

IN4MATX 133: User Interface Software

Lecture 6:
DOM Manipulation &
Package Management

Announcements

- Start on A2 as soon as you can
 - Should have most of the background that you need for part 1 after today, Tuesday's Typescript slides are also up
- #assignment2 channel on Slack is up

Announcements

- A few logistics reminders
 - There are too many of you to respond to direct messages on Slack, if there's something personal send us an email instead
 - We can grant extensions for assignments, but we require 24 hours notice
 - We will not grant late extension requests for future assignments

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
- Describe the role of package managers in web development
- Use the Node Package Manager (NPM) to install packages

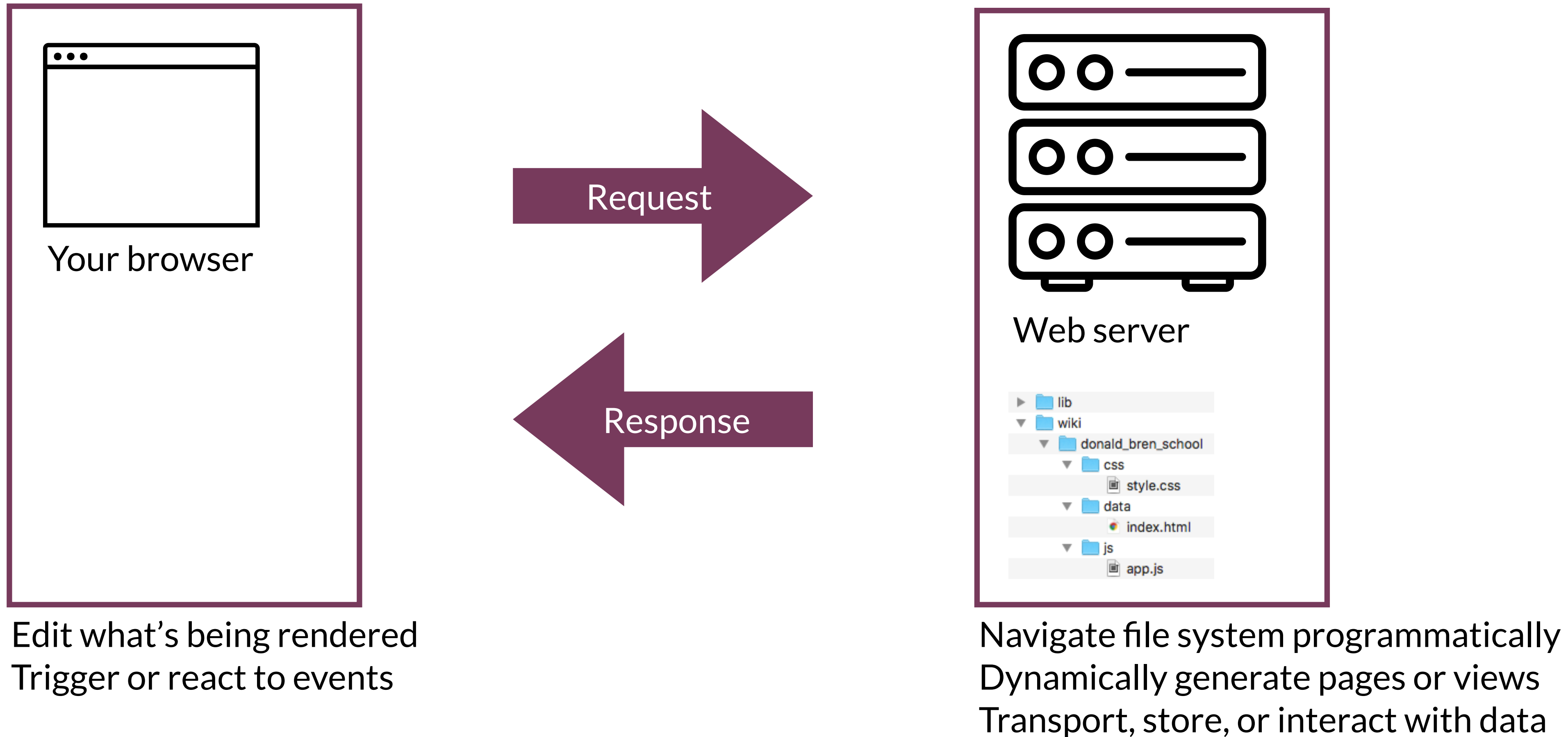
**Thus far, JavaScript looks
just like any other language**

**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



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
```



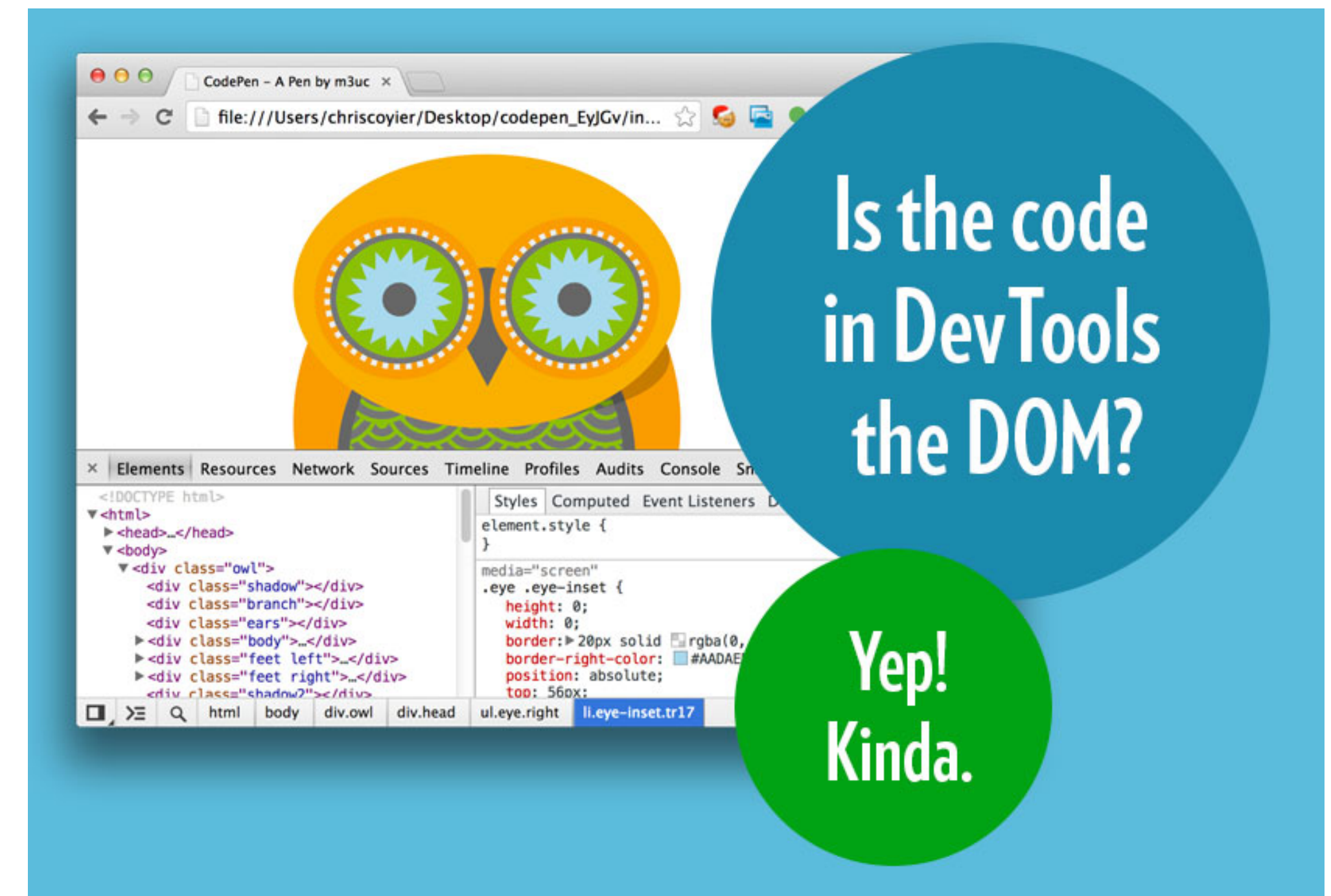
Again, better to keep your program
inside the window

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'));
```


DOM manipulation demo



Hello, IN4MATX 133!

Fill in this paragraph and add another one.

Question

 12345

Which snippet would edit the p to display “1, 2, 3, 4, 5”?

```
<p class="para" id="intro">1, 2, 3</p>
```

- ☐ A `document.getElementById("intro").append(", 4, 5");`
- ☐ B `document.getElementsByClassName("para").innerHTML = "1, 2, 3, 4, 5";`
- ☐ C `document.getElementsByTagName("p")[0].innerHTML = "1, 2, 3, 4, 5";`
- ☐ D Two of the above
- ☐ E All of the above

Which snippet would edit the p to display “1, 2, 3, 4, 5”?

```
<p class="para" id="intro">1, 2, 3</p>
```

- ☐ A `document.getElementById("intro").append(", 4, 5");`
- ☐ B `document.getElementsByClassName("para").innerHTML = "1, 2, 3, 4, 5";`
- ☐ C `document.getElementsByTagName("p")[0].innerHTML = "1, 2, 3, 4, 5";`
- ☐ D Two of the above
- ☐ E All of the above

A

0%

B

0%

C

0%

D

0%

E

0%

Question

Which snippet would edit the `p` to display “1, 2, 3, 4, 5”?

```
<p class="para" id="intro">1, 2, 3</p>
```

- ☐ A `document.getElementById("intro").append(", 4, 5");`
- ☐ B `document.getElementsByClassName("para").innerHTML = "1, 2, 3, 4, 5";`
- ☐ C `document.getElementsByTagName("p")[0].innerHTML = "1, 2, 3, 4, 5";`
- ☒ D Two of the above
- ☐ E All of the above

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!');
```

↑
Remember, parameters are optional

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

↑
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>

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: JavaScript

```
document.addEventListener( 'DOMContentLoaded', function (event) {  
    alert( "Page loaded!" );  
} );
```

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 Math.js before you use it in a script

`<head>`

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/mathjs/9.0.0/math.js"></script>
```

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

Switching topics: Package Management

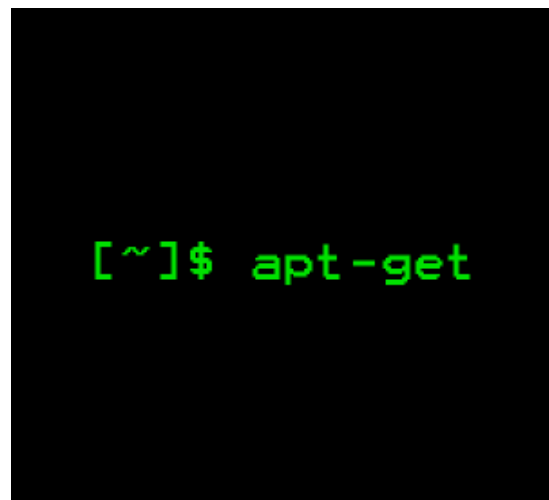
Importing packages so far

- 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/9.0.0/math.min.js"></script>`
- Downloading from the source
 - e.g., if you downloaded Bootstrap rather than using a CDN

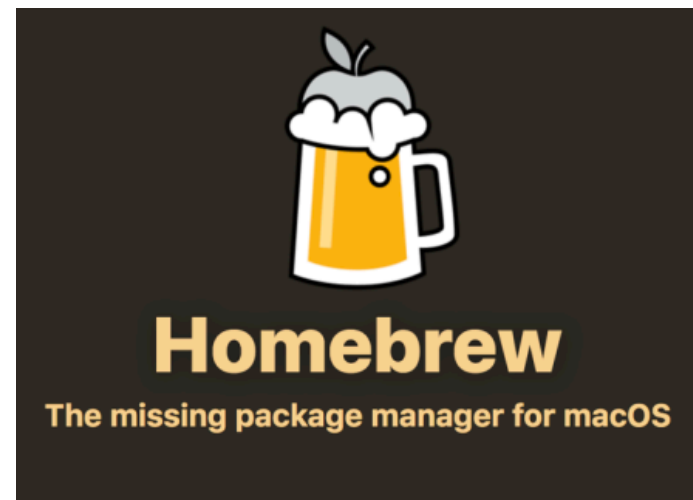
Package managers

- 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

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

NPM vs YARN

Created By
Gant Laborde of Infinite Red

What you need to know

```
npm install === yarn
```

Install is the default behavior.

```
npm install taco --save === yarn add taco
```

The Taco package is saved to your **package.json** immediately.

```
npm uninstall taco --save === yarn  
remove taco
```

—**save** can be defaulted in NPM by npm config set save true but this is non-obvious to most developers. Adding and removing from **package.json** is default in Yarn.

```
npm install taco --save-dev === yarn  
add taco --dev
```

```
npm update --save === yarn upgrade
```

What you know about Yarn

```
npm init === yarn init
```

```
npm link === yarn link
```

```
npm outdated === yarn outdated
```

```
npm publish === yarn publish
```

```
npm run === yarn run
```

```
npm cache clean === yarn cache clean
```

```
npm login === yarn login (and logout)
```

```
npm test === yarn test
```

<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/>
 - <https://dockyard.com/blog/2020/02/14/you-probably-don-t-need-moment-js-anymore>
- 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
 - Large, as it includes the packages themselves
 - Add the folder to the `.gitignore` file

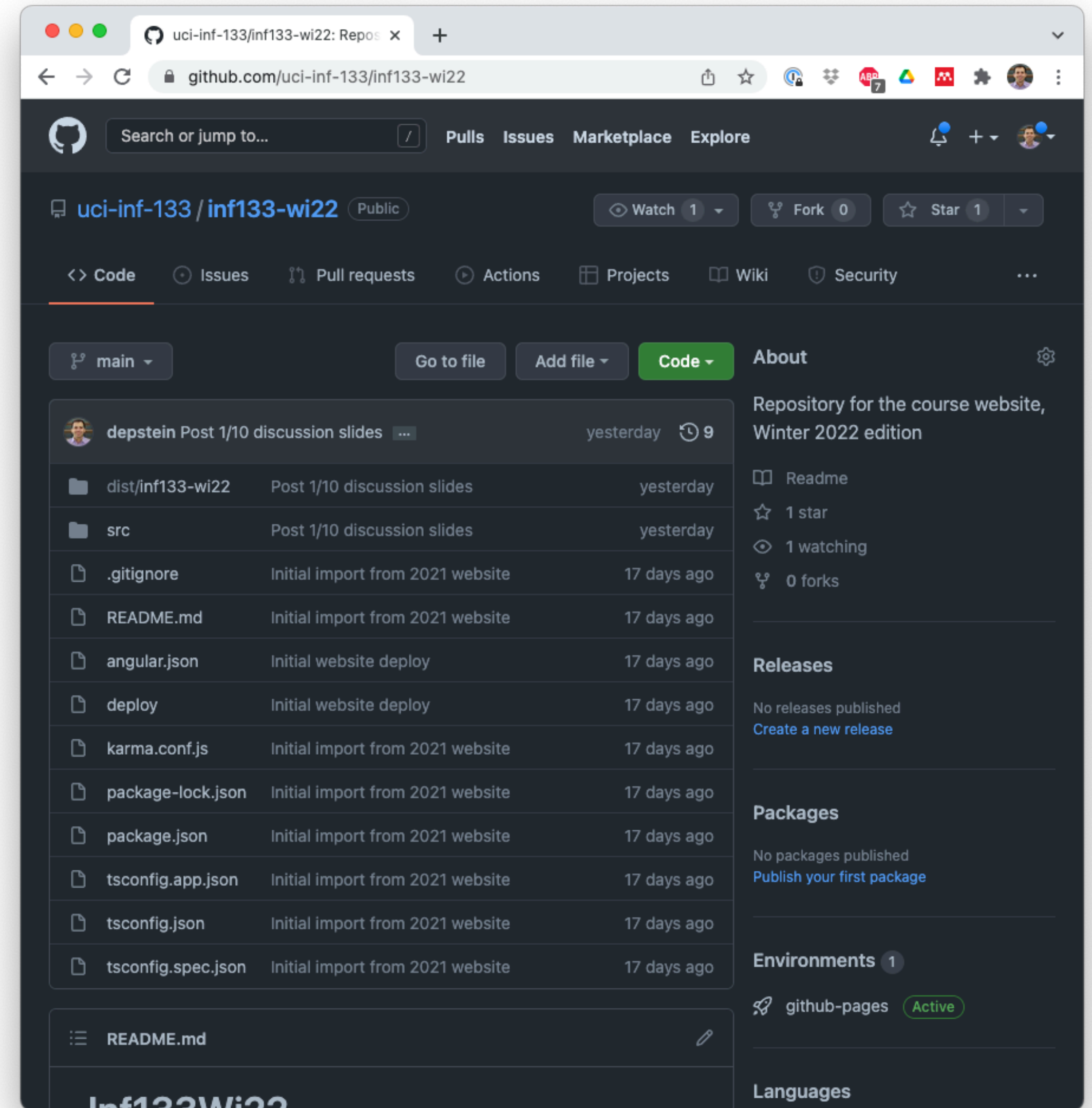
Using npm

- 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

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-wi25>



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-wi22",
  "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 >, <, >=, =

Question

Which is correct?

`npm install
packagename`

`npm install -g
packagename`

A 1, 2, 3, 4

1, 4

B 1, 2, 3

4

C 1

4

D 1, 2, 3

2, 3, 4

E 1

1, 4

- 1 Downloads package to `node_modules`
- 2 Adds package to `package.json`
- 3 Adds package's dependencies to `package-lock.json`
- 4 Package is usable as a program from the command line

	<code>npm install packagename</code>	<code>npm install -g packagename</code>	
A	1, 2, 3, 4	1, 4	1 Downloads package to node_modules
B	1, 2, 3	4	2 Adds package to package.json
C	1	4	3 Adds package's dependencies to package-lock.json
D	1, 2, 3	2, 3, 4	4 Package is usable as a program from the command line
E	1	1, 4	

A

0%

B

0%

C

0%

D

0%

E

0%

Question

Which is correct?

`npm install
packagename`

`npm install -g
packagename`

A 1, 2, 3, 4

1, 4

B 1, 2, 3

4

C 1

4

D 1, 2, 3

2, 3, 4

E 1

1, 4

- 1 Downloads package to `node_modules`
- 2 Adds package to `package.json`
- 3 Adds package's dependencies to `package-lock.json`
- 4 Package is usable as a program from the command line

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
- Describe the role of package managers in web development
- Use the Node Package Manager (NPM) to install packages

IN4MATX 133: User Interface Software

Lecture 6: DOM Manipulation & Package Management

Utility functions

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]]
```

Lo