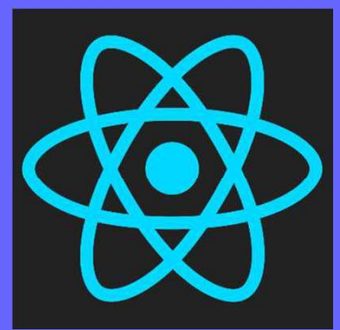
7

Props in React

Objectives

After completing this lesson, you should be able to do the following:

- Default Props
- PropsTypes
- Passing Down Props Using Spread Operator



Introduction to Props

- Components are reusable so you can create a component that returns any JSX you want to and include it in any part of your application.
- For example let's say we need to reuse the **Greet Component** all you have to do is include the grid tag as many times as you want if duplicate it.

```
JS Greet.js U X
my-app > src > components > JS Greet.js > ...
1  import React from 'react';
2
3  // function Greet() {
4  //     return <h1>Hello React !</h1>
5  // }
6
7  export const Greet = () => <h1> Hello React !</h1>
8
9  //export default Greet
```

```
JS App.js M X
my-app > src > JS App.js > App > render
1  import React, { Component } from 'react';
2  import logo from './logo.svg';
3  import './App.css';
4  import { Greet } from './components/Greet';
5  import Welcome from './components/Welcome';
6  import Hello from './components/Hello';
7
8  class App extends Component {
9    render() {
10     return (
11       <div className="App">
12
13         <Greet />
14         <Greet />
15         <Greet />
16
17         {/* <Welcome /> */}
18
19         {/* <Hello /> */}
20       </div>
```

Hello React !

Hello React !

Hello React !

Props Intro

- It would be great is if we could pass in the name of the person we wanted to greet that way reusing the same component we could greet three different people instead of greeting React three times.
- That is where **Props** comes into picture.
- **Props** short for properties is the optional input that your component can accept it also allows the component to be dynamic.

props are used to pass data and methods from a parent component to a child component.

- **Interesting things about props**
- 1. They are immutable.
- 2. They allow us to create reusable components.

Illustration

- **Props** to pass a name from **App Component** to the **Greet Component** and render that name in the browser.
- To specify props for a component we specify them as attributes to specify a name property we simply add the name attribute to the attribute we assign the value

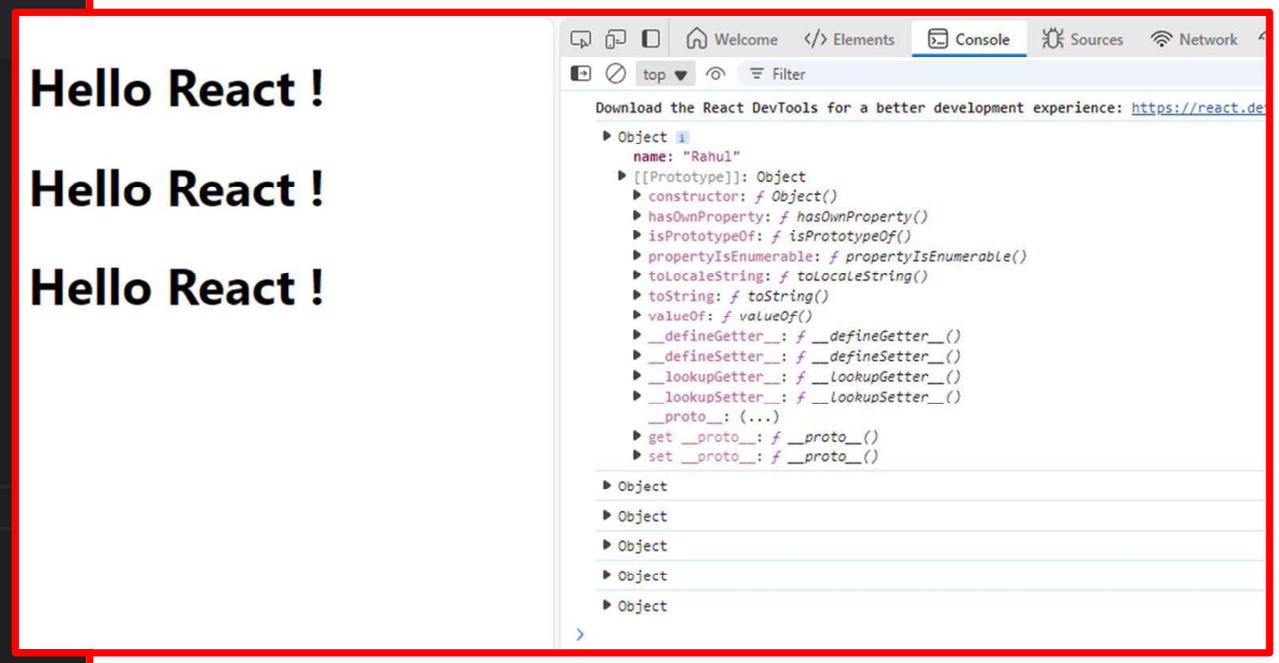
```
8   class App extends Component {
9     render() {
10      return (
11        <div className="App">
12
13          <Greet name="Rahul"/>
14          <Greet name="Steven"/>
15          <Greet name="Imran"/>
16
17          {/* <Welcome /> */}
18
19          {/* <Hello /> */}
20        </div>
```

- First Step add a Parameter to the Functional Component
  - We Name with any name but Conventional is **props.**
- Second Step is to use this Parameter in the Function Body.

JS App.js M JS Greet.js U

my-app > src > components > JS Greet.js > Greet

```
1   import React from 'react';
2
3   // function Greet() {
4   //     return <h1>Hello React !</h1>
5   // }
6
7   export const Greet = (props) => {
8       console.log(props);
9       return <h1> Hello React !</h1>
10  }
11
12  //export default Greet
```



- We need to ask React to Evaluate JSX Expression
- { } : is JSX Feature, which is Really Helpful and Used a Lot in React Application

my-app > src > components > JS Greet.js > Greet

```
1   import React from 'react';
2
3   // function Greet() {
4   //     return <h1>Hello React !</h1>
5   // }
6
7   export const Greet = (props) => {
8       console.log(props);
9       return <h1> Hello { props.name } Welcome to React !</h1>
10  }
11
12  //export default Greet
```

Hello Rahul Welcome to React !

Hello Steven Welcome to React !

Hello Imran Welcome to React !

Advantage of Props

- The reusability of components makes much more sense
- Now that we understand props each component can have an HTML template and we can pass in the data we want the Component

Lets Add as many Props we Need

```
class App extends Component {
  render() {
    return (
      <div className="App">

        <Greet name="Rahul" heroName ="Bat Man"/>
        <Greet name="Steven" heroName = "Super Man"/>
        <Greet name="Imran" herorName = "Spider Man"/>
```

```
6
7  export const Greet = (props) => {
8      console.log(props);
9      return <h1> Hello { props.name } A.K.A { props.heroName } Welcome to React !</h1>
10 }
```

Hello Rahul A.K.A Bat Man Welcome to React !

Hello Steven A.K.A Super Man Welcome to React !

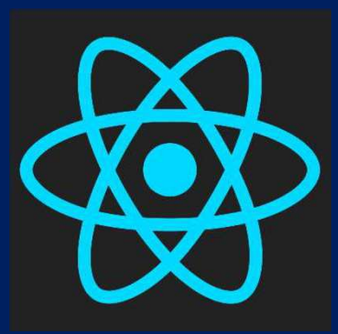
Hello Imran A.K.A Welcome to React !

Children Props

```
class App extends Component {
  render() {
    return (
      <div className="App">

        <Greet name="Rahul" heroName ="Bat Man">
          <p> This is Children Props</p>
        </Greet>
        <Greet name="Steven" heroName = "Super Man">
          <button>Action</button>
        </Greet>
```

```
export const Greet = (props) => {
    console.log(props);
    return (
    <div>
        <h1> Hello { props.name } A.K.A { props.heroName } Welcome to React !</h1>
        { props.children }
    </div>
    )
}
```



Props in Class Component

- Unlike the functional component where we specify the props parameter in a class component the properties are available through **this.props** which is reserved in Class Components

App.js M | Greet.js U | Welcome.js U

my-app > src > App.js > App > render

```
9   render() {
10    return (
11      <div className="App">
12
13        <Greet name="Rahul" heroName ="Bat Man">
14          <p> This is Children Props</p>
15        </Greet>
16        <Greet name="Steven" heroName = "Super Man">
17          <button>Action</button>
18        </Greet>
19        <Greet name="Imran" herorName = "Spider Man"/>
20
21        <Welcome name="Rahul" heroName ="Bat Man"/>
22        <Welcome name="Steven" heroName = "Super Man"/>
23        <Welcome name="Imran" herorName = "Spider Man" />
24
25        {/* <Hello /> */}
26      </div>
27    );
28  }
```

App.js M | Greet.js U | Welcome.js U

my-app > src > components > Welcome.js > Welcome > render

```
1  import React, { Component } from 'react';
2
3  class Welcome extends Component {
4      render() {
5          return <h1> Hello { this.props.name } A.K.A { this.props.heroName } Welcome to React !</h1>
6      }
7  }
8
9  export default Welcome
```

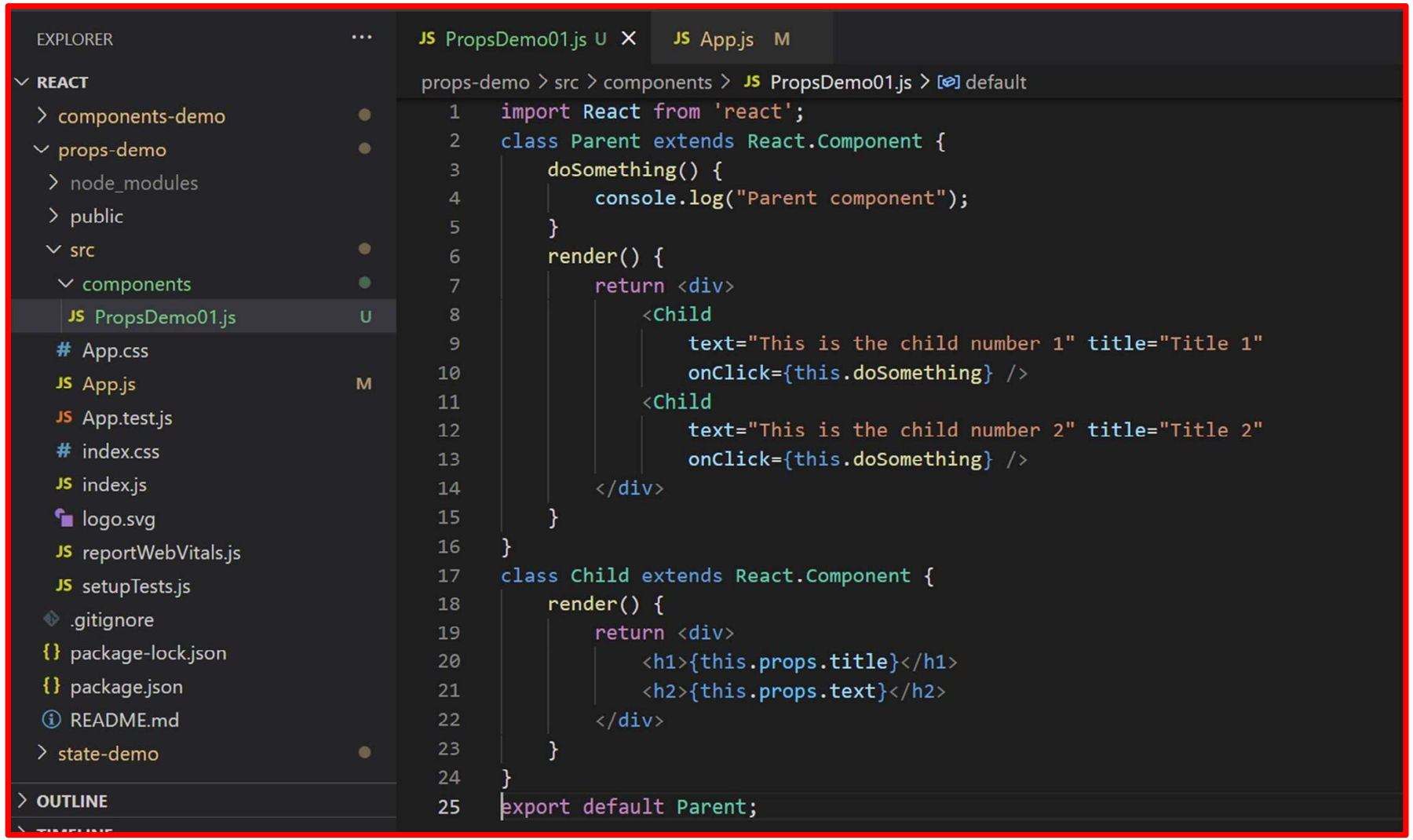
- When specifying the component you can include additional attributes **React** is going to bundle all such attributes into an object which we by convention call as props in the component definition.
- We can access any attribute we want to and render dynamic content now there is one strict rule though when it comes to props
- That is **props** are **immutable** or in simpler words their **Value Cannot Be Changed**

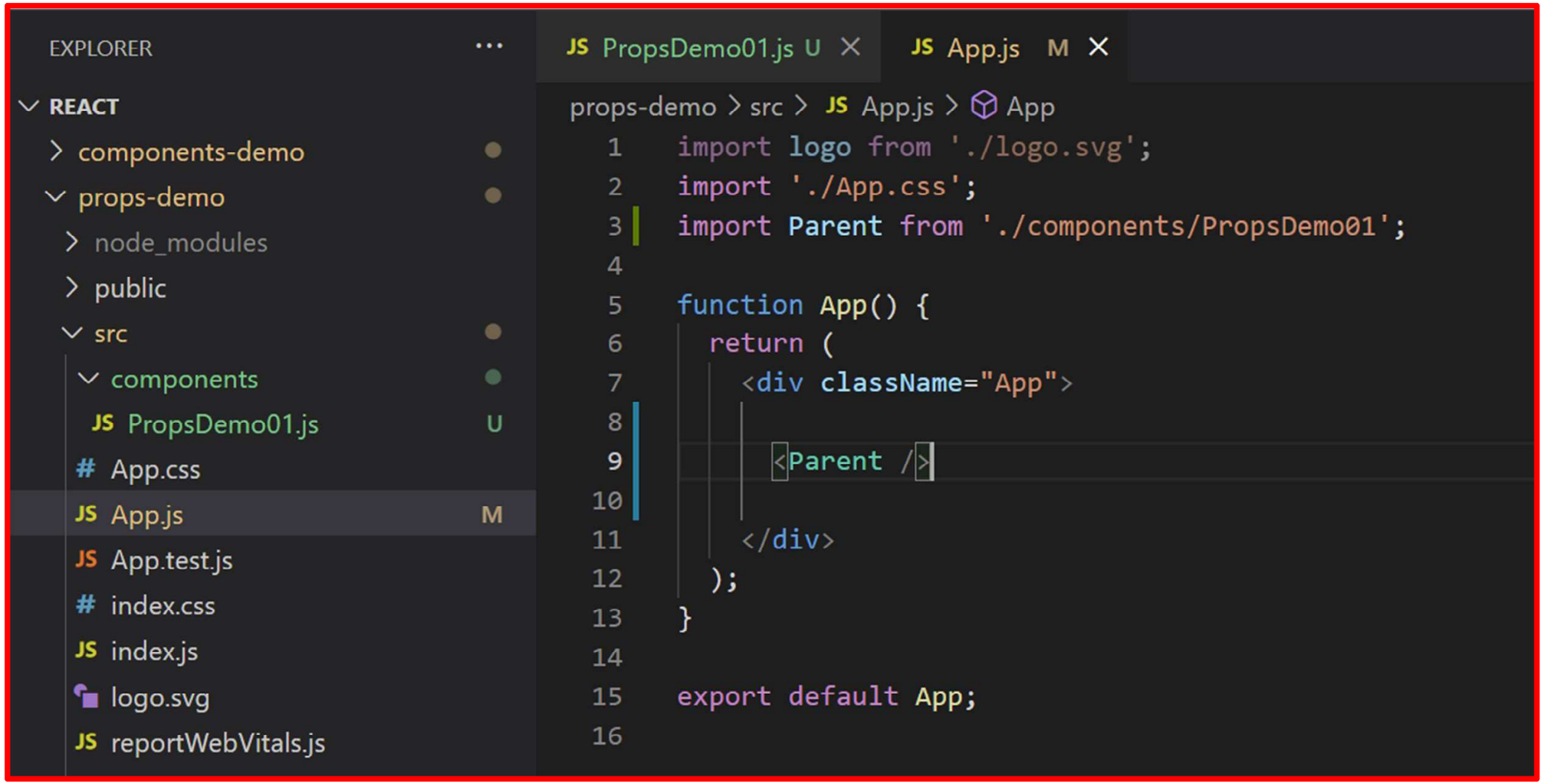
```jsx
export const Greet = (props) => {
    console.log(props);
    props.name = "Santhosh"
    return (
    <div>
        <h1> Hello { props.name } 
        { props.children }
    </div>
    )
}
```

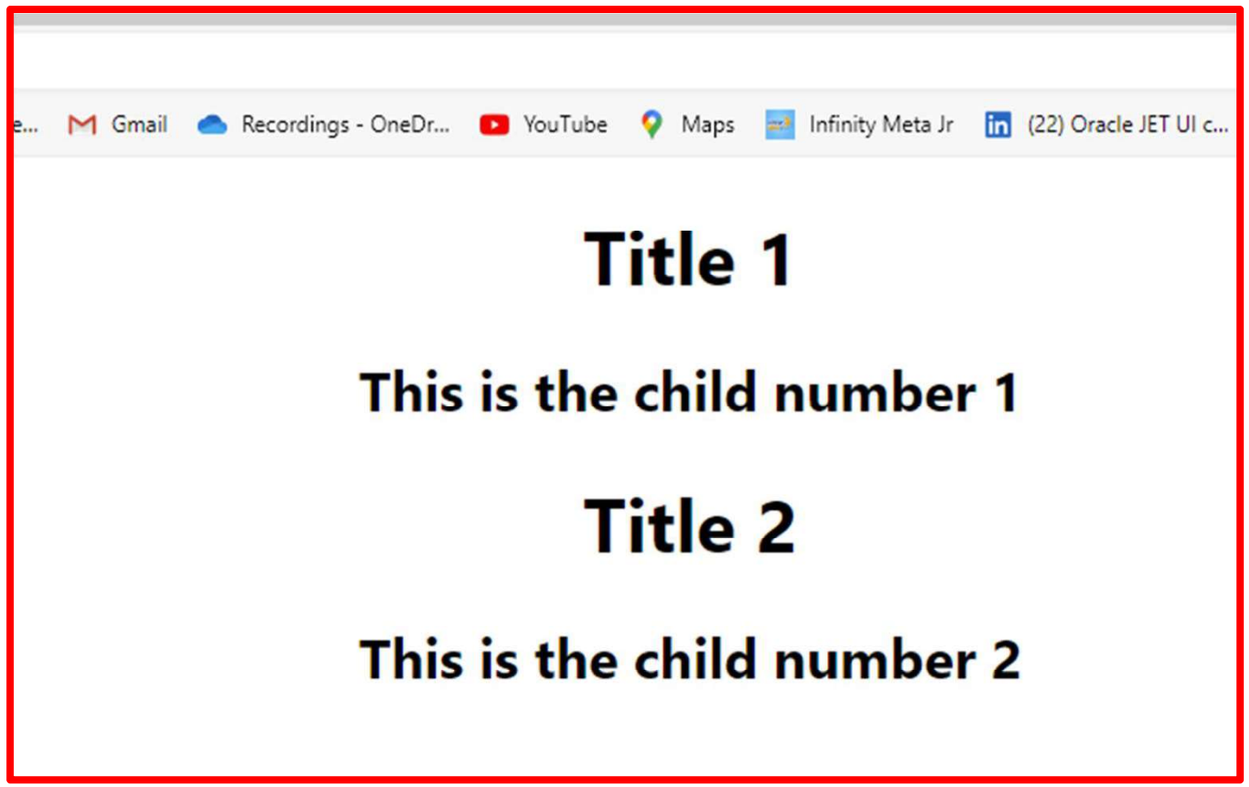
Uncaught runtime errors:

ERROR

```
Cannot assign to read only property 'name' of object '#<Object>'
TypeError: Cannot assign to read only property 'name' of object '#<Object>'
    at Greet (http://localhost:3000/main.902a7758a8b6daed19d0.hot-update.js:29:14)
    at react-stack-bottom-frame (http://localhost:3000/static/js/bundle.js:15854:18)
    at renderWithHooks (http://localhost:3000/static/js/bundle.js:6064:20)
    at updateFunctionComponent (http://localhost:3000/static/js/bundle.js:7757:17)
    at beginWork (http://localhost:3000/static/js/bundle.js:8343:16)
    at runWithFiberInDEV (http://localhost:3000/static/js/bundle.js:3835:68)
    at performUnitOfWork (http://localhost:3000/static/js/bundle.js:10416:93)
    at workLoopSync (http://localhost:3000/static/js/bundle.js:10309:38)
    at renderRootSync (http://localhost:3000/static/js/bundle.js:10293:7)
    at performWorkOnRoot (http://localhost:3000/static/js/bundle.js:10057:42)
```


EXPLORER
REACT
components-demo
props-demo
node_modules
public
src
components
PropsDemo01.js U
App.css
App.js M
App.test.js
index.css
index.js
logo.svg
reportWebVitals.js
setupTests.js
.gitignore
package-lock.json
package.json
README.md
state-demo
OUTLINE
PropsDemo01.js U
App.js M
props-demo > src > components > PropsDemo01.js > default
1 import React from 'react';
2 class Parent extends React.Component {
3 doSomething() {
4 console.log("Parent component");
5 }
6 render() {
7 return
8 <Child
9 text="This is the child number 1" title="Title 1"
10 onClick={this.doSomething} />
11 <Child
12 text="This is the child number 2" title="Title 2"
13 onClick={this.doSomething} />
14
15 }
16 }
17 class Child extends React.Component {
18 render() {
19 return
20 <h1>{this.props.title}</h1>
21 <h2>{this.props.text}</h2>
22
23 }
24 }
25 export default Parent;


EXPLORER
REACT
components-demo
props-demo
node_modules
public
src
components
PropsDemo01.js U
App.css
App.js M
App.test.js
index.css
index.js
logo.svg
reportWebVitals.js
PropsDemo01.js U
App.js M
props-demo > src > App.js > App
1 import logo from './logo.svg';
2 import './App.css';
3 import Parent from './components/PropsDemo01';
4
5 function App() {
6 return (
7
8
9 <Parent />
10
11
12);
13 }
14
15 export default App;
16


e...
Gmail
Recordings - OneDr...
YouTube
Maps
Infinity Meta Jr
(22) Oracle JET UI c...
Title 1
This is the child number 1
Title 2
This is the child number 2

Default props

- defaultProps allows you to set default, or fallback, values for your component props. defaultProps are useful when you call components from different views with fixed props, but in some views you need to pass different value.

Syntax

ES5

```
var MyClass = React.createClass({
  getDefaultProps: function() {
    return {
      randomObject: {},
      ...
    };
  }
}
```

ES6

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

MyClass.defaultProps = {
    randomObject: {},
    ...
}
```

ES7

```
class MyClass extends React.Component {
    static defaultProps = {
        randomObject: {},
        ...
    };
}
```

- The result of getDefaultProps() or defaultProps will be cached and used to ensure that **this**.props.randomObject will have a value if it was not specified by the parent component.

PropTypes

- propTypes allows you to specify what props your component needs and the type they should be. Your component
- will work without setting propTypes, but it is good practice to define these as it will make your component more
- readable, act as documentation to other developers who are reading your component, and during development,
- React will warn you if you you try to set a prop which is a different type to the definition you have set for it.
- Some primitive propTypes and commonly useable propTypes are -

- If you attach isRequired to any propType then that prop must be supplied while creating the instance of that component.
- If you don't provide the **required** propTypes then component instance can not be created.

ES6

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

MyClass.propTypes = {
    randomObject: React.PropTypes.object,
    callback: React.PropTypes.func.isRequired,
    ...
};
```

ES7

```
class MyClass extends React.Component {
     static propTypes = {
        randomObject: React.PropTypes.object,
        callback: React.PropTypes.func.isRequired,
        ...
    };
}
```

More complex props validation

- In the same way, PropTypes allows you to specify more complex validation

Validating an object

```
...
    randomObject: React.PropTypes.shape({
        id: React.PropTypes.number.isRequired,
        text: React.PropTypes.string,
    }).isRequired,
...
```

Validating on array of objects

```
...
    arrayOfObjects: React.PropTypes.arrayOf(React.PropTypes.shape({
        id: React.PropTypes.number.isRequired,
        text: React.PropTypes.string,
    })).isRequired,
...
```

Passing down props using spread operator

- Instead of

```
var component = <Component foo={this.props.x} bar={this.props.y} />;
```

- Where each property needs to be passed as a single prop value you could use the spread operator ... Supported for arrays in ES6 to pass down all your values.

```
var component = <Component {...props} />;
```

- Remember that the properties of the object that you pass in are copied onto the component's props.
- The order is important. Later attributes override previous ones.

```
var props = { foo: 'default' };
var component = <Component {...props} foo={'override'} />;
console.log(component.props.foo); // 'override'
```

Summary

In this lesson, you should have learned how to:

- Default Props
- PropsTypes
- Passing Down Props Using Spread Operator