# IN4MATX 133: User Interface Software

Lecture:
Package Management
& TypeScript

# Goals for this Lecture

## By the end of this lecture, you should be able to…

- Describe the role of package managers in web development
- Use the Node Package Manager (NPM) to install packages
- Write code which follows object-oriented principles in TypeScript
- Explain the advantages and disadvantages of using TypeScript

# 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

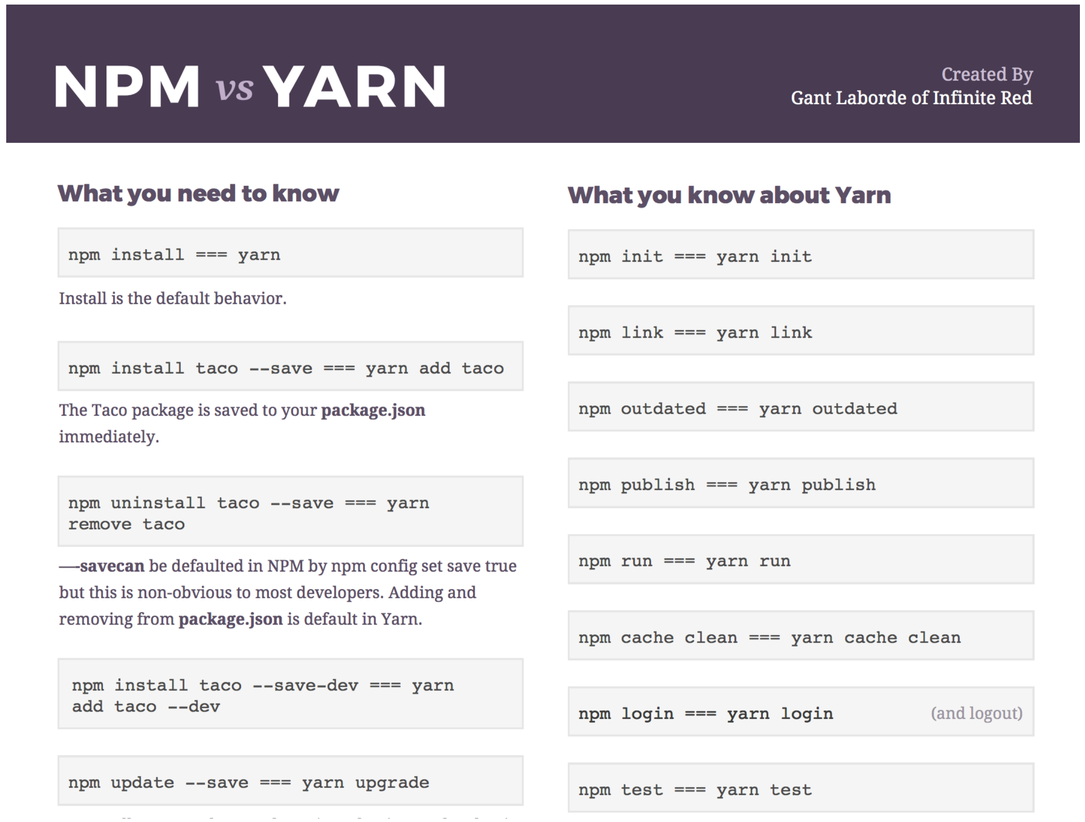
https://www.quora.com/Why-do-so-many-different-programming-languages-have-their-own-package-manager

# 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



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
```

—**savecan** 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/
- 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

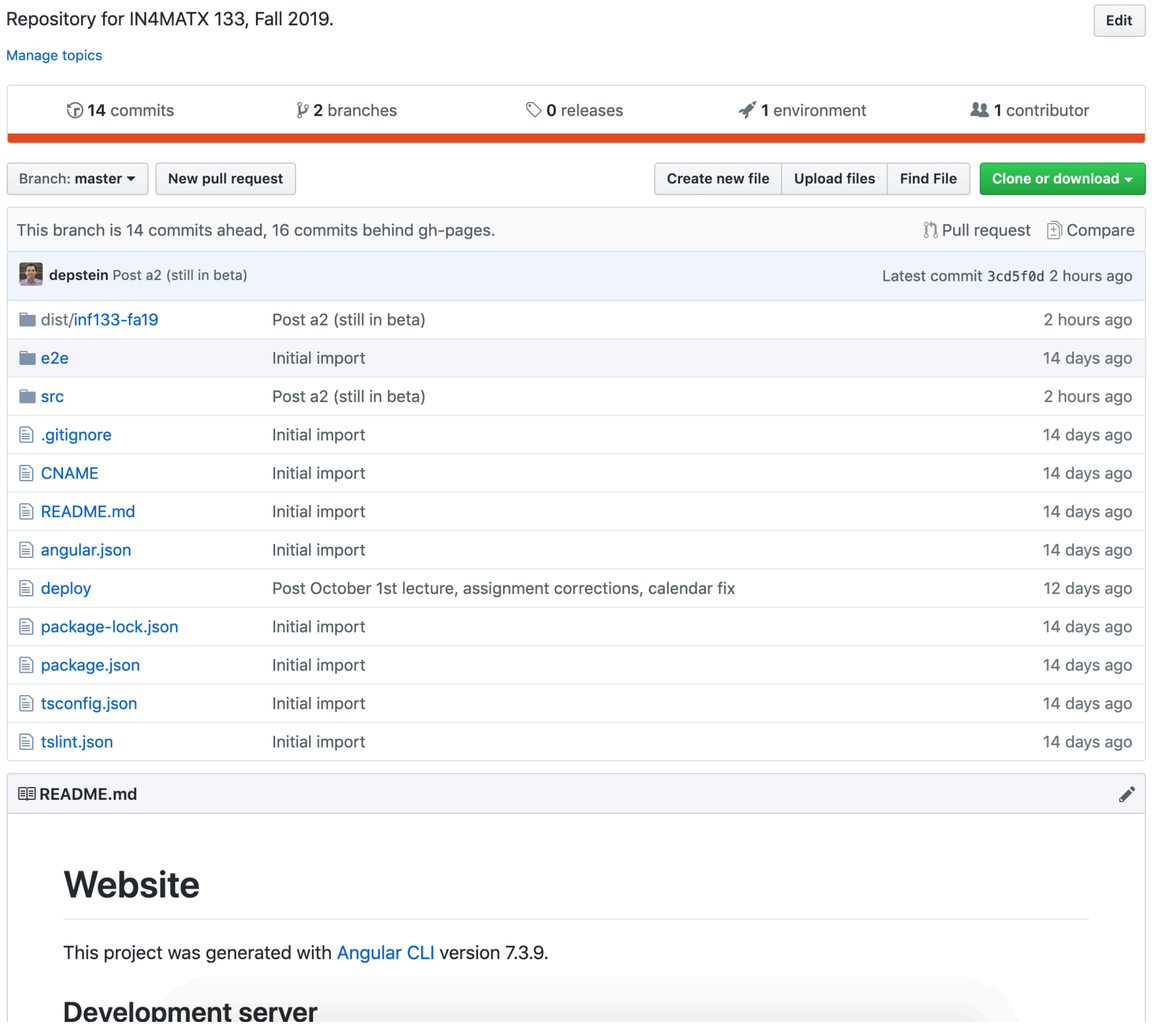
# Using npm

- 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 >, <, >=, =

# Using npm to install TypeScript

- `npm install -g typescript`
- Installs `tsc`, the TypeScript transpiler
- Could also install via HomeBrew (mac) or Chocolatey (pc)

https://www.typescriptlang.org/#download-links

# So what is TypeScript?

# About TypeScript

- Released by Microsoft in 2012
- Originally only supported in Visual Studio
    - Now in Eclipse, Sublime, Webstorm, Atom, etc.
- Latest version is TypeScript 4.0, released in Aug 2020

# About TypeScript


- Introduces static types, type checking, and objects
  - These are all optional!
- A strict superset of JavaScript
  - Syntactically-correct JavaScript will also run in TypeScript
  - Means TypeScript can use popular JavaScript libraries
- “Transcompiles” or “Transpiles” to JavaScript
  - Takes TypeScript code and converts it to equivalent JavaScript code

# Typing

- hello.ts

```
var courseNumber:number = 133;

console.log('Hello, IN4MATX ' + courseNumber + '!');
```

- Transpiling: `tsc hello.ts`

- Generates hello.js

```
var courseNumber = 133;
console.log('Hello, IN4MATX ' + courseNumber + '!');
```

# Typing

- Pre-defined types
  - Number
  - Boolean
  - String
  - Void (generally a function return type)
  - Null
  - Undefined
  - Any

# Typing

- Typing is optional

```
var courseNumber:any = 133;
var courseNumber = 133;

console.log('Hello, IN4MATX ' + courseNumber + '!');
```

# Typing

- Functions can specify argument types and return types

```
function area(shape: string, width: number, height: number)
{
    var area = width * height;
    return "I'm a " + shape + " of area " + area + "
cm^2.";
}

// "better" function
function area(shape: string, width: number,
height: number):string {
    var area:number = width * height;
    return "I'm a " + shape + " of area " + area + "
cm^2.";
}
```

# Typing

- Types enable error checking

```
// "better" function
function area(shape: string, width: number, height:
number):string {
    var area:number = width * height;
    return "I'm a " + shape + " of area " + area + "
cm^2.";
}

document.body.innerHTML = area(15, 15, 'square');
```

```
error: Argument of type ’15’ is not assignable to parameter of type ‘string’
```

# Classes

- Also just like in Java, with a constructor and methods

```
class Shape {
    area: number;
    color: string;
    name: string;

    constructor (name: string, width: number, height: number ) {
        this.name = name
        this.area = width * height;
        this.color = "pink";
    };
    shoutout() {
        return "I'm " + this.color + " " + this.name +  " with an area of " +
this.area + " cm squared.";
    }
}

var square = new Shape("square", 30, 30);
```

# Classes

- Will make a function() with prototype methods when transpiled

```
//shape.js
var Shape = /** @class */ (function () {
    function Shape(name, width, height) {
        this.name = name;
        this.area = width * height;
        this.color = "pink";
    }
    ;
    Shape.prototype.shoutout = function () {
        return "I'm " + this.color + " " + this.name + " with an area of
" + this.area + " cm squared.";
    };
    return Shape;
}());
var square = new Shape("square", 30, 30);
```

# tsconfig.json

- Indicates a TypeScript project
  - Indicates what files or folders to transpile
  - Pick transpiler options, such as whether to remove comments
  - Specify what version of ECMAScript to transpile to
  - `tsc --project tsconfig.json`

https://www.typescriptlang.org/docs/handbook/tsconfig-json.html

# Benefits of TypeScript

- Type checking
  - Transpiler can show warnings or errors *before* code is executed
  - Because your editor knows the types,
    it can autocomplete methods and API features
  - Easier to refactor code
- Object-oriented
  - Easier to manage/maintain large code bases
  - Simple enough to use; same principles as Java and other OO languages

# Drawbacks of TypeScript

- Transpiling is occasionally a pain
    - Can be slow for large projects
- Might not work with new JavaScript libraries out of the gate
    - It took a little while for TypeScript to interface nicely with Angular and React, for example
    - Now it’s the default for Angular

# Other noteworthy JavaScript transpilers

- Dart
  - Developed by Google
  - Introduces typing and similar object-oriented practices
  - Transpiles to JavaScript with dart2js
- CoffeeScript
  - Open-source development
  - “Python-like” variable assignment, loops, and conditionals
  - Mostly meant to make JavaScript syntax prettier/cleaner


# Goals for this Lecture

## By the end of this lecture, you should be able to…

- Describe the role of package managers in web development
- Use the Node Package Manager (NPM) to install packages
- Write code which follows object-oriented principles in TypeScript
- Explain the advantages and disadvantages of using TypeScript

# A few more TypeScript details

# Type declaration files

- Because types get stripped out when transpiling, a “declaration file” (`.d.ts`) can be created
  - Important when someone else will use your code as a library
  - Declaration file helps their code check types in your library
  - `tsc --declaration test.ts`
- You can install typings declarations for many libraries from npm
  - `npm install --save @types/your-library-here`

# Type declaration files

```ts
//test.ts
function area(shape: string, width: number, height: number):string {
    var area:number = width * height;
    return "I'm a " + shape + " of area " + area + " cm^2.";
}

document.body.innerHTML = area('square', 15, 15);
```

```ts
// transpiled test.js
function area(shape, width, height) {
    var area = width * height;
    return "I'm a " + shape + " of area " + area + " cm^2.";
}
document.body.innerHTML = area('square', 15, 15);
```

```ts
// generated test.d.ts
declare function area(shape: string, width: number, height: number): string;
```

# Interfaces

- Just like in Java, describes the “inputs” and “outputs” of an object

```
interface Shape {
    name: string;
    width: number;
    height: number;
    color?: string; // ? Specifies an "optional" value
}

function area(shape : Shape):string {
    var area = shape.width * shape.height;
    return "I'm " + shape.name + " with area " + area + " cm squared";
}

console.log(area({name: "rectangle", width: 30, height: 15}));
console.log(area({name: "square", width: 30, height: 30, color:
“red"}));
```

# Inheritance

- Like in Java, classes and interfaces can be extended

```
class Shape3D extends Shape {
    volume: number;

    constructor (name: string, width: number, height: number, length: number ) {
        super( name, width, height ); //calls base class constructor
        this.volume = length * this.area;
    };

    shoutout() { //overrides the base class
        return "I'm " + this.name +  " with a volume of " + this.volume + " cm cube.";
    }

    superShout() { //calls base class shoutout method
        return super.shoutout();
    }
}

var cube = new Shape3D("cube", 30, 30, 30);
console.log( cube.shoutout() );
console.log( cube.superShout() );
```

# Generics

- Also work the same a Java

```
function identity<T>(arg: T): T {
    return arg;
}

let output = identity<string>("myString");  // type of output will be 'string'
let output = identity("myString");  // type of output will be 'string'
```

https://www.typescriptlang.org/docs/handbook/generics.html