QUESTION 1

class MyComponent extends React.Component {

constructor(props) {

// set the default internal state

this.state = {

clicks: 0

};

}

componentDidMount() {

this.refs.myComponentDiv.addEventListener('click', this.clickHandler);

}

componentWillUnmount() {

this.refs.myComponentDiv.removeEventListener('click', this.clickHandler);

}

clickHandler() {

this.setState({

clicks: this.clicks + 1

});

}

render() {

let children = this.props.children;

return (

<div className="my-component" ref="myComponentDiv">

<h2>My Component ({this.state.clicks} clicks})</h2>

<h3>{this.props.headerText}</h3>

{children}

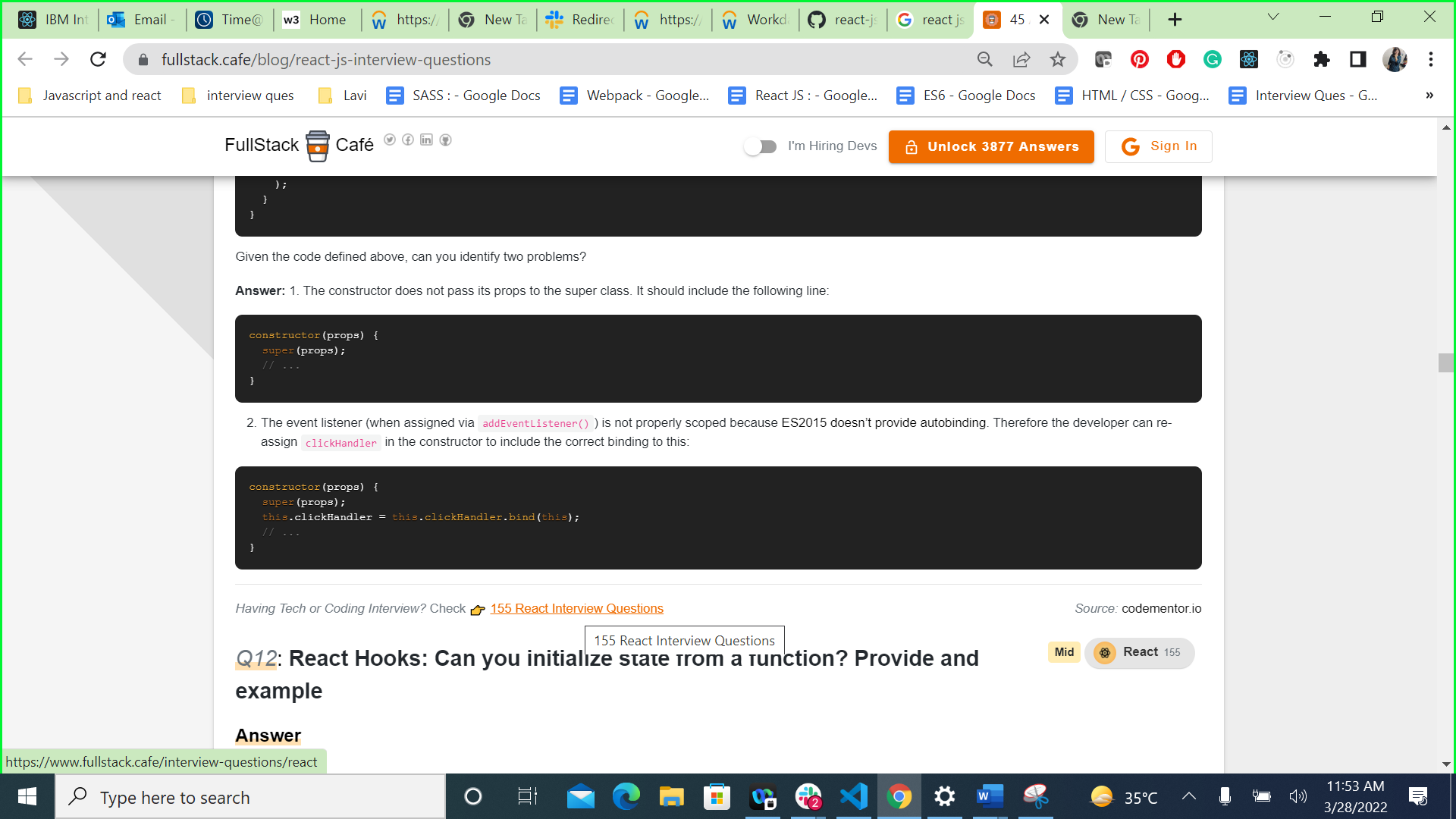
</div>

);

}

}

1. Identify 2 problems



QUESTION 2

import react from "react";

var username={name:'test'}

class App extends react.Component{

  constructor(){

    super();

    this.state={username:{name:'chhavi'}}

  }

  render(){

    var u=this.username;

    console.log(u);

    return(

      <div>

      </div>

    )

  }

}

export default App;

O/P – UNDEFINED

QUESTION 3

const TeamName = ({team=''}) => {

  const [teamName, setTeamName] = useState(team);

  return <div>{teamName && teamName}</div>;

};

function App(){

  const [val, setVal] = useState("Red Bull");

  return <><TeamName team={val}></TeamName></>

}

export default App;

O/P --- Red Bull ( destructing)

QUESTION 4

 const Example = () => {

  const [val, setVal] = useState('initial value');

  useEffect(() => {

   setVal('new value')

  }, []);

  useEffect(() => {

    console.log(val);

   }, [val]);

  return <>{console.log('render called')}</>;

};

export default Example;

O/P

render called

initial value

render called

new value

QUESTION 5

function Person(firstName, lastName) {

  this.firstName = firstName;

  this.lastName = lastName;

}

const member = new Person('Lydia', 'Hallie');

Person.getFullName = function() {

  return `${this.firstName} ${this.lastName}`;

};

console.log(member.getFullName());

In JavaScript, functions are objects, and therefore, the method getFullName gets added to the constructor function object itself. For that reason, we can call Person.getFullName(), but member.getFullName throws a TypeError.

If you want a method to be available to all object instances, you have to add it to the prototype property:

Person.prototype.getFullName = function() {

return `${this.firstName} ${this.lastName}`;

};

QUESTION 6

function checkAge(data) {

  if (data === { age: 18 }) {

    console.log('You are an adult!');

  } else if (data == { age: 18 }) {

    console.log('You are still an adult.');

  } else {

    console.log(`Hmm.. You don't have an age I guess`);

  }

}

checkAge({ age: 18 });

o/p

When testing equality, primitives are compared by their value, while objects are compared by their reference. JavaScript checks if the objects have a reference to the same location in memory.

The two objects that we are comparing don't have that: the object we passed as a parameter refers to a different location in memory than the object we used in order to check equality.

This is why both { age: 18 } === { age: 18 } and { age: 18 } == { age: 18 } return false.

QUESTION 7

const a = {};

const b = { key: 'b' };

const c = { key: 'c' };

a[b] = 123;

a[c] = 456;

console.log(a[b]);

Object keys are automatically converted into strings. We are trying to set an object as a key to object a, with the value of 123.

However, when we stringify an object, it becomes "[object Object]". So what we are saying here, is that a["[object Object]"] = 123. Then, we can try to do the same again. c is another object that we are implicitly stringifying. So then, a["[object Object]"] = 456.

Then, we log a[b], which is actually a["[object Object]"]. We just set that to 456, so it returns 456.

QUESTION 8

const promise1 = Promise.resolve('First')

const promise2 = Promise.resolve('Second')

const promise3 = Promise.reject('Third')

const promise4 = Promise.resolve('Fourth')

const runPromises = async () => {

  const res1 = await Promise.all([promise1, promise2])

  const res2  = await Promise.all([promise3, promise4])

  return [res1, res2]

}

runPromises()

  .then(res => console.log(res))

  .catch(err => console.log(err))

The Promise.all method runs the passed promises in parallel. If one promise fails, the Promise.all method rejects with the value of the rejected promise. In this case, promise3 rejected with the value "Third". We’re catching the rejected value in the chained catch method on the runPromises invocation to catch any errors within the runPromises function. Only "Third" gets logged, since promise3 rejected with this value.