WORKSHOP

TODO LIST

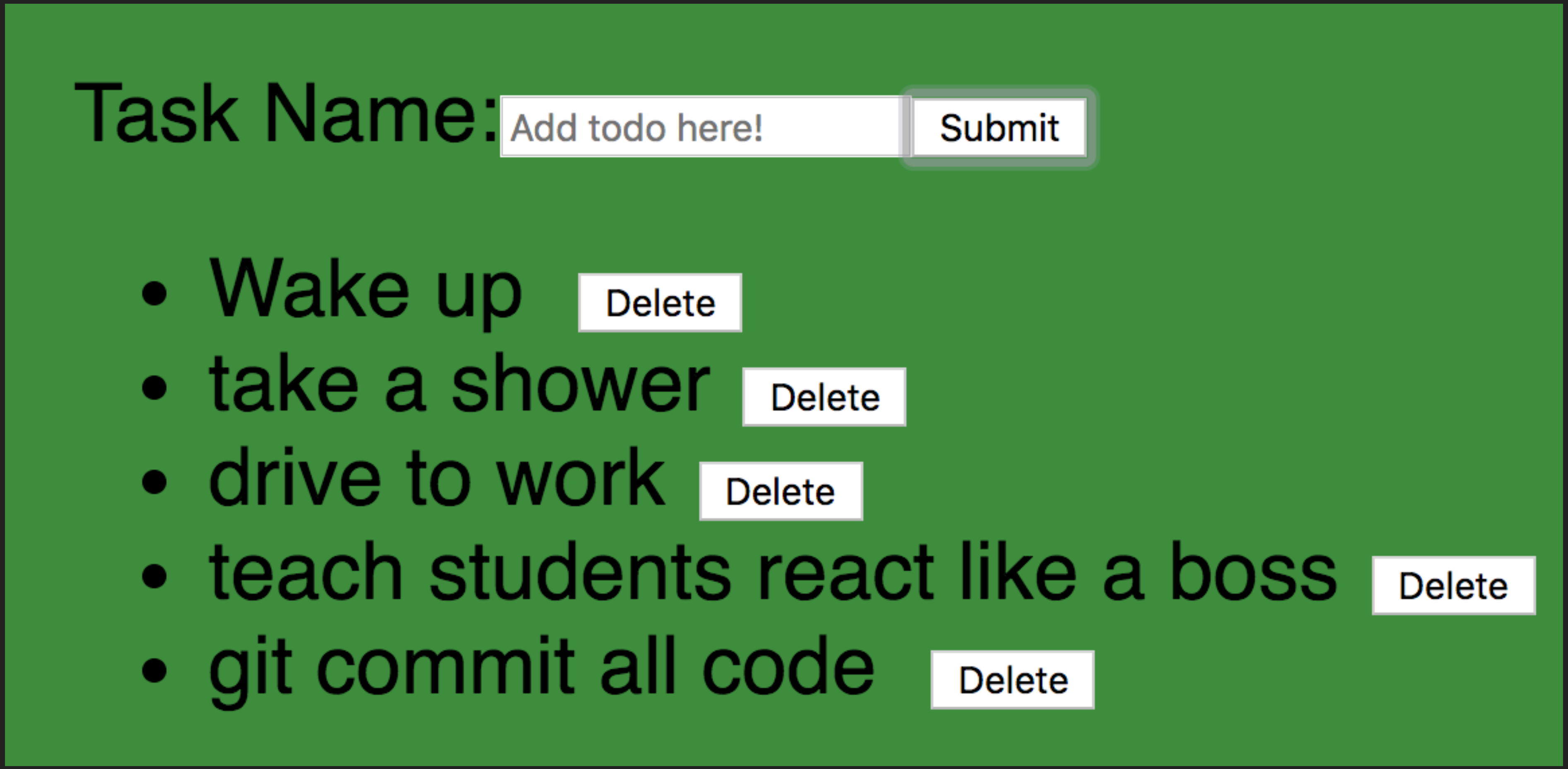
CREATE-REACT-APP

- Now that we have installed and used create-react-app. Lets create a project called Counter.
- Type - npx create-react-app todo in our console.
- Follow the suggested prompt of running the commands

  ```
  cd todo
  npm start
  ```
- Now we should see our server running and the react-app boilerplate site being displayed.

OUR GOAL

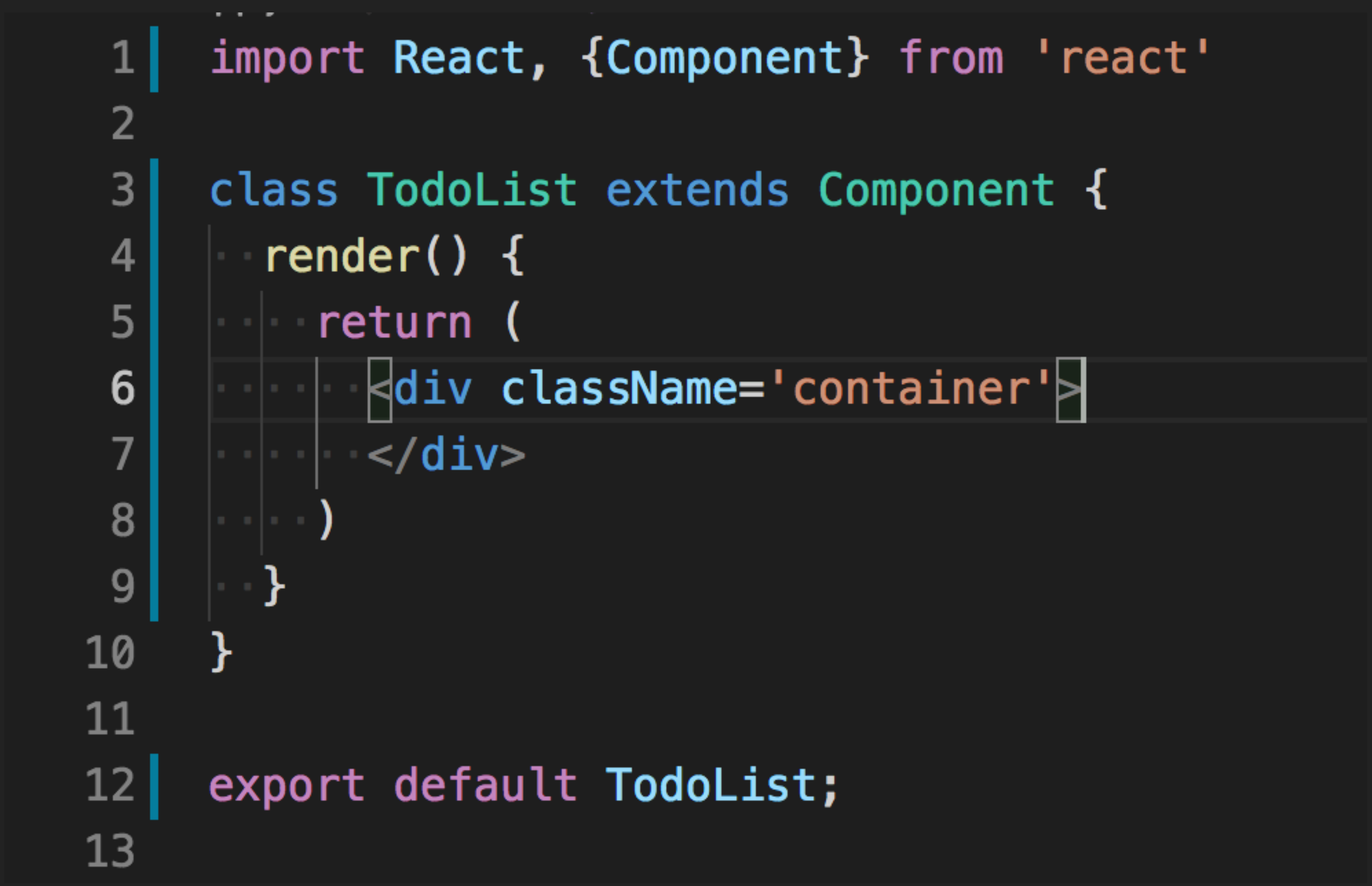
- This is what our todo application will look like if completed correctly



- Wake up Delete
- take a shower Delete
- drive to work Delete
- teach students react like a boss Delete
- git commit all code Delete

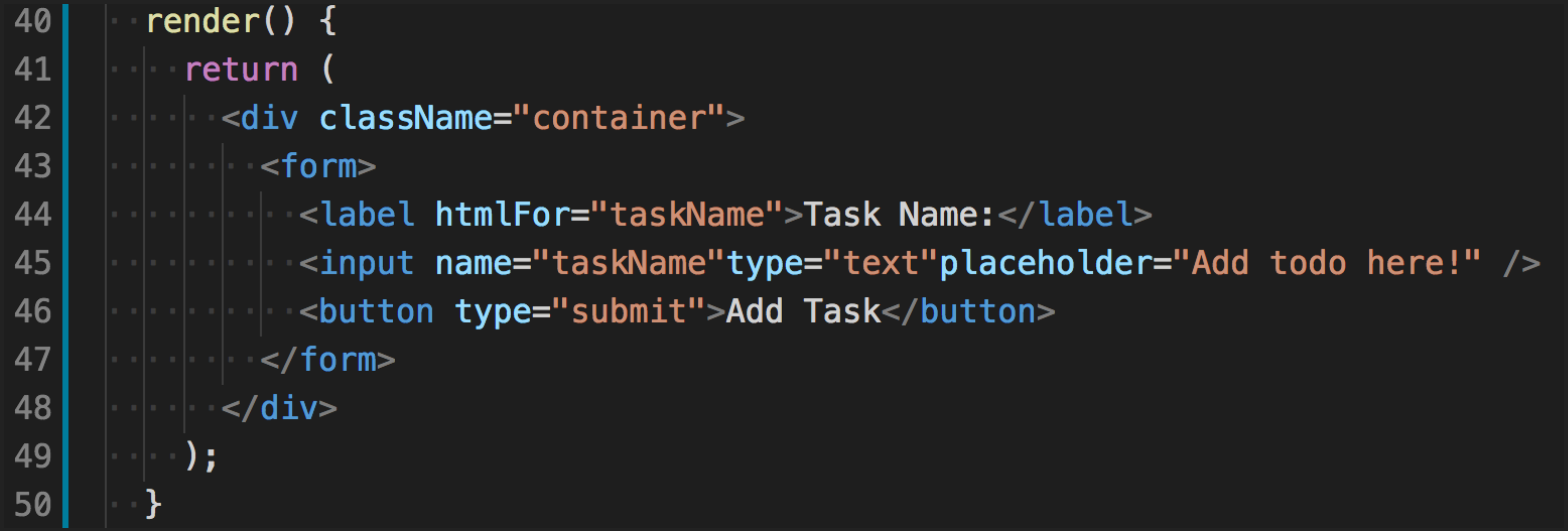
- First, think about what type of component that you'll be creating for this workshop.
- Secondly, think about the functionality of this program and what type of methods we will use to accomplish create, update, and delete todo.
- Third, think about how you are going to format this data and how we will display it correctly in JSX.

- Let's start off by deleting everything in App.js, and building the base of our application.



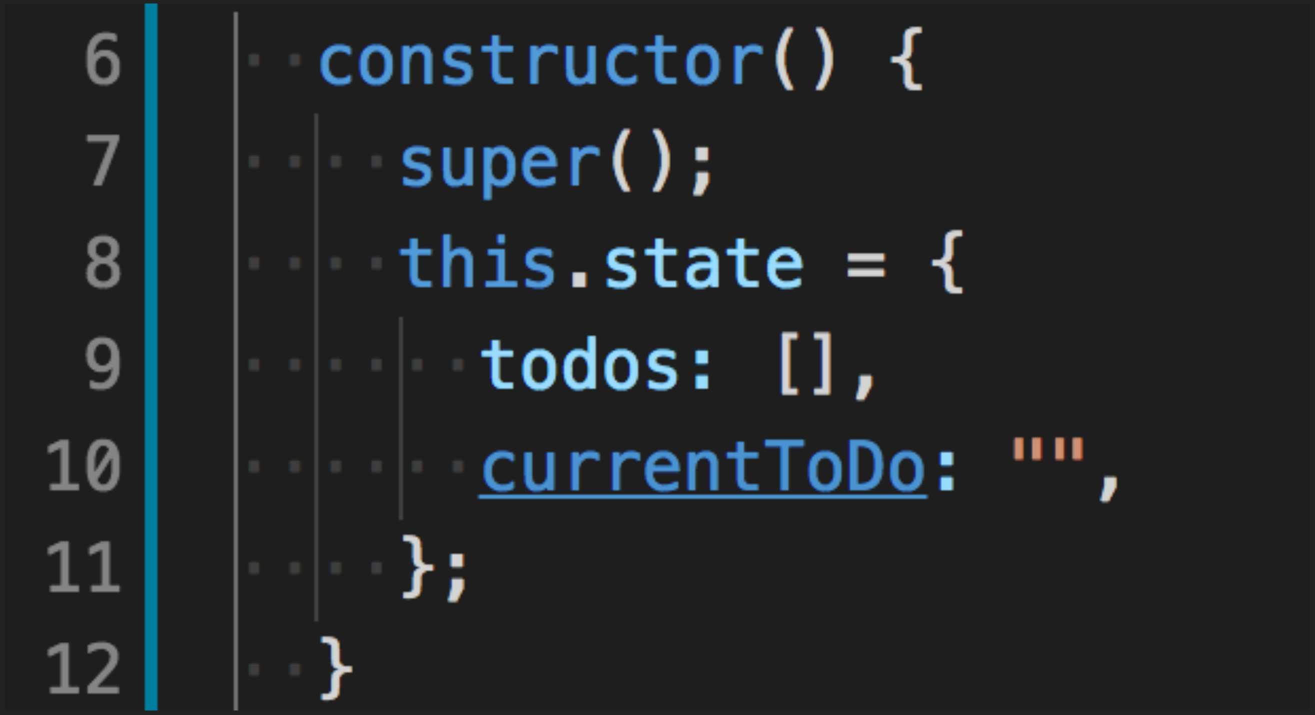
```
import React, {Component} from 'react'

class TodoList extends Component {
  render() {
    return (
      <div className='container'>
      </div>
    )
  }
}

export default TodoList;

```

- Now that we have our lovely app's styling set. We should start building out the UI. Lets first create our form.

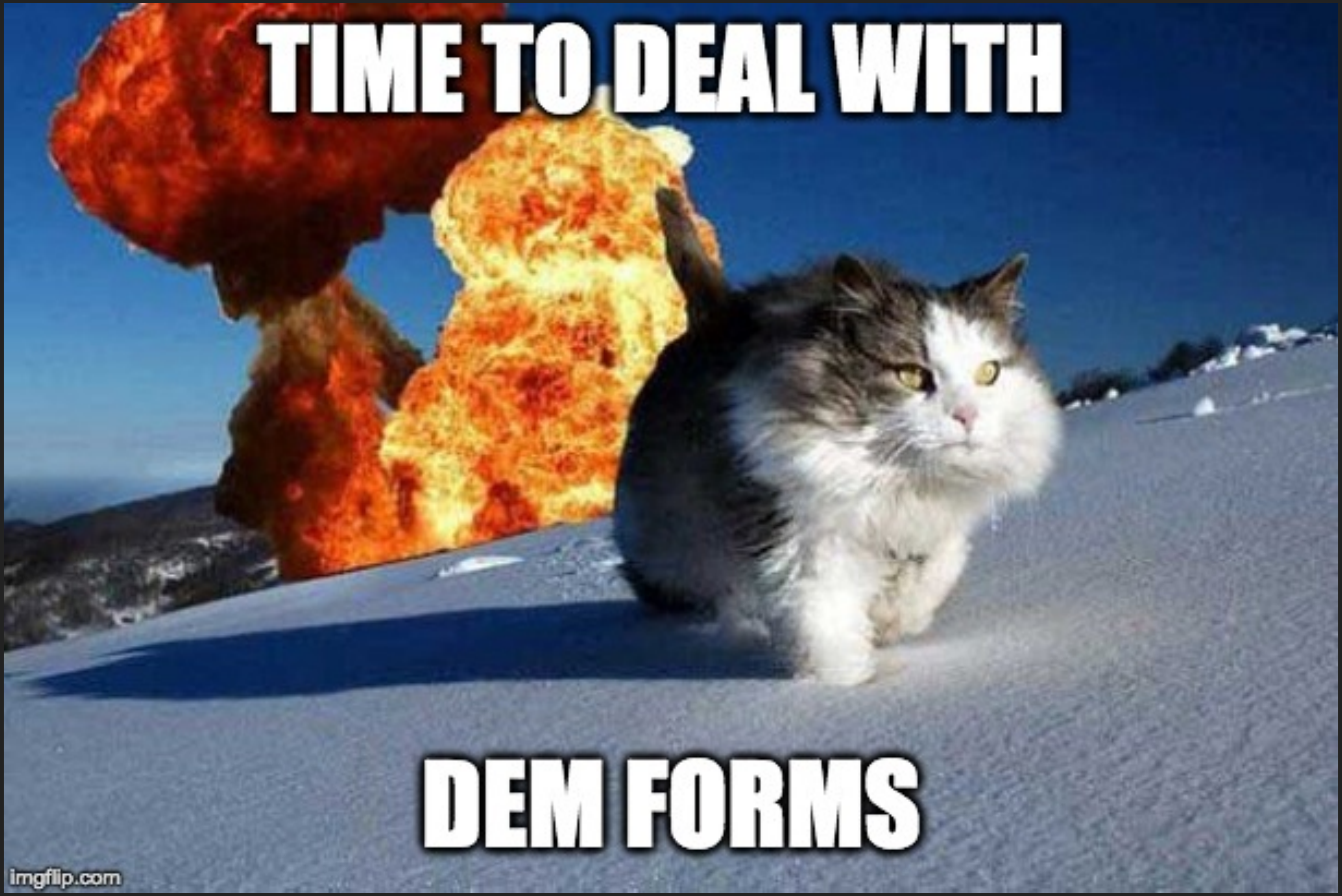


```
  render() {
    return (
      <div className="container">
        <form>
          <label htmlFor="taskName">Task Name:</label>
          <input name="taskName"type="text"placeholder="Add todo here!" />
          <button type="submit">Add Task</button>
        </form>
      </div>
    );
  }
```

- We know this application is going to use state, so let's set that up as well. First - set up state with a key of todos and a value of an empty array. This allows us to build out our todos list and map through it to list the individual items in the todos array. We also need to have a key of currentToDo that we want to read from the form and then add to our todo list. This key will have a datatype of string.

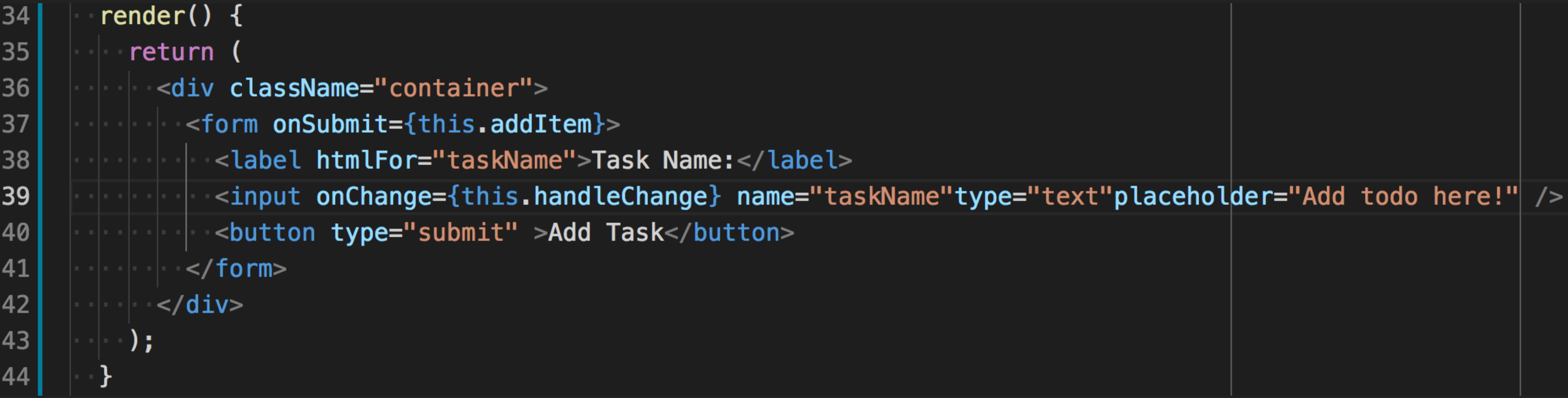


```
  constructor() {
    super();
    this.state = {
      todos: [],
      currentToDo: "",
    };
  }
```

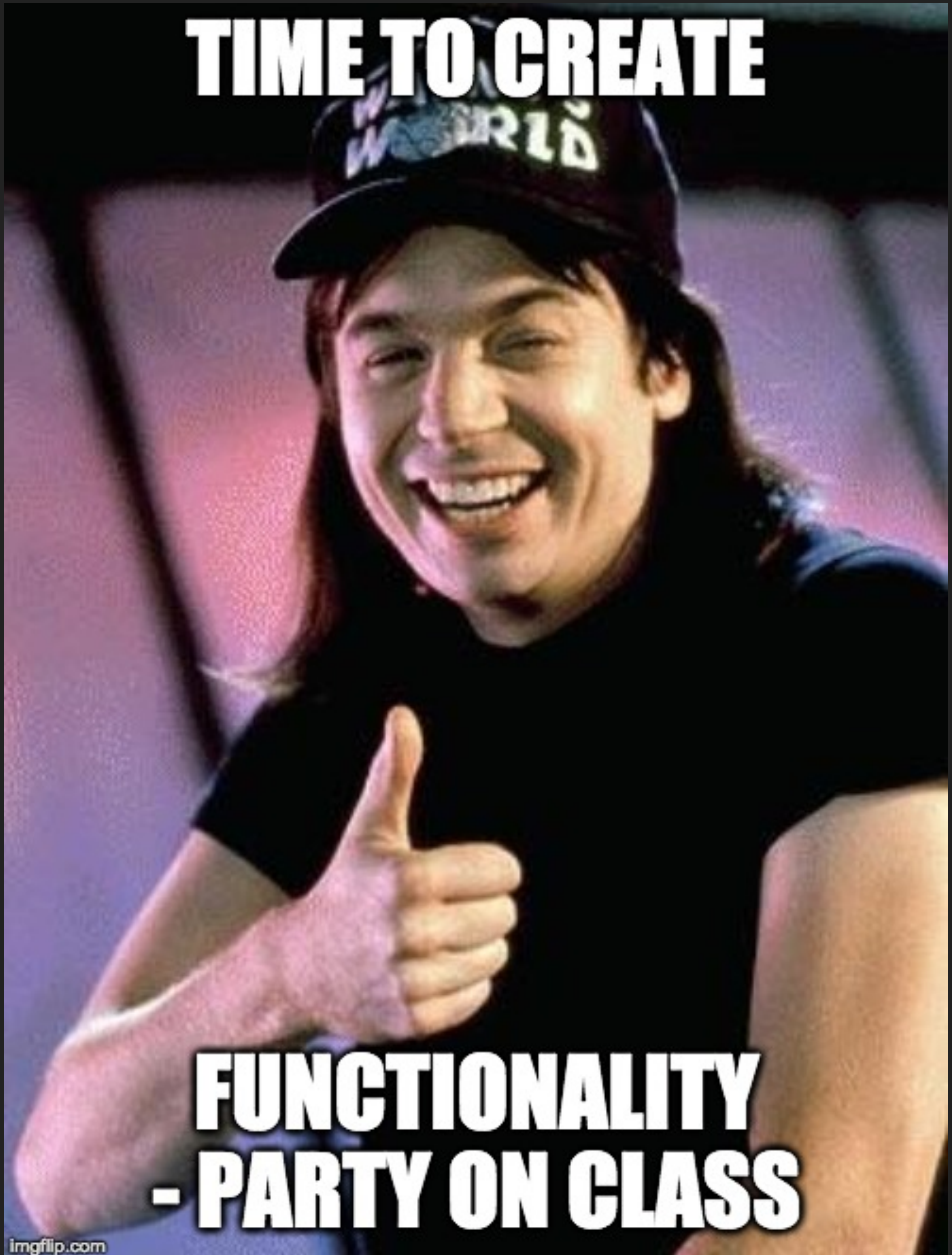

TIME TO DEAL WITH
DEM FORMS
imgflip.com

- When dealing with forms we need to set up functionality where the user adds data into our form, as well as when the user submits the form. We handle these events by setting up functionality that is executed when the onChange event listener as well as the onSubmit are invoked. onChange and onSubmit are event listeners that we can attach to the from and the input to trigger functionality. But it's up to us to create this functionality. This way our state can reflect when the users submits the new task and that the input isn't read only.

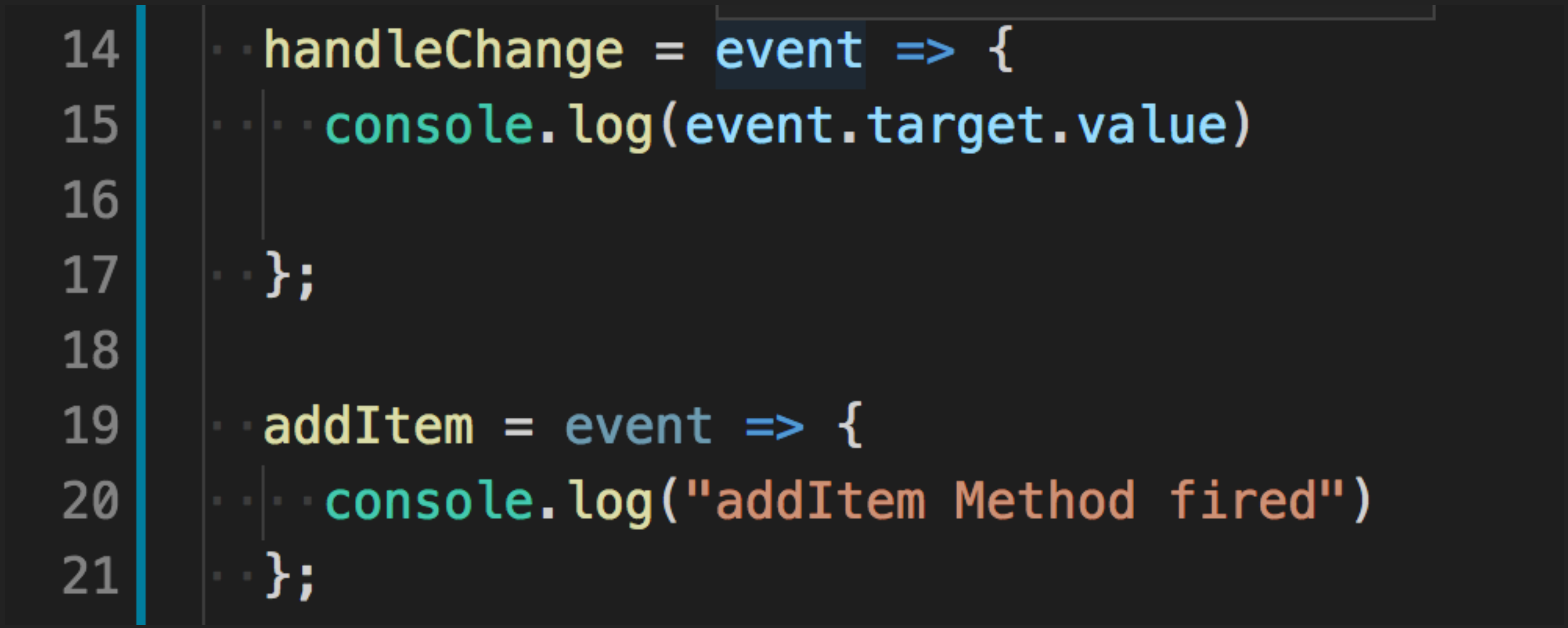
- Let's add these listeners to the form element as well as the button element.



```
  render() {
    return (
      <div className="container">
        <form onSubmit={this.addItem}>
          <label htmlFor="taskName">Task Name:</label>
          <input onChange={this.handleChange} name="taskName"type="text"placeholder="Add todo here!" />
          <button type="submit" >Add Task</button>
        </form>
      </div>
    );
  }
```

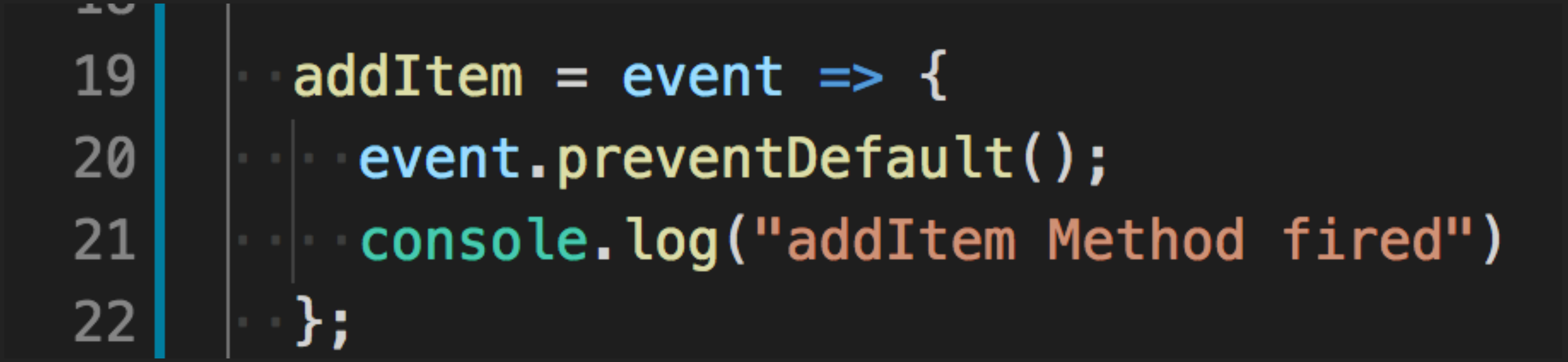

TIME TO CREATE
FUNCTIONALITY - PARTY ON CLASS
imgflip.com

- The two methods that I attached to our listeners were addItem and handleChange. Let's define these. Both these methods will have the argument of event. This event is either when the onSubmit or the onChange event listener is fired & event is what is passed/returned from that function.



```
  handleChange = event => {
    console.log(event.target.value)

  };

  addItem = event => {
    console.log("addItem Method fired")
  };
```

- Now if we try and see what is being console.log'd we see that the form is just resubmitting. This is where stop propagation comes into play. Reloading is the default behavior of the form submission. So we need to modify the addItem method.



```
  addItem = event => {
    event.preventDefault();
    console.log("addItem Method fired")
  };
```

TIME TO START
CODING!!!
imgflip.com

- Tasks:
  - Complete the handleChange and the addItem methods.
  - Then reflect your added todos in a list.
  - Then add a delete method that removes any todo in the list.
  - Then create a TasksList component that accepts props to build out the list instead of doing it in our main app.
  - Then create a ToDoForm component that accepts props to fire the correct methods when the corresponding event listeners are fired.

- If you've gotten this far you're doing awesome sauce!
- Complete an Update method and add a update button to each list item that will allow you to update the individual item.