General Assembly

Events + State

- Mid course drinks + survey on Wednesday
    - Want pizza? (Something else?) Bring dollarydoos Wednesday
- Class Rules reminder
- Thanks for Exit Tickets
- How was everyone's weekend?
    - Mine: When my friend told me to stop acting like a flamingo I had to put my foot down.

JS1

Objectives

- Understand advanced event usage
- Use event delegation for child element events
- Introduce State Management
- Understand MVC
- Use templating to generate DOM

JS1

Slackbots Recap

- Botkit
- npm
- I write missleading instructions
- Listen for keywords and respond
- Introduced First Project (due end of June)

JS1

Events + State

- First; a recap

Events + State

Event Listeners

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

- We *can* use `.onclick`, but...

Events + State

Event Listeners

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

- [vocab]: `addEventListener`
- `.addEventListener` is better
    - Doesn't overwrite previous event listeners

Events + State

Event Object

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

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

- [vocab]: `event.target`
- `event.target` is the original node that triggered the event

Events + State

Default Event Action

- Some events don't cause the browser to do anything by default
    - eg; 'hover'
- Others *do*
    - eg; default when 'click' on an <a> tag === open that URL
    - But not *all* 'click' events do that
- Called "Cancellable Events"

Events + State

Default Event Action

```
<a id="cancel-me" href="http://mdn.io">MDN</a>
```

```
var hyperlink = document.querySelector('#cancel-me')
hyperlink.addEventListener('click', (event) => {

})
```

- Listening to a click event like normal
- Clicking the <a> now would open the URL

Events + State

Default Event Action

```html
<a id="cancel-me" href="http://mdn.io">MDN</a>
```

```javascript
var hyperlink = document.querySelector('#cancel-me')
hyperlink.addEventListener('click', (event) => {
  event.preventDefault()
})
```

- [vocab]: `event.preventDefault()`
- Clicking <a> now would do nothing
    - The *default* action has been prevented
- Let's try it...

Events + State

Default Event Action

```
var hyperlinks = document.querySelectorAll('a')
var linksArray = Array.from(hyperlinks)
linksArray.forEach((links) => {
  link.addEventListener('click', (event) => {
    event.preventDefault()
  })
})
```

- [Think Pair Share 5min] What does this do?
- Try it: Execute on any page with links then click them
- Other example: Prevent default `click` action on a `checkbox` (ie; prevent it from being checked)

Events + State

Event Dispatch

Given HTML: bit.ly/tasty-fruit

- Capture Phase
- Target Phase
- Bubble Phase

- [on board]: DOM tree
- Events don't start at Node that triggered it
  - Start at `document` and *trickle down*
  - Known as *Capture Phase*
- Once hit node that triggered it, start to Bubble
  - Known as *Bubble Phase*
  - Go back UP the DOM tree
- [vocab]: Capture/Target/Bubble phase

Event Dispatch

Capture Phase

```
var body = document.querySelector('body')
body.addEventListener('click', (event) => {
  console.log(event.target)
}, true)
```

- To handle an event, set 3rd param of `addEventListener` to `true`
- This example captures *any* `click` on *any* element in the `<body>`
- Behaves the same as before
- But will be executed before other phases

Event Dispatch

Target Phase

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

- Same as we saw earlier
- What most folks are familiar with

Event Dispatch

Bubble Phase

```
var body = document.querySelector('body')
body.addEventListener('click', (event) => {
  console.log(event.target)
})
```

- Almost same as Capture phase, but no `true` as 3rd param
- Bubble phase is *cancellable*
  - The other phases are *not* cancellable

Event Dispatch

Bubble Phase

```
var button = document.querySelector('#some-button')
button.addEventListener('click', (event) => {
  event.stopPropagation()
})
```

- [vocab]: `event.stopPropagation()`
- Cancels event from propagating further up DOM tree
- But, still executes all listeners at current level

Event Dispatch

Given HTML: bit.ly/tasty-fruit

```
document.querySelector('body').addEventListener('click', (event) => {
  console.log('clicked body')
}, true)
document.querySelector('li').addEventListener('click', (event) => {
  console.log('clicked li')
  event.stopPropagation()
})
document.querySelector('ul').addEventListener('click', (event) => {
  console.log('clicked ul')
})
```

```
> clicked body
> clicked li
```

- [Think & Pair]: What does this output?
  - Tip: Draw a DOM tree + events to visualize Prints:

```
clicked body clicked li
```

- Why is this usefull? ...

Events + State

Event Delegation

- allows capturing events on children
- So children can be removed / added as we please
    - No new listeners need to be added

```
var body = document.querySelector('body')
body.addEventListener('click', (event) => {



  console.log('clicked li, handled in body')
  console.log(event.target)

})
```

- Given same *Tasty Fruit* page
    - This example has a problem
    - [ask class] What is it?
        - Will trigger on <h2> clicks too

```
var body = document.querySelector('body')
body.addEventListener('click', (event) => {


  console.log('clicked li, handled in body')
  console.log(event.target)
})
```

```
var body = document.querySelector('body')
body.addEventListener('click', (event) => {

  if (event.target.matches('li')) {
    console.log('clicked li, handled in body')
    console.log(event.target)
  }
})
```

- [vocab]: `<Element>.matches`
- `<Element>.matches` to the rescue!
- Supports same query as `document.querySelector`
- Still a problem!
    - [Try it in Chrome console on fruits page]
    - Anyone figure out what the issue is?
        - Doesn't fire when clicking `Blood` because it is a `<span>` *inside* an `<li>`, not an `<li>` itself
        - Have find closest `<li>` in DOM

```
var body = document.querySelector('body')
body.addEventListener('click', (event) => {
  var closestMatch = closest(event.target, 'li')
  if (closestMatch) {
    console.log('clicked li, handled in body')
    console.log(closestMatch)
  }
})
```

- Desired outcome leads to...

```
var body = document.querySelector('body')
body.addEventListener('click', (event) => {

  var closestMatch = closest(event.target, 'li')
  if (closestMatch) {
    console.log('clicked li, handled in body')
    console.log(closestMatch)
  }

})
function closest(element, query) {
  while (element !== document) {
    if (element.matches(query)) {
      return element
    }
    element = element.parentNode
  }
  return null
}
```

- [vocab]: `closest`
- [Walk through each line]: Use DOM tree to illustrate

```
var body = document.querySelector('body')
body.addEventListener('click', (event) => {

  var closestMatch = closest(event.target, 'li')
  if (closestMatch) {
    console.log('clicked li, handled in body')
    console.log(closestMatch)
  }

})
function closest(element, query) {
  while (element !== document) {
    if (element.matches(query)) {
      return element
    }
    element = element.parentNode
  }
  return null
}
```

- We can make this chunk reusabled

```
function delegate(selector, eventName, targetSelector, listener) {
  var delegatedTo = document.querySelector(selector)
  delegatedTo.addEventListener(eventName, (event) => {
    var closestMatch = closest(event.target, targetSelector)
    if (closestMatch) {
      event.delegateTarget = closestMatch
      listener(event)
    }
  })
}
```

```
function closest(element, query) {
  // ...
}
```

- [vocab]: `delegate` / `event.delegateTarget`
- `closest` hasn't changed
- `delegate` is a helper function derived from previous snippet
- `event.delegateTarget` to compliment `event.target`
- Simplifies code

Event Delegation

1. Save this page bit.ly/tasty-fruit
2. Add the script as per bit.ly/js1-utilities
3. Use the DevTools Console to an event listener to `<body>` which listens for clicks on `<li>`s *and their children*
4. Check your solution works when clicking *"Blood"* in *"Blood Orange"*

Event Delegation

```
delegate('body', 'click', 'li', (event) => {
  console.log('clicked li, handled in body')
  console.log(event.delegateTarget)
})
```

- Listens on `body` for clicks anywhere within an `li`.
- Prints out the `li` which had the event

Event Delegation

```
$('body').on('click', 'li', function (event) {
  console.log('clicked li, handled in body')
  console.log(this)
})
```

- This is the equivalent in jQuery
    - But we don't need to pull in all of jQuery's bulk!

- Om nom nom nom

Event Delegation

```
delegate('body', 'click', 'li', (event) => {
  console.log('clicked li, handled in body')
  console.log(event.delegateTarget)
})
```

- Our new `delegate` method is nice, but why?
- Only have to add single lisener, not once per child element
  - Simplifies code
  - DRY
  - `document.createElement` / `ulEl.appendChild` - listeners are already listening
- Don't rely on DOM as single source of truth...

Events + State

State

- [vocab]: "State", "State Transition", "Events"
- *State* is how your application looks at a particular time
- A *State Transition* is moving from one state to another
- *Events* cause state transitions

State

[Light off]

- Example from first lesson; psuedocode
- Current state of application; light is off

State

```
[Light off] -> (button pressed) -> [Light on]
[Light on]  -> (button pressed) -> [Light off]
```

- Event occurs, transitions to new state; light is on
- Same event triggers both state transitions
- Needs a conditional to check which transition to do
- What data do we check in conditional?

Given this HTML:

```html
<div id='container'>
  <div id='light'>off</div>
  <button>Click me</button>
</div>
```

- [on whiteboard] make a copy

```
var button = document.querySelector('button')
var light = document.querySelector('#light')


button.addEventListener('click', () => {
  if (light.innerHTML === 'on') {
    light.innerHTML = 'off'
  } else {
    light.innerHTML = 'on'
  }
})
```

- Can check DOM, but is global variable, might be changed

```
var button = document.querySelector('button')
var light = document.querySelector('#light')
var lightState = 'off'

button.addEventListener('click', () => {
  if (lightState === 'on') {
    lightState = 'off'
  } else {
    lightState = 'on'
  }
  light.innerHTML = lightState
})
```

- Now we're keeping state in our application, not DOM

```
(function() {
  var button = document.querySelector('button')
  var light = document.querySelector('#light')
  var lightState = 'off'

  button.addEventListener('click', () => {
    if (lightState === 'on') {
      lightState = 'off'
    } else {
      lightState = 'on'
    }
    light.innerHTML = lightState
  })
})()
```

- Wrap in IIFE to keep state private
- Good, but still relies on the DOM (for `'button'` & `'#light'` elements)
  - Particularly to set `.innerHTML`
  - Let's split DOM and state management...

```
(function() {
  var button = document.querySelector('button')
  var container = document.querySelector('#container')
  var lightState = 'off'

  button.addEventListener('click', () => {
    if (lightState === 'on') {
      lightState = 'off'
    } else {
      lightState = 'on'
    }
    render(lightState, container)
  })


  function render(data, into) {
    into.innerHTML = '<div id="light">' + data + '</div>'
    into.innerHTML += '<button>Click me</button>'
  }
})()
```

- Separates render & DOM from state
- Note no `light` element selector anymore
- [ask class]: Spot the bug?
    - `button`'s event listener will go away after the first time `render()` is called

```
(function() {

  var container = document.querySelector('#container')
  var lightState = 'off'

  delegate('#container', 'click', 'button', () => {
    if (lightState === 'on') {
      lightState = 'off'
    } else {
      lightState = 'on'
    }
    render(lightState, container)
  })


  function render(data, into) {
    into.innerHTML = '<div id="light">' + data + '</div>'
    into.innerHTML += '<button>Click me</button>'
  }
})()
```

- `delegate` will persiste between `render` calls
- Note no need to get `button` element anymore either

```
(function() {

  var container = document.querySelector('#container')
  var lightState = 'off'
  var buttonText = 'Turn On'
  delegate('#container', 'click', 'button', () => {
    if (lightState === 'on') {
      lightState = 'off'
      buttonText = 'Turn On'
    } else {
      lightState = 'on'
      buttonText = 'Turn Off'
    }
    render(lightState, buttonText, container)
  })
  function render(data, button, into) {
    into.innerHTML = '<div id="light">' + data + '</div>'
    into.innerHTML += '<button>' + button + '</button>'
  }
})()
```

- Allows setting other state easily
- no DOM manipulation required!

```
(function() {

  var container = document.querySelector('#container')
  var state = {light: 'off', button: 'Turn On'}

  delegate('#container', 'click', 'button', () => {
    if (state.light === 'on') {
      state.light = 'off'
      state.button = 'Turn On'
    } else {
      state.light = 'on'
      state.button = 'Turn Off'
    }
    render(state, container)
  })
  function render(data, into) {
    into.innerHTML = '<div id="light">' + data.light + '</div>'
    into.innerHTML += '<button>' + data.button + '</button>'
  }
})()
```

- Easier to manage state as a single object
- This split is known as MVC...

```
(function() {

  var container = document.querySelector('#container')
  var state = {light: 'off', button: 'Turn On'}

  delegate('#container', 'click', 'button', () => {
    if (state.light === 'on') {
      state.light = 'off'
      state.button = 'Turn On'
    } else {
      state.light = 'on'
      state.button = 'Turn Off'
    }
    render(state, container)
  })

  function render(data, into) {
    into.innerHTML = '<div id="light">' + data.light + '</div>'
    into.innerHTML += '<button>' + data.button + '</button>'
  }

})()
```

- [vocab]: *Model*
- Model is the state object

```javascript
(function() {

  var container = document.querySelector('#container')
  var state = {light: 'off', button: 'Turn On'}

  delegate('#container', 'click', 'button', () => {
    if (state.light === 'on') {
      state.light = 'off'
      state.button = 'Turn On'
    } else {
      state.light = 'on'
      state.button = 'Turn Off'
    }
    render(state, container)
  })
  function render(data, into) {
    into.innerHTML = '<div id="light">' + data.light + '</div>'
    into.innerHTML += '<button>' + data.button + '</button>'
  }
})()
```

- [vocab]: *View*
- View is what's rendered to the user

```
(function() {

  var container = document.querySelector('#container')
  var state = {light: 'off', button: 'Turn On'}

  delegate('#container', 'click', 'button', () => {
    if (state.light === 'on') {
      state.light = 'off'
      state.button = 'Turn On'
    } else {
      state.light = 'on'
      state.button = 'Turn Off'
    }
    render(state, container)
  })
  function render(data, into) {
    into.innerHTML = '<div id="light">' + data.light + '</div>'
    into.innerHTML += '<button>' + data.button + '</button>'
  }
})()
```

- [vocab]: *Controller*
- Controller is where Model is updated and View is re-rendered
- Enemy Of The State talk by Amy Palamountain
- Be warned: MVC is defined differently all over the place
    - Some say put M/V/C in diff files. I prefer to keep together.

Events + State

Exercise / Homework

Re-do the TODO List Homework, with these additions:

- Event delegation
- Separate State from DOM
- Use a View Template

```
(function() {

  var container = document.querySelector('#container')
  var state = {light: 'off', button: 'Turn On'}

  delegate('#container', 'click', 'button', () => {
    if (state.light === 'on') {
      state.light = 'off'
      state.button = 'Turn On'
    } else {
      state.light = 'on'
      state.button = 'Turn Off'
    }
    render(state, container)
  })

  function render(data, into) {
    into.innerHTML = '<div id="light">' + data.light + '</div>'
    into.innerHTML += '<button>' + data.button + '</button>'
  }

})()
```

- Let's look at just the render function

Events + State

Templates

```
function render(data, into) {
  into.innerHTML = '<div id="light">' + data.light + '</div>'
  into.innerHTML += '<button>' + data.button + '</button>'
}
```

- [vocab]: *Template*
- Better way to build string

Events + State

ES6 Template Strings

```
function render(data, into) {
  into.innerHTML = `<div id="light">${data.light}</div>`
  into.innerHTML += `<button>${data.button}</button>`
}
```

- Uses ` instead of ' or "
- Wrap JS expressions in ${ }

- ` pronounced: 'backtick'

Events + State

ES6 Template Strings

```
function render(data, into) {
  into.innerHTML = `
<div id="light">${data.light}</div>
<button>${data.button}</button>
`
}
```

- Supports multi line strings!

- Will include the new line at start and end of string too

Events + State

ES6 Template Strings

```
function render(data, into) {
  into.innerHTML = `
<div id="light">${data.light.toUpperCase()}</div>
<button>${data.button}</button>
`
}
```

- Inside ${ } is any Javascript expression

- Eg; Some method that returns a string

Events + State

ES6 Template Strings

```
function render(data, into) {
  into.innerHTML = `
<div id="light">${data.light === 'on' ? `ON!!!` : `OFF`}</div>
<button>${data.button}</button>
`
}
```

- Inside ${ } is any Javascript expression

- Eg; a ternary that returns a string, etc.

Events + State

Templating Libraries

- Mustache
- Handlebars.js
- Pug *(formerly Jade)*

- Beyond template strings
- Libraries that do more advanced templating

Events + State

Exercise / Homework

Re-do the TODO List Homework, with these additions:

- Event delegation
- Separate State from DOM
- Use a View Template

- [share `lesson-08-events-and-state-practice.zip`]

JS1

Objectives

- Revisit each of the objectives on board

JS1

Next Lesson

- Callbacks
- Async JS
- Intro to Promises

JS1

Questions?

JS1

Exit Tickets

http://ga.co/js1syd

- [share in Slack]

General Assembly

JS1

- Extra resources:
  - DOM Event specification (Long and technical, but really interesting!)
  - Enemy Of The State talk by Amy Palamountain
  - DOM Utility functions