The Reducer

We'll delve into the purpose of the reducer. It takes in two arguments: the STATE of the app and an ACTION. The reducer interprets the action and performs changes to the app accordingly.

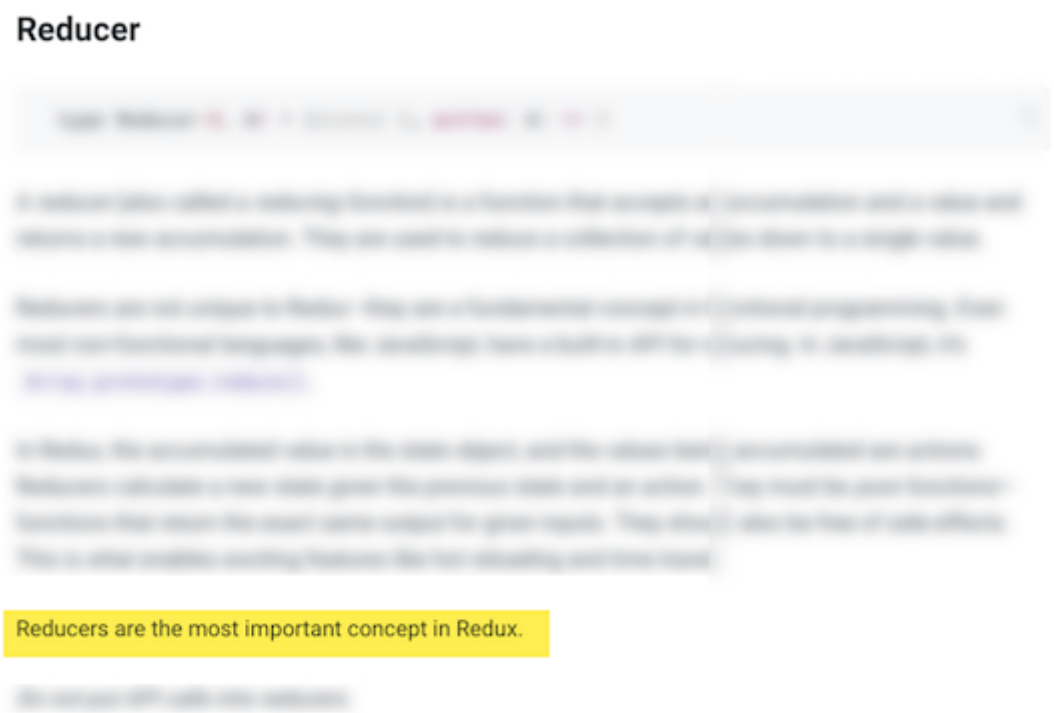
We will go into greater details pretty soon, but I'll keep this short for now.

When you hear the word, reducer, what comes to your mind? reduce?

Yeah, that's what I thought. It sounds like reduce.

Well, according to the Redux official docs:

> **Reducers** are the most important concept in Redux.



Our Cashier is a pretty important person, huh?

So, what's the deal with the Reducer. What does it do?

In more technical terms, a reducer is also called a reducing function. You may not have noticed, but you probably already use a reducer - if you're

not have noticed, but you probably already use a reducer - if you're conversant with the `Array.reduce()` method.

Here's a quick refresher.

Consider the code below.

It is a popular way to get the sum of values in a Javascript Array.

```
let arr = [1,2,3,4,5]
let sum = arr.reduce((x,y) => x + y)

console.log(sum) //15
```

Under the hood, the function passed into `arr.reduce()` is called a reducer.

In this example, the reducer takes in two values, an accumulator and a `currentValue` where x is the accumulator and y, the currentValue.

In likewise manner, the Redux Reducer is just a function. A function that takes in two parameters. The first being the STATE of the app, and the other the ACTION.

But where does the STATE and ACTION passed into the REDUCER come from?

When I was learning Redux, I asked myself this question a few times.

Firstly, take a look at the `Array.reduce()` example again:

```
let arr = [1,2,3,4,5]

let sum = arr.reduce((x,y) => x + y)

console.log(sum) //15
```

The `Array.reduce()` method is responsible for passing in the needed arguments, x and y into the function argument, the reducer. So, the arguments didn't come out of thin air.

The same may be said for Redux.

The Redux reducer is also passed into a certain method. Guess what is it?

Here you go!

createStore(reducer)

The `createStore` factory function. There's a little more involved in the process as you'll soon see.

Like `Array.reduce()`, `createStore()` is responsible for passing the arguments into the reducer.

If you aren't scared of technical stuff, here's the stripped down version of the implementation of createStore within the Redux source code.

```
function createStore(reducer) {

        var state;
        var listeners = []

        function getState() {
            return state
        }

        function subscribe(listener) {
            listeners.push(listener)
            return unsubscribe() {
                var index = listeners.indexOf(listener)
                listeners.splice(index, 1)
            }
        }

        function dispatch(action) {
            state = reducer(state, action)
            listeners.forEach(listener => listener())
        }

        dispatch({})
        return { dispatch, subscribe, getState }

}
```

Don't beat yourself up if you don't get the code above. What I really want to point out is within the dispatch function.

Notice how the reducer is called with state and action.

With all that being said, the most minimal code for creating a Redux store is this:

```
import { createStore } from "redux";
```

```
const store = createStore(reducer); //this has been updated to
                                    //include the created reducer.
```