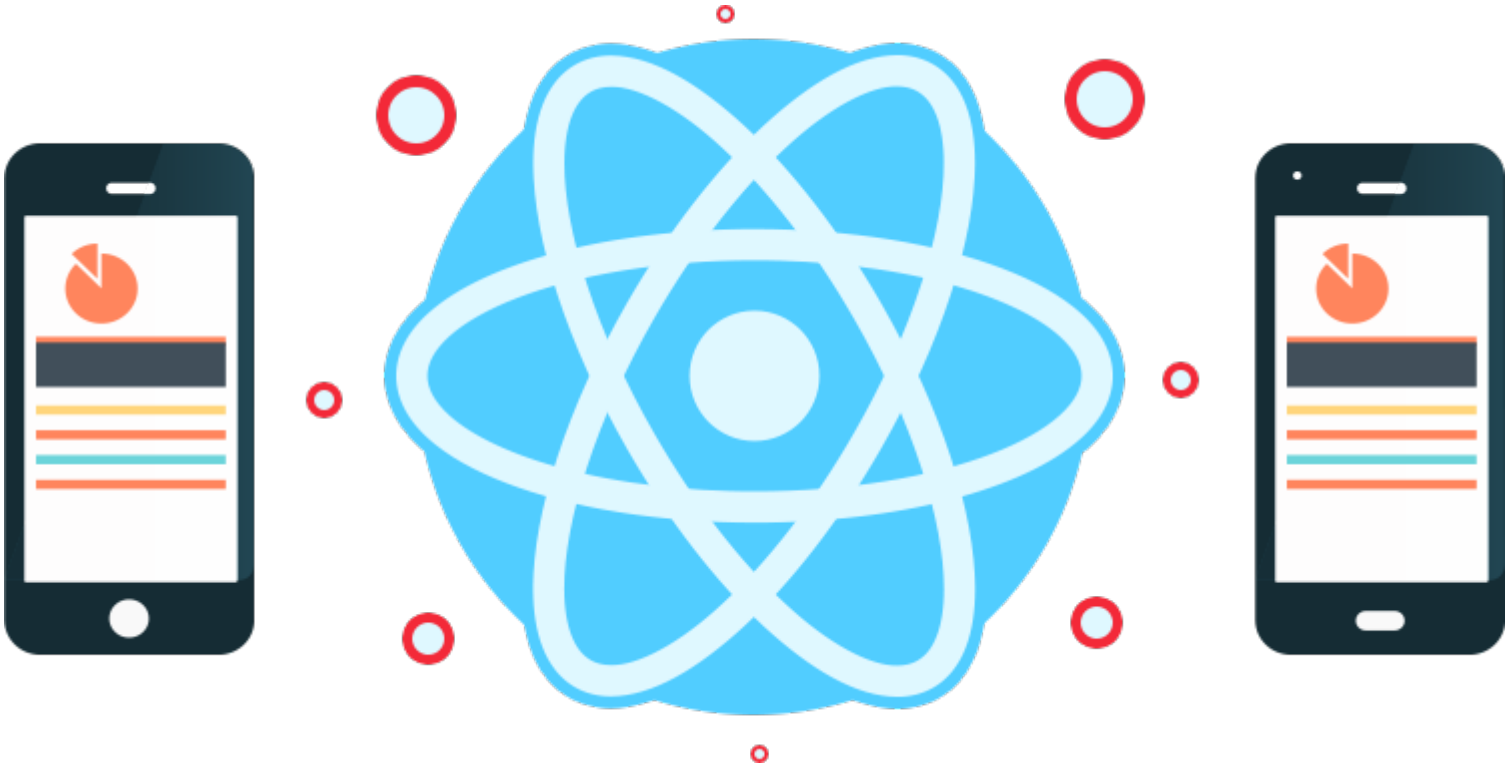
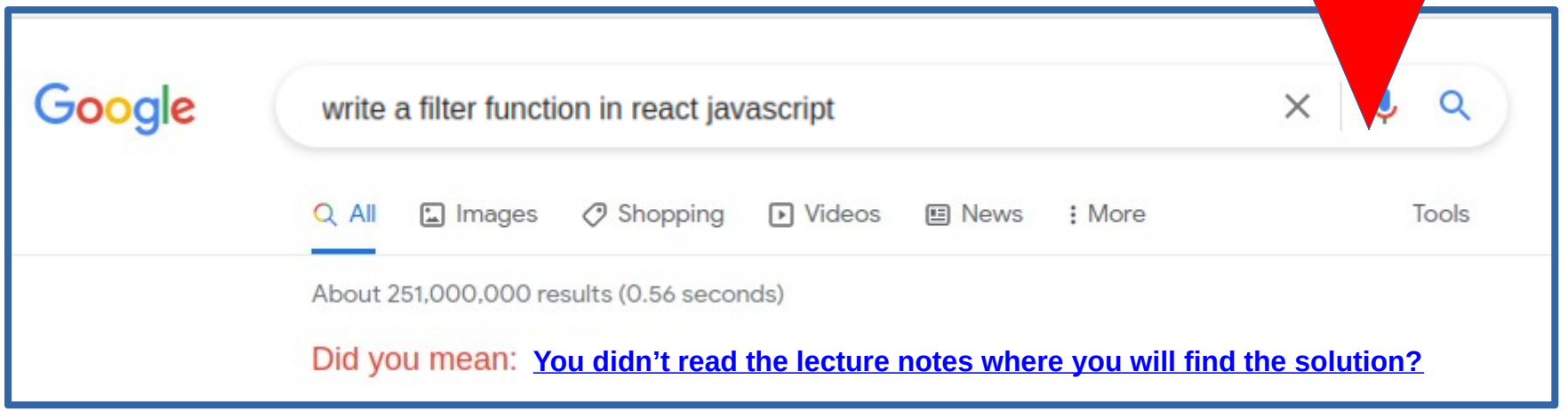
CS385 Mobile Application Development (Lecture 5)



Peter Mooney

Lab 1


Google
write a filter function in react javascript
Google Search
I'm Feeling Lucky
Google offered in: Gaeilge


Google
write a filter function in react javascript
All
Images
Shopping
Videos
News
More
Tools
About 251,000,000 results (0.56 seconds)
Did you mean: You didn't read the lecture notes where you will find the solution?

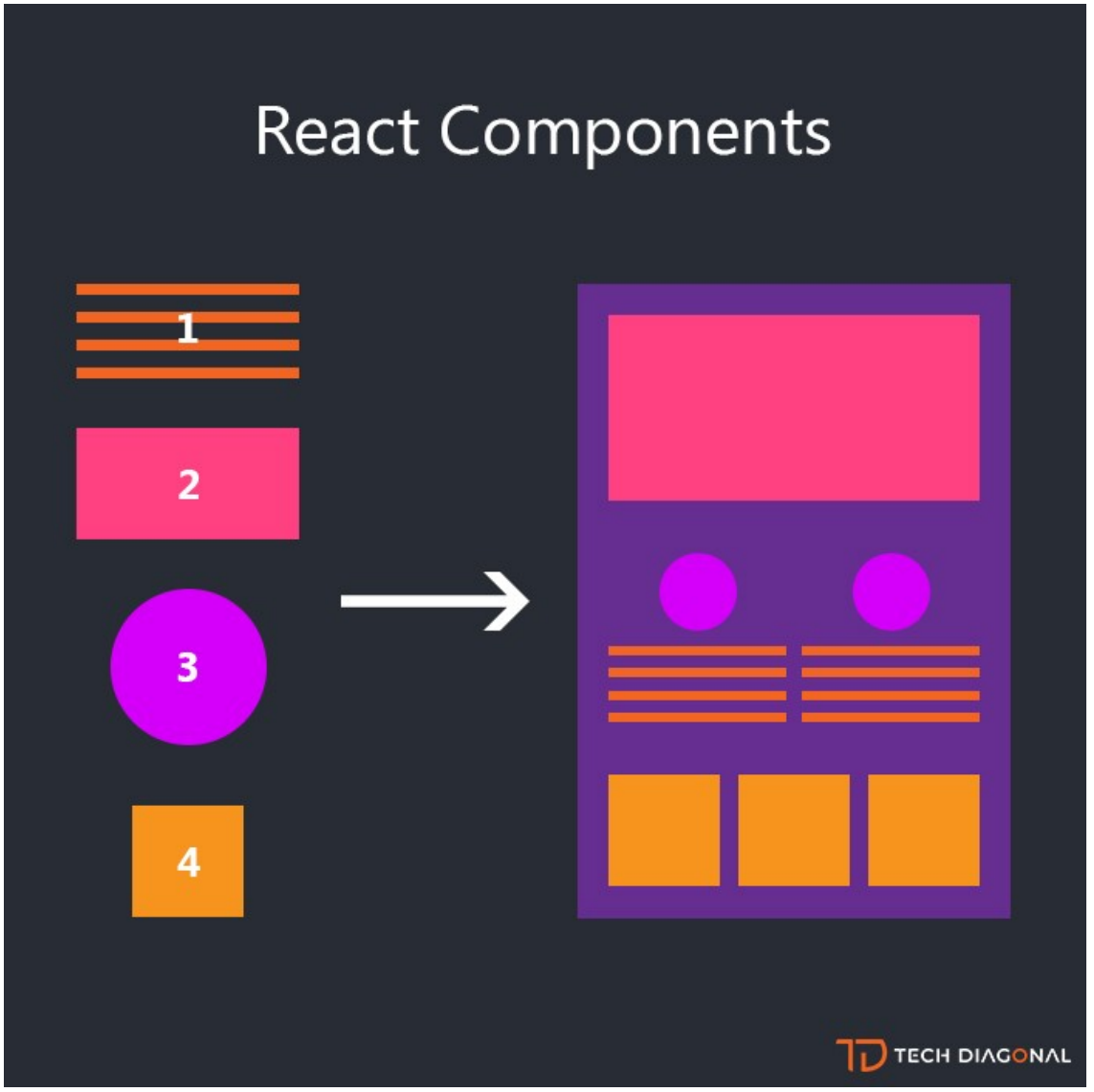
Lecture 5 and 6 introduce some of the most important aspects of React Javascript

- We will learn about
  - **Components in React applications** – what are they and how do we create them (Parent and Child)
  - **The use of `props`** – which is the principal means of passing data and functionality between components
  - Starting to **maintain state in React applications**
  - **Understanding how state, props, and components essentially work together** to provide the functionality in a React application.

We need one full example which includes components, props and state

- It is difficult to teach and learn these concepts in isolation. So, **we'll build a simple example which includes all of the concepts**
- It is important that you try this code out yourself in CodeSandBox.
- **It is important that you understand these ideas conceptually as they appear in almost every Javascript-based app framework!**

Components in React

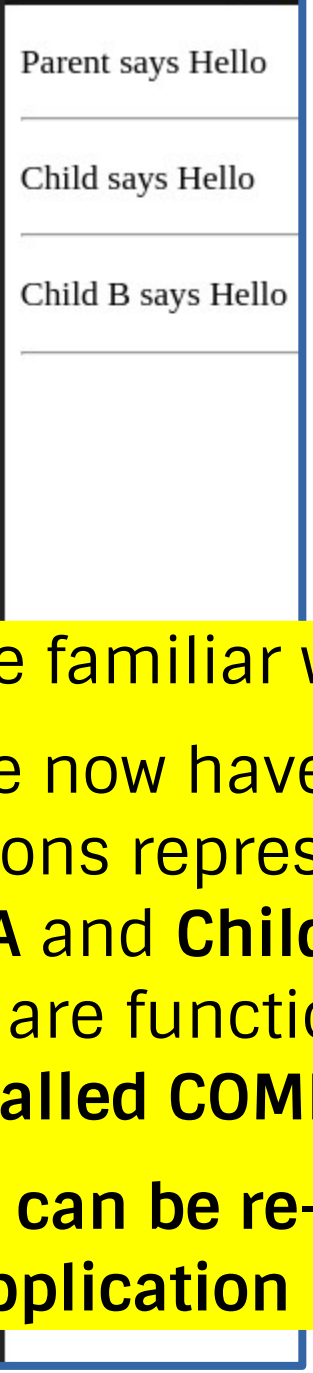


- **Mobile applications in React are made up of components (which can be re-used multiple times in the application)**

Part 1 – a parent component and two child components

```
6   // App is the parent component
7   function App() {
8     return (
9       <>
10        <p>Parent says Hello</p><hr />
11        <ChildA />
12        <ChildB />
13      </>
14    );
15  }
16
17  function ChildA() { // ChildA is a child component
18    return (
19      <>
20        <p>Child says Hello</p><hr />
21      </>
22    );
23  }
24
25  function ChildB() { // Childb is a child component
26    return (
27      <>
28        <p>Child B says Hello</p><hr />
29      </>
30    );
31  }
```

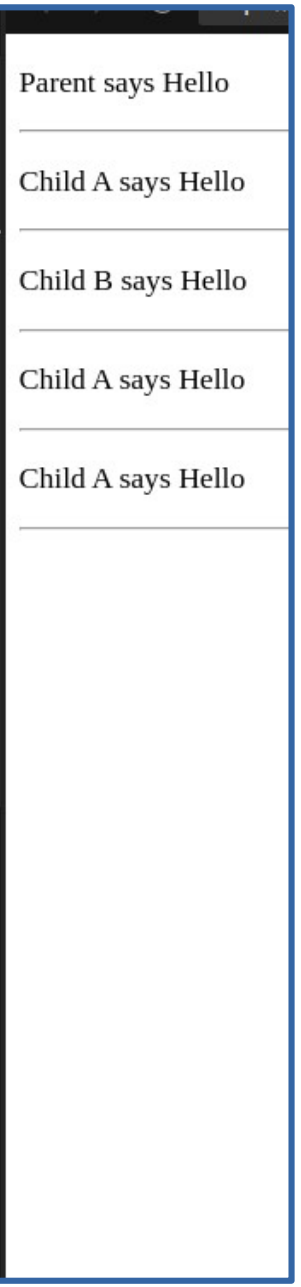
Look on Line 11 and Line 12 how we can ‘call’ or ‘invoke’ the two child components. We use a HTML tag approach (they are really JSX)



- We are familiar with **App()**
- But we now have two new functions representing **ChildA** and **ChildB** – these are functions but also **called COMPONENTS.**
- **These can be re-used in our application**

Part 1 – Here we reuse the Child A component (see line 13 + 14)

```
6   // App is the parent component
7   function App() {
8     return (
9       <>
10        <p>Parent says Hello</p><hr />
11        <ChildA />
12        <ChildB />
13        <ChildA />
14        <ChildA />
15      </>
16    );
17  }
18
19  function ChildA() { // ChildA is a child component
20    return (
21      <>
22        <p>Child A says Hello</p><hr />
23      </>
24    );
25  }
26
27  function ChildB() { // Childb is a child component
28    return (
29      <>
30        <p>Child B says Hello</p><hr />
31      </>
32    );
33  }
```



Components are really useful in larger React application

- They are generally used to “encapsulate” code or logic into nice packages.
- Components are just React functions that return JSX (like `App()`).
- These components can actually do anything we would do in the `App()` function.
- **There is a component hierarchy** – `App()` is usually the parent – then every other component defined is usually at least a child (could be a grand-child)
- **This gives rise to the need for PARENT-CHILD communication – or simply a way to pass data and functionality between the PARENT and CHILD functions or components**

To faciliate parent-child communication we use a feature of React called `props`

- In React props are defined for every function or component (we don't always need to use `props`).
- Think of `props` as a way of gathering or managing the parameters of a function.
- **`Props`, in React, faciliates parent-child communication between components.**

Part 2: Parent-Child communication using props – first example

```
7  function App() {
8    let parentX = 100;  let parentY = 88;
9    return (
10     <>
11       <p>Parent says Hello</p><hr />
12       <ChildA xFromParent={parentX}/>
13       <ChildB yFromParent={parentY}/>
14     </>
15   );
16 }
17 // props is now defined for the child A
18 function ChildA(props) { // ChildA is a child component
19   return (
20     <>
21       <p>Child A says Hello</p>
22       <p>Parent x = {props.xFromParent}</p><hr />
23     </>
24   );
25 }
26 // props is now defined for the child B
27 function ChildB(props) { // Childb is a child component
28   return (
29     <>
30       <p>Child B says Hello</p>
31       <p>Parent Y = {props.yFromParent}</p>
32       <hr />
33     </>
34   );
35 }
```

Parent says Hello

Child A says Hello

Parent x = 100

Child B says Hello

Parent Y = 88

- There are two variables in App0 – the parent. **We want to pass or communicate this data to the Children components**.
- So we use props (line 18 and 27). **Props faciliates communication**.
- **Notice how the variables are passed** – line 12, 13
- **Notice how the variables or data are recieved in the children components** (line 22, line 31)

KEY CONCEPT – props allow input into functions and components

- Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.
- RECAP: Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.

- **Whether you declare a component as a function or a class, it must never modify its own props.**
- **All React components must act like pure functions with respect to their props.**

We can use `props` to pass objects, arrays of objects, and even handlers to functions

```
6  // App is the parent component
7  function App() {
8  let parentX = {name: "Peter", course: "cs385"};
9  let parentY = {name: "Siobhan", course: "cs621"};
10   return (
11     <>
12       <p>Parent says Hello</p><hr />
13       <ChildA xFromParent={parentX} />
14       <ChildB yFromParent={parentY} />
15     </>
16   );
17 }
18 // props is now defined for the child A
19 function ChildA(props) {
20   // ChildA is a child component
21   return (
22     <>
23       <p>Child A says Hello</p>
24       <p>Parent X Name = {props.xFromParent.name}</p><hr />
25     </>
26   );
27 }
28 // props is now defined for the child B
29 function ChildB(props) {
30   // Childb is a child component
31   return (
32     <>
33       <p>Child B says Hello</p>
34       <p>Parent Y Course = {props.yFromParent.course}</p><hr />
35     </>
36   );
```

- Here we see the same example. **This time, `parentX` and `parentY` are JSON objects.**
- IMPORTANT – **notice how we access the properties of these parent objects in the children** (line 24 and line 34)

We can use `props` to pass an array of objects from Parent to Child

```
function App() {
  let parentX = [
    { name: "Peter", course: "cs385" },
    { name: "Siobhan", course: "cs621" }
  ];
  return (
    <>
      <p>Parent says Hello</p><hr />
      <ChildA xFromParent={parentX} />
    </>
  );
}
// props is now defined for the child A
// we use a map function as the props is an array
function ChildA(props) {
  // ChildA is a child component
  return (
    <>
      <p>Child A says Hello</p>
      {props.xFromParent.map((i, index) => (
        <p key={index}>
          Name: {i.name}, Course: {i.course}
        </p>
      ))}
    </>
  );
```

Parent says Hello

Child A says Hello

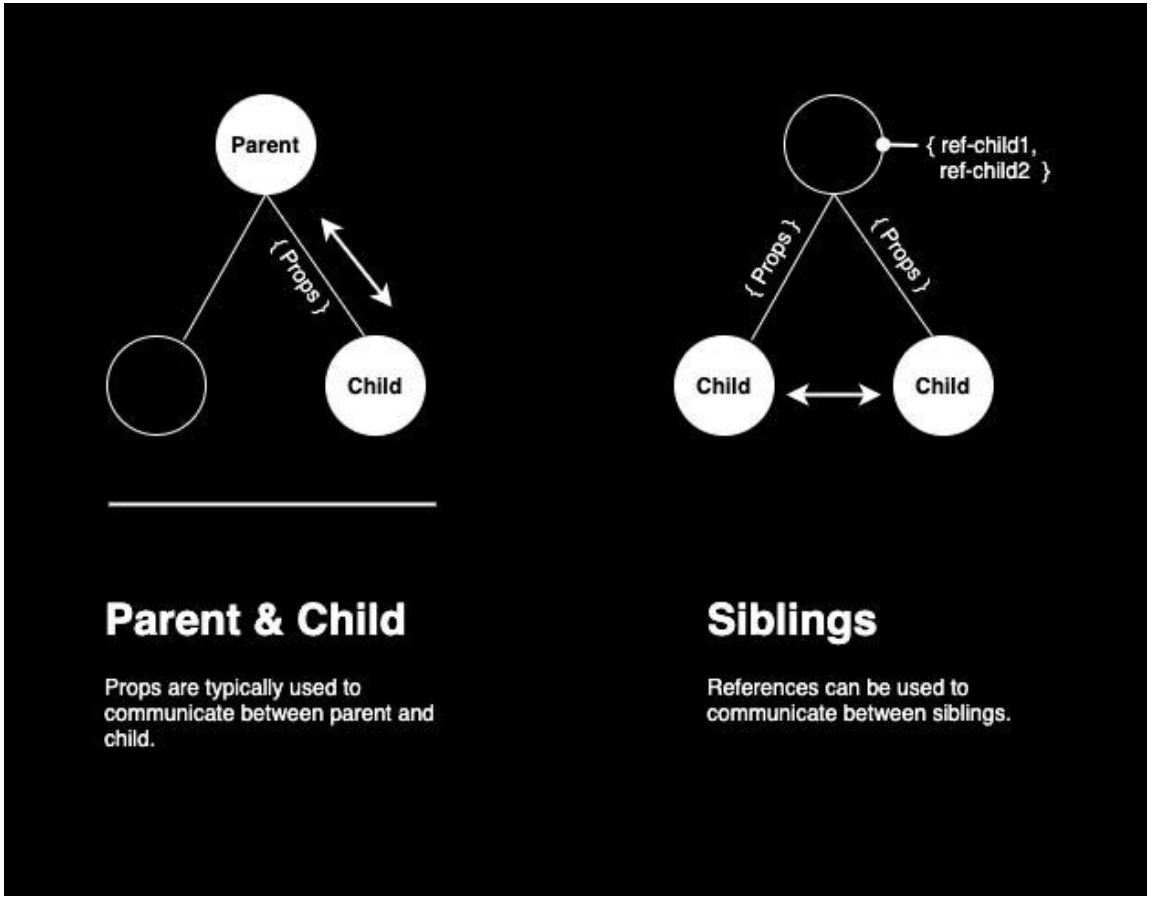
Name: Peter, Course: cs385

Name: Siobhan, Course: cs621

- Here, **the parent has an array of JSON objects.**
- **We want the child component to RENDER the array**. So we need to use **`props`** to pass the array **`parentX`** to the Child component **`ChildA`**

As our application code becomes larger, we will need components to organise our code and functionality

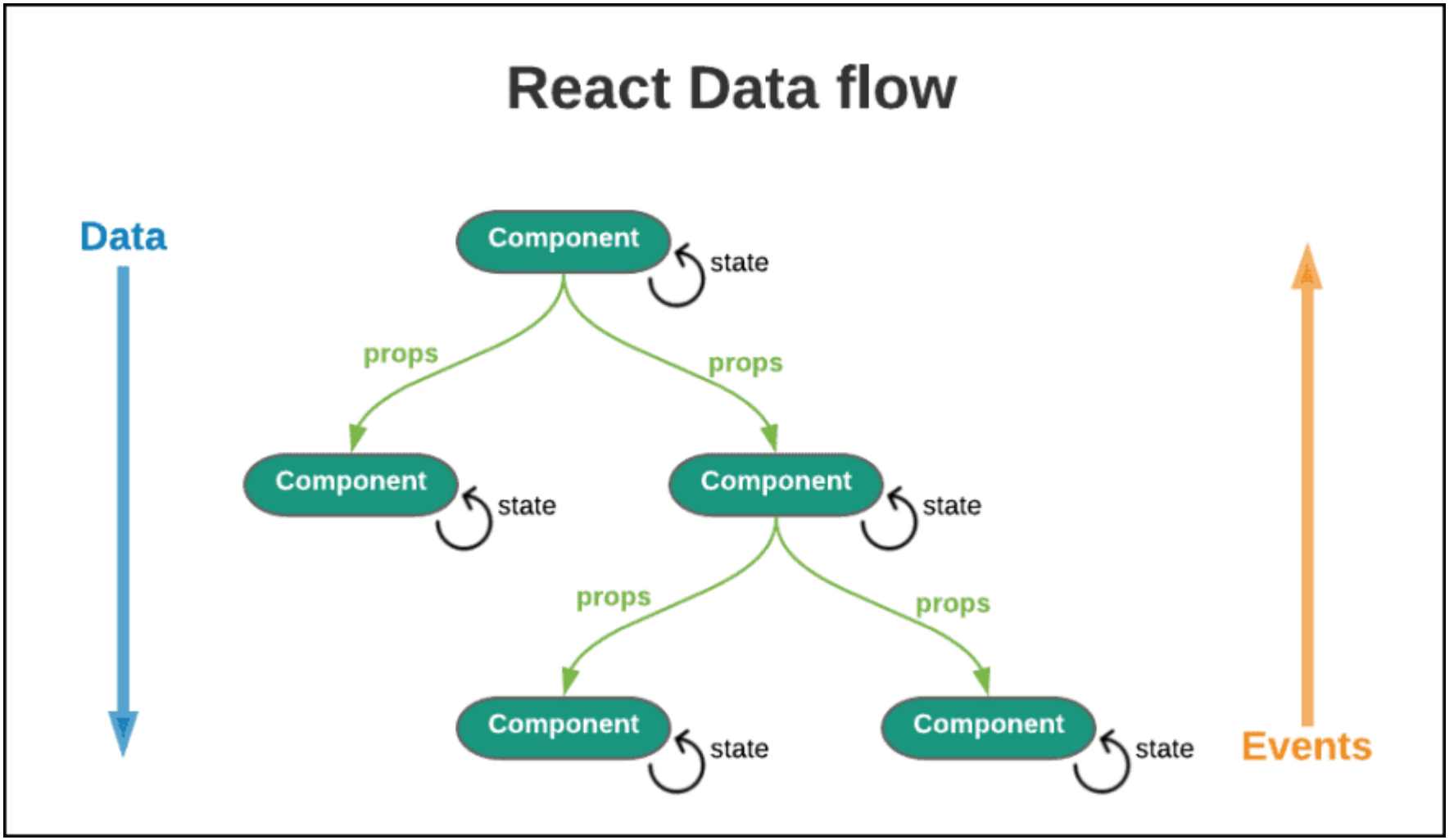
- As mentioned, components are used to organise or separate code into distinct units or functions. This is good software development practice, especially for larger projects



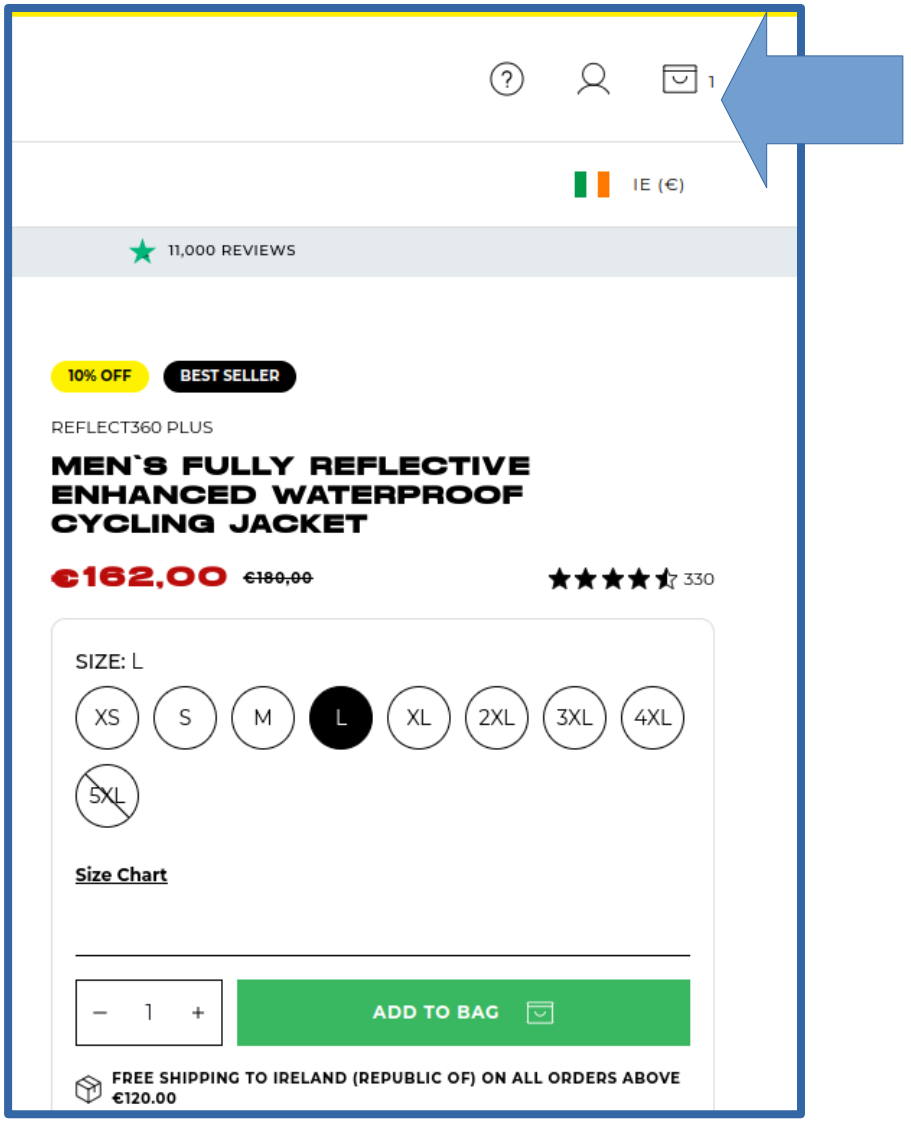
Up to this point we have seen how `props` faciliate communication or data passing between components or functions

- All of our examples, so far, have featured static data – that is, the data does not change or get updated in our code.
- We are simply rendering the values or contents of the variables passed or communicated by props.
- **Let's start to look at how we can begin to allow for updating or dynamic variables** (and communication in React)

Maintaining state in React Javascript applications

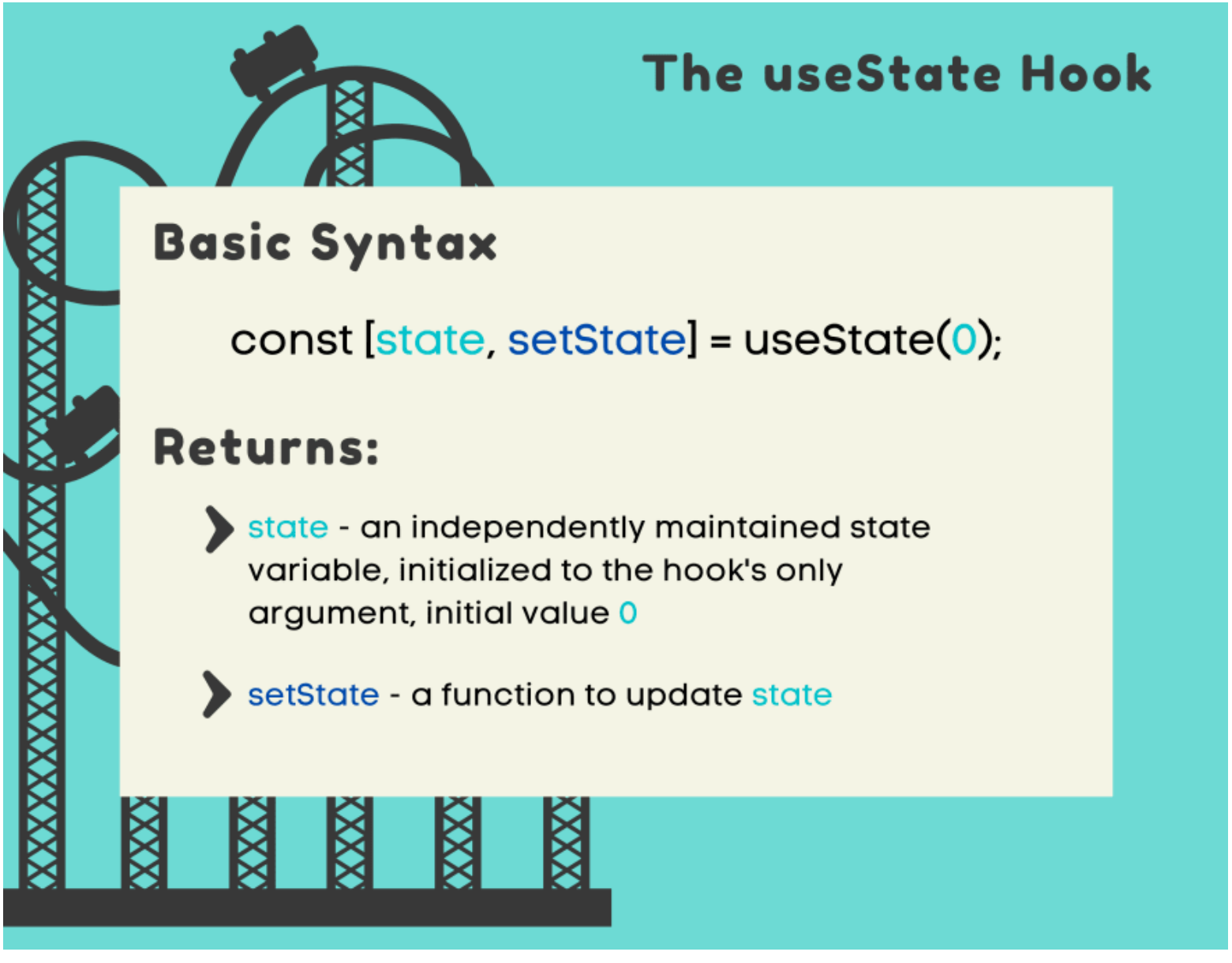


How to think about state? Imagine an online store.



- Example: I've just purchased a cycling jacket. It's in my basket in the online store.
- But I want to browse for other items.
- How does the online store application REMEMBER the contents of my basket?
- It **uses a concept called "state" where variables or data are stored and maintained right across the application (that is over all components)**

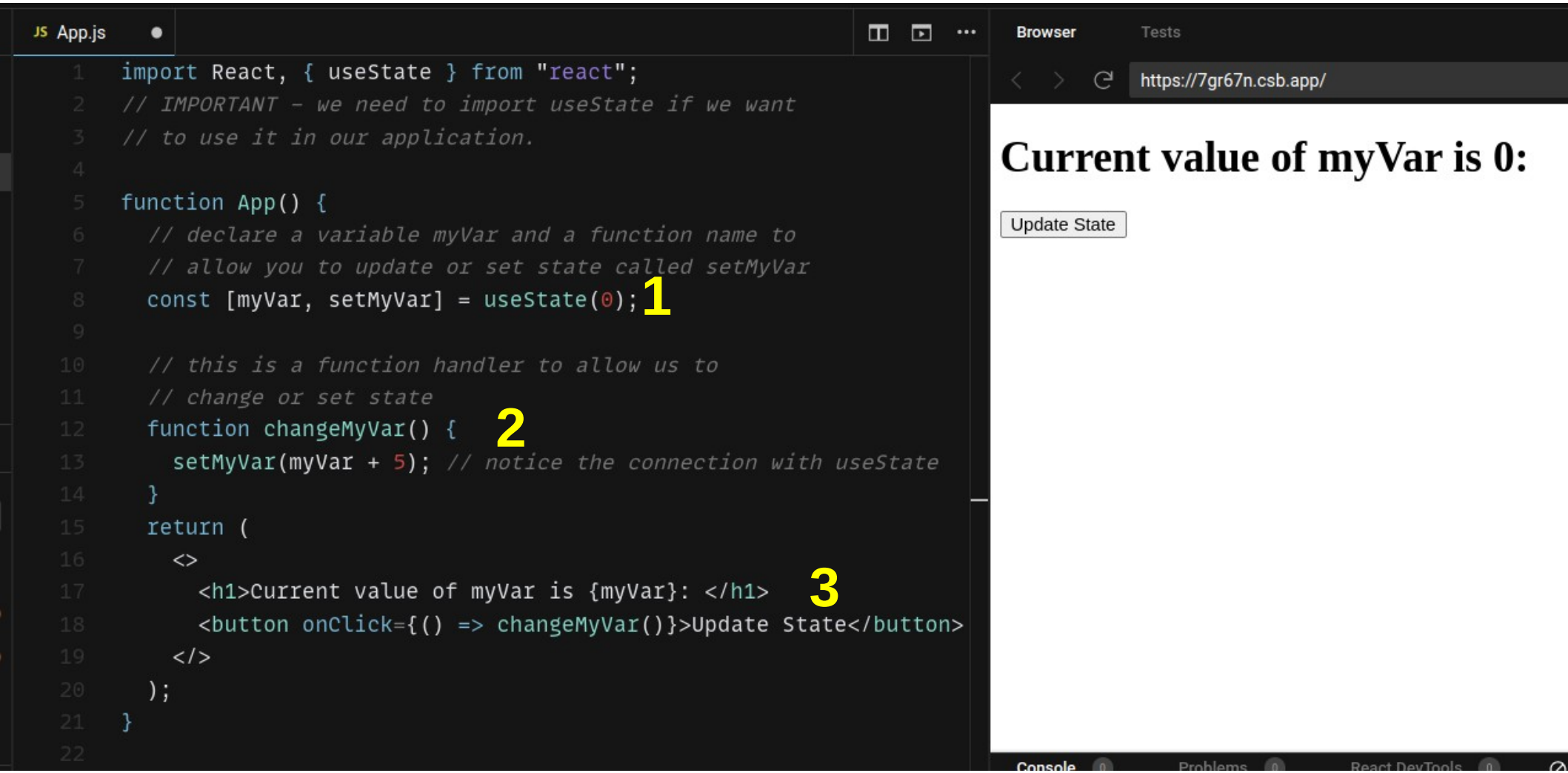
React provides HOOK called `useState` to help us maintain state



Key concept – we will useState to allow for a variable to be updated and maintained between parent and child

- For this example – it is necessary to include our first User Interface element – a BUTTON.
- We'll use the BUTTON to change the valuable of the variable – so everytime the user presses the BUTTON the variable will be updated.
- **This is a simple example – but we have multiple parts to the example.**

Example – using `useState` for the first time – with a `<button>`



Concept 1: Very important – here we use `useState` to declare our state variable, give it an initial value, and name a function to allow update

App.js

```
1  import React, { useState } from "react";
2  // IMPORTANT - we need to import useState if we want
3  // to use it in our application.
4
5  function App() {
6    // declare a variable myVar and a function name to
7    // allow you to update or set state called setMyVar
8    const [myVar, setMyVar] = useState(0);
9
10   // this is a function handler to allow us to
11   // change or set state
12   function changeMyVar() {
13     setMyVar(myVar + 5); // notice the connection with useState
14   }
15   return (
16     <>
17       <h1>Current value of myVar is {myVar}: </h1>
18       <button onClick={() => changeMyVar()}>Update State</button>
19     </>
20   );
21 }
22
```

1

2

3

Browser Tests

https://7gr67n.csb.app/

Current value of myVar is 0:

Update State

The hook `useState` is provided by React.

On line 8 we decide the initial value is 0.

The variable is called **myVar**

If we want to update **myVar** – we can use a function called **setMyVar** and pass the updated value

Console Problems React DevTools

Concept 1: Very important – here we use `useState` to declare our state variable, give it an initial value, and name a function to allow update

App.js

```
1  import React, { useState } from "react";
2  // IMPORTANT - we need to import useState if we want
3  // to use it in our application.
4
5  function App() {
6    // declare a variable myVar and a function name to
7    // allow you to update or set state called setMyVar
8    const [myVar, setMyVar] = useState(0);
9
10   // this is a function handler to allow us to
11   // change or set state
12   function changeMyVar() {
13     setMyVar(myVar + 5); // notice the connection with useState
14   }
15   return (
16     <>
17       <h1>Current value of myVar is {myVar}: </h1>
18       <button onClick={() => changeMyVar()}>Update State</button>
19     </>
20   );
21 }
22
```

1

2

3

Browser Tests

https://7gr67n.csb.app/

Current value of myVar is 0:

Update State

The hook `useState` is provided by React.

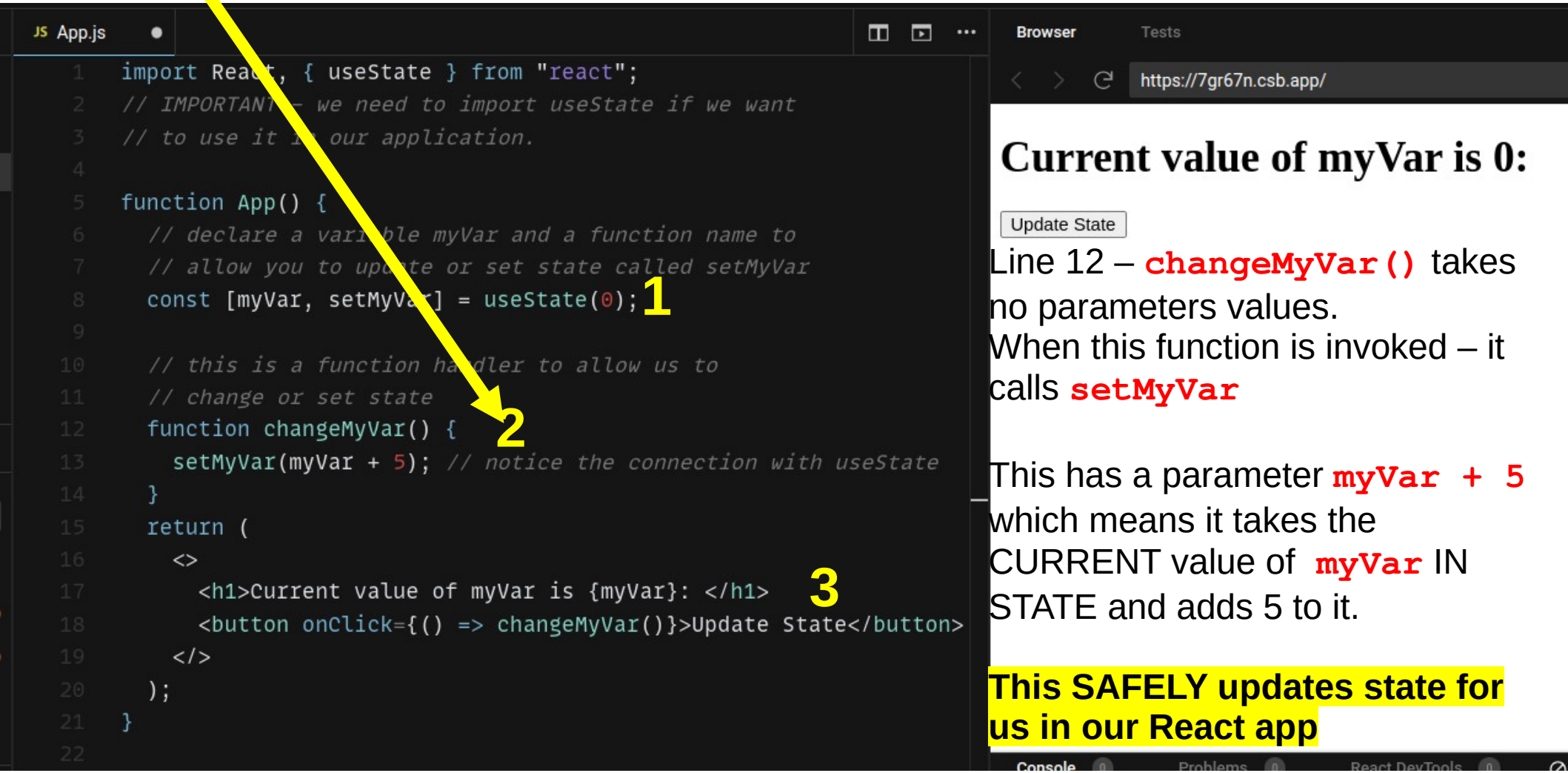
On line 8 we decide the initial value is 0.

The variable is called **myVar**

If we want to update **myVar** – we can use a function called **setMyVar** and pass the updated value

Console Problems React DevTools

Concept 2: This is where we provide a HANDLER function. This allows us to safely access the `setMyVar` function (provided by `useState` hook)



Line 12 – `changeMyVar()` takes no parameters values.
When this function is invoked – it calls `setMyVar`

This has a parameter `myVar + 5` which means it takes the CURRENT value of `myVar` IN STATE and adds 5 to it.

This SAFELY updates state for us in our React app

Concept 3: Here, a BUTTON CLICK ACTION (`onClick`) is used to invoke `changeMyVar` – the value of `myVar` is rendered and shown to update

```
1  import React, { useState } from "react";
2  // IMPORTANT - we need to import useState if we want
3  // to use it in our application.
4
5  function App() {
6    // declare a variable myVar and a function name to
7    // allow you to update or set state called setMyVar
8    const [myVar, setMyVar] = useState(0);
9
10   // this is a function handler to allow us to
11   // change or set state
12   function changeMyVar() {
13     setMyVar(myVar + 5); // notice the connection with useState
14   }
15   return (
16     <>
17       <h1>Current value of myVar is {myVar}: </h1>
18       <button onClick={() => changeMyVar()}>Update State</button>
19     </>
20   );
21 }
22
```

1

2

3

Browser Tests

https://7gr67n.csb.app/

Current value of myVar is 0:

Update State

In **HTML we declare a button using the <BUTTON> tag.**

Button elements then need some action to be performed – in the case of buttons this is `onClick` (usually)

Using the syntax on Line 18 we call the handler function on line 12 `changeMyVar()` so that we may update the value of the state variable `myVar`.

Console Problems React DevTools

Demo – file from Moodle

`Lecture5-UseState.js`

Concept 4: We can choose HOW we update the value of myVar here. It could be the result of additional code or computation

JS App.js

```
import React, { useState } from "react";
// IMPORTANT - we need to import useState if we want
// to use it in our application.

function App() {
  // declare a variable myVar and a function name to
  // allow you to update or set state called setMyVar
  const [myVar, setMyVar] = useState(0);

  // this is a function handler to allow us to
  // change or set state
  function changeMyVar() {
    setMyVar(myVar + 5); // notice the connection with useState
  }
  return (
    <>
      <h1>Current value of myVar is {myVar}: </h1>
      <button onClick={() => changeMyVar()}>Update State</button>
    </>
  );
}
```

1

2 ,4

3

Browser Tests

https://7gr67n.csb.app/

Current value of myVar is 0:

Update State

The use of setMyVar from the useState hook is very important.

It ensures that the state variable myVar is updated correctly by React.

However, it also allows us to include additional, more involved code or computation – depending on our application. The next slide is an example

Console Problems React DevTools

Concept 4: Notice how in this example – we have added some additional computation. It's not very exciting but shows how we can add complexity

App.js

```
1  import React, { useState } from "react";
2  // IMPORTANT - we need to import useState if we want
3  // to use it in our application.
4
5  function App() {
6    // declare a variable myVar and a function name to
7    // allow you to update or set state called setMyVar
8    const [myVar, setMyVar] = useState(0);
9
10   // this is a function handler to allow us to
12   function changeMyVar() {
13     let x = Math.random() * 100;
14     console.log(x);
15     if (x >= 50) setMyVar(myVar + 1);
16     // notice the connection with useState
17     else setMyVar(myVar - 1);
18   }
19   return (
20     <>
21       <h1>Current value of myVar is {myVar}: </h1>
22       <button onClick={() => changeMyVar()}>Update State</button>
23     </>
24   );
26
27 export default App;
```

Browser Tests

https://7gr67n.csb.app/

Current value of myVar is 1:

Update State

Now, in `changeMyVar()` we add or subtract 1 based on the value generated by x.

This isn't really a very useful function – but it means we can supply whatever code we need to update the state variable `myVar`.

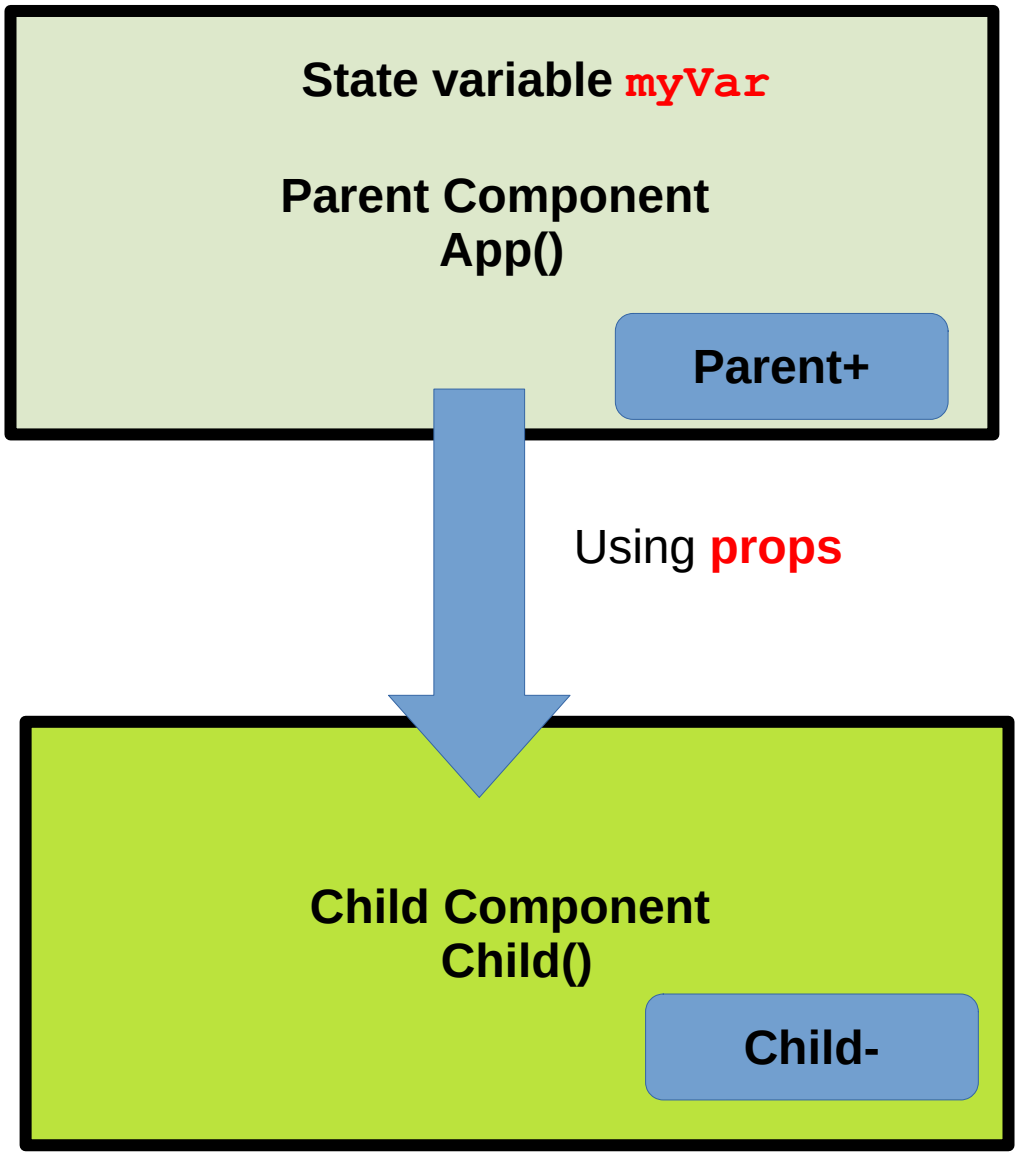
Console 0 Problems 0 React DevTools 0

```
24.96692941727805
21.64919584828313
63.71570687668442
85.59739909866293
```

Finally, let's join up what we have learned with Components, `props`, and `useState` to see a full example of Parent-Child Communication in React

Overall schematic diagram of our parent-child app example

The `App()` function or component has a handler or update function called `changeMyVar` which takes a single integer parameter



The **Parent+** button should add 1 to the current value of `myVar`

The **Child-** button should subtract 1 from the current value of `myVar`

Parent-Child app example – full source code

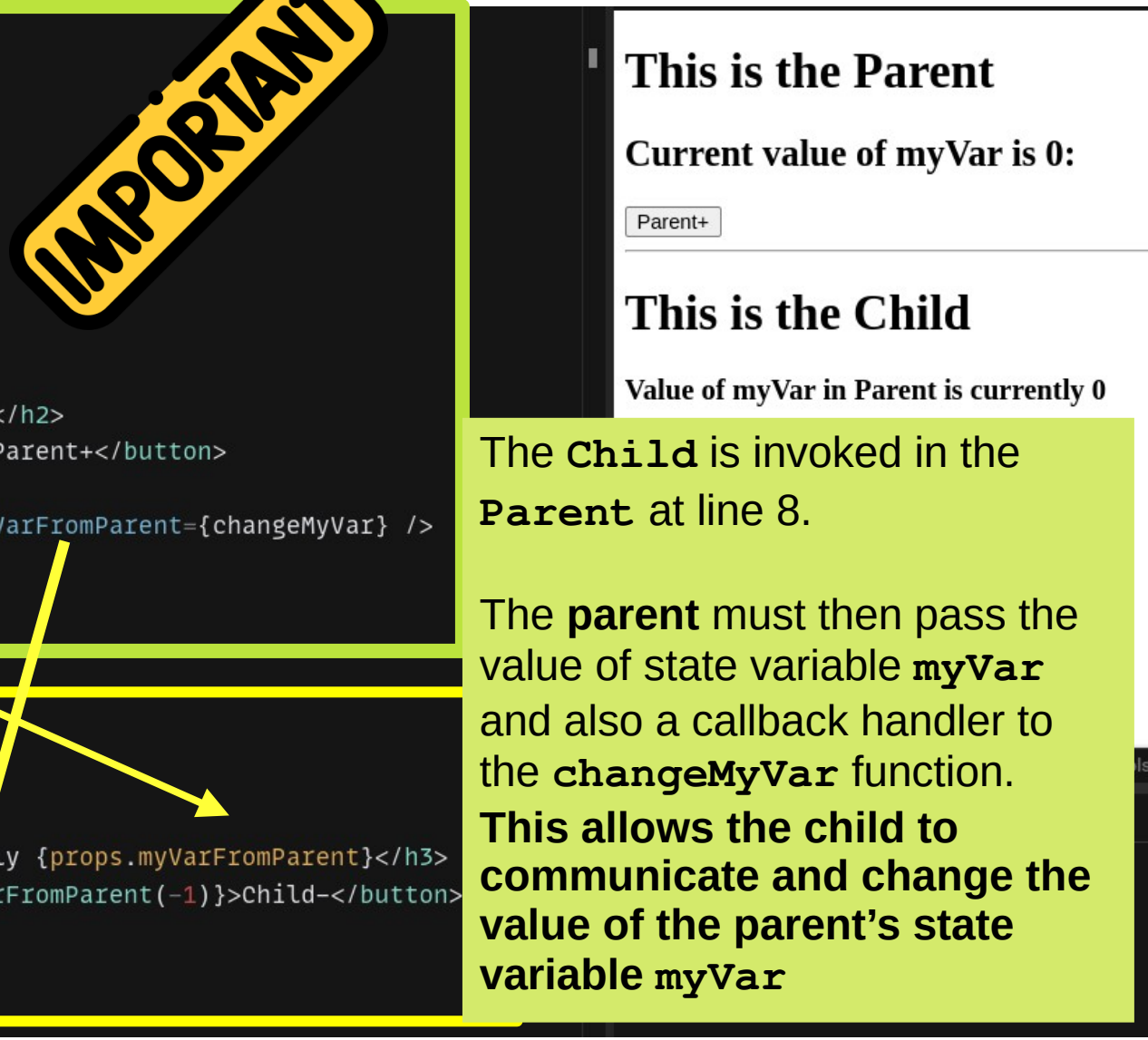
```jsx
function App() {
  const [myVar, setMyVar] = useState(0);

// Change my var has a parameter variable i
  function changeMyVar(i) {
    setMyVar(myVar + i);
  }
  return (
    <>
      <h1>This is the Parent</h1>
      <h2>Current value of myVar is {myVar}: </h2>
      <button onClick={() => changeMyVar(1)}>Parent+</button>
      <hr />
      <Child myVarFromParent={myVar} changeMyVarFromParent={changeMyVar} />
    </>
  );
}
// This is the Child component - using props
function Child(props) {
  return (
    <>
      <h1>This is the Child</h1>
      <h3>Value of myVar in Parent is currently {props.myVarFromParent}</h3>
      <button onClick={() => props.changeMyVarFromParent(-1)}>Child-</button>
    </>
  );
}
```

It is props that faciliates the parent-child communication

```
5  function App() {
6    const [myVar, setMyVar] = useState(0);
7
8  // Change my var has a parameter variable i
9    function changeMyVar(i) {
10     setMyVar(myVar + i);
11   }
12   return (
13     <>
14       <h1>This is the Parent</h1>
15       <h2>Current value of myVar is {myVar}: </h2>
16       <button onClick={() => changeMyVar(1)}>Parent+</button>
17       <hr />
18       <Child myVarFromParent={myVar} changeMyVarFromParent={changeMyVar} />
19     </>
20   );
21 }
22 // This is the Child component - using props
23 function Child(props) {
24   return (
25     <>
26       <h1>This is the Child</h1>
27       <h3>Value of myVar in Parent is currently {props.myVarFromParent}</h3>
28       <button onClick={() => props.changeMyVarFromParent(-1)}>Child-</button>
29     </>
30   );
31 }
```

IMPORTANT



The **`Child`** is invoked in the **`Parent`** at line 8.

The **parent** must then pass the value of state variable **`myVar`** and also a callback handler to the **`changeMyVar`** function.

This allows the child to communicate and change the value of the parent's state variable `myVar`

Key Concept – understanding this simple example is key to parent-child communication

```jsx
function App() {
  const [myVar, setMyVar] = useState(0);

// Change my var has a parameter variable i
  function changeMyVar(i) {
    setMyVar(myVar + i);
  }
  return (
    <>
      <h1>This is the Parent</h1>
      <h2>Current value of myVar is {myVar}: </h2>
      <button onClick={() => changeMyVar(1)}>Parent+</button>
      <hr />
      <Child myVarFromParent={myVar} changeMyVarFromParent={changeMyVar} />
    </>
  );
}
// This is the Child component - using props
function Child(props) {
  return (
    <>
      <h1>This is the Child</h1>
      <h3>Value of myVar in Parent is currently {props.myVarFromParent}</h3>
      <button onClick={() => props.changeMyVarFromParent(-1)}>Child-</button>
    </>
  );
}
```

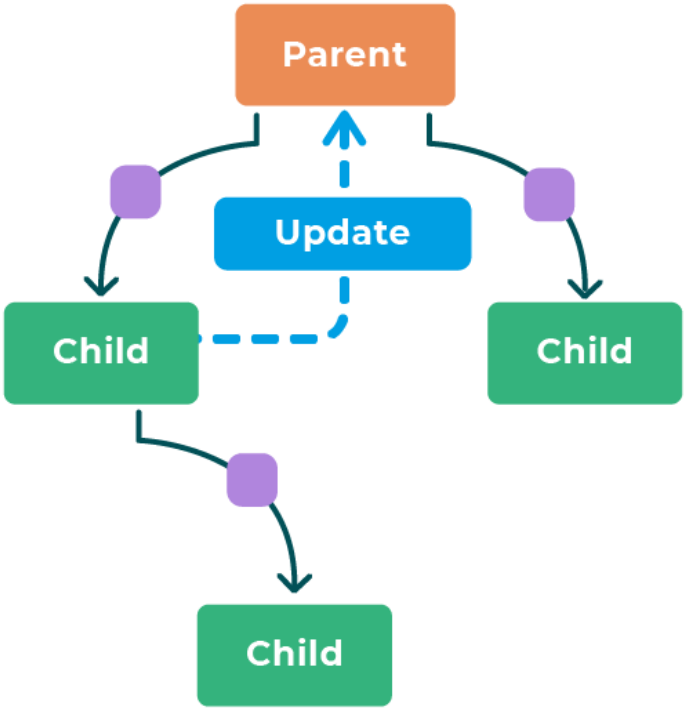
IMPORTANT

Ensure that you can understand how the `myVar` is being changed by both the PARENT and the CHILD.

The two components COMMUNICATE by using props. This is line 8 – this is where the communication happens.

Demo of

Lecture5-Parent-Child-myVar.js



Lecture 5 – key concepts

COMPONENTS

- Components allow us to divide our code into small packages or units.
- **Each component can represent a different part of the User Interface**
- Components can communicate with each other via **props** and handler functions (as we have seen)
- Components are just React Javascript functions that return JSX (like App0)

Lecture 5 – key concepts `props`

- Without **props**, a parent component could not communicate variables (data) or handles to functions to child components.
- **Using props allows programmers to seamlessly create communication between components – in a very clear and robust manner**.
- When using **props** you need to look out for the names of the variables or handles to functions you are using. Errors in names can be hard to track.

Lecture 5 – key concepts

`useState`

- The only safe way to maintain state in a React application is by using the `useState` hook.
- `useState` does two very important jobs – it **sets the initial value of a state variable** AND it **creates the set method** to allow this variable to be updated.

```
3  function App() {
4    const [myVar, setMyVar] = useState(0);
5    const [alpha, setAlpha] = useState(-10);
6
7    function changeMyVars() {
8      setMyVar(myVar + 2);
9      setAlpha(alpha - 10);
10   }
11   return (
12     <>
13       <h1>myVar={myVar}, alpha={alpha}</h1>
14       <button onClick={() => changeMyVars()}>Update</button>
15     </>
16   );
17 }
```

Lecture 5 – key concepts

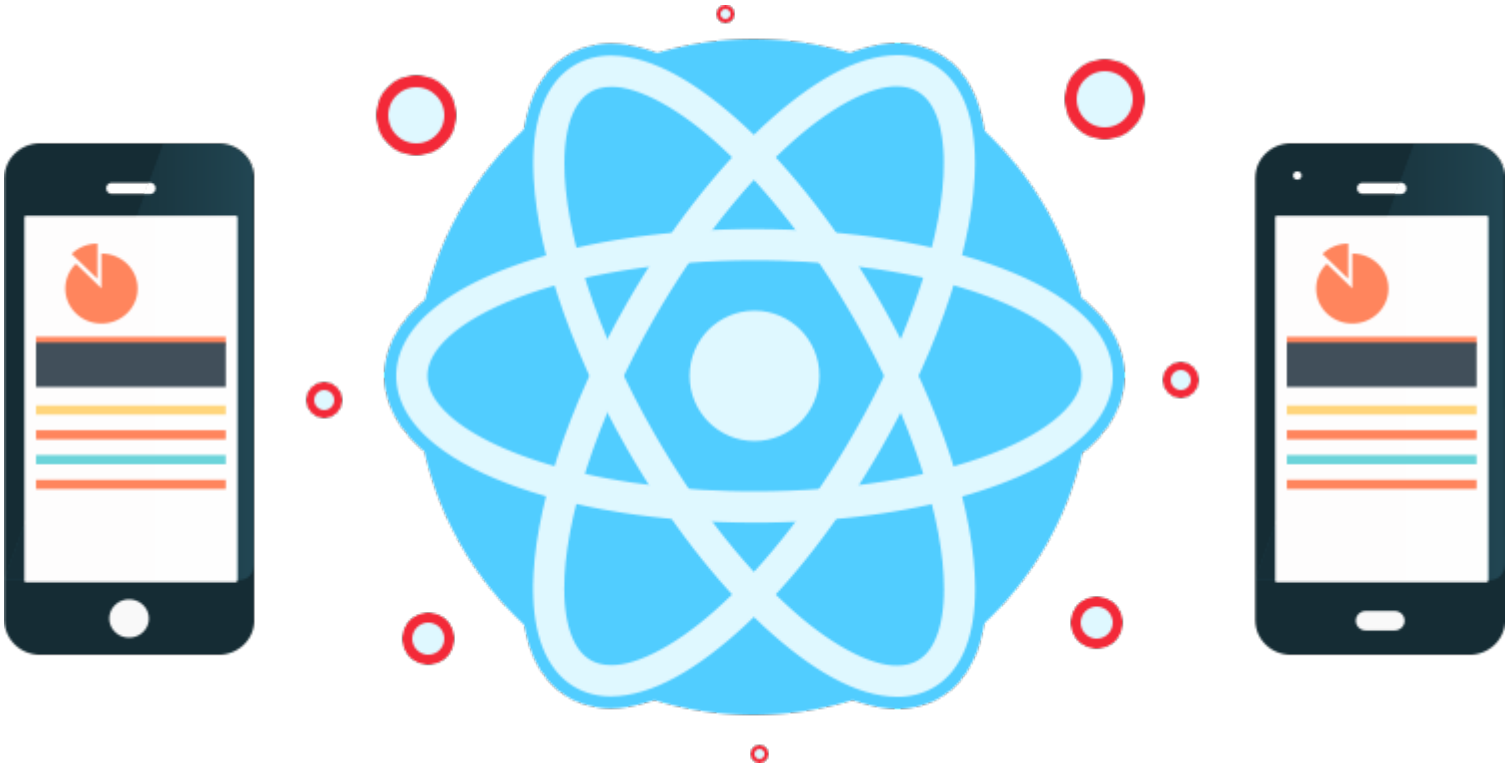
Parent-Child communications

- It may not be very obvious now, but **as you begin to write more complex React Javascript code Parent-Child communication will become crucial**.
- Parent-Child communication is simply two things:
  - (1) **separating our code into smaller unit**s where these units may represent different User Interface elements.
  - (2) **a means of moving variables or handles to functions between these components**.

Final words, lecture 5

- Very important content here – central to developing any advanced React/Javascript application
- **Parent-Child communication is ABSOLUTELY VITAL in large applications** – we simply have to divide our code and logic into components
- **State management is also critically important aspect of application development** – by working with the `useState` hook we ensure that data in state is maintained correctly by our application.
- The use of props faciliates communication – **ANYTHING can be passed via `props` within Javascript**

CS385 Mobile Application Development (Lecture 5)



Peter Mooney