteCount++
});
```

- Notice no `<form>` element
  - Stops `<button>` from submitting by default
  - Could also use `type="button"`
- One issue: global `noteCount`!
  - Fix: Wrap it all in an IIFE

JS1

Objectives

- Understand What is The DOM
- Understand how & when to load JS
- Get familiar with DevTools
- Practice traversing The DOM
- Understand what jQuery is / is not useful for
- Practice creating and inserting DOM elements
- Intro to DOM events

- We start pulling together our knowledge for real-world usage now

- Not this Dom, unfortunately

DOM + jQuery

The DOM

- "*Document Object Model*"
- Bridge between JS & HTML

```
<html>
  <head>
    <title>Tasty Fruit</title>
  </head>
  <body>
    <img id="fruits" src="http://bit.ly/1V62kTZ">
    <h1>Some <i>nice</i> fruits</h1>
  </body>
</html>
```

- This is HTML
- [on whiteboard]: Tree
  - This is the DOM representation of the HTML
- *nodes* are different points in the tree
  - Can be an element (eg; `img`, `h1`), text, etc
  - Can have *attributes* (eg; `id`, `src`)

```
<html>
  <head>
    <title>Tasty Fruit</title>
    <script>
      console.log('Hello world')
    </script>
  </head>
  <body>
    <img id="fruits" src="http://bit.ly/1V62kTZ">
    <h1>Some <i>nice</i> fruits</h1>
  </body>
</html>
```

```
> Hello world
```

- Use a `<script>` element for JS
- Browser executes (aka; parses / reads) DOM from top-to-bottom, the same as JS
  - But there's no *hoisting* for DOM like in JS
- When sees `<script>`, changes to JS mode
  - Once done *executing* JS, continues on with DOM

```
<html>
  <head>
    <title>Tasty Fruit</title>
    <script>
      console.log(document.getElementById('fruits'))
    </script>
  </head>
  <body>
    <img id="fruits" src="http://bit.ly/1V62kTZ">
    <h1>Some <i>nice</i> fruits</h1>
  </body>
</html>
```

```
> null
```

- output: `null`
  - `#fruits` element is not yet available (DOM hasn't been parsed up to that point)

```
<html>
  <head>
    <title>Tasty Fruit</title>
    <script>
      document.addEventListener('DOMContentLoaded', function() {
        console.log(document.getElementById('fruits'))
      })
    </script>
  </head>
  <body>
    <img id="fruits" src="http://bit.ly/1V62kTZ">
    <h1>Some <i>nice</i> fruits</h1>
  </body>
</html>
```

```
> <img id="image" src="http://bit.ly/1V62kTZ">
```

- `'DOMContentLoaded'` event occurs *after* all the DOM is parsed
- Output now is the expected `#fruits` element
- Benefits:
  - Creates a function scope for us to use (avoids global variables!)
  - JS inside function isn't executed until after DOM parsed
- Down Sides:
  - Still has to *read* / parse all the JS, but only actually executes the JS outside the function
    - Can take time
  - User might not see DOM for a while if JS takes long time to parse

```
<html>
  <head>
    <title>Tasty Fruit</title>
  </head>
  <body>
    <img id="fruits" src="http://bit.ly/1V62kTZ">
    <h1>Some <i>nice</i> fruits</h1>
    <script>
      document.addEventListener('DOMContentLoaded', function() {
        console.log(document.getElementById('fruits'))
      })
    </script>
  </body>
</html>
```

```
> <img id="image" src="http://bit.ly/1V62kTZ">
```

- Move `<script>` to before `</body>`
- DOM is all parsed by then (except `</body></html>`, but that's ok)
    - User can see DOM before JS starts executing

```
<html>
  <head>
    <title>Tasty Fruit</title>
  </head>
  <body>
    <img id="fruits" src="http://bit.ly/1V62kTZ">
    <h1>Some <i>nice</i> fruits</h1>
    <script>
      console.log(document.getElementById('fruits'))
    </script>
  </body>
</html>
```

```
> <img id="image" src="http://bit.ly/1V62kTZ">
```

- Benefits:
  - Can get rid of the event listener now
  - DOM elements are all created
  - User sees DOM output before JS is executed by browser
- Some *rare* cases you need to have JS in `<head>`
  - Almost always should be just before `</body>`

```
<html>
  <head>
    <title>Tasty Fruit</title>
  </head>
  <body>
    <img id="fruits" src="http://bit.ly/1V62kTZ">
    <h1>Some <i>nice</i> fruits</h1>
    <script src="js/app.js"></script>
  </body>
</html>
```

File: `js/app.js`

```
console.log(document.getElementById('fruits'))
```

- Can pull JS out into separate file
  - Like we've seen so far with our homeworks
- Follows same rules: top-to-bottom DOM, switch to JS when it encounters any, then back to DOM when done with JS
- One extra step: Has to download the `js/app.js` file now too
- Benefits:
  - Browser can cache the file, doesn't have to re-download

The DOM

`document`

- This is how you access the DOM
- It is an object
- With methods, just like from Objects lesson
- Internally, it uses something similar to the Module pattern
  - You can't access the internal representation of DOM
  - But you *can* access via functions `document.getElementById()`, etc

The DOM

bit.ly/tasty-fruit

- Save this example code
- Open it in Chrome

The DOM

DevTools

- [Show class] `fruits.html` in browser
- Chrome DevTools == best debugging for websites
  - So good that node has recently incorporated them!
  - F12 to open on sane platforms.
  - Right Click > Inspect Element, or Tools > Developer Tools on insane platforms (such as OSX)
- Console == same as Node REPL, but on any website
- Elements == the current state of the DOM (looks like HTML)
- Sources == all the JS loaded by the page

The DOM

`document.getElementById(<id>)`

- [class] try these out in Console tab...

The DOM

`document.getElementById(<id>)`

For example:

```
document.getElementById('fruits')
```

Will find:

```
<img id="fruits" src="http://bit.ly/1V62kTZ" />
```

- Only a single element

The DOM

`document.getElementsByClassName(<className>)`

The DOM

`document.getElementsByClassName(<className>)`

For example:

```
document.getElementsByClassName('highlight')
```

Will find:

```
[
  <span class="highlight">nice</span>,
  <span class="highlight">Blood</span>
]
```

- A collection of elements

The DOM

`document.getElementsByTagName(<tagName>)`

The DOM

`document.getElementsByTagName(<tagName>)`

For example:

```
document.getElementsByTagName('li')
```

Will find:

```
[
  <li>Apples</li>,
  <li>Pears</li>,
  <li><span class="highlight">Blood</span> Oranges</li>
]
```

- A collection of elements

The DOM

`document.querySelector(<cssSelector>)`

- [vocab]: querySelector
- Does the same as all the above combined, using css selectors

The DOM

`document.querySelector(<cssSelector>)`

For example:

```
document.querySelector('.highlight')
```

Will find:

```
<span class="highlight">nice</span>
```

- Only ever a single element
  - Even if multiple match, only the first is returned
- Another example...

The DOM

`document.querySelector(<cssSelector>)`

For example:

```
document.querySelector('#fruits')
```

Will find:

```
<img id="fruits" src="http://bit.ly/1V62kTZ" />
```

- Don't really need `document.getElementById` anymore

The DOM

`document.querySelectorAll(<cssSelector>)`

- [vocab]: querySelectorAll
- Same as `.querySelector`, but returns a collection of *all* items that match

The DOM

`document.querySelectorAll(<cssSelector>)`

For example:

```
document.querySelectorAll('.highlight')
```

Will find:

```
[
  <span class="highlight">nice</span>
  <span class="highlight">Blood</span>
]
```

- Another example...

The DOM

`document.querySelectorAll(<cssSelector>)`

For example:

```
document.querySelectorAll('li')
```

Will find:

```
[
  <li>Apples</li>,
  <li>Pears</li>,
  <li><span class="highlight">Blood</span> Oranges</li>
]
```

- `document.querySelectorAll` is what you want to use most of the time

The DOM

Collections

NodeLists (mdn.io/nodeList)

```
Array.from()
```

- The collections above are not actually Arrays
- But sometimes look like them:
  - Has a `.length` property
  - Can access individual elements by bracket notation: `[0]`
  - Sometimes has a `.forEach()` method
- But doesn't have `.map` / `.filter` / etc
- Actually *NodeLists*
- Convert to an Array with `Array.from()`
- Example...

The DOM

Collections

NodeLists (mdn.io/nodeList)

```
Array.from(document.querySelectorAll('li'))
```

- used like `Object.keys` from earlier lesson
- Can then `.map` / `.filter` etc the result

The DOM

Traversing the Tree

```
<element>.children
```

```
<element>.childNode
```

```
<element>.parentNo
```

```
<element>.siblings
```

- [on whiteboard]: Point to tree
- Nodes can have children, and can have parents
- `.children` returns only *elements* (think; *tags*)
- `.childNodes` returns elements, text, and html comments
- `.parentNode` returns the parent node
- Try it out on a result from `document.querySelector`

The DOM

Traversing the Tree

```
<element>.children
```

```
<element>.childNode
```

```
<element>.parentNo
```

```
<element>.siblings
```

- [vocab]: children, childNodes, parentNode
- There is no `.siblings` in DOM!
- Enter jQuery...

DOM + jQuery

jQuery

- Since 2006
- v3.0 just released
- The utility belt of JS
- Smoothed JS's rough edges due to different implementations across browsers
- Not required anywhere near as much these days

DOM + jQuery

jQuery

```
<script src="https://code.jquery.com/jquery-3.0.0.min.js"></script>
```

```
jQuery
```

- Include this script in the page (near `</body>` of course!)
- Get a global variable `jQuery`

DOM + jQuery

jQuery

```
jQuery('#fruits ~ ul > li:nth-child(2)')
document.querySelectorAll('#fruits ~ ul > li:nth-child(2)')
```

```
[
  <li>Pears</li>
]
```

- Selector doesn't have to be that complex. Using it for illistrative purposes only
- Finds equivalent elements:
  - The 2nd `li` inside the first `ul` which comes after `#fruits`
- jQuery also *does not* return an array

DOM + jQuery

jQuery

```
jQuery('#fruits ~ ul > li:nth-child(2)')
```

Is equivalent to...

```
$('#fruits ~ ul > li:nth-child(2)')
```

- [on board]: `jQuery === $`
- Almost always used as `$`

DOM + jQuery

jQuery

```
$('#fruits ~ ul > li:nth-child(2)')
```

- Will use $ from now on
- jQuery adds some extra methods to result, like...

DOM + jQuery

jQuery

```
$('#fruits ~ ul > li:nth-child(2)').siblings()
```

```
[
  <li>Apples</li>,
  <li><span class="highlight">Blood</span> Oranges</li>
]
```

- Solves our `.siblings()` problem from earlier

DOM + jQuery

jQuery

```
$('#fruits ~ ul > li:nth-child(2)').siblings()
```

How?

- But, we don't really need jQuery
- [think pair share]: Using what we have now, how could we find siblings without jQuery? Psuedocode / english description is fine.

DOM + jQuery

jQuery

```
var element =
  document.querySelectorAll('#fruits ~ ul > li:nth-child(2)')[0]
var childElements = Array.from(element.parentNode.children)
var siblings = childElements.filter(function(child) {
  return child !== element
})
```

- Could rewrite

```
function(child) {
return child !== elemen
}
```

as `child => child !== element` (is the same)

DOM + jQuery

jQuery

```
function siblings(selector) {
  var element = document.querySelectorAll(selector)[0]
  var childElements = Array.from(element.parentNode.children)
  return childElements.filter(function(child) {
    return child !== element
  })
}
siblings('#fruits ~ ul > li:nth-child(2)')
```

```
[
  <li>Apples</li>,
  <li><span class="highlight">Blood</span> Oranges</li>
]
```

- Turn it into a function
- Bonus: Returns an actual Array (not a NodeList)
- Some minor diff to jQuery, but we could add those features if we needed

DOM + jQuery

jQuery

- Animation
- AJAX

- Popular uses for jQuery

DOM + jQuery

~~jQuery~~

- ~~Animation~~
- ~~AJAX~~

- Don't need jQuery most of the time
- Use built-in DOM methods
- Can build our own DOM library if we absolutely need
- Use GSAP (GreenSock) or CSS transitions for animation
- Use `.fetch()` for AJAX (will learn in later lesson)
- Not saying it's useless. If you want to; go for it.

DOM + jQuery

Creating DOM elements

- So far created all elements with HTML, which is converted to DOM elements

DOM + jQuery

Creating DOM elements

```
document.createElement
```

- [vocab] createElement
- For example...

DOM + jQuery

Creating DOM elements

```
document.createElement('li')
```

- Creates a `<li>` element
- But doesn't do anythin with it (it's not in the DOM)

DOM + jQuery

Creating DOM elements

```
var newListItem = document.createElement('li')
document.querySelector('ul').appendChild(newListItem)
```

- [vocab]: appendChild
- Append items with `.appendChild`
- Will append to the selected element
- Also [vocab]:
  - `.insertBefore()`
  - `.replaceChild()`
  - `.innerHTML`

DOM + jQuery

Events

- How we respond to interactions by the user

DOM + jQuery

Events

```
var button = document.querySelector('#some-button')
button.onclick = function() {
  console.log('button clicked')
}
```

```
> button clicked
```

- A simple on click event

DOM + jQuery

Events

```
var button = document.querySelector('#some-button')
button.onclick = function() {
  console.log('button clicked')
}

// ... later
button.onclick = function() {
  console.log('No click for you!')
}
```

```
> No click for you!
```

- Overwrote the event handler :/
- The variable `.onclick` on the DOM object overridden with a new function

DOM + jQuery

Events

```
var button = document.querySelector('#some-button')
button.addEventListener('click', function() {
  console.log('button clicked')
})
```

- We *add* an event listener
- Will not override any existing listeners

DOM + jQuery

Events

```
var button = document.querySelector('#some-button')
button.addEventListener('click', function() {
  console.log('button clicked')
})

// ... later
button.addEventListener('click', function() {
  console.log('All the clicks for you!')
})
```

```
> button clicked
> All the clicks for you!
```

DOM + jQuery

Event Object

```
var button = document.querySelector('#some-button')
button.addEventListener('click', function(event) {
  console.log('button clicked')
})
```

- Every event listener given an event object
- Is an object (everything is!)
- Has some useful things associated with it...

DOM + jQuery

Event Object

```
var button = document.querySelector('#some-button')
button.addEventListener('click', function(event) {
  console.log(event.target)
})
```

```
> <button id="some-button">Click Me</button>
```

- `.target` gives the DOM element the event occured on
- [ask class]: What use is that?
  - Event listener functions can be re-usable! DRY code!
  - For example...

DOM + jQuery

Event Object

```
var button = document.querySelector('#some-button')
var button2 = document.querySelector('#another-button')
button.addEventListener('click', function(event) {
  console.log(event.target)
})
button2.addEventListener('click', function(event) {
  console.log(event.target)
})
```

- Instead of this, repeating the function

DOM + jQuery

Event Object

```
var button = document.querySelector('#some-button')
var button2 = document.querySelector('#another-button')
var clickHandler = function(event) {
  console.log(event.target)
}
button.addEventListener('click', clickHandler)
button2.addEventListener('click', clickHandler)
```

- One function to rule them all!

DOM + jQuery

Practice Time!

- [share zip in slack]

DOM + jQuery

Homework

No homework this lesson!

JS1

Objectives

- Revisit each of the objectives on board

JS1

Next Lesson

Slackbots! :D

JS1

Questions?

JS1

Exit Tickets

http://ga.co/js1syd

- [share in Slack]

General Assembly

JS1