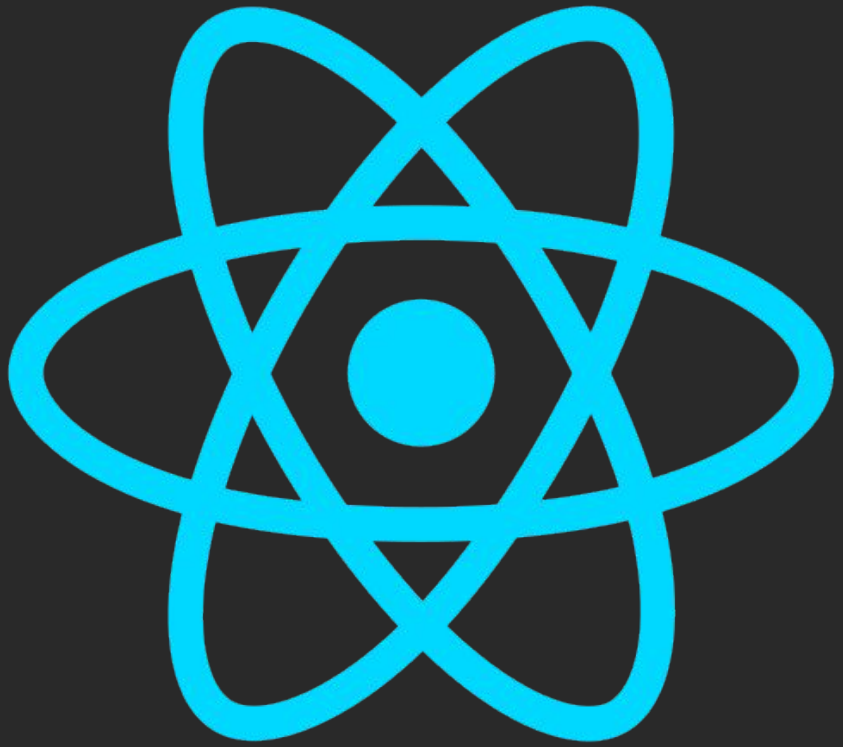
React Basics

IMY 220 ● Class Discussion



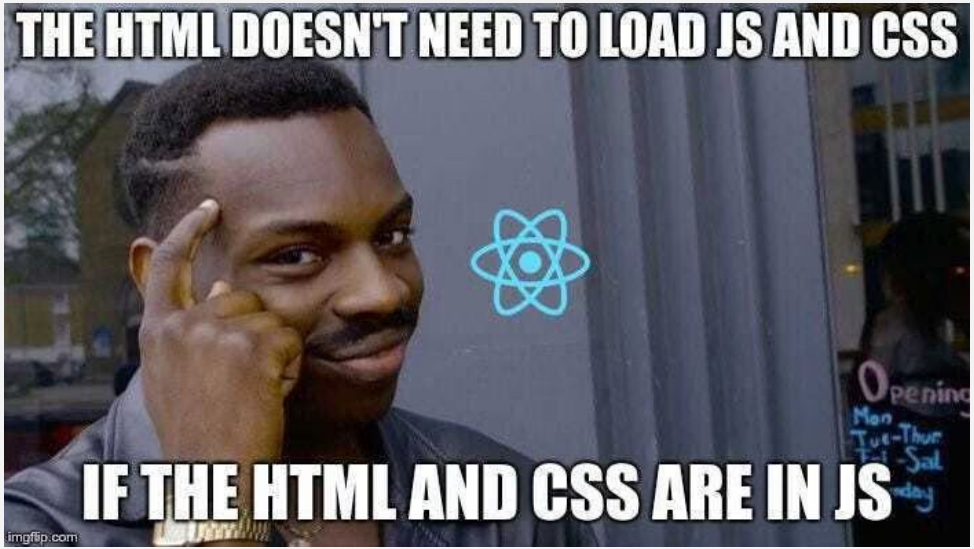
Quiz!

https://quiz.com/77046fbc-0b36-4882-84e1-48326ed70858

Lecture Recap

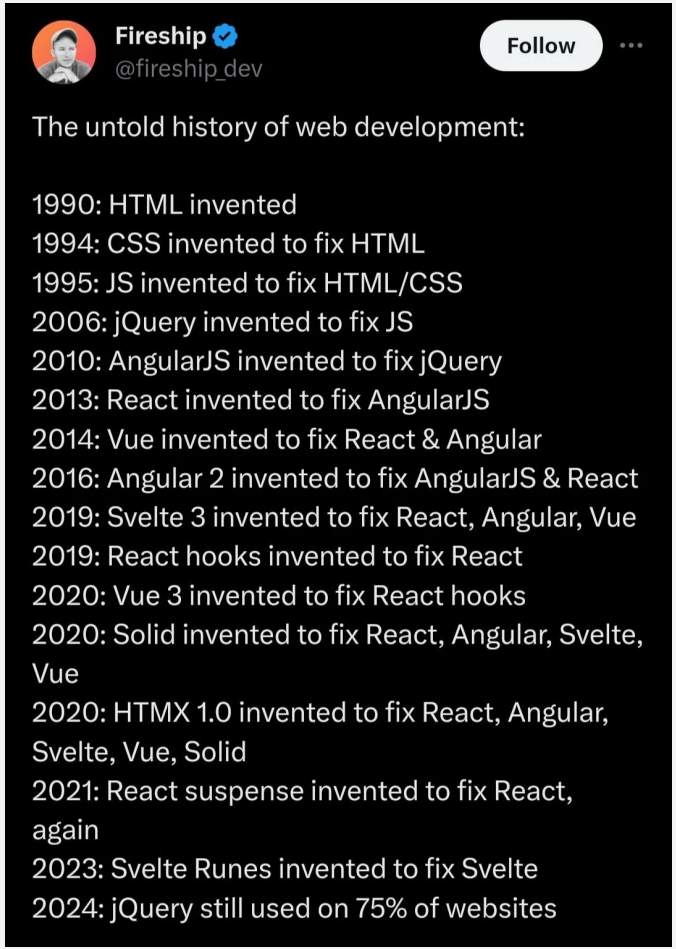
- React Basics
- Creating components
- Using components inside other components
- Props
- Types of components
- Basic component state

Download the files off ClickUP for the exercises.



What is React?

- **Frontend** JavaScript Framework
- Developed and Maintained by Facebook
- React facilitates writing code in **small isolated pieces** called components
  - Similar concept to ‘modules’
  - OOP Approach
- Very popular framework today (among many others)
  - Virtual DOM **
  - JSX



“JavaScript Framework”

“JavaScript Framework”

“JavaScript frameworks are an essential part of modern front-end web development, providing developers with tried and tested ***tools for building scalable, interactive web applications****.*

Many modern companies use frameworks as a standard part of their tooling, so many front-end development jobs now require framework experience.”

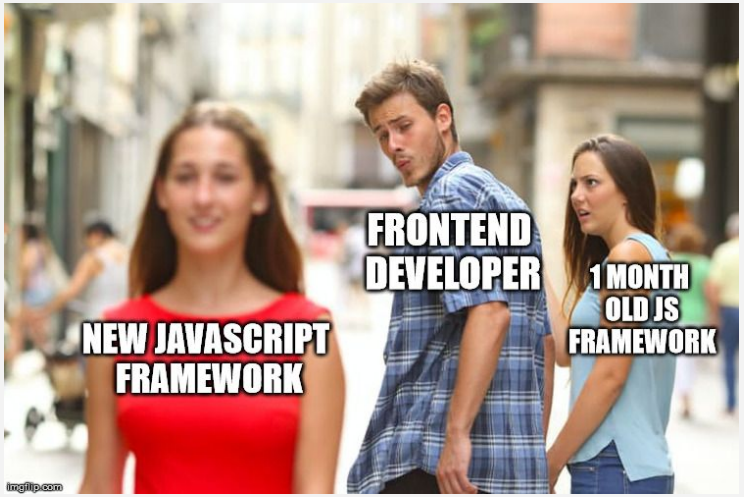
https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks

React is a **Frontend** Framework - Build UI with it

- Other JS Frameworks needed to build backend parts of applications
- Server-side features are coming soon though

“JavaScript Framework”

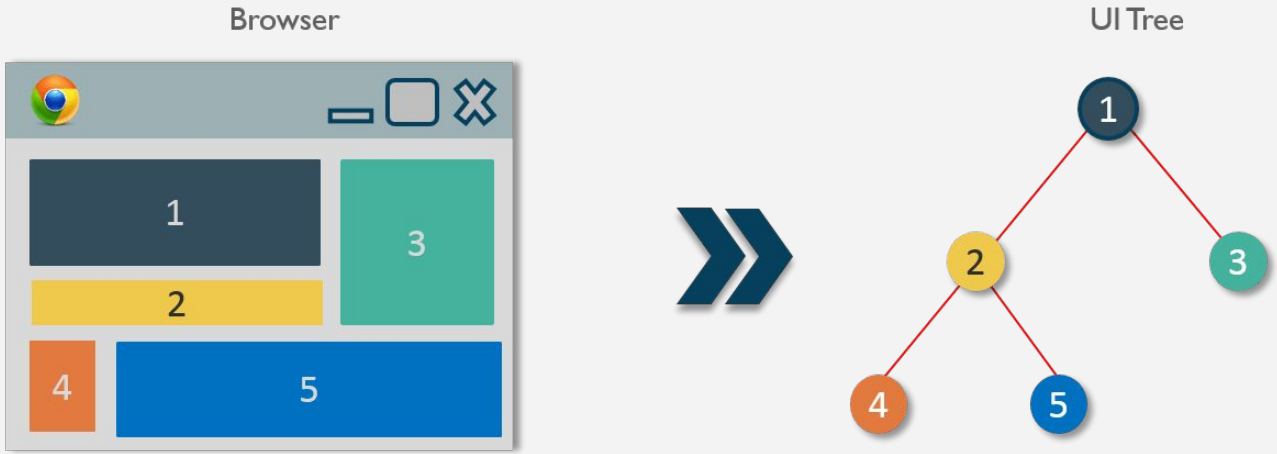
- New ones come out all the time
- Each are kind of the same, but kind of different
- Come back to this in later lectures
- **Why React?**



Components

- **Core** of every React Application
- **Reusable** pieces of code
- **Encapsulate** one ‘object’ / function.
- Like OOP **classes / objects**
- **Self-contained** (ideally).

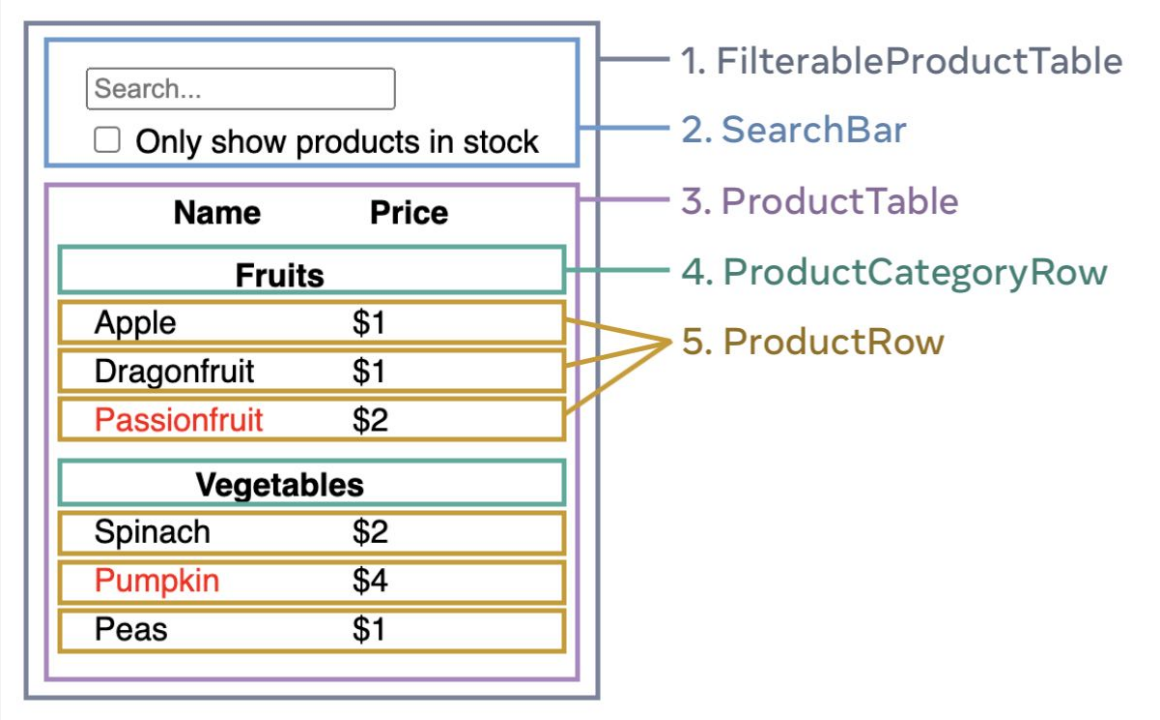
- Tell React how to Render stuff to the DOM (**Virtual DOM** **)
- **JSX** = JavaScript XML
  - Write React components more intuitively



Components

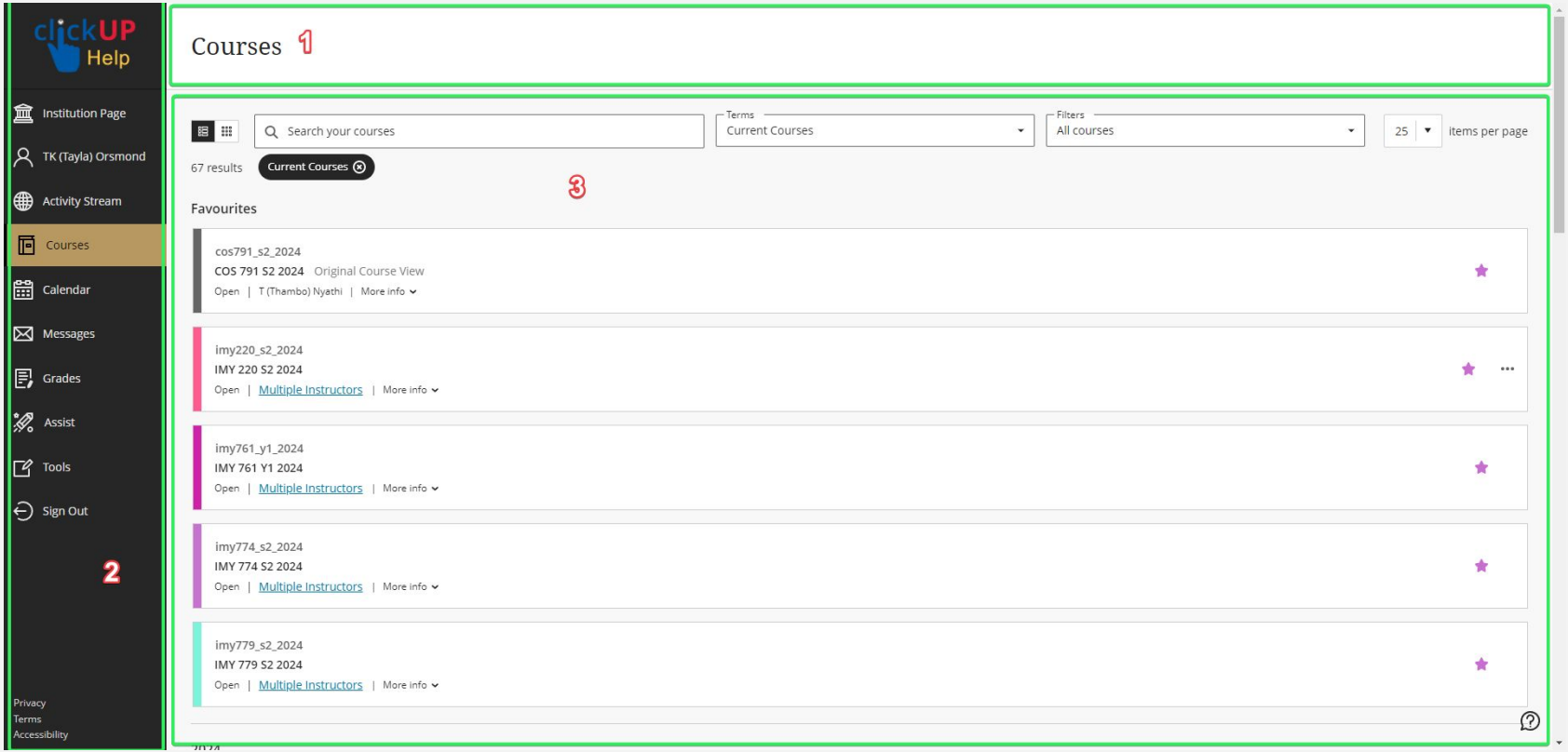
One ‘page’ doesn’t have all the functionality anymore.

Divide our ‘pages’ / app into functional **pieces**.

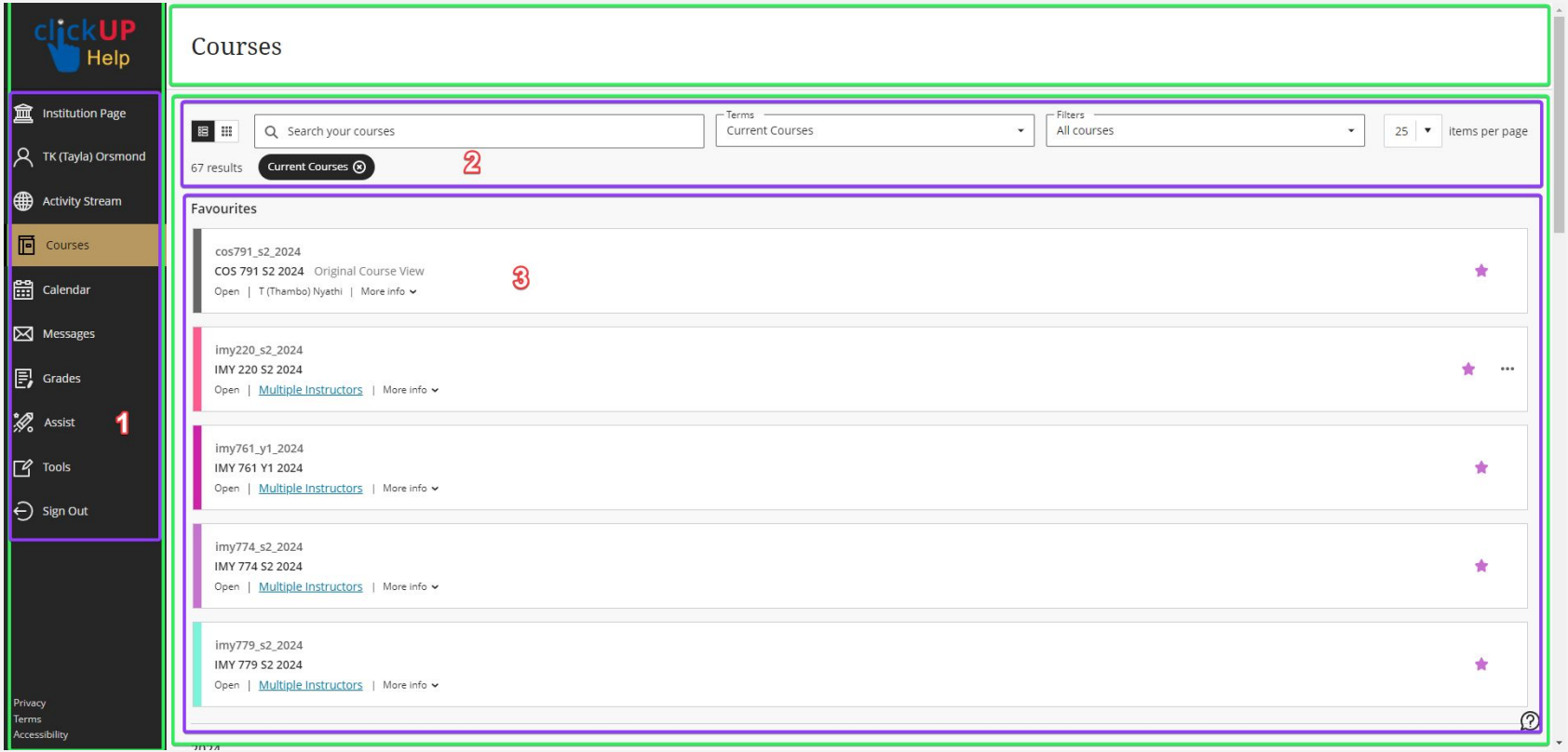


https://react.dev/learn/thinking-in-react

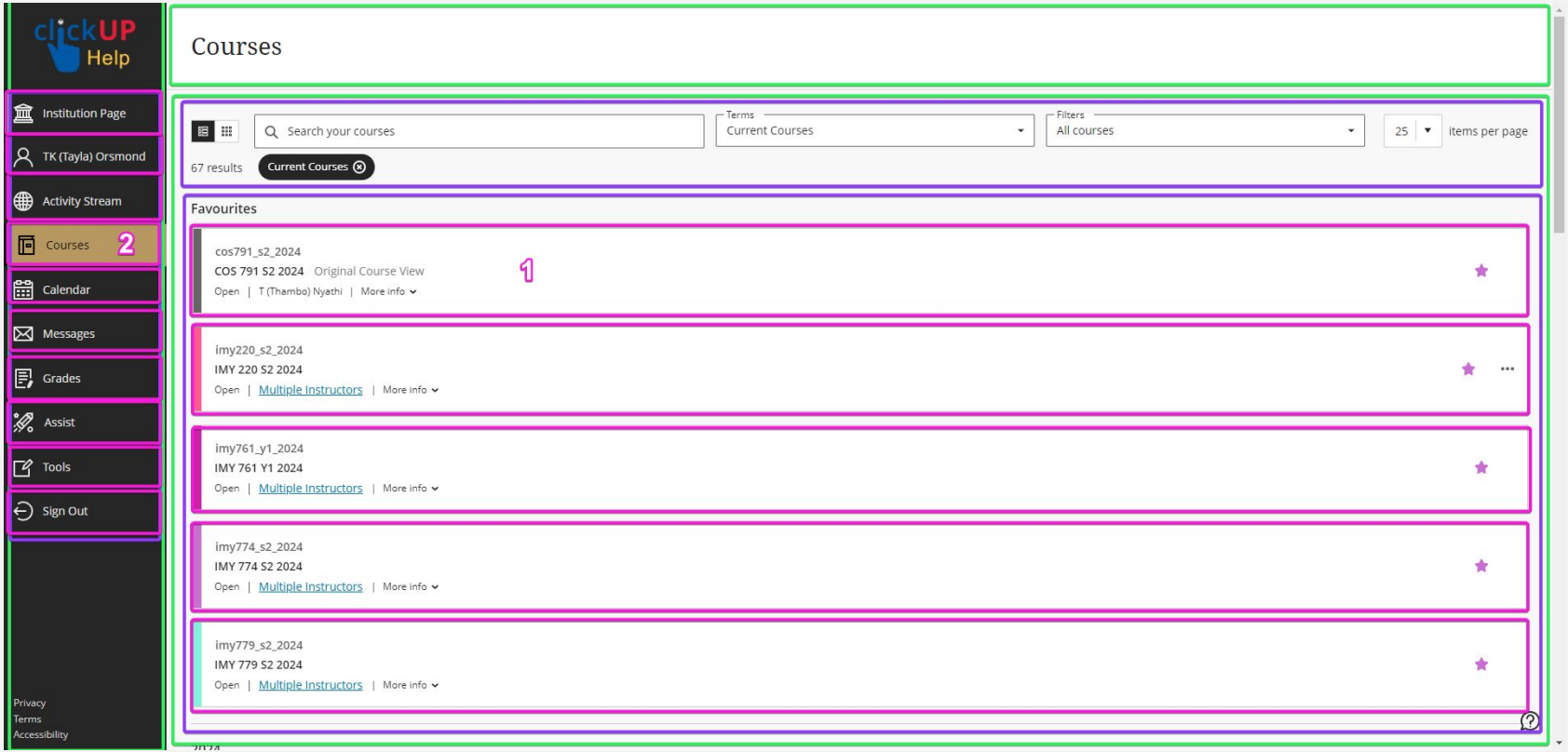
Components


clickUP Help
Institution Page
TK (Tayla) Orsmond
Activity Stream
Courses
Calendar
Messages
Grades
Assist
Tools
Sign Out
2
Privacy
Terms
Accessibility
Courses 1
Search your courses
Terms
Current Courses
Filters
All courses
25
items per page
67 results
Current Courses
3
Favourites
cos791_s2_2024
COS 791 S2 2024 Original Course View
Open | T (Thambo) Nyathi | More info
imy220_s2_2024
IMY 220 S2 2024
Open | Multiple Instructors | More info
imy761_y1_2024
IMY 761 Y1 2024
Open | Multiple Instructors | More info
imy774_s2_2024
IMY 774 S2 2024
Open | Multiple Instructors | More info
imy779_s2_2024
IMY 779 S2 2024
Open | Multiple Instructors | More info

Components


clickUP Help
Courses
Institution Page
TK (Tayla) Orsmond
Activity Stream
Courses
Calendar
Messages
Grades
Assist
1
Tools
Sign Out
Privacy
Terms
Accessibility
Search your courses
Terms
Current Courses
Filters
All courses
25
items per page
67 results
Current Courses
2
Favourites
cos791_s2_2024
COS 791 S2 2024
Original Course View
Open | T (Thambo) Nyathi | More info
3
imy220_s2_2024
IMY 220 S2 2024
Open | Multiple Instructors | More info
imy761_y1_2024
IMY 761 Y1 2024
Open | Multiple Instructors | More info
imy774_s2_2024
IMY 774 S2 2024
Open | Multiple Instructors | More info
imy779_s2_2024
IMY 779 S2 2024
Open | Multiple Instructors | More info

Components


clickUP Help
Institution Page
TK (Tayla) Orsmond
Activity Stream
Courses
2
Calendar
Messages
Grades
Assist
Tools
Sign Out
Privacy
Terms
Accessibility
Courses
Search your courses
Terms
Current Courses
Filters
All courses
25
items per page
67 results
Current Courses
Favourites
cos791_s2_2024
COS 791 S2 2024
Original Course View
1
Open | T (Thambo) Nyathi | More info
imy220_s2_2024
IMY 220 S2 2024
Open | Multiple Instructors | More info
imy761_y1_2024
IMY 761 Y1 2024
Open | Multiple Instructors | More info
imy774_s2_2024
IMY 774 S2 2024
Open | Multiple Instructors | More info
imy779_s2_2024
IMY 779 S2 2024
Open | Multiple Instructors | More info

Components


clickUP Help
Courses
Kind of unecessary - more work to pass data
Institution Page
TK (Tayla) Orsmond
Activity Stream
Courses
Calendar
Messages
Grades
Assist
Tools
Sign Out
Privacy
Terms
Accessibility
Search your courses
Terms
Current Courses
Filters
All courses
25
items per page
67 results
Current Courses
Favourites
cos791_s2_2024
COS 791 S2 2024
Original Course View
Open | T (Thambo) Nyathi | More info
imy220_s2_2024
IMY 220 S2 2024
Open | Multiple Instructors | More info
imy761_y1_2024
IMY 761 Y1 2024
Open | Multiple Instructors | More info
imy774_s2_2024
IMY 774 S2 2024
Open | Multiple Instructors | More info
imy779_s2_2024
IMY 779 S2 2024
Open | Multiple Instructors | More info

Creating Components

```
class ComponentName extends React.Component {...}
```

1. **Capital letter** (very important, React will sometimes try render an HTML element otherwise - also convention)
2. `extends React.Component` (using some of the basic React component library functionality - same thing as inheriting from a C++, Java or JS class)
3. `{...}` (JS class functionality goes here - with some added functions React provides in the base class (we'll get to this in later lectures))

Creating Components

Two must-haves:

constructor()

1. `constructor(props)` - anything you pass to the component as props…
2. `super(props)` - …must also be passed to the parent (in this case React.component)

render()

1. Returns JSX (HTML and JS in `{}`)
2. Tells React what elements to render and how

⭐ Creating Components: Exercise

Note: Using babel CDN (do NOT do this in production - very slow!)

- *Just to get used to it*

Use the files off of ClickUP. In the file called **App.js**, implement (5mins):

- A simple contact card component called `Contact` that takes a person's name, surname and email and displays it as shown here.
- The values can be **hardcoded** for now.

*Render this component on the index.html page by finishing the code in App.js to **render the Contact component inside the "#root" div**.*

Contact Alice Apple

Email: alice.apple@email.com

Using Components

Components can be used inside other components!

- Encapsulate functionality and reuse it in other parts of your code
- `export class ComponentName {…} // like with Node modules`
- `export default ComponentName // if only one component in file`
- `import ComponentName from ‘./Component’ // .js is implicit`
- Use component JSX in **render()**

Important:

- Only **one** root element (`React.createElement` function under the hood only takes one root parameter)
- To not bloat the HTML with s, use React **fragments**
  - OLD method in videos: <React.fragment>
  - ⭐⭐⭐ **NEW method: `<Fragment></Fragment>` or `<></>`**

Using Components

How do we pass data between components?

- **props**
  - Basically like arguments in a function
  - Used in the constructor to set state - state used in **render()**
  - For simple components, props can be used right in **render()**
  - In the component **JSX passed in like 'attributes'**
    - *Aside: className not class*

```
<ClassName prop1="primitive prop" prop2={Js prop} />
```

As many props as we want.

⭐ Using Components: Exercise

Note: Using babel CDN (do NOT do this in production - very slow!)

- *Just to get used to it*

Use the same files off of ClickUP. Implement in **App.js** (5-10mins):

- Pass name, surname and email values to `Contact` component using **props**. Props can be used directly in render() since this component doesn't need state.
- Create a `ContactList` **component** list that takes a JS array of contact objects as its prop and renders a list of **Contact** components.

***Render** the **ContactList** component to index.html using the same method as before (but instead of Contact, it's ContactList). Pass in the provided list as a **prop**.*

You do not need to export / import since they are in the same file.

Contact Peter Plum

Email: peter.plum@email.com

Contact Alice Apple

Email: alice.apple@email.com

Contact Percy Pear

Email: percy.pear@email.com

Contact Olive Orange

Email: olive.orange@email.com

Contact Beatrice Banana

Email: beatrice.banana@email.com

Contact Graham Grape

Email: graham.grape@email.com

Contact Fiona Fig

Email: fiona.fig@email.com

Component State

Components can have **state**

- **React interactivity**
- Tells React what state the application is in at a point, and **when React should re-render the browser (update DOM) to reflect the updated state**.

Several things can trigger state change / re-render:

1. **Initial** component load (first-time)
2. Page **reload** (same as initial)
3. State variable updates to a **different value** than its current one
4. **Forcing** reload (not recommended)

Component State

This gets pretty complicated pretty fast.

We will talk about this again in future lectures.

For now, let’s look at simple state.

Component State

`this.state`

- `state` is similar to `props` but has built-in functionality for updating / re-rendering (using virtual DOM)
  - **VERY BAD IDEA to mutate (change) `props`**
- **Has to be initialised as a JS object in the constructor `{}`**

Modify state:

- **Must call `setState()` NOT modify state variables directly (i.e., like vars)**
- React handles the state for you - automatically updates the elements for you whenever you call setState() - will NOT work if you try modify directly
- Complexities: Batch updates, value comparison, etc. (Future Lecture)

Component State - Events

JS event handlers to handle user input

- `onclick` => `onClick` (React version of the event handler)
- Define the function used onClick in the class (self-contained functionality)

JS Class functions **not automatically bound to class** (does not have access to component's `this`)

- Have to manually bind them

`.bind()` - change what `this` refers to in a specific function

- Set the scope of `this` to that of the passed in argument - i.e., change what `this` refers to
- Correct place to bind = **constructor**

Functions can then modify state with `setState()`

Component State - Refs

Refs = Interact with child elements

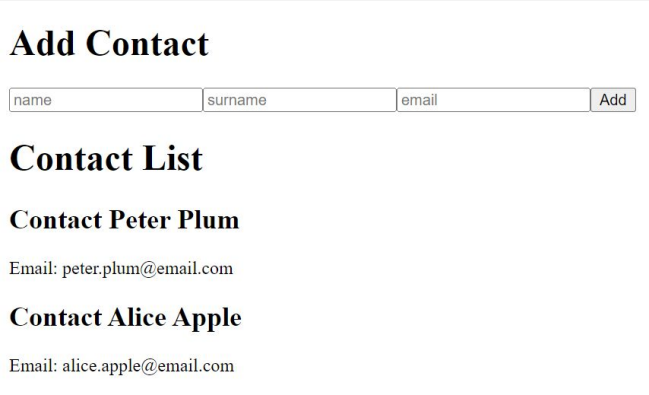
- Access values from child elements such as inputs / focus inputs
- **Refs are created with `React.createRef()`**

```
this.refName = React.createRef()
```

```
<input … ref={this.refName} />
```

- Access element: `this.refName.current`
- Access element value (if input): `this.refName.current.value`

⭐ Component State - Exercise



Note: Using babel CDN (do NOT do this in production - very slow!)

- *Just to get used to it*

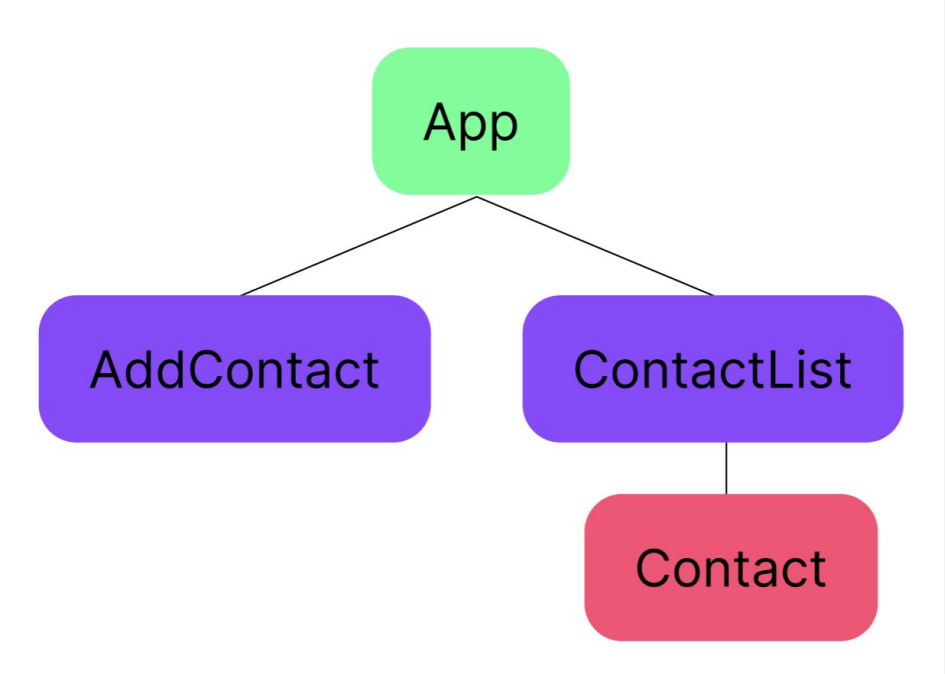
Use the same files off of ClickUP. In the file called **App.js** implement (10-15mins):

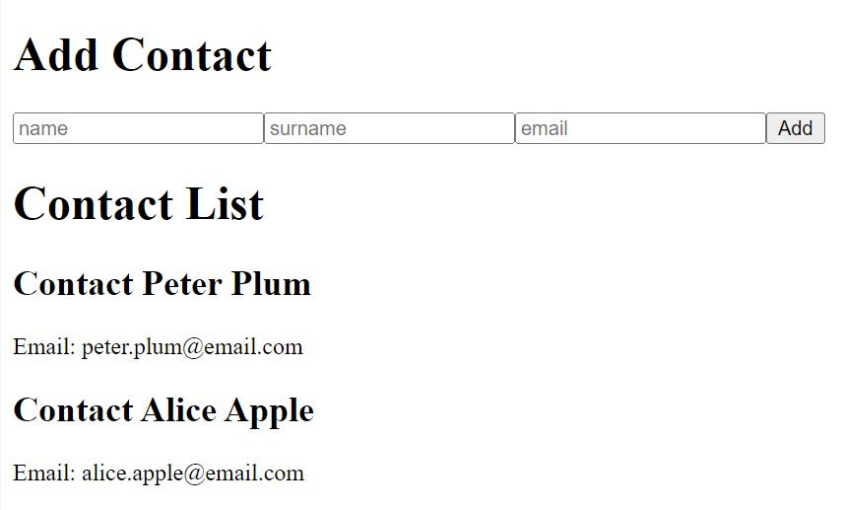
- A component called **`AddContact`** that contains a **form** to add contacts to the list. It should define a function called **addContact** that it uses to add contacts to the list by **getting the form values (hint: ref)**, and then calling **onContactAdded** that is passed in as a prop.
- A component called **App** that acts as the parent component.
  - It should have two children: **AddContact** and **ContactList**.
  - It should take in a list of contacts as its prop and use this to initialize its state. It should pass this state to its child **ContactList**.
  - It should also define a **onContactAdded** function which updates its state (adds a contact to the array). It should pass this function to its other child **AddContact**.

*Render the **App** component to index.html, passing it the provided list of contacts.*

⭐ Components - Exercise

Final structure should look like this:





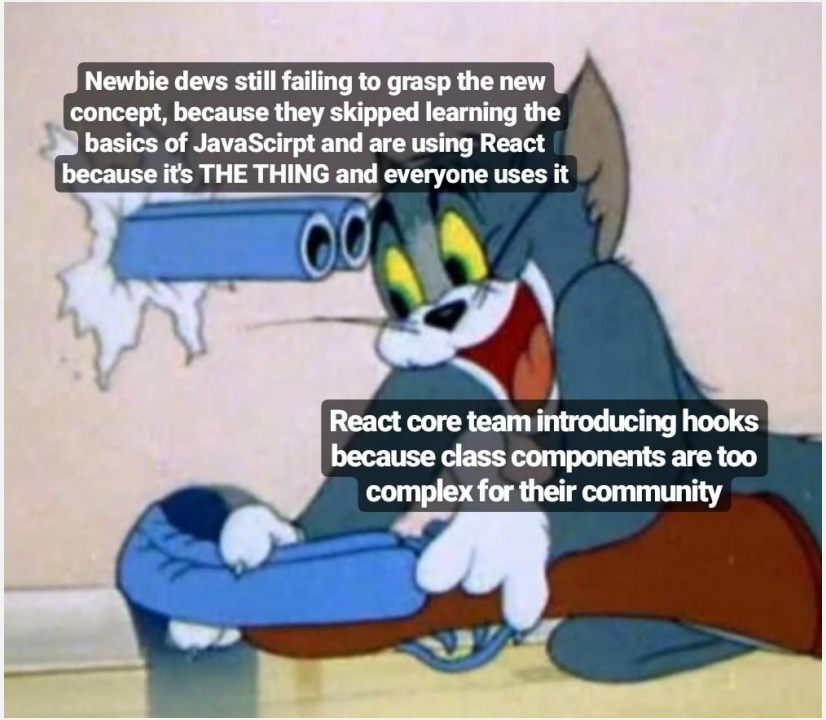
Types of Components (in the Wild)

Functional components

- Newer
- Generally preferred (even by React themselves)
- Two forms: `function()=` and `const = () =>`

Class components

- Simpler to understand (OOP)
- Still run into some of the same errors if you use functional over class
- Recommended for you



Types of Components (in the Wild) - Differences

Class

```
class App extends React.component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div>
        <h1>My App</h1>
        <p>My app is awesome</p>
      </div>
    );
  }
}
```

Functional

```
function App(props) {
  return (
    <div>
      <h1>My App</h1>
      <p>My app is awesome</p>
    </div>
  );
}
```

Types of Components (in the Wild) - Differences

Class

```
class App extends React.component {
  constructor(props) {
    super(props);
    this.state = {
      prop1: this.props.prop1,
    };
    this.ref1 = React.createRef();
    this.function1 = this.function1.bind(this);
  }

  function1() {
    this.setState({ prop1: "new value" });
  }

  render() {
    return (
      <div>
        <h1>My App</h1>
        <p>My app is awesome</p>
        <p>Prop1 {this.state.prop1}</p>
        <button onClick={this.function1}>Change</button>
      </div>
    );
  }
}
```

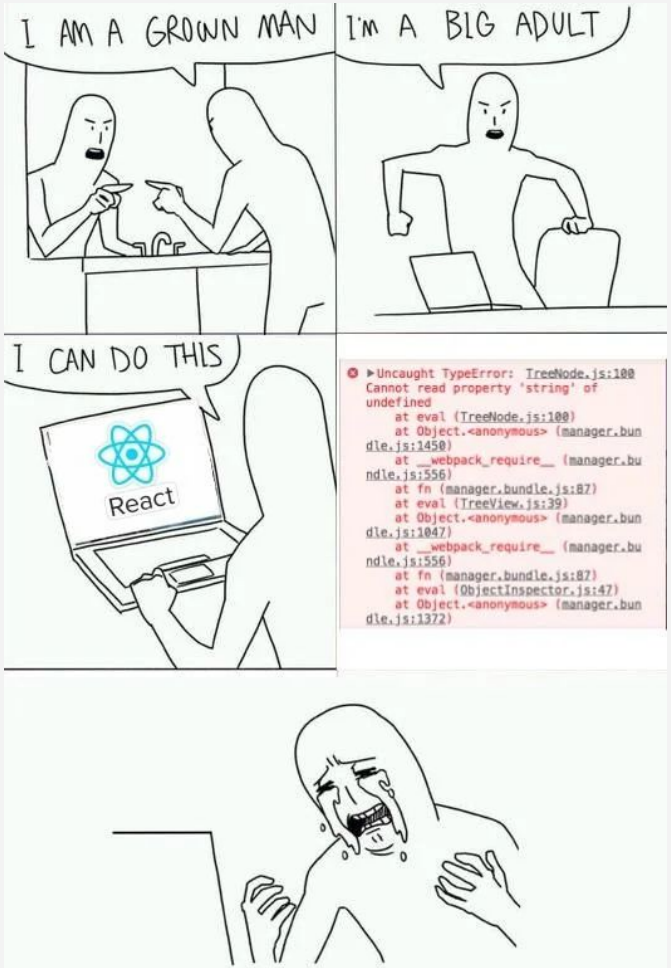
Functional

```
function App({prop1}) {
  const [state1, setState1] = useState(prop1);
  const ref1 = useRef();

  //const function1 = () => { setState1("new value"); };

  useEffect(() => {
    ref1.current = "new value";
    setState1("new value");
  }, []);

  return (
    <div>
      <h1>My App</h1>
      <p>My app is awesome</p>
      <p>Prop1 {state1}</p>
      <p>Ref1 {ref1.current}</p>
      <button onClick={function1}>Change</button>
    </div>
  );
}
```

Troubleshooting React 🔥🔥🔥

You are likely to run into issues when dealing with React State and Asynchronous behaviour

- COS 226 is your friend when understanding WHY these issues arise.
- I will try my best to guide you through some of these issues when we get there.
- Lots of frameworks exist for managing React's state for more complex applications.
- For our simple application (semester project) you will be fine without them.



Extension Exercises We Don’t have Time for:

- Edit Contact with a form that auto-populates values
- Reverse the Contact list
- Search to filter the list
- Display one Contact from a list of clickable Contact items (requires adding a new component)
- React Assignments

React docs (be aware they use functional components): https://react.dev/learn/tutorial-tic-tac-toe

Questions?

bufferhead-code / **nextjs-use-php**

Public

Use PHP code right within your React / Next.js App. With "use php";

```
return (
    <button
        formAction={async () => {
            'use php'
            (new PDO('mysql:host=loca
                ->prepare("INSERT INT
                ->execute(array('new'
            }}>
        Insert Bookmark
    </button>
)
```