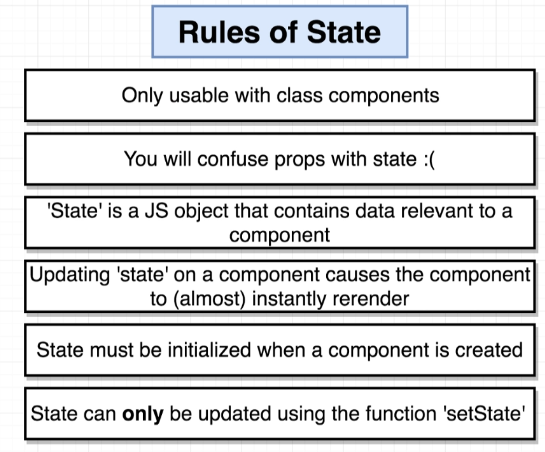
STATE in React Components –

In the last section, we refactored our functional App component to a class-based one – but doing this alone will not solve the problem of get current position () taking some amount of time.

This problem is going to get solved by Reacts State system.



1. Technically, can be used with functional components using ‘hooks’ system.
2. Props and states are two different things but very easy to mix up.
3. State is a JS object that contains data related to a ‘singular’ component -> in our case, we care about is user’s current latitude.
4. Updating state leads to re-rendering that’ll solve the issue we had earlier.
5. State is going to contain some information relevant to component initially and is updated using setState().

**Initialising State through Constructor** –

According to the rules, State must be initialized when a component is created. After that, we make use of our state in the render method.

At some point in the future, we’ll also make sure to update our state with setState() [after we successfully retrieve our location].

React will throw an error if we do not define a render method.

There’s another function that we can define inside of our class-based components – ‘constructor’.

* Specially named function particular to JavaScript language, not specific to ‘React’.
* It is the first function that is going to be called any time an instance of the class is created => any time we create a new instance of the App component and show it on the screen, this constructor function is automatically and instantly called before anything else.
* This makes it a good location for us to initialise our state – when our component is first created.

Now, when we define the constructor method, it is automatically called with the **props** object – same props that we saw previously in the functional component.

NOTE :

Inside the constructor, before doing anything else, we must write “**super(props)**” –

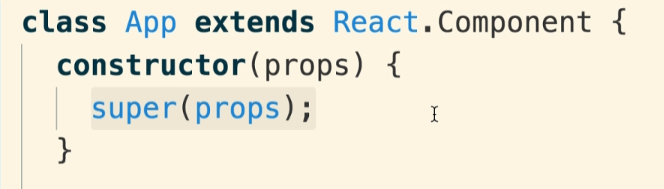
Our App component is extending (i.e. borrowing functionality) from React Component base class. This base class has a constructor function of its own that goes through some amount of setup OR has some code inside to set up a React component for us.

When we define a constructor function inside our App class, we’re essentially overriding (replacing) the constructor function that is inside of React.Component class.

However, we want to make sure that all the setup code inside of Component class’ constructor function still gets called. So, to make sure that the parent’s constructor gets called, we call super with props.

Super is a reference to the parent’s constructor function.

(Must be done every time we create a constructor inside class-based component)



 {} – state object. It is going to eventually contain some different pieces of data, some different properties that are important and relevant to the component.

Here, the most relevant piece of data that we have is the latitude – so we might want to initialize our state object to include a property named ‘latitude’ and give it a default value of null.



**Updating State Properties** –

After creating the state object, we then assigned it to this.state property. Now that it is assigned, we can freely reference the state object and properties inside of it from any function inside our App component.



The general idea is to make use of this state property inside JSX or any other function inside our component.

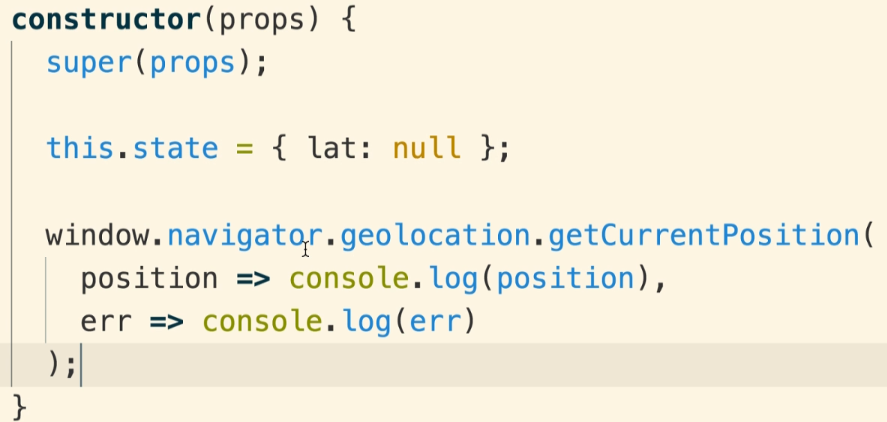
Now, after we succeed in getting the position, we want to update the ‘lat’ property inside our state object.

We also know that anytime we update our state object, our component will almost instantly re-render itself.

NOTE :

The ‘render’ method inside our Component is going to be called somewhat frequently – we must know where to put calls that take some amount of time – get current position function.

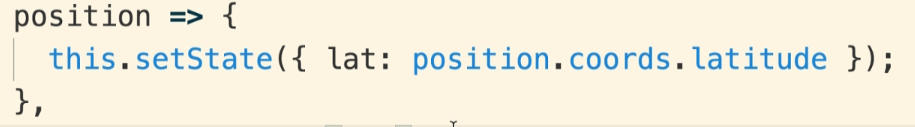
We never ever want to start to initialize some work or some request from a call in render method AS the render method is going to be called many number of times.

So, we move the current position generator code into the constructor function. 

This makes sure that the instant our App component shows on the screen i.e. the instant we create an instance of this component, it’ll attempt to start to get the user’s location. Later on, when we call the render method again to update our component, we’re not going to double fetch the current location.

Now, the first argument to get current position is a call-back function – so we try to take latitude out of the position object and use it to update the state object – done using setState() function.

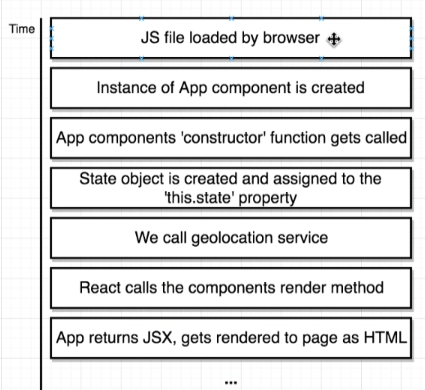
**this.setState({object that has the updated state we want});**



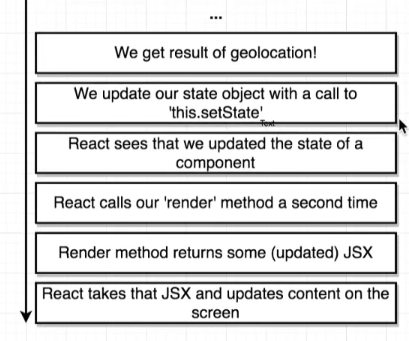
To update the state, we always use this.setState() method. We never ever use direct assignment to our state object like –> **~~this.state.lat = position.coords.latitude.~~**(wrong)

The only exception to this rule is when we **initialize** our state inside of our constructor function.

**App Lifecycle** –



* Loaded up the index.html file on the Browser, and that requested our JS file with all the React code inside of it [JS file loaded by Browser and executed]
* When JS file was executed, we created a new instance of the App component.
* The first function that is called inside the component is the constructor function.
* After calling super function, we initialize our state object [a new JS object having all different properties inside of it that we want to use inside this component as state]. This object is assigned to this.state property.
* Now we call the geolocation service.
* React then takes the component and calls it render method.
* The component returns some amount of JSX and that gets rendered to the page as HTML.
* Then we wait for some time …
* Now we get the result of geolocation request.
* We execute the call-back wherein we take the position object and pull the latitude out of it and assign it using setState
* The state gets updated – React sees this and re-renders the component.
* Re-render => React calls our render method again.
* Render method returns some JSX (updated one) and is again shown on screen as HTML.



Our App component was rendered two times – once, when our application first booted up, and a second time when we updated our state.

There was a very small gap in time where our component was rendered onto the screen without any number beside latitude.

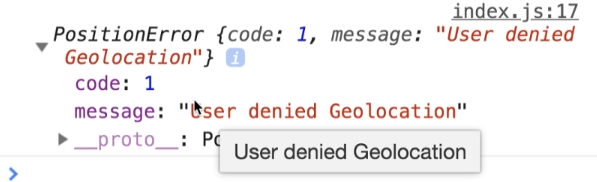
**Handling Error Call-back** –

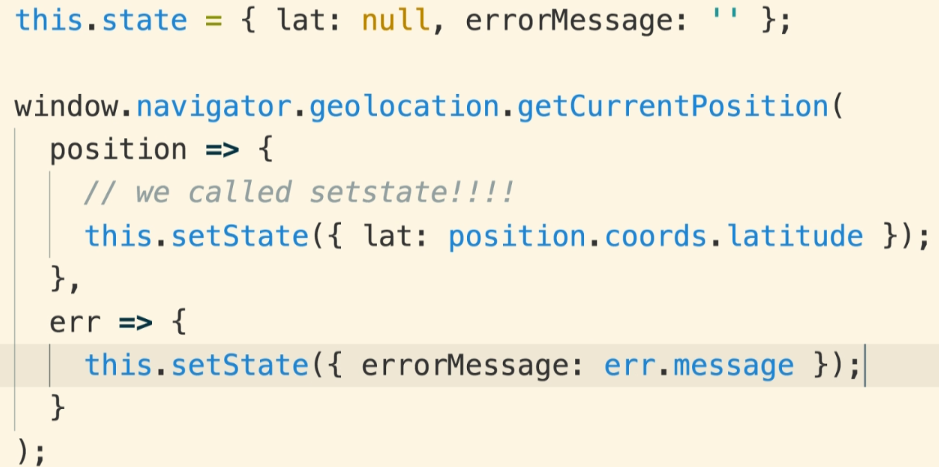
If anything goes wrong with get current position request, we enter the error call-back function. We probably would want to re-render our component and directly tell the user something wrong has happened and we can’t tell user’s current location.

How to re-render our component?? -> **State**. (Anytime we want to re-render, update state)

We add a new state property called ‘errorMessage’ having default value as empty string ‘’.

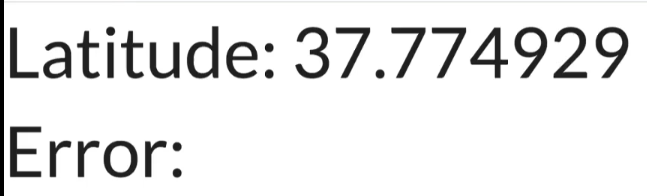
If anything goes wrong while attempting to get the position, we’ll update this errorMessage property. This will cause the component to re-render and we can attempt to show the error message inside of our render method.





NOTE :

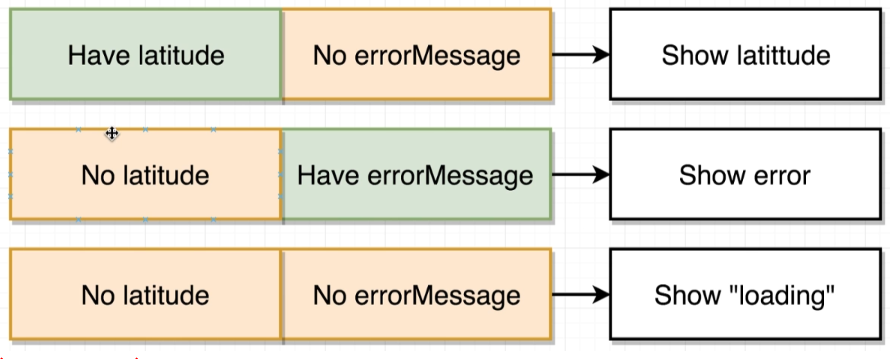
Whenever we update our state, we’re not required to update every property in it. Updating one property leaves the other one’s untouched.



**Conditionally Rendering Content** –

It’s not good for us to put the word error even when there is none – unsettling for our users.

Combinations of state we can have –



So in order to conditionally return different JSX, all we have to do is put a simple collection of if statements into the render method.

