

Javascript, SVG

JavaScript?!

- Just another programming language like Java, Python, C++, R, etc.
- “Really” not so much different from Python and Java.
 - executes the code line-by-line (Python)
 - all variables are declared in a universal data type (Python)
 - requires symbols like {, }, ; (Java)

JavaScript?!

- High-level, general purpose language
- **Imperative:** you write algorithms as step-by-step instructions (lines of code) using JavaScript's syntax, and the computer will interpret these instructions in order to execute the algorithm.
- **Interpreted:** translates the high-level language into machine language *on the fly at runtime*
- Can run in the browser, or on a server (Node.js)

Add JavaScript code to a html document

- The answer is <script> tag. Add the line below in <body> tag.

```
<script type="text/javascript">  
    /* this is the place for you to write your code! */  
</script>
```

- You can attach a separate JavaScript file to a html document (similarly to a CSS). You will learn about it during the in-class exercise today

Multiple ways for vanilla html/css/js



```
<!DOCTYPE html>
<html>
<head>
  <!-- include here to run before the page appears -->
  <script src="js/script.js"></script>
  <!-- use defer to run after all markup loads -->
  <script src="script.js" defer></script>
</head>
<body>
  ... html content ...

  <!-- include here to run "after" html appears -->
  <script src="js/script.js"></script>
</body>
</html>
```

Variable Assignment

- **var**: globally scoped or function/locally scoped
- Nobody uses **var** anymore, but you may see it in old code
- **let**: block scoped
 - *Can be updated* but not re-declared (within its scope)
- **const**: variables that maintain constant values (cannot change).
 - cannot be updated or re-declared

let



```
let greeting = "say Hi";  
// this is fine  
greeting = "say Hello instead";  
// this will fail  
let greeting = "say Hello instead";
```

const



```
const greeting = "say Hi";  
// this will fail  
greeting = "say Hello instead";  
// this will also fail  
const greeting = "say Hello instead";
```


Arrays

- Similar to Python lists
- Ordered, one-dimensional sequences of values

```

//an array of names
let names = ['Jared', 'Bofu', 'Andy', 'Trey'];

//an array of numbers (can contain "duplicate" values)
let numbers = [1, 2, 2, 3, 5, 8];

//arrays can contain different types (including other arrays!)
let things = ['tears', 2.5, true, [3, 4, 'hey']];

//access using bracket notation
console.log( names[1] ); // 'Bofu'
console.log( things[3][2] ); // 'hey'
```

Objects

- Unordered sequences of key-value pairs

```
let myObject = {  
  city: "san diego",  
  b: 2,  
  nest: {a: 1, b: 2, c: 3},  
  fruits: ['apple',  
  'peach'], "jared",  
  key: "value"  
}
```

Control Structures



```
if(condition){  
    //statements  
}  
else if(alternativeCondition)  
{ //statements  
}  
else {  
    //statements  
}
```

Control Structures

The condition can be any expression that evaluates to a Boolean value.



```
//check if a `person` variable has a `name` property  
if (dog.name) {  
    console.log('Dog does have a name!');  
}
```

Control Structures

Ternary: single expression if statement

```
let x; //declare variable
if (condition) {
    x = 'foo';
} else {
    x = 'bar';
}

//can be condensed into:
let x = condition ? 'foo' :
'bar';
```

Loops



```
// initialization, condition, update
for (let step = 0; step < 5; step++) {
  // Runs 5 times, with values of step 0
  through 4.
  console.log("step", step);
}
```

```
// "step 0"
// "step 1"
// "step 2"
// "step 3"
// "step 4"
```

Functions

- Named sequences of statements used to *abstract* code
- (Think python functions)

```
function sayHello(name) {  
    return `Hello, ${name}`;  
}  
  
//expected: argument is assigned a value  
sayHello("Jared"); // "Hello, Jared"  
  
//argument not assigned a value (left undefined)  
sayHello(); // "Hello, undefined"  
  
//extra arguments (values) are not assigned to variables, so are ignored  
sayHello("Jared", "Thursday"); // "Hello, Jared"
```

Arrow Functions

- Named sequences of statements used to *abstract* code
- (Think python functions)

```

● ● ●

// Traditional function
function add100(a, b) {
  return a + b + 100;
};

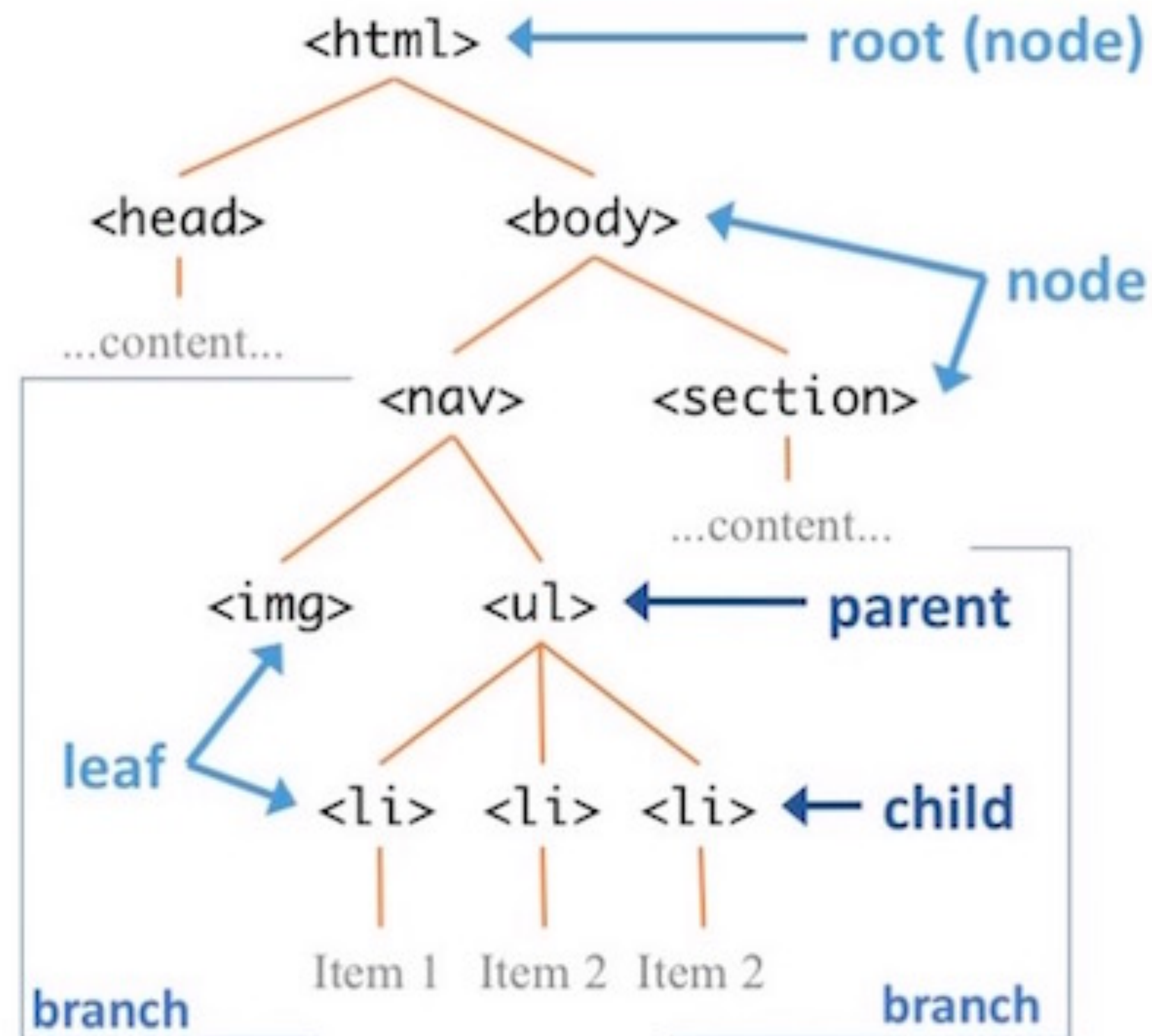
// Arrow function
const add100 = (a, b) => a + b + 100;
```


Manipulate html with JavaScript

- When a html document is up on a web browser, the browser creates a Document Object Model (DOM)
- DOM is a hierarchical data structure which organizes different html tags, attributes, and text of the html document
- JavaScript offers a set of methods for DOM
 - These methods allow us to access/change different elements, attributes, and texts on a html document.

DOM

We will learn to use JavaScript to dynamically update the DOM (and therefore, the rendered HTML)



Search html with JS DOM methods

- Multiple ways to search and modify html elements ([link](#))
 - by **tag name** `const element = document.getElementsByTagName("p");`
 - by **id** `const element = document.getElementById("intro");`
 - by **class name** `const x = document.getElementsByClassName("intro");`
 - by **CSS selectors** `const x = document.querySelectorAll("p.intro");`

Modify html with JS DOM methods

- Change HTML content
- Change the value of an existing attribute
- Set a new attribute (link)

```
<html>
<body>

<p id="p1">Hello World!</p>

<script>
document.getElementById("p1").innerHTML = "New text!";
</script>

</body>
</html>
```

JavaScript can Change HTML

New text!

The paragraph above was changed by a script.

```
element.setAttribute("class", "democlass");
```

```
<!DOCTYPE html>
<html>
<body>



<script>
document.getElementById("myImage").src = "landscape.jpg";
</script>

</body>
</html>
```

Scalable Vector Graphics (SVG)

Scalable Vector Graphics (SVG)

- an XML-based vector image format for two-dimensional graphics **with support for interactivity and animation**
- Key features: Resolution-independent, smaller file sizes, easily editable, and widely supported by modern browsers.
- Compared to regular HTML elements (e.g. `<div>`'s), it is **much** easier to draw shapes
- SVG works well with D3 too, and that is why we are going to use SVG for all the drawings throughout the course.

Why SVG?

- **Scalability:** SVG images maintain their quality when scaled, making them ideal for responsive design.
- **Accessibility:** SVGs can be easily made accessible for screen readers and are indexable by search engines.
- **Interactivity:** SVG elements can be manipulated using CSS and JavaScript for enhanced user experiences.
- **Performance:** SVG files are usually smaller than raster images, resulting in faster load times (though performance suffers if *too many* SVG elements on screen).

Basic SVG Elements

- SVG has a lot of elements, just like html.
- For dataviz, we care mostly about the shapes:
 - `<svg>`, `<rect>`, `<circle>`, `<ellipse>`, `<line>`, `<polyline>`, `<polygon>`, `<path>`, `<text>`, `<g>`
- When we learn D3, we will map our data elements to these shapes, creating visualizations

Draw a rectangle

- Claim the drawing canvas `<svg>` in `<body>`

```
<svg width="500" height="100">  
</svg>
```

- Add a `<rect>`

```
<rect x="0" y="0" width="100" height="30"/>
```

- `x` and `y` - a point (x, y) where the rectangle starts
- `width` - how far the shape goes towards the right
- `height` - how far the shape goes towards the bottom

Add a style to the rectangle

- use attributes

```
<rect x="0" y="0" width="250" height="50"  
fill = "yellow" stroke="green" stroke-width = "3"/>
```

- Some commonly used style attributes are listed below.
 - fill: A color value. Just as with CSS, colors can be specified as named colors, hex values, or RGB or RGBA values.
 - stroke: A color value.
 - stroke-width: A numeric measurement. Typically in pixels.
 - opacity: A numeric value between 0.0 (completely transparent) and 1.0 (completely opaque).

Beyond Vanilla

Vanilla HTML

- **“Vanilla HTML”**: Refers to the frontend stack we’ve used so far in this course:
 - HTML
 - CSS
 - JavaScript

Vanilla HTML

- **“Vanilla HTML”**: Refers to the frontend stack we’ve used so far in this course:
 - HTML
 - CSS
 - JavaScript
- Basically everything referenced from some .html file

Limitations of Vanilla HTML

- In traditional HTML development, the DOM and UI updates are tightly coupled.
- Complex UI interactions and state management become challenging and error-prone.
- Lack of componentization makes code organization and reuse difficult.

Limitations of Vanilla HTML

- **In traditional HTML development, the DOM and UI updates are tightly coupled.**
- When we want to update UI, we often manipulate the DOM directly

Limitations of Vanilla HTML

- This can lead to issues:
 1. Manual DOM manipulation can be tedious and error-prone, especially for complex UIs.
 2. Directly updating the DOM can cause unnecessary reflows and repaints, impacting performance.
 3. Keeping track of UI changes and managing the state of the application can be challenging.

Limitations of Vanilla HTML

- **Complex UI interactions and state management become challenging and error-prone.**
- As web applications become more sophisticated, handling complex UI interactions and managing application state becomes increasingly challenging. With vanilla HTML and JavaScript, developers often end up writing extensive and convoluted code to manage these complexities.

Limitations of Vanilla HTML

- This can lead to issues:
 1. Callback hell: In a callback-based architecture, managing multiple asynchronous operations can result in deeply nested and hard-to-read code.
 2. Inconsistent UI updates: Manually handling UI updates based on application state changes can easily lead to inconsistencies and synchronization issues.
 3. Difficulty Debugging tightly coupled UI & state management

Limitations of Vanilla HTML

- **Lack of componentization makes code organization and reuse difficult**
- Vanilla HTML lacks built-in mechanisms for code organization and reuse:

Limitations of Vanilla HTML

- This can lead to issues:
 1. UI elements and logic tend to be tightly coupled and scattered throughout the codebase.
 2. Code duplication becomes common, leading to maintenance challenges and increased chances of introducing bugs.
 3. Reusing UI elements or sections across different pages or components requires manual copying and pasting of code

Solution

- **Frameworks:** pre-written libraries that provide a structured and standardized way to build web applications, offering a set of tools and abstractions to simplify and streamline the development process

Solution

- **Frameworks:** pre-written libraries that provide a structured and standardized way to build web applications, offering a set of tools and abstractions to simplify and streamline the development process
- Examples: React, Svelte, Vue, etc.

New Development Environments

Beyond Vanilla

- As we move away from vanilla html and towards frameworks, we will be using mostly (if not entirely) JavaScript.
- Thus, we'll need more tools to manage typical programming tasks:
 - Package management
 - Dependency management
 - Server-side execution
 - Development tooling
 - etc.

Beyond Vanilla

- Basically, we're going to be using JavaScript like most of you are used to using Python: as a programming language with all kinds of concerns around developer workflow and tooling.
- Let's go over some new tools

Node.js

- **Node.js:** a JavaScript runtime environment that allows developers to execute JavaScript code outside of a web browser. It is used for server-side scripting, command-line tools, and building scalable network applications, providing a runtime environment for executing JavaScript code on the server and enabling developers to leverage JavaScript's versatility beyond the client-side web development.



npm: Node Package Manager

- **npm:** a package manager for JavaScript that comes bundled with Node.js. It is used to discover, install, manage, and share reusable code packages and libraries (think of Python's **pip**)



Babel

- **Babel:** a JavaScript compiler that transforms modern JavaScript code into an older, widely supported version, allowing developers to write using the latest language features while ensuring compatibility across different environments.

BABEL

ESLint

- **ESLint:** A popular linter tool that helps enforce code quality and consistency by identifying and reporting coding errors, potential bugs, and style violations.



Bundlers

- **Bundlers:** tools that combine and optimize multiple modules or files into a single bundle, enabling efficient delivery and execution of code in web applications.
- Our JS projects will have lots of files, and bundlers make sure they are optimized to load on web-pages (minimized, tree-shaken, babel, etc.)

Bundlers

- Many bundlers exist!
- The most popular is (was) **webpack**, but it's pretty confusing.
- I prefer **vite**, which is new and super fast
- **Parcel** is another great choice if you're new, because it abstracts a lot of annoying config away



Our New Dev Environment

cli

- Modern web-dev feels much more like writing python than writing html files.
- We'll kick everything off from the command line.

cli

- Modern web-dev feels much more like writing python than writing html files.
- We'll kick everything off from the command line.



```
npm create vite@latest
```


cli

- We can follow the options and set up a React project here

```
(base) → lecture19 npm create vite@latest
Need to install the following packages:
  create-vite@4.3.2
Ok to proceed? (y) y
✓ Project name: ... vite-project
? Select a framework: > - Use arrow-keys. Return to submit.
  Vanilla
  Vue
> React
  Preact
  Lit
  Svelte
  Others
```

cli

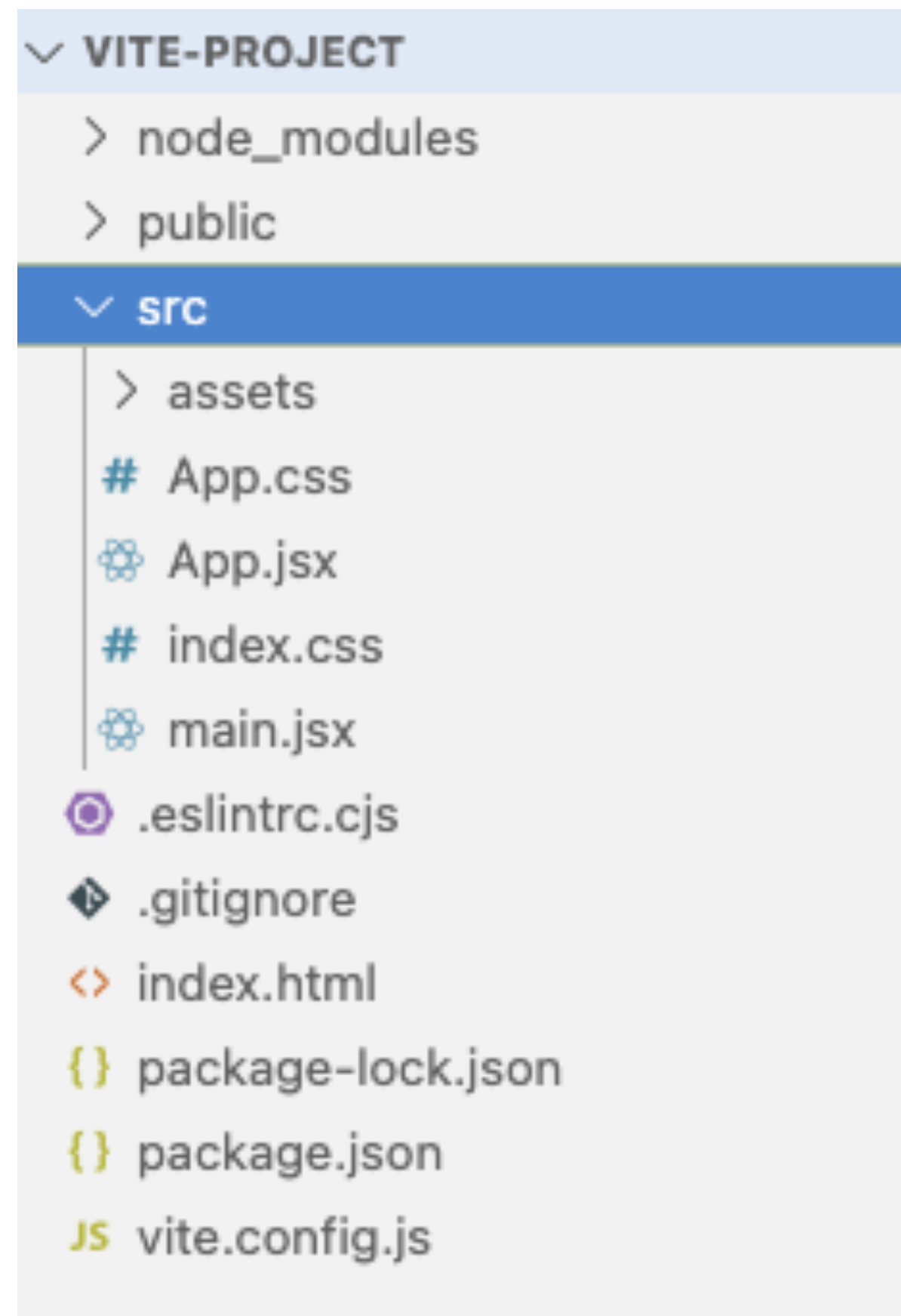
- We can follow the options and set up a React project here



```
cd vite-project  
npm install  
npm run dev
```

cli

- We'll go over React soon, but let's look at what we get/do



package.json

- **package.json**: a manifest for a JavaScript project, listing its dependencies, defining project details, and providing scripts for various development tasks.

package.json

```
{
  "name": "vite-project",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint src --ext js,jsx --report-unused-disable-
directives --max-warnings 0",
    "preview": "vite preview"
  },
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0"
  },
  "devDependencies": {
    "@types/react": "^18.0.37",
    "@types/react-dom": "^18.0.11",
    "@vitejs/plugin-react": "^4.0.0",
    "eslint": "^8.38.0",
    "eslint-plugin-react": "^7.32.2",
    "eslint-plugin-react-hooks": "^4.6.0",
    "eslint-plugin-react-refresh": "^0.3.4",
    "vite": "^4.3.9"
  }
}
```

Other tools

- Static typing, css frameworks, meta-frameworks, etc.
- *JavaScript Fatigue*