IN4MATX 133: User Interface Software

Lecture 6:
DOM Manipulation &
Package Management

Today's goals

By the end of today, you should be able to…

- Describe the different roles JavaScript has in client-side and server-side development
- Explain the role of the Document Object Model (DOM)
- Write code which edits the DOM using built-in JavaScript functions and jQuery
- Describe the role of package managers in web development
- Use the Node Package Manager (NPM) to install packages

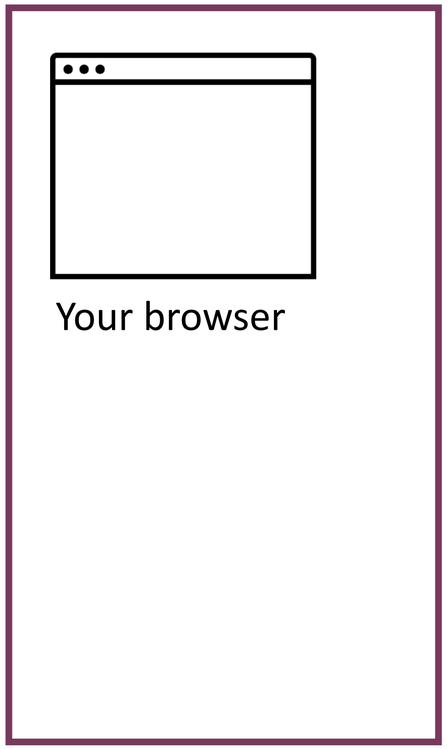
Thus far, JavaScript looks just like any other language

What about JavaScript makes it used so widely on the web?

JavaScript has many functions for editing webpages

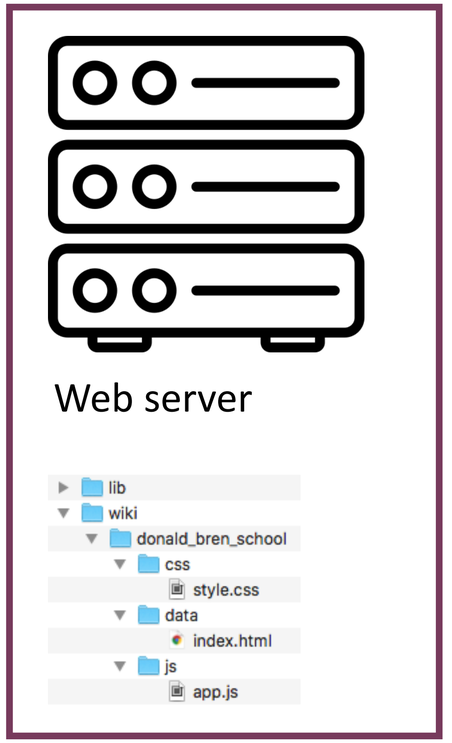
Today, JavaScript is used both client-side and server-side

Client-side and server-side JavaScript



Request

Response



Edit what's being rendered
Trigger or react to events

Navigate file system programmatically
Dynamically generate pages or views
Transport, store, or interact with data

Client-side

- Can be seen by the user
- Changes happen in real-time in the browser
- Examples: AJAX, Angular, React, Vue.js

Server-side

- Cannot be seen by the user
- Changes happen on the server in response to HTTP requests
- Examples: Node, ASP.NET

It can be confusing to follow your code if JavaScript is on both sides

Client-side object: Window

- The window object refers to the browser itself.
  You can access properties and execute functions on it

```
/* example properties */
var width = window.innerWidth;   //viewport width
var height = window.innerHeight; //viewport height

/* example functions */
window.alert("Boo!");

var confirmed = window.confirm("Are you sure?");
var quest = window.prompt("What is your quest?”);
```

Bad form, put it on your page instead

Client-side object: Window

- It's possible to use window to control the browser, but behavior varies drastically by browser

```
var xPos = window.screenX; //offset from screen edge
var yPos = window.screenY; //offset from screen edge
var scroll = window.scrollY; //pixels scrolled down
var url = window.location.href; //url for this page

window.scrollTo(0,1000); //scroll to position
window.open(url); //open a new window loading the URL
window.close(); //close window
```

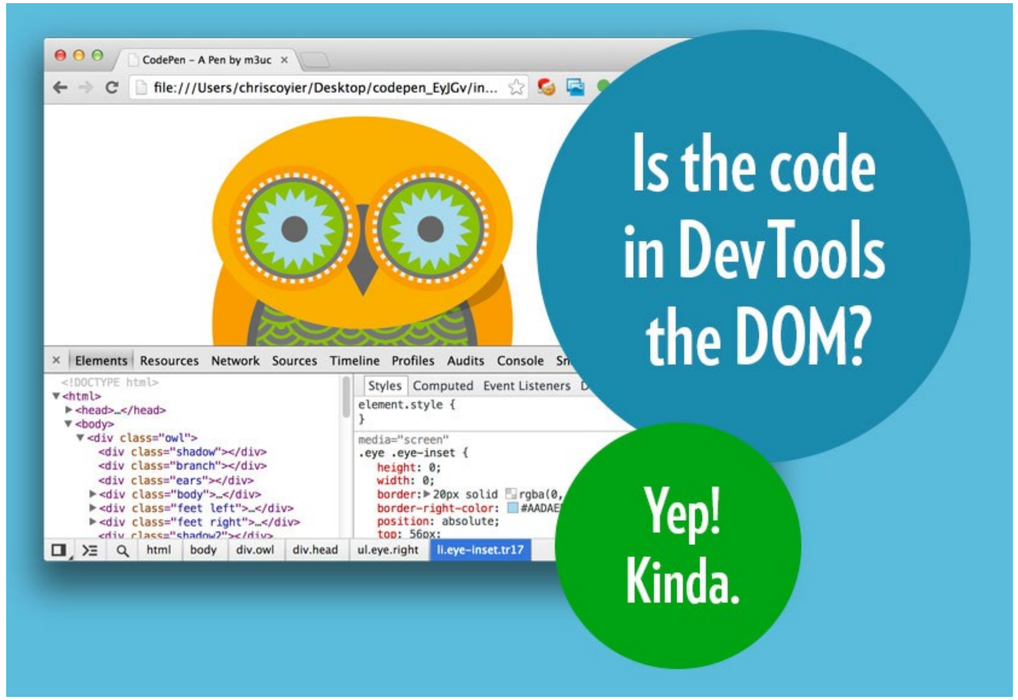
Again, better to keep your program inside the window

Server-side: has no Window

- No window object exists in server-side JavaScript
- On a server, there would be nothing to scroll to and no one to alert
- We will see server-side development more next week

HTML Document Object Model (DOM)

- “A standard for how to get, change, add, or delete HTML elements”
- “the HTML you write is parsed by the browser and turned into the DOM”
- Client-side JavaScript can then edit the DOM
  - Server-side JavaScript might specify what HTML to show, but will not edit the DOM



https://css-tricks.com/dom/

JavaScript in HTML

- Can insert inline using the `<script>` tag

```
<head>
  <script>
    alert("Hello, world!");
  </script>
</head>
<body>

</body>
```

JavaScript in HTML

- Your script should wait until after the page has loaded
  - Otherwise elements you're trying to access might not exist

```
<head>
  <script>
    function pageLoaded()  {
      alert("Page Loaded!");
    }
  </script>
</head>
<body onload="pageLoaded();">

</body>
```

JavaScript in HTML

- Functions can respond to events

```
<head>
  <script>
    function buttonClicked() {
      alert("Button Clicked!");
    }
  </script>
</head>
<body>
  <!--Bad style! Don't do this-->
  <button onclick="buttonClicked()">Click me!</button>
</body>
```

JavaScript in HTML

- Like CSS, better to load an external script

```
<head>
  <script src="js/script.js"></script>
</head>
```

Editing the DOM

- document object model
- Write into the DOM with `document.write`

```
<script>
  document.write("<h1>Hello, World!</h1>");

  var myCourse = "IN4MATX 133";
  document.write("<h1>Hello, " + myCourse + "!");
</script>
```

Selecting elements

- We can reference individual HTML elements by calling selector functions

```
//element with id="foo"
var fooElem = document.getElementById('foo');

//elements with class="row"
var rowElems = document.getElementsByClassName('row');

//<li> elements
var liElems = document.getElementsByTagName('li');
```

Selecting elements

- We can reference individual HTML elements by calling selector functions

```
/*easiest to select by reusing CSS selectors! */
var cssSelector = 'header p, .title > p';

//selects FIRST element that matches css selector
var elem = document.querySelector(cssSelector);

//matches ALL elements that match css selector
var elems = document.querySelectorAll(cssSelector);
```

Editing elements

- Properties and functions of elements can manipulate them

```
/* properties to access the CONTENT of the element */

var elem = document.querySelector('p');

var text = elem.textContent; //the text content of the elem
elem.textContent = "I'm different!"; //change the content


var html = elem.innerHTML; //content including tags
elem.innerHTML = "I'm <strong>different</strong>";

var parent = elem.parentNode; //get the parent node
```

Editing elements

- Can add/remove classes, IDs, and inline style

```
<style>/*Bad form! Just for demo*/
  .title {
    font-style: italic;
  }
</style>
<h1>Hello, IN4MATX 133!</h1>
<script>
  var elements = document.getElementsByTagName("h1");
  for(var i = 0; i < elements.length; i++) {
    elements[i].classList.add("title");
    elements[i].style.color="blue";
  }
</script>
```

Changing the DOM

```
//create a new <p> (not yet in the tree)
var newP = document.createElement('p');
newP.textContent = "I'm new!";

//create Node of textContent only
var newText = document.createTextNode("I'm blank");

var div = document.querySelector('div#container');
div.appendChild(newP); //add element INSIDE (at end)
div.appendChild(newText);

//add node BEFORE element (new, old)
div.insertBefore(document.createTextNode("First!"), newP);

//replace node (new, old)
div.replaceChild(document.createTextNode('boo'), newText);

//remove node
div.removeChild(div.querySelector('p'));
```

Validating data

- Check form fields before sending to a server
  - Provide instant feedback, reduce server back-and-forth

```
<script>
 function validateForm() {
  var x = document.forms["myForm"]["fname"].value;
  if(x==null || x=="") {
   alert("Name must be filled out");
   return false;
  }
 }
</script>
<form name="myForm" onsubmit="return validateForm()" method="post">
 <label>Name: </label>
 <input type="text" name="fname">
 <input type="submit" value="Submit">
</form>
```

Gather and use information

- Remember: this is client-side!

```
<script>
  var x = document.getElementById("demo");
  function getLocation() {
      if (navigator.geolocation) {
          navigator.geolocation.getCurrentPosition(showPosition);
      } else {
          x.innerHTML = "Geolocation is not supported by this browser.";
      }
  }
  function showPosition(position) {
      x.innerHTML = "Latitude: " + position.coords.latitude +
      "<br>Longitude: " + position.coords.longitude;
  }
</script>
```

How do we make interactive pages?

Listeners

- Can attach a listener to that method, specifying that we want to do something when that element causes an event

```
//respond to "click" events
elem.addEventListener('click', callback);

//often use an anonymous callback function
elem.addEventListener('click', function(){
  console.log('clicky clicky!');
});
```

Listeners

- Listener callback function will be passed an **event** parameter
  - Sometimes useful, but can often be ignored

```
elem.addEventListener('click', function(event) {
    console.log('You clicked me!');

    //get what was clicked;
    var clickedElem = event.target;
});
```

Remember, parameters are optional

The “target” (source) of the event

Event types

```
'click'      //mouse or button clicked

'dblclick'   //double-clicked

'hover'      //mouse hover

'focus'      //element gains focus (important!)

'mouseenter' //mouse is moved over an element

'mouseleave'  //mouse leaves the element

'mousedown' //mouse button is pressed

'keydown'   //key is pressed

//... and more!
```

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

Manipulation in pure JavaScript is verbose

- If you’re editing a lot of tags, your code can get very long and difficult to read

jQuery

- Predefines methods for manipulating the DOM
  - Include before your own script
- Remember:
  - Integrity: hashes to ensure the downloaded file matches what's expected
  - Crossorigin: some imports require credentials, anonymous requires none

```
<script
  src="https://code.jquery.com/jquery-3.3.1.min.js"
  integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="
  crossorigin="anonymous"></script>
```

jQuery selector

- Use the `jQuery()` function to select DOM elements.
  The parameter is a CSS selector String (like `querySelector`)
  - More common to use the `$()` shortcut

```
//selects element with id="foo" (e.g., <p id="foo">)
var fooElem = jQuery('#foo');

//selects all <a> elements (returns an array)
var allLinksArray = jQuery('a');

//selects element with id="foo" (e.g., <p id="foo">)
var fooElem = $('#foo');

//selects all <a> elements (returns an array)
var allLinksArray = $('a');
```

jQuery selector

- jQuery handles all CSS selectors, as well as some additional pseudoclasses

```
$('#my-div') // by id
$('div') // by type
$('.my-class') // by class
$('header, footer') // group selector
$('nav a') // descendant selector
$('p.red') // scoped selector

$('section:first') // first <section> element
                   // not a css selector!
```

jQuery and elements

- Similar to pure JavaScript, jQuery provides methods to access and modify attributes and CSS

```
//Pure JavaScript
var txt = document.querySelector('#my-div').textContent;
document.querySelector('#my-div').textContent = 'new
info!';
document.querySelector('#my-div').innerHTML = '<em>new
html!</em>';

//jQuery
var txt = $('#my-div').text();          //get the textContent
$('#my-div').text('new info!');         //change the
textContent
$('#my-div').html('<em>new html!</em>'); //change the HTML
```

jQuery and the DOM tree

```
//create an element (not in DOM)
var newP = $('<p class="new">This is a new element</p>');

//add content to DOM
$('main').append(newP); //add INSIDE, at end
$('main').append('<em>new</em>'); //can create element on the fly

$('main').prepend('<em>new</em>'); //add INSIDE, at beginning

$('main').before('<header>'); //insert BEFORE

$('footer').insertAfter('main'); //insert target AFTER param

$('main').wrap('<div class="container"></div>'); //surround

$('footer').remove(); //remove element
$('main').empty(); //remove all child elements
```

Works without closing tag

Selects existing element, so will move!

jQuery event handling

- jQuery also provides convenience methods for registering event listeners

Like `addEventListener('click')`

```
$('#button').click(function(event) {
    console.log('clicky clicky!');

    //what was clicked
    var element = $(event.target);
});
```

`event.target` is equivalent to `this`

Get as jQuery-style element to call jQuery methods on it ("wrap it")

Document ready: JavaScript

- Remember earlier: your script should wait until after the page has loaded
  - Otherwise elements you're trying to access might not exist

```
<head>
  <script>
    function pageLoaded()  {
      alert("Page Loaded!");
    }
  </script>
</head>
<body onload="pageLoaded();">

</body>
```

Document ready: jQuery

```
$(document).ready(function() {
  //your program goes here
  //this need not be an anonymous function

});
```

Document ready: jQuery

```
//shortcut: just pass the function to the jQuery method
$(function() {
  //your program goes here
  //this need not be an anonymous function

});

//shortest cut: use the abbreviated syntax
$(() => {
  //your program goes here
  //this need not be an anonymous function

});
```

Importing JavaScript

- When your script uses document ready, convention is to load it in the `<head>` tag
  - Important to think about ordering, particularly for libraries
  - e.g., import JQuery before you use it in a script

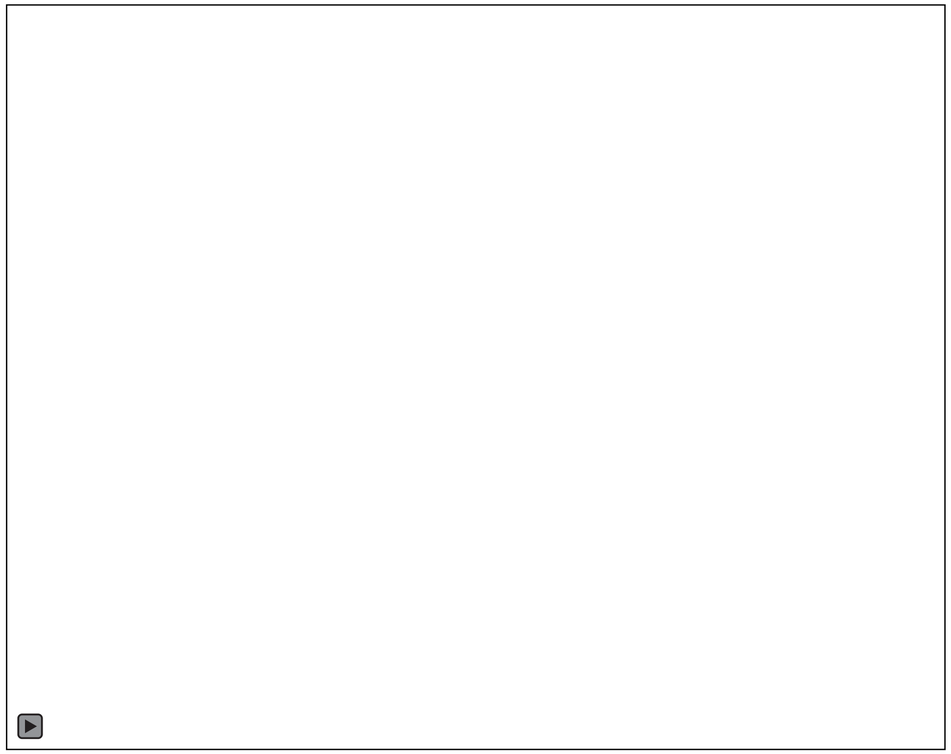
```
<head>
  <script src="https://code.jquery.com/jquery-
3.3.1.min.js"></script>
  <script src="index.js"></script>
</head>
```

jQuery effects

- jQuery supports adding transitions to modify the appearance of effects

```
$('#id').fadeIn(1000);    //fade in over 1 second
$('#id').fadeOut(500);    //fade out over 1/2 sec
$('#id').slideDown(200); //slide down over 200ms
$('#id').slideUp(500);    //slide up over 500ms
$('#id').toggle();        //toggle whether displayed
```

jQuery demo



Switching topics: Package Management

Importing packages so far

- Through content delivery networks (CDNs)
  - Pasting a “script” tag into the `<head>` of our HTML files
  - `<script src="https://cdnjs.cloudflare.com/ajax/libs/mathjs/5.2.0/math.min.js"></script>`
- Downloading from the source
  - e.g., if you downloaded Bootstrap rather than using a CDN

Package managers

- Provide an easy way to install software on your computer
  - Both new programs and libraries
- Simplify the process of updating software to the latest version
  - A challenge: packages depend on other packages,
    and often varied versions of those packages
  - Your package manager should deal with this for you
- They’re essentially app stores, except all the content is free

OS-level package managers

apt-get (Unix)

homebrew (macOS)

chocolatey (Windows)

Language-level package managers

pip (Python)

RubyGems (ruby)

npm (JavaScript)

yarn (also JavaScript)

Why are there
so many package managers?

So many package managers

- There’s some value in keeping language or domain-specific contexts
  - Certain languages interface better with certain file formats
- Most managers are driven by community efforts
  - New package manager solves some problem of a previous one
- But a lot of these are excuses; in reality, it’s often a frustrating mess

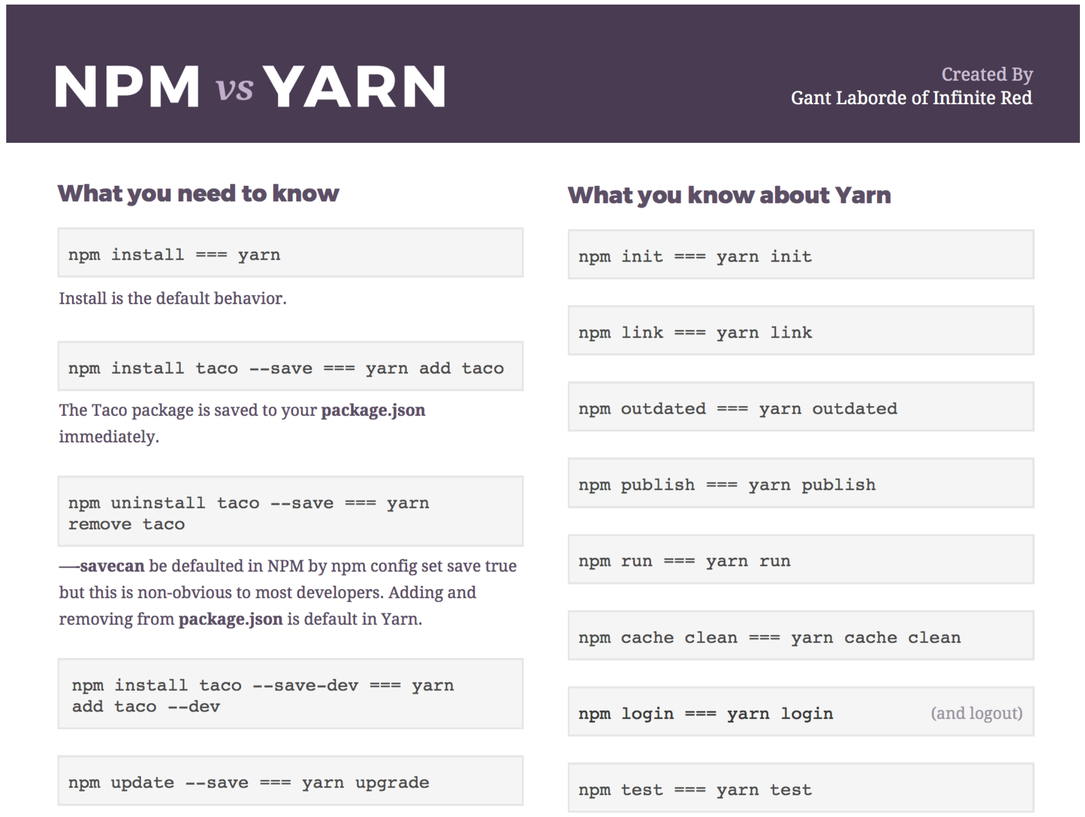
https://www.quora.com/Why-do-so-many-different-programming-languages-have-their-own-package-manager

npm and Yarn: web package managers

- npm was introduced as the package manager for Node.js (server-side JavaScript)
  - Yarn was developed later, released by Facebook as open-source
  - Uses the same registry (list of packages)
- Have a lot of useful libraries for developing webpages and web interfaces
  - Has packages for both server-side and client-side JavaScript development
- Occasionally used to install system-wide software
- package and library are often used interchangeably, which can be misleading

Yarn as an “upgrade” to npm

- Yarn intentionally uses the same concepts as npm
  - Faster, more secure
- But npm is still more widely used
  - Facebook developed Yarn, some people don’t like their involvement
  - We’ll stick to npm in this course



https://shift.infinite.red/npm-vs-yarn-cheat-sheet-8755b092e5cc

Some example web libraries

- Moment js: for managing time and timezones
  - https://momentjs.com/docs/
- Math js: for any math, unit conversion etc.
  - http://mathjs.org/docs/
- Express: for routing your website to different content (other pages or files)
  - https://expressjs.com/

npm concepts

- `package.json` file: the libraries installed in a given project
  - Kept in the root folder of your project by convention
- `package-lock.json` file
  - Used to keep track of the specific versions of other libraries that the libraries you've installed require
  - "the libraries of your libraries"
- `node_modules` folder: all the libraries you've installed in your project

npm and git

- Maybe you've seen the `.gitignore` file
  - Specifies what files should *not* be committed to your repository
- *Do* commit the `package.json` and `package-lock.json` files
  - Allows someone else to install the same package versions you used
- *Do not* commit the `node_modules` directory
  - Would be redundant; `package.json` specifies what versions to download
  - Add the folder to the `.gitignore` file

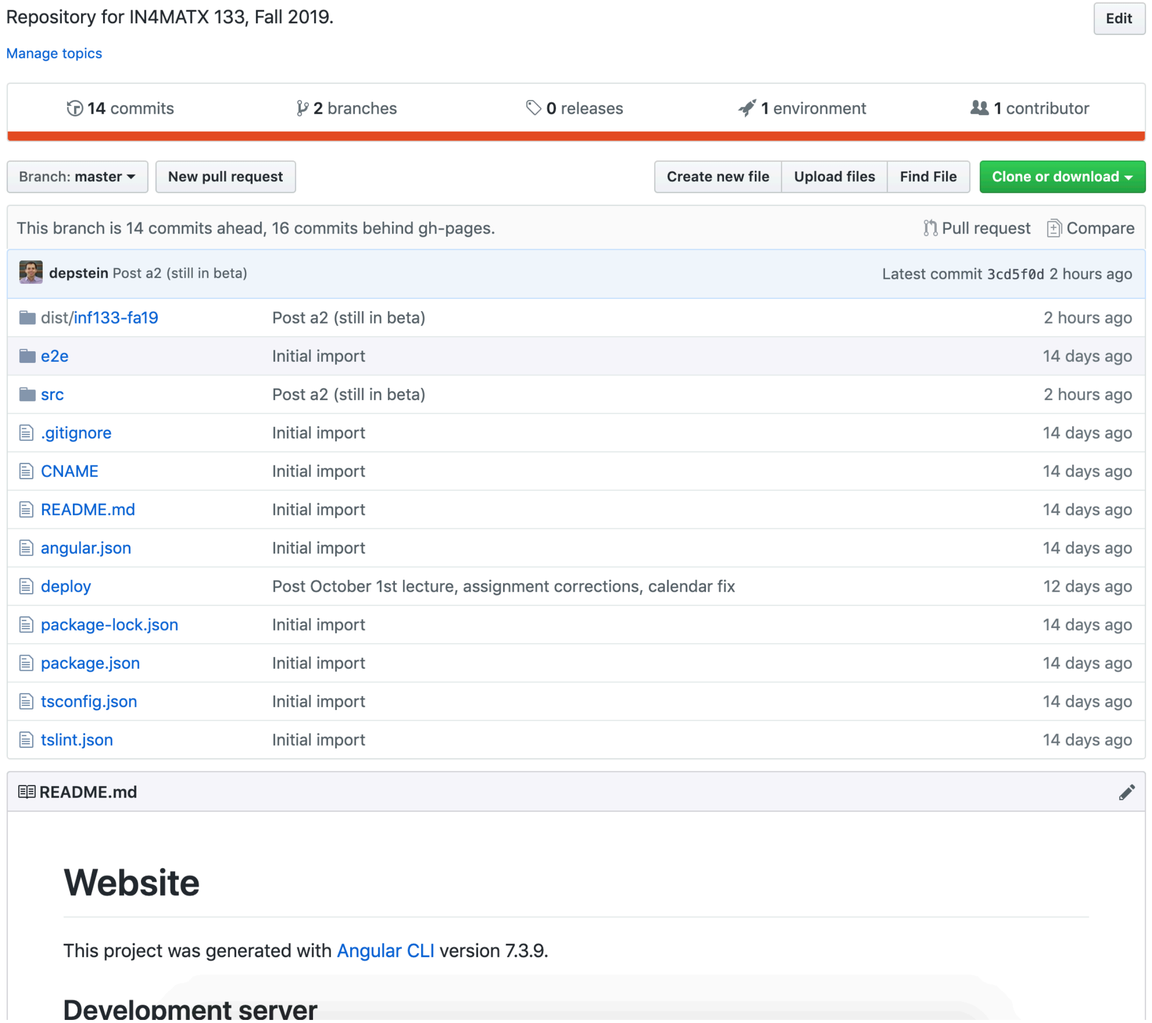
Using npm

- Runs in your operating system's command line
- Use in the root directory of your project (`cd path/to/project`)
- Install packages: `npm install packagename`
  - Will install package into your project's `node_modules/` folder
- Get the latest version of a package: `npm update`
  - Important for patching security vulnerabilities

https://www.npmjs.com/get-npm

Using npm

- Let's say we wanted to run the course webpage
  - Assume we've installed npm, then clone the repository
- Run `npm install` in the project's root directory
  - Will add all of the libraries the webpage depends on to `node_modules/`



https://github.com/uci-inf-133/inf133-fa18

Using npm

- npm can also install *global* packages, which are just software on your computer
  - `npm install -g packagename`
  - Usually programs which run via command line
- These global packages are programs rather than libraries, so they're not added to `package.json` or `node_modules/`
  - Though your project might depend on them to run
- Global packages are often redundant with OS-level package managers
- A2 only requires global packages

package.json

- Do not edit manually unless you know what you're doing!

```
{
  "name": "inf133-fa19",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
  },
  "dependencies": {
    "@angular/animations": "~7.2.0",
    "@angular/common": "~7.2.0",
    "@angular/compiler": "~7.2.0",
    "@angular/core": "~7.2.0",
    "@angular/forms": "~7.2.0",
    "@angular/platform-browser": "~7.2.0",
    "@angular/platform-browser-dynamic": "~7.2.0",
    "@angular/router": "~7.2.0",
    "bibtex-parse-js": "0.0.24",
    "component": "^1.1.0",
    "core-js": "^2.5.4",
    "moment": "^2.24.0",
    "ngx-moment": "^3.4.0",
    "rxjs": "~6.3.3",
    "tslib": "^1.10.0",
    "zone.js": "~0.8.26"
  }
}
```

~: Version number is "approximately the same as" (e.g., 7.2.X)

^: Version number is "compatible with" (e.g., 1.X.X)

Also explicit >, <, >=, =

Today's goals

By the end of today, you should be able to…

- Describe the different roles JavaScript has in client-side and server-side development
- Explain the role of the Document Object Model (DOM)
- Write code which edits the DOM using built-in JavaScript functions and jQuery
- Describe the role of package managers in web development
- Use the Node Package Manager (NPM) to install packages

Utility functions

jQuery utility functions

- jQuery includes many utility functions to simplify syntax

```
//check if an item is in an array
$.inArray(4, [3,4,3] );

//this is like .filter, but works on old browsers
$.grep( [3,4,3], function(item) {
    return item > 3;
});

//iterate over arrays or objects -- works for either!
$.each( [1,3,3], function(key, value) {
    console.log('Give me a '+value);
});

$.each( {dept:'IN4MATX',num:'133'}, function(key, value) {
    console.log(key+' name: '+value);
});
```

http://api.jquery.com/category/utilities/

Even more utilities: Lodash

- A handy library for working with basic data structures

```
_.flatten([1, [2, [3, [4]], 5]]);
// => [1, 2, [3, [4]], 5]

var zipped = _.zip(['a', 'b'], [1, 2], [true, false]);
// => [['a', 1, true], ['b', 2, false]]

_.unzip(zipped);
// => [['a', 'b'], [1, 2], [true, false]]
```

https://lodash.com/