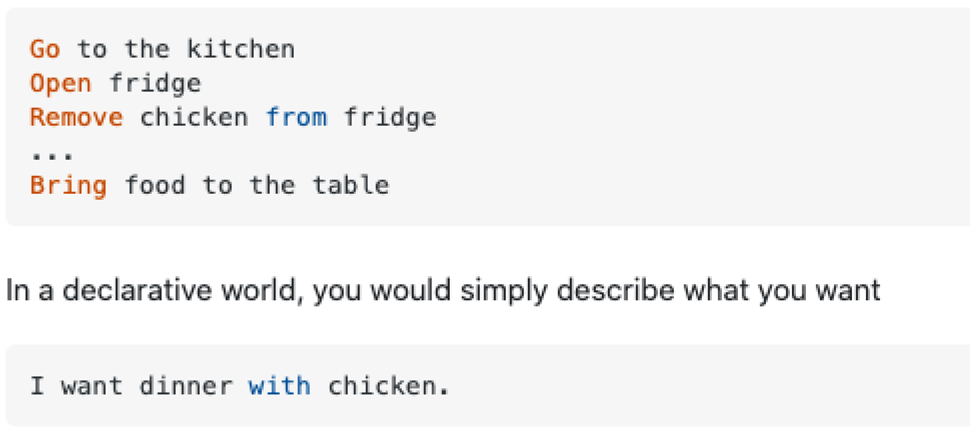
2. Imperative Approach vs Declarative Approach

- Let us start from an example

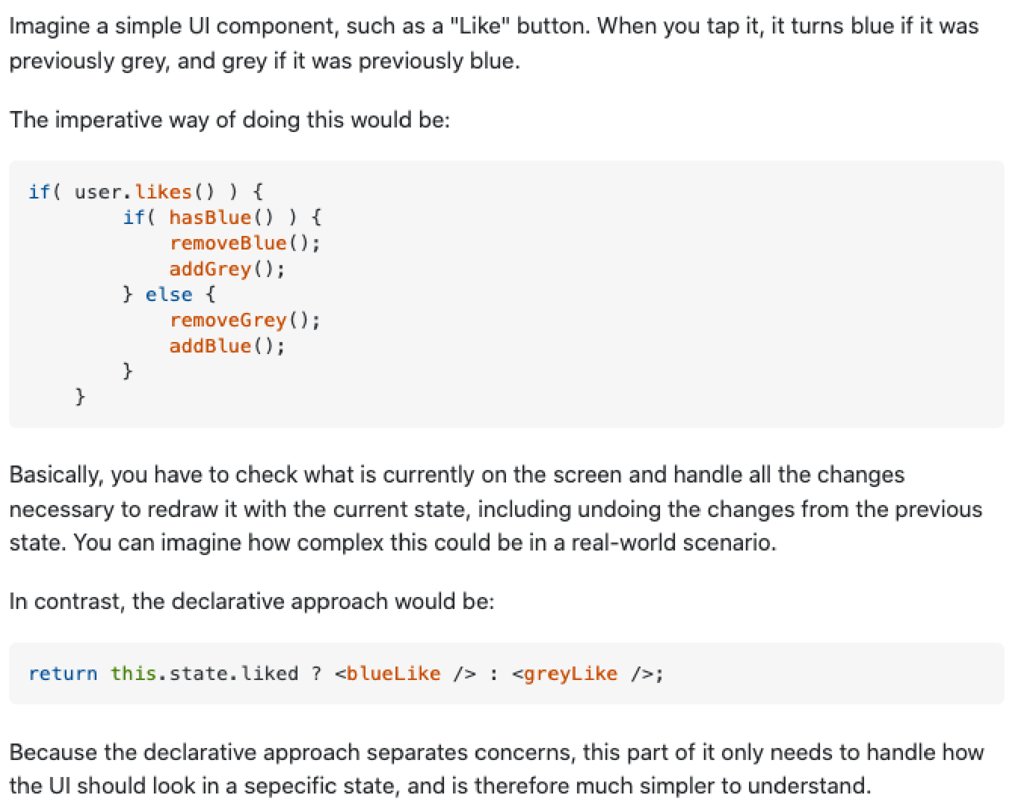


```
Go to the kitchen
Open fridge
Remove chicken from fridge
...
Bring food to the table
```

In a declarative world, you would simply describe what you want

```
I want dinner with chicken.
```

  -
  - The first one is the imperative approach whereas the second one is the declarative approach
- Let us see another example



Imagine a simple UI component, such as a "Like" button. When you tap it, it turns blue if it was previously grey, and grey if it was previously blue.

The imperative way of doing this would be:

```
if( user.likes() ) {
        if( hasBlue() ) {
            removeBlue();
            addGrey();
        } else {
            removeGrey();
            addBlue();
        }
    }
```

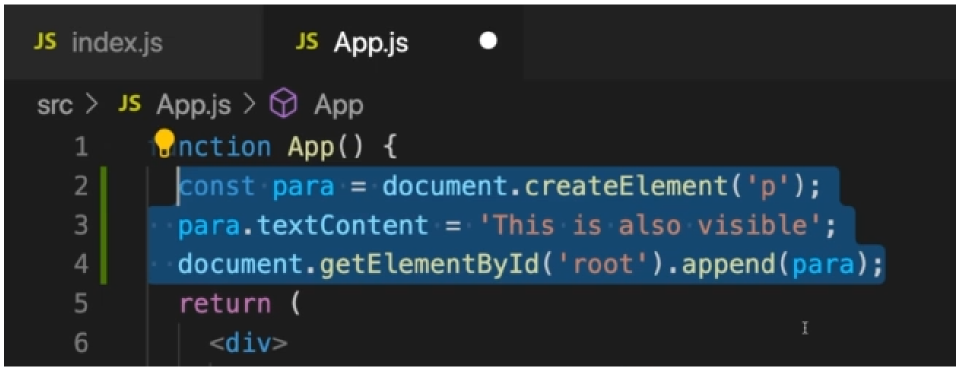
Basically, you have to check what is currently on the screen and handle all the changes necessary to redraw it with the current state, including undoing the changes from the previous state. You can imagine how complex this could be in a real-world scenario.

In contrast, the declarative approach would be:

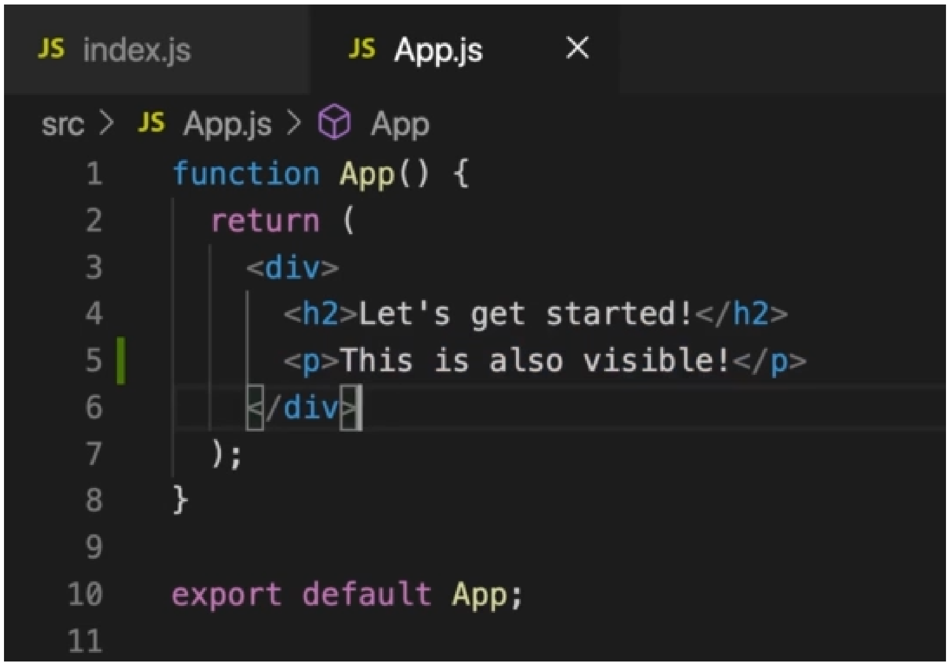
```
return this.state.liked ? <blueLike /> : <greyLike />;
```

Because the declarative approach separates concerns, this part of it only needs to handle how the UI should look in a sepecific state, and is therefore much simpler to understand.

  -
  - Here in the imperative approach, if user likes/dislikes, then we have to select the like buttons and check whether it has blue class in it, if yes then we need to remove that class and then add grey else we need to remove grey and add blue

        - But whereas in declarative approach it renders just based on the state of the component we don’t need to specify each and every step, this is the power of components an dits state
- Another example,
    - Say we need to add some para in our app component based on some event
    - The imperative approach is
        - 

```
src > App.js > App
nction App() {
  const para = document.createElement('p');
  para.textContent = 'This is also visible';
  document.getElementById('root').append(para);
  return (
    <div>
```

    - The declarative approach is
        - 

```
src > App.js > App
function App() {
  return (
    <div>
      <h2>Let's get started!</h2>
      <p>This is also visible!</p>
    </div>
  );
}

export default App;
```

            - Here the p tag is added in our component’s target state, here we didn’t specify any states yet but this is how it works.