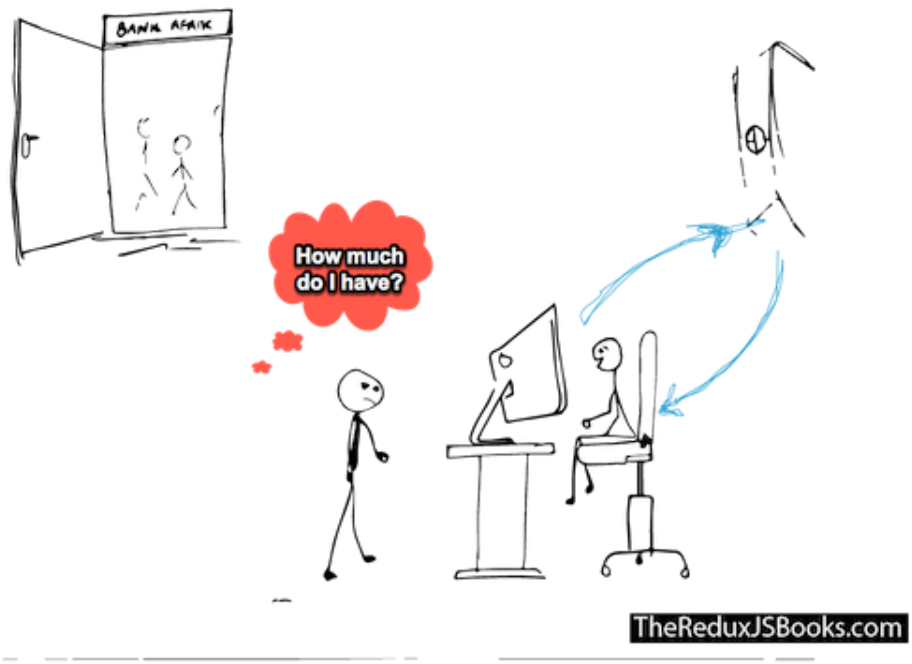
# The Second createStore Argument

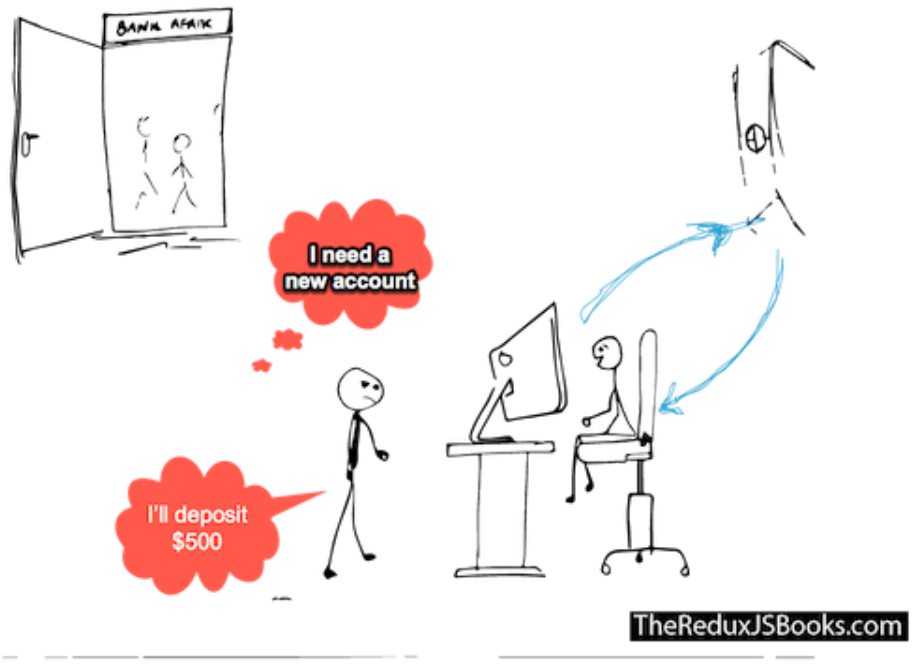
We will now learn:
a) how to pass the initialState argument into createStore().
b) how to manage our app's state completely through Redux

> When you visit the Cashier in the bank, if you asked them for your account balance, they'll look it up and tell it to you.




> But how?

> When you first created an account with your bank, you either did so with some amount of deposit or not.




Let's call this the initial deposit into your account.

Back to Redux.

In the same way, when you create a redux STORE (our own money keeping VAULT), there's the option of doing so with an initial deposit.

In Redux terms, this is called the `initialState` of the app.

Thinking in code, `initialState` is the second argument passed into the `createStore` function call.

```
const store = createStore(reducer, initialState);
```

Before making any monetary ACTION, if you requested your bank account balance, the INITIAL DEPOSIT will always be returned to you.

Afterwards, anytime you perform any monetary ACTION, this initial deposit will also be updated.

Now, the same goes for Redux

Now, the same goes for Redux.

The object passed in as `initialState` is like the initial deposit to the Vault. This initialState will always be returned as the STATE of the application unless you update the state by performing an ACTION.

We will now update the application to pass in an initial state:

```
const initialState = { tech: "React " };
const store = createStore(reducer, initialState);
```

Note how `initialState` is just an object, and it is exactly what we had as the default state in the React App before we began refactoring.

Now, here's all the code we have at this point - with the reducer also imported into `App.js`.

App.js:

```
import React, { Component } from "react";
import HelloWorld from "./HelloWorld";
import reducer from "./reducers";
import { createStore } from "redux";

const initialState = { tech: "React " };
const store = createStore(reducer, initialState);

class App extends Component {
  render() {
    return <HelloWorld tech={this.state.tech}/>
  }
}

export default App;
```

reducers/index.js

```
export default state  => {
  return state
}
```

If you're coding along and try to run the app now, you'll get an error. Why?

Have a look at the tech prop passed into `< HelloWorld />`. It still reads, this.state.tech.

There's no longer a state object attached to `< App />`, so that will be undefined. Let's fix that.

The solution is quite simple. Since the STORE now manages the state of our application, this means the application STATE object must be retrieved from the STORE. But how?

Whenever you create a store with `createStore()`, the created store has **3 exposed methods**.

One of these is `getState()`.

At any point in time, calling the getState method on the created STORE will return the current state of your application.

In our case, `store.getState()` will return the object

```
{tech: "React"}
```

since this is the INITIAL STATE we passed into the `createStore()` method when we created the STORE.

You see how all this come together now?

Hence the tech prop will be passed into `< HelloWorld />` as shown below:

App.js

```
import React, { Component } from "react";
import HelloWorld from "./HelloWorld";
import { createStore } from "redux";

const initialState = { tech: "React " };
const store = createStore(reducer, initialState);

class App extends Component {
  render() {
    return <HelloWorld tech={store.getState().tech}/>
  }
}
```


reducers/index.js:

```
export default state => {
  return state
}
```

Here's the result of refactoring the Hello World app completely:

```
export default state => {
  return state
}
```

And that is it! You just learned the Redux basics and successfully refactored a simple React app to use Redux.

The React application now has its state managed by Redux. Whatever needs to be gotten from the STATE object will be grabbed from the STORE as shown earlier.

Hopefully, you understood this whole refactoring process.

For a quicker overview, have a look at this Github diff.

With the Hello World project, we have taken a good look at some of essential Redux concepts. Even though it's such a tiny project, it provides a decent foundation to build upon!