

Components

A React "Component" returns JSX/HTML

- A js function
 - "function-based component" or
 - "functional component"
- Old style is "class-based"
 - We won't be using those
 - Almost no one does: old
- React Docs are (now) very high quality!
 - See **<https://reactjs.org/>**

Components are Elements

A React Component can be used as an Element in JSX

- Open/close or self-closing
 - NO: `<Greeting>` (Needs a close somewhere)
 - YES: `<Greeting/>`
 - YES: `<Greeting></Greeting>`
- Element name matches function name
 - MixedCase, not camelCase
 - YES: `<Greeting/>` or `<CatVideos/>`
 - NO: `<greeting/>` or `<catVideos/>`

HTML Elements in JSX are actually JSX

- Work like actual elements
 - Mostly (But it's good)
- All elements, HTML-based or not, are **consistent**
- All elements can be open/close or be self-closing
- **All elements require a close of some sort in JSX!**
- NO: `<input name="name">` (Valid HTML, invalid JSX)
- YES: `<input name="name"/>`
- YES: `<input name="name"></input>` (but why?)

Components are not files

OFTEN a `.jsx` file is exactly 1 component

- This is not required by React

Course Requirements:

- One `.jsx` file === one component
- Filename must match component name
- Component must be MixedCase

Outside of course, then can change

Components are a single container

Can have any nested elements/components

- MUST have a **single parent container element**
- YES:

```
function Greeting() {  
  return (<div>Hello<p>World</p></div>); // div is single container  
}
```

- NO:

```
function Greeting() {  
  return (<div>Hello</div><p>World</p>); // two sibling containers  
}
```

- OR be a **fragment**
 - Wrapped in a single non-element container

Example of single parent container

This works:

```
function CatFacts() {  
  return (  
    <div className="cat-facts">  
      <h1>Cat Facts</h1>  
      <div className="subtitle">Number 3 will shock you</div>  
      <ul>  
        <li>Cats can rotate their ears 180 degrees</li>  
        <li>Felines can purr or roar, but not both</li>  
        <li>Humans domesticated dogs,  
          but cats domesticated humans</li>  
      </ul>  
    </div>  
  );  
}  
  
export default CatFacts;
```

Example without single parent container

This will give you an error:

```
function CatFacts() {  
  return (  
    <h1>Cat Facts</h1>  
    <div className="subtitle">Number 3 will shock you</div>  
    <ul>  
      <li>Cats can rotate their ears 180 degrees</li>  
      <li>Felines can purr or roar, but not both</li>  
      <li>Humans domesticated dogs,  
        but cats domesticated humans</li>  
    </ul>  
  );  
}  
  
export default CatFacts;
```

You need to use fragments

"Just put all our of component output in a `<div>`?"

- No
- If the parent container element isn't useful
 - Not semantic
 - Not styled or impacting styling
 - Not listening to events
- Use a **fragment** instead

How to use a Fragment

```
function Demo() {  
  return (  
    <>  
      <p>  
        These p tags will have no containing  
        element from this component  
      </p>  
      <p>And React will not complain</p>  
    </>  
  );  
}
```

- `<>` and `</>`
- React treats like a containing element
- But no element in output HTML

When NOT to use Fragment

- Parent container element
 - **is semantic**, or
 - **is styled**, or
 - **impacts styling**, or
 - **listens to events**
- Use appropriate wrapping container element

Example: A `<Card>` element will have styling

imports

Even without React, we want multiple files to organize

- Big files = hard to manage/maintain

Vite includes a **bundler** program (Rollup)

- Lets us use many files in dev
- Outputs to fewer files in prod

Syntax is not browser JS

- Bundler converts

Importing JSX

Write a `Test.jsx` in `src/`

```
function Test() {  
  return (  
    <p>Hello World</p>  
  );  
}  
export default Test;
```

Top of `App.jsx`:

```
import Test from './Test';
```

Near end of `App.jsx`, before `</>`:

```
</p>  
<Test/>  
</>
```

The parts of importing

- Say what you want to export
- Say what you are importing
 - And from where
- Use what you've imported

We will start with discussing **component imports** first

- Other imports are different rules

Say what you are exporting

At end of file:

```
export default VARIABLE_NAME;
```

Example:

```
function CatVideo() { /* ... */ }  
export default CatVideo;
```

- Exported default variable should match filename
 - For ease of use, not system requirement
- There are other export options
 - We won't use them yet
- This isn't JS that works in browser
 - Converted by tools that Vite gives us

Say what you are importing

...and from where

```
import CatVideo from './CatVideo';  
import Component from './Filename';
```

- Can be single or double quotes
- **Component** is the name you will use
 - **Course Requirement:** must match filename
- **Filename** is the filename
 - You need an explicit path (**./**)
 - Can be a different directory
 - Do not need a file extension
 - Will import **.jsx** or **.js**

Using your imported Component

Use an imported Component in a HTML-like JSX tag:

```
import Test from './Test';

function Demo() {
  return (
    <div className="demo">
      <Test/>
    </div>
  );
}
export default Demo;
```

Any file can import other files

- Gets weird/breaks if you make a circle
 - A uses B, and B uses A
 - Don't do that

importing CSS

CRA allows you to import CSS files

```
import './App.css';
```

- Makes the CSS available on the HTML page
 - No `<link>` required
- Filename can be anything
 - Does not have to be MixedCase
 - Must have `.css` extension
 - Must have a path (e.g. `./`)

Organizing your CSS files

- Many options
 - All in `src/index.css`
 - Mostly in css imported by `App.jsx`
 - CSS for each Component
 - Imported in those components?
- React has even more CSS options not seen yet

Course Requirements for CSS in React

Course Requirements:

- Any filenames for .css files
- import css into whatever JSX files you want
- MUST have some organization
 - Not all one big file when lots of CSS
- MUST be in imported `.css` files
 - Should feel like CSS in course so far
 - No styled components, CSS modules, etc
 - No style attributes on HTML/JSX

Importing Images

Importing images LOOKS like importing Components:

```
import someImage from './cat-pic.jpg';
```

There are important differences:

- You pick a variable name to import as
- The filename needs to be complete
 - Including file extension
 - And with explicit path
- Variable holds the path to the image as a string:
 - ``

Images: public/ or src/?

Vite gives us some options:

- Can import images with absolute paths
 - Will use files in `public/`
 - Filenames not changed when built
 - Use for images that will NOT change
- Can import images with relative paths
 - Will use files in `src/`
 - Filenames are cache-busted when built
 - Use for images that MAY change

importing JS

We will cover this more later

- but all components are JS

Component Props

Components have attribute-like values:

```
<Greeting target="world"/>
```

These are called "props"

- Allow you to pass values to Components
- Allows for flexibility and reuse

```
<Greeting target="class"/>  
<Greeting target="world"/>
```

```
<p>Hello class</p>  
<p>Hello world</p>
```

Prop values

Unlike HTML, props can hold more than strings

- non-strings must be in `{}`

Unlike HTML, props should ALWAYS have a value

- not there/not there like `disabled` or `checked`

```
<CatList cats={['Jorts', 'Jean', 'Nyancat']}/>
```

```
<ul class="cats">
  <li>Jorts</li>
  <li>Jean</li>
  <li>Nyancat</li>
</ul>
```


Reading passed props

A Component function is passed an object of all props

```
<CatList cats={['Jorts', 'Jean', 'Nyancat']}/>
```

```
function CatList( props ) {  
  const list = props.cats.map( cat => {  
    return (  
      <li>{cat}</li>  
    );  
  });  
  
  return (  
    <ul className="cats">  
      {list}  
    </ul>  
  );  
}  
export default CatList;
```

Destructuring props

Common to **destructure** props object to get variables

```
function CatList( { cats } ) {  
  const list = cats.map( cat => {  
    return (  
      <li>{cat}</li>  
    );  
  });  
  
  return (  
    <ul className="cats">  
      {list}  
    </ul>  
  );  
}  
export default CatList;
```

Error Messages in React are usually helpful

- Check browser console after adding
- `<CatList cats={['Jorts', 'Jean', 'Nyancat']} />`

Warning: Each child in a list should have a unique "key" prop

Check the render method of `CatList`.
See <https://reactjs.org/link/warning-keys>
for more information

- Actually really helpful!
- Complete with link to learn more!

Errors vs Warnings

- Technically, that was a **warning**
 - Doesn't stop the program from running
 - May not be *working*
- **Errors** stop a program from running
 - Try not closing a Component/element

Even though a warning doesn't stop the program

- **You should resolve warnings right away**
- It is literally a likely bug
 - Could impact what you're doing now

What is this warning saying?

- Wants `key` prop on each component in list
 - `key` must have a unique value
- React rewrites HTML when data changes
 - It wants to do so EFFICIENTLY
 - If you give me a list, then later give me list
 - Which added/removed vs changed?
- We need to identify the items of a list
 - And `list` is an array (list) of `` elements

Can I use the index as the key?

- No
 - Well, Yes, but you shouldn't
- It will silence the warning
- But is actually WORSE
 - If an element is removed
 - Index will not LIE
 - Index does not uniquely identify
 - Index can refer to different elements

Do not use index for a key prop of a list

What DO I use as a key prop?

Use a value uniquely connected to the data in element

- Accurate: "is this the same list item as last time"
- Complex records normally have an identifier
 - Ex: NEUID
- Simple records build one from data
 - Might be combination of fields
 - Or just one field:

```
const list = cats.map( cat => {  
  return (  
    <li key={cat}>{cat}</li>  
  );  
});
```

All About key Prop

- Use when outputting array of elements in JSX
 - Pass `key={}` on each element
 - Use a value that identifies the element
 - Do not pass `index` as `key`

Events

Components are JS that outputs HTML

- So how do we attach event listeners to HTML?

"on" Handlers

```
function doMeow() {  
  console.log('meow');  
}  
  
function Meow() {  
  return (  
    <p onClick={doMeow}>Meow</p>  
  );  
}  
  
export default Meow;
```

But WAIT!

Didn't we say NOT to use "onclick" in HTML?!

Yes!

- But this isn't HTML
- It LOOKS like HTML, but isn't
 - `onClick` vs `onclick`
- Differences are subtle but real
 - React will translate it more like `.addEventListener()`

Comparing

Bad:

```
<p onclick="function() { console.log('meow') }">Meow</p>
```

- Editing JS in HTML
 - All in a string of attribute value
 - Hard to interact with other JS

Good:

```
<p onClick={ () => console.log('meow') }>Meow</p>
```

- Editing JS in JSX (which is just JS)
- No weird scope or variable changes

Only HTML elements can get events

Events don't happen to Components

- You can pass props
- No built in behavior, just a name
- Component can apply to returned HTML element
 - Which DOES have built-in behavior

```
function Meow({ onClick }) {  
  return (  
    <p onClick={onClick}>Meow</p>  
  );  
}  
  
export default Meow;
```

Wait, What?

- Components can be passed props like `onClick`
 - But it is just a name
 - No Behavior
- Component CAN use/pass the passed prop
- Native Elements DO have behavior for `onClick`

```
function Meow({ onClick }) {  
  return (  
    <p onClick={onClick}>Meow</p>  
  );  
}  
  
export default Meow;
```

```
<Meow onClick={ () => console.log('meow') }/>
```

Passed event handlers can have any name

- `onEVENT` props only matter on native elements
- Otherwise they are just props
- We can pass such props with ANY name
- Effectively named callbacks

```
function Meow({ onMeow }) {  
  return (  
    <p onClick={onMeow}>Meow</p>  
  );  
}  
  
export default Meow;
```

```
<Meow onMeow={ () => console.log('meow') }/>
```

Summary - Components

Components:

- Functions that return HTML/JSX
 - or class-based component
- Can be nested
- Passed **props**
- Must have a single parent element
 - Or be **fragment**
- Must be named in MixedCase
- FOR THIS COURSE:
 - 1 component per `.jsx` file (must be `.jsx`)
 - Filename matches component name

Summary - imports/exports

- A component can be exported from a file
- A component can be imported from an export
- A CSS file can be imported
 - Many options on how to organize/approach
 - CSS imports not needed in all components
- An image path can be imported
- All imports need an explicit path

FOR THIS COURSE:

- CSS classes should be `kebab-case` or BEM
- No styled-components, CSS Modules, etc

Summary - props

Components have **props** passed in JSX

- Received in `props` object passed to JS function
 - Often **destructured** to named variables
- Props can hold string or non-string values
- Event handler props have no behavior on components
 - But can be passed to HTML elements

Summary - event handlers

Event handlers go on HTML tags in JSX

- Looks like HTML JS attributes
 - But aren't
- Must be `onEVENT` syntax
 - EVENT is a capitalized event name
 - So `onEVENT` will be camelCase
 - e.g. `onClick`, `onInput`, `onChange`
- Event handler props don't work on components
 - But can be put on HTML elements