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 docs/tutorials exist

Components are Elements (ish)

A React Component can be used as an Element in JSX

- Open/close or self-closing
  - `<Greeting></Greeting>` or `<Greeting/>`
- Consistent!
  - html elements in JSX are ALSO consistent!
- Element name matches function name
  - MixedCase, not camelCase
    - YES: `<Greeting/>` or `<CatVideos/>`
    - NO: `<greeting/>` or `<catVideos/>`

Components are not files

OFTEN a `.jsx` file is exactly 1 component

- This is not required by React itself
- On some occasions, a good reason not to

For this course, it IS required

- **Required:** Component is `.jsx` file
- **Required:** 1 file === 1 Component
- **Required:** filename matches component name
- After course, then you can change

Components return 1 parent element/fragment

Can have many nested elements

- But MUST have a single parent container element
- OR be a "fragment"
    - More on that later

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;
```

imports

Create React App uses webpack

- Also includes some extra config options
- Allows for non-standard imports

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';
```

inside `return` of `App.jsx`:

```
    <Test/>
```

Component import/export

- Components are JS functions
    - With MixedCase names (starts with capital)
- export/import as default
    - Nothing specific to React
        - Just uses MixedCase name

Component Naming

React Requirement:

- Component function name is `MixedCase` not `camelCase`

This Course Requirements

- Components are in `.jsx` file
- Exactly 1 component per `.jsx` file
- Component name === filename
  - Including being MixedCase

importing CSS

CRA allows you to import CSS files

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

- Makes the CSS available on the HTML page
- filename can be anything
    - does not have to be MixedCase
    - must have `.css` extension
    - must have a path (e.g. `./`)
- Do not need to have CSS with each component
    - can use `src/index.css`
    - or put all css in css file(s) imported in `App.jsx`

React has other options for CSS, more on that later

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 path
- Variable holds the path to the image as a string:
    - `<img src={someImage} alt="smug cat"/>`

Import other JS

Any plain `.js` files

- such as `services.js`
- imported just like normal

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`

```
<MovieSequels count={3} />
```

```
<ul class="sequels">
  <li>Cats: The Musical</li>
  <li>Cats: The Musical 2</li>
  <li>Cats: The Musical 3</li>
</ul>
```

Reading passed props

A Component function is passed an object of all props

```
<MovieSequels count={3} />
```

```
function MovieSequels( props ) {
  const list = [];

  for(let sequel = 1; sequel <= props.count; sequel  += 1) {
    const title = sequel === 1 ? '' : sequel;
    list.push( <li>Cats: The Musical {title}</li> );
  }

  return (
    <ul className="sequels">
      {list}
    </ul>
  );
}
export default MovieSequels;
```

Destructuring props

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

```
function MovieSequels( { count } ) {
  const list = [];

  for(let sequel = 1; sequel <= count; sequel  += 1) {
    const title = sequel === 1 ? '' : sequel;
    list.push( <li>Cats: The Musical {title}</li> );
  }

  return (
    <ul className="sequels">
      {list}
    </ul>
  );
}
export default MovieSequels;
```

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
- Differences are subtle but real

Comparing

Bad:

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

- Editing JS in HTML
  - Hard to find
  - Hard to edit

Good:

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

- Editing JS in JSX (which is just JS)
  - Right where you would put it
- Function is an actual function value

Only HTML elements can get events

Events don't happen to Components

- But you can pass props
- Component can apply to returned element

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

export default Meow;
```

Naming Event Handlers

- Common to use different names based on context

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

export default Meow;
```

- Caller doesn't know "how" `onMeow` is called
- But does decide what `onMeow` does

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

Summary - Components

Components:

- Functions that return HTML/JSX
    - or class-based component
- Can be nested
- Passed "props"
- Must have single parent element/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
    - Each CSS imports only has to happen once
- An image path can be imported
- All your imports need an explicit path

FOR THIS COURSE:

- CSS classes must be `kebab-case` (or BEM)

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
- No automatic prop behavior on Components
    - Not real HTML elements!
    - Props are passed to Component
    - Can be put on 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 MixedCase event name
    - e.g. `onClick`, `onInput`, `onChange`, `onSubmit`
- No automatic Component handler prop behavior
    - Event Handler props passed to Component
    - Can be put on actual HTML elements