upGrad
#RahoAmbitious

# Full Stack Software Development

upGrad

**Course:** Advanced Frontend Development Using React

**Lecture On:** React State and Lifecycle

## In the last class, we discussed...

- Props and how to use them inside a functional as well as a class component
- Events and event handling
- Developing functionality for adding a subscriber

upGrad

## Poll 1

Which of the following is correct syntax for a focus event handler, *onFocusHandler*?

A. `<button onfocus={onFocusHandler()}>`

B. `<button onFocus={onFocusHandler}>`

C. `<button onFocus={this.onFocusHandler}>`

D. `<button onfocus={this.onFocusHandler()}>`

# Poll 1 (Answer)

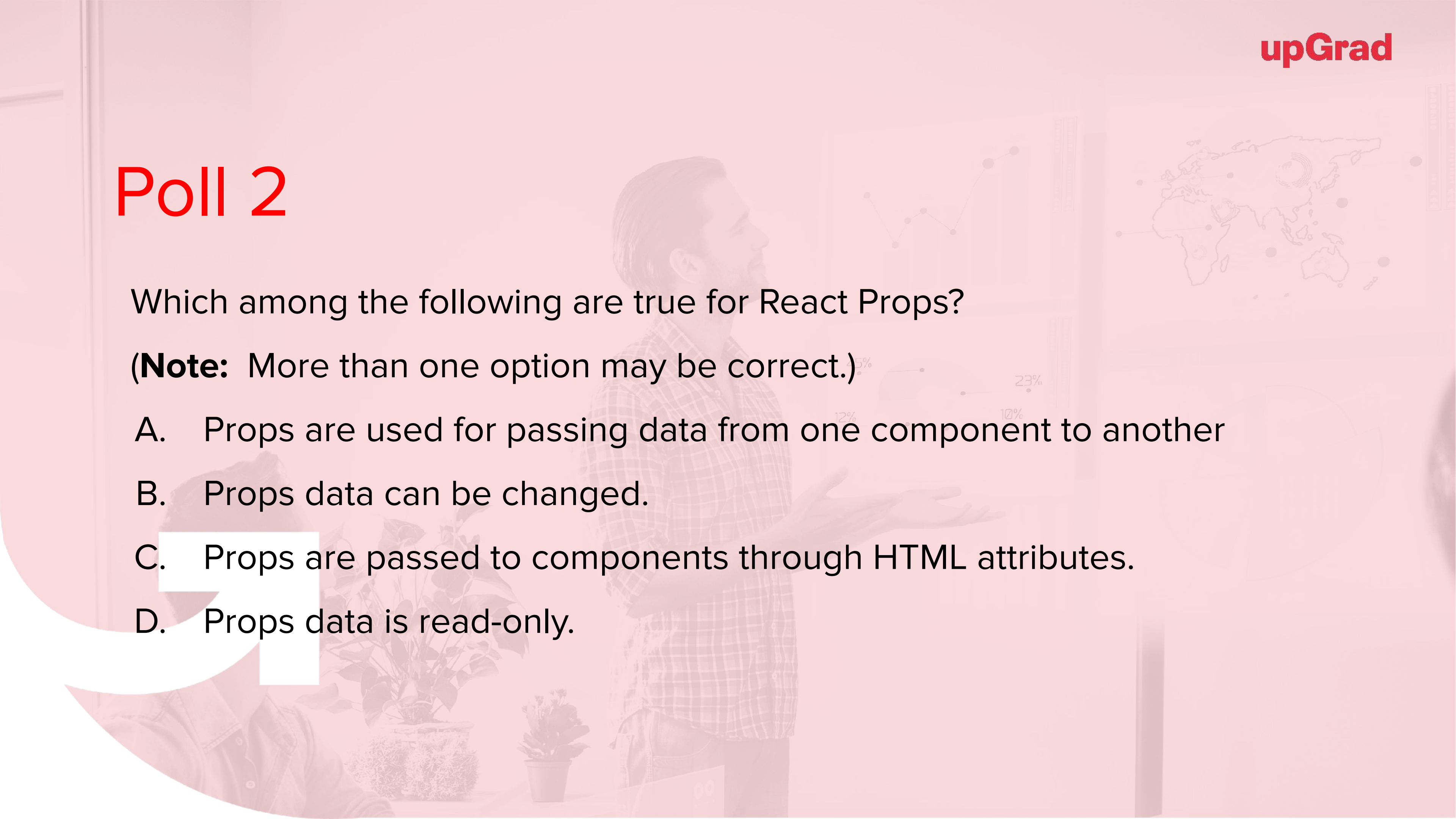
Which of the following is correct syntax for a focus event handler, *onFocusHandler*?

A. <button onfocus={onFocusHandler()}>

B. <button onFocus={onFocusHandler}>

**C. <button onFocus={this.onFocusHandler}>**

D. <button onfocus={this.onFocusHandler()}>



## Poll 2

Which among the following are true for React Props?

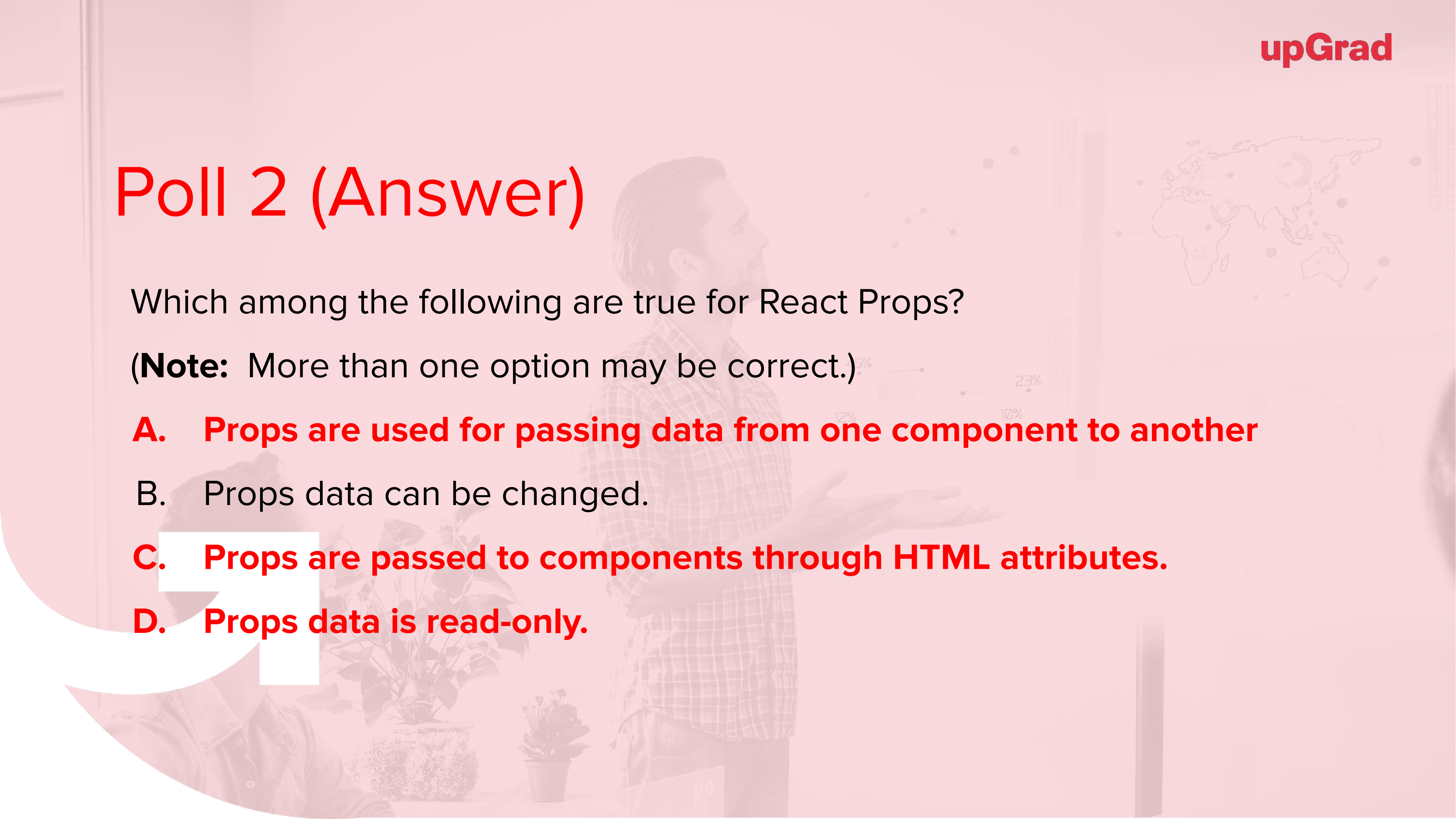
(**Note:** More than one option may be correct.)

A. Props are used for passing data from one component to another

B. Props data can be changed.

C. Props are passed to components through HTML attributes.

D. Props data is read-only.



upGrad

## Poll 2 (Answer)

Which among the following are true for React Props?

(**Note:** More than one option may be correct.)

**A. Props are used for passing data from one component to another**

B. Props data can be changed.

**C. Props are passed to components through HTML attributes.**

**D. Props data is read-only.**

## Today’s Agenda

1. Introduction to State
2. Characteristics of State
3. Setting the state using the *setState()* method
4. Lifecycle of components and the processes of mounting, updating and unmounting

# State

## State

upGrad

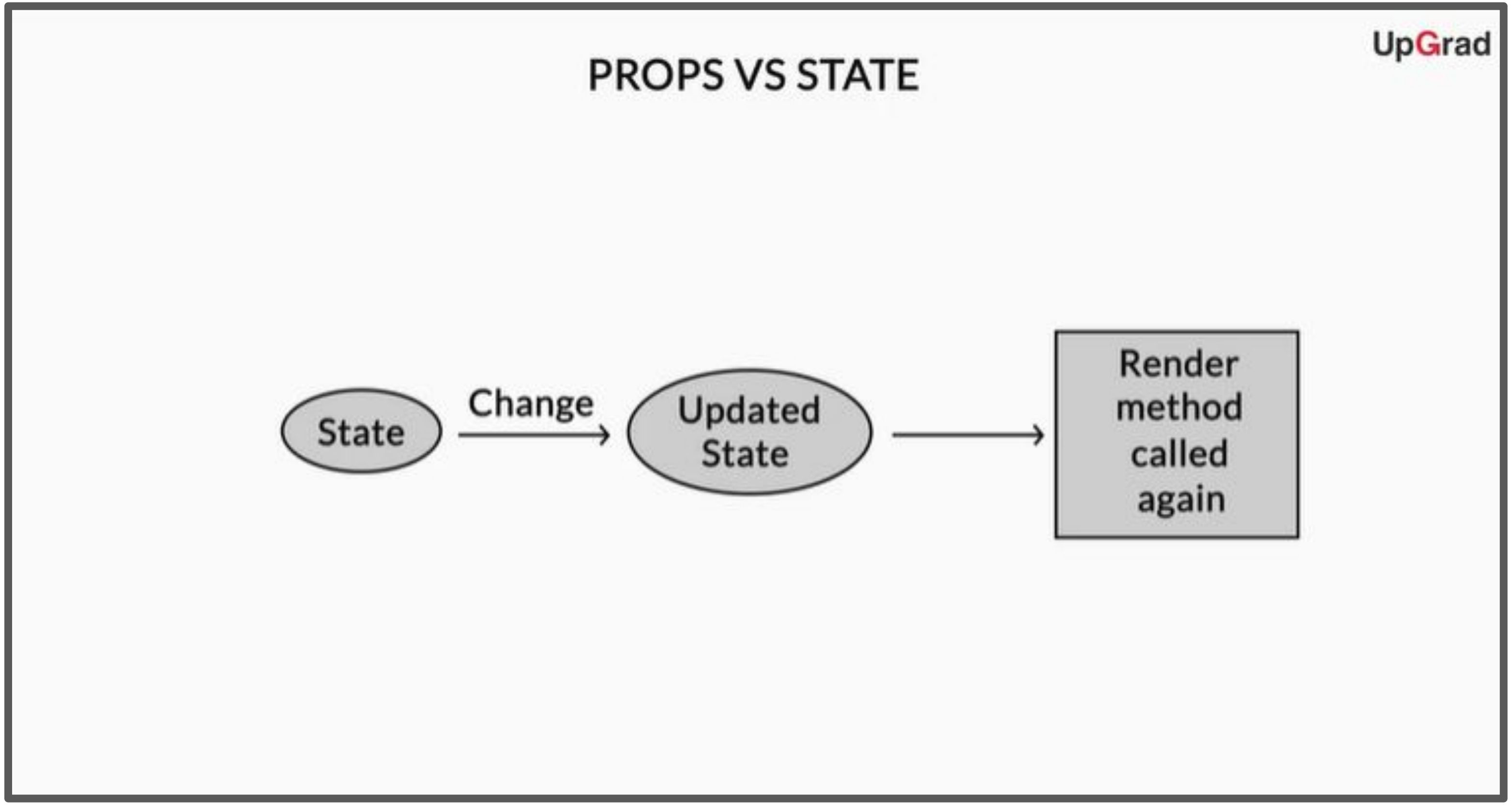
### State vs Props

In this session, we will look at one of the most basic ideas behind React - state

**Now, you might be wondering that props also maintain state in a way. Then, what is the difference between the two?**

- State is internal and controlled within a component unlike props that are external and are controlled by whatever renders a component

## State


UpGrad
PROPS VS STATE
State
Change
Updated State
Render method called again

## State

upGrad

### Stateful Components

- Remember that a change in state makes the *render()* method to be called again. This is one of the reasons why state can only be maintained inside a class component which consists of a *render()* method
- State is also only available to those components that extend from the Component class which can happen in class components only
- Thus, class components are called ‘stateful’

## State

upGrad

### Stateless Components

- This also leads to the conclusion that functional components cannot contain the state because they do not extend from the Component base class; thus, they do not have any *render()* method. Therefore, they are termed ‘stateless’
- From React 16.8, you could make a functional component stateful through hooks. You will learn about using hooks in the upcoming sessions

## State

upGrad

### Use of State in the Phone Directory Application

Let’s understand this concept even more clearly inside the Phone Directory application.

- In the Phone Directory application, the state changes when the user adds the name and phone number inside the input boxes
- The name and the phone are also updated so that they change when a user fills or makes changes in the input boxes
- This provides a better user experience

## State

upGrad

### Use of State in the Phone Directory Application

We need to modify the *AddSubscriber* page so that whenever a name or phone number is added, the state of the subscriber details changes and is reflected instantaneously

- In the *AddSubscriber.js* file, initialize state and write it inside the *constructor* since it is the first method that is called, making it the right place to initialise everything including the state.

```
constructor() {
  super();

  this.state = {
    id: 0,
    name: '',
    phone: ''
  };
}
```

**Note:** You need to reference super() method before referencing 'this' keyword inside the constructor of a class.

Code Reference

## State

upGrad

### Use of State in the Phone Directory Application

- Bind event listeners to the input textboxes.

```
<input id="name" type="text" className="input-control" name="name"
onChange={this.inputChangedHandler} /><br /><br />

<input id="phone" type="text" className="input-control" name="phone"
onChange={this.inputChangedHandler} /><br /><br />
```

- Define this event listener.

```
inputChangedHandler = (e) => {
  const state = this.state;
  state[e.target.name] = e.target.value;
  this.setState(state);
  console.log(this.state);
}
```

Code Reference

## State

upGrad

### Use of State in the Phone Directory Application

Now, let’s update the name and phone so that they change as and when a user fills or makes changes in the input boxes.

- Inside *render()* method, add this statement in the beginning to capture state properties inside variables. This is ES6 syntax

```
render() {
    const { name, phone } = this.state;
```

Code Reference

## State

upGrad

### Use of State in the Phone Directory Application

- Add these variables inside the subscriber-info fields so that they display the current state of the subscriber

```
<span className="subscriber-info">Name: {name}</span><br />
<span className="subscriber-info">Phone: {phone}</span>
```

*Go to browser and notice how beautifully state changes without any need for page reload*

Code Reference

## State

upGrad

### Use of *super(props)* in Constructor

In many cases, the state needs to be initialized with the passed prop value, and hence, *super(props)* is mandatory to access *this.props* inside constructor

```
constructor(props) {
    super(props);
    this.state = {
      //your code here
     };
}
```

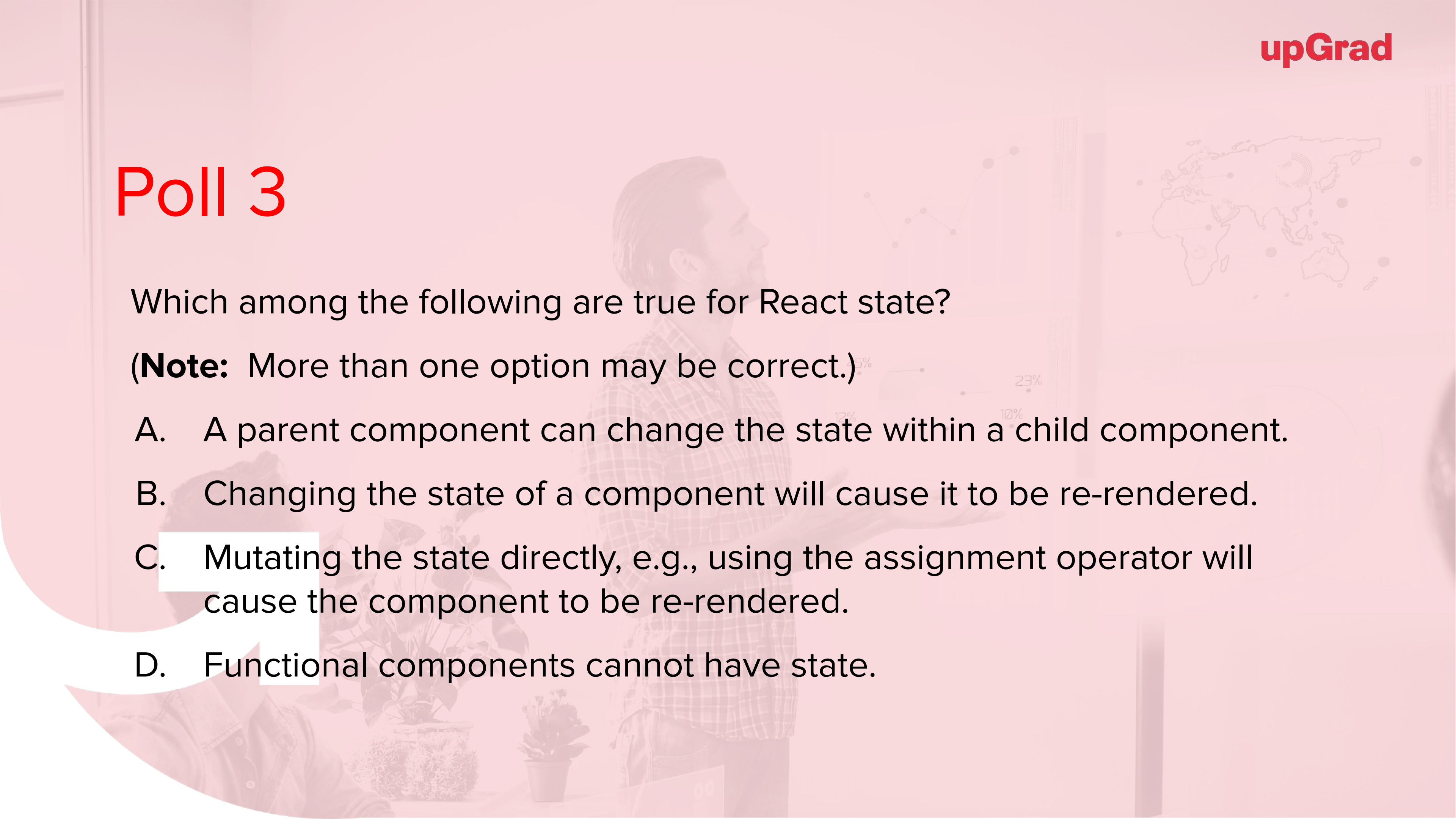
# State

upGrad

## Characteristics of a State

Some of the important points regarding state can be summarised as follows:

- A state can be maintained inside a class component only
- A state is always initialised inside the class constructor
- If you define the constructor of a class, you need to call the *super()* method in the first statement of the constructor definition. This method calls the constructor of the parent class
- To set the state, you must always use the *setState()* method and must never directly manipulate the application's state. However, *setState()* should never be called inside the constructor
- Whenever a state is changed, the *render()* method of the class is called again



## Poll 3

Which among the following are true for React state?

(**Note:** More than one option may be correct.)

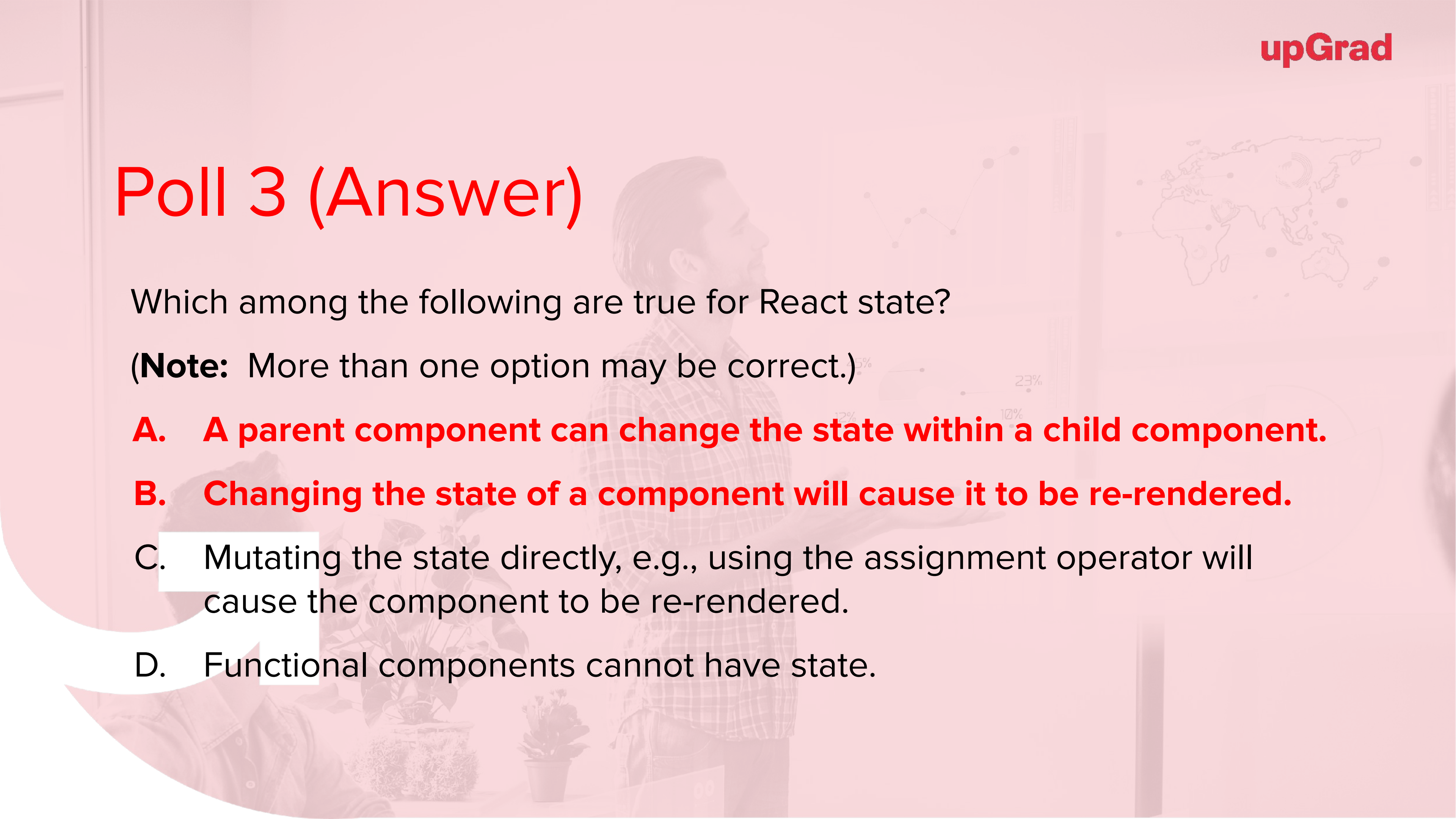
A. A parent component can change the state within a child component.

B. Changing the state of a component will cause it to be re-rendered.

C. Mutating the state directly, e.g., using the assignment operator will cause the component to be re-rendered.

D. Functional components cannot have state.

upGrad



## Poll 3 (Answer)

Which among the following are true for React state?

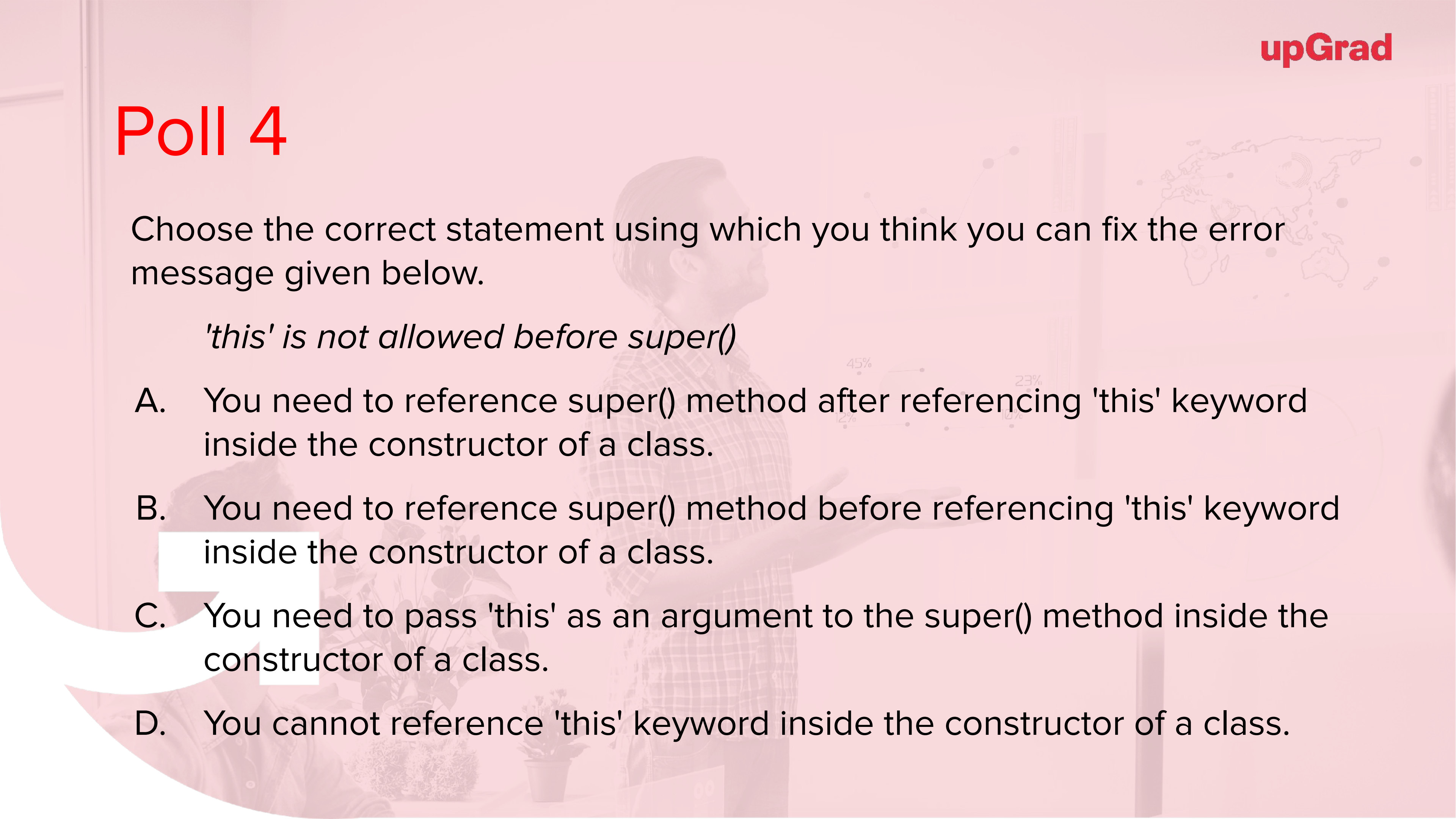
(**Note:** More than one option may be correct.)

**A. A parent component can change the state within a child component.**

**B. Changing the state of a component will cause it to be re-rendered.**

C. Mutating the state directly, e.g., using the assignment operator will cause the component to be re-rendered.

D. Functional components cannot have state.



## Poll 4

Choose the correct statement using which you think you can fix the error message given below.

*'this' is not allowed before super()*

A. You need to reference super() method after referencing 'this' keyword inside the constructor of a class.

B. You need to reference super() method before referencing 'this' keyword inside the constructor of a class.

C. You need to pass 'this' as an argument to the super() method inside the constructor of a class.

D. You cannot reference 'this' keyword inside the constructor of a class.

# Poll 4 (Answer)

Choose the correct statement using which you think you can fix the error message given below.

*'this' is not allowed before super()*

A. You need to reference super() method after referencing 'this' keyword inside the constructor of a class.

**B. You need to reference super() method before referencing 'this' keyword inside the constructor of a class.**

C. You need to pass 'this' as an argument to the super() method inside the constructor of a class.

D. You cannot reference 'this' keyword inside the constructor of a class.

## Poll 5

upGrad

What will be logged in the browser console when the following code snippet is executed?

```
class Answer extends React.Component {
     constructor(props) {
          super();
          console.log('Inside constructor: ',this.props);
     }
     render() {
          console.log('Inside render: ', this.props);
          return (
               <div>Hello world!</div>
          );
     }
}
ReactDOM.render(<Answer value={42} />, document.body);
```

A. Inside constructor: undefined
Inside render: { value: 42 }

C. Inside constructor: { value: 42 }
Inside render: { value: 42 }

A. Inside constructor: {}
Inside render: undefined

C. Inside constructor: {}
Inside render: {}

# Poll 5 (Answer)

upGrad

What will be logged in the browser console when the following code snippet is executed?

```
class Answer extends React.Component {
     constructor(props) {
          super();
          console.log('Inside constructor: ', this.props);
     }
     render() {
          console.log('Inside render: ', this.props);
          return (
               <div>Hello world!</div>
          );
     }
}
ReactDOM.render(<Answer value={42} />, document.body);
```

**A. Inside constructor: undefined**
**Inside render: { value: 42 }**

C. Inside constructor: { value: 42 }
Inside render: { value: 42 }

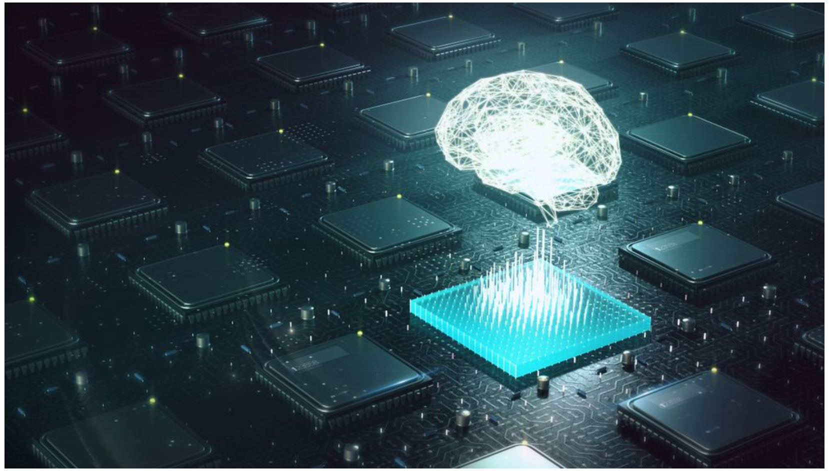
A. Inside constructor: {}
Inside render: undefined

C. Inside constructor: {}
Inside render: {}

## Component Lifecycle

## Component Lifecycle

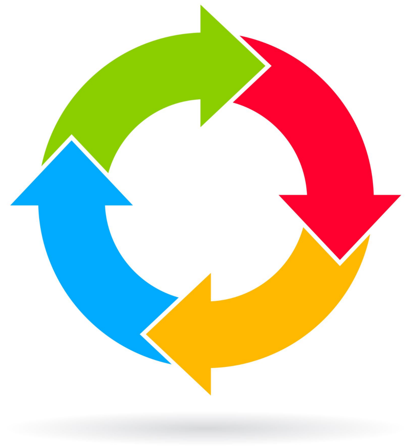
- Let’s dive a little deep into the React concepts
- Previously, you have learnt about states
- Now, you will learn about the lifecycle of components



## Component Lifecycle

upGrad

- The components in React undergo a series of changes that defines their lifecycle
- The lifecycle in React varies from one process to another, and in total, the three processes are **mounting, updating and unmounting**



## Component Lifecycle

- **Mounting** refers to the instance of a component being created and inserted into DOM
- **Updating** denotes the instance of a component being updated by props or a state
- **Unmounting** refers to the component being removed from DOM

## Component Lifecycle

upGrad

### Mounting

Inside a mounting process, a component's lifecycle is defined by the following methods, which are called in the given order:

1. **constructor()**
2. **render()**
3. **componentDidMount()**

## Component Lifecycle

upGrad

### Phone Directory Application

Now, in the Phone Directory application, let’s manipulate state and see its effect on a component’s lifecycle

- In the *index.js* file and render the *App* component, instead of the *AddSubscriber* component

```
ReactDOM.render(<App />, document.getElementById('root'));
```

**Note:** This would start displaying the App component which shows the list of subscribers.

Code Reference

## Component Lifecycle

upGrad

### Phone Directory Application

- Go to *App.js* file and remove the variable ‘subscribers’

```
// let subscribers = [
//   {
//     id: 1,
//     name: "Shilpa Bhat",
//     phone: "8888888888"
//   },
//   {
//     id: 2,
//     name: "Srishti Gupta",
//     phone: "9999999999"
//   }
// ]
```

Code Reference

## Component Lifecycle

upGrad

### Phone Directory Application

- Write *constructor* function inside *App* component. Define state which will maintain the state of the *App* component to display all the subscribers

```
constructor(props) {
  super(props);

  this.state = {
    subscribersListToShow: []
  };

  console.log("Constructor called!");
}
```

- Change this variable inside render() method and map it with state

```
this.state.subscribersListToShow.map(sub => {
```

Code Reference

# Component Lifecycle

upGrad

## Phone Directory Application

- Now, add componentDidMount() method inside the App component which is also a lifecycle method

```
componentDidMount() {
    console.log("componentDidMount called!");
    let newSubscriber = {
      id: 1,
      name: "Shilpa Bhat",
      phone: "8888888888"
    }
    let subscribersList = this.state.subscribersListToShow;
    subscribersList.push(newSubscriber);
    this.setState({ subscribersListToShow: subscribersList });
}
```

***Note:*** *We are changing state in this method by pushing a subscriber in the state of the component.*

Code Reference

## Component Lifecycle

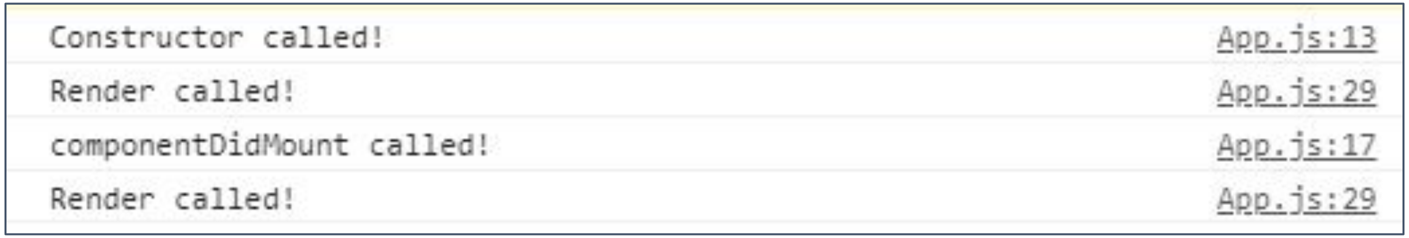
upGrad

### Phone Directory Application

- Add a console.log() statement inside render() method too. This method will be called whenever render() method is called

```
render() {
   console.log("Render called!");
```

- See the changes in the console tab in DevTools



```
Constructor called!                                        App.js:13
Render called!                                             App.js:29
componentDidMount called!                                  App.js:17
Render called!                                             App.js:29
```

Notice how the *render()* method is called again after the state has been changed in the *componentDidMount()* method

Code Reference

# Component Lifecycle

upGrad

## Phone Directory Application

To sum it all up:

- A change has been made in the state. A new subscriber is added in the list using *push*
- The state is set using *this.setState*. The state is initialised in the constructor using *this.state*
- At the remaining locations, *this.setState* is used
- The change in the state has been made inside componentDidMount. The method is triggered when the page is mounted
- This change in state leads to the render() method being called again

## Component Lifecycle

upGrad

### Phone Directory Application (Hands-on)

- Remove all the *console.log()* methods inside your application
- Also, remove the *componentDidMount()* method that you wrote just now inside the *App.js* file

Code Reference

## Component Lifecycle

upGrad

### Updating

Any component can be updated by causing changes to props or state. Inside an updating process, the following methods are called when a component is re-rendered:

1. **render()**
2. **componentDidUpdate()** - not called for the initial render

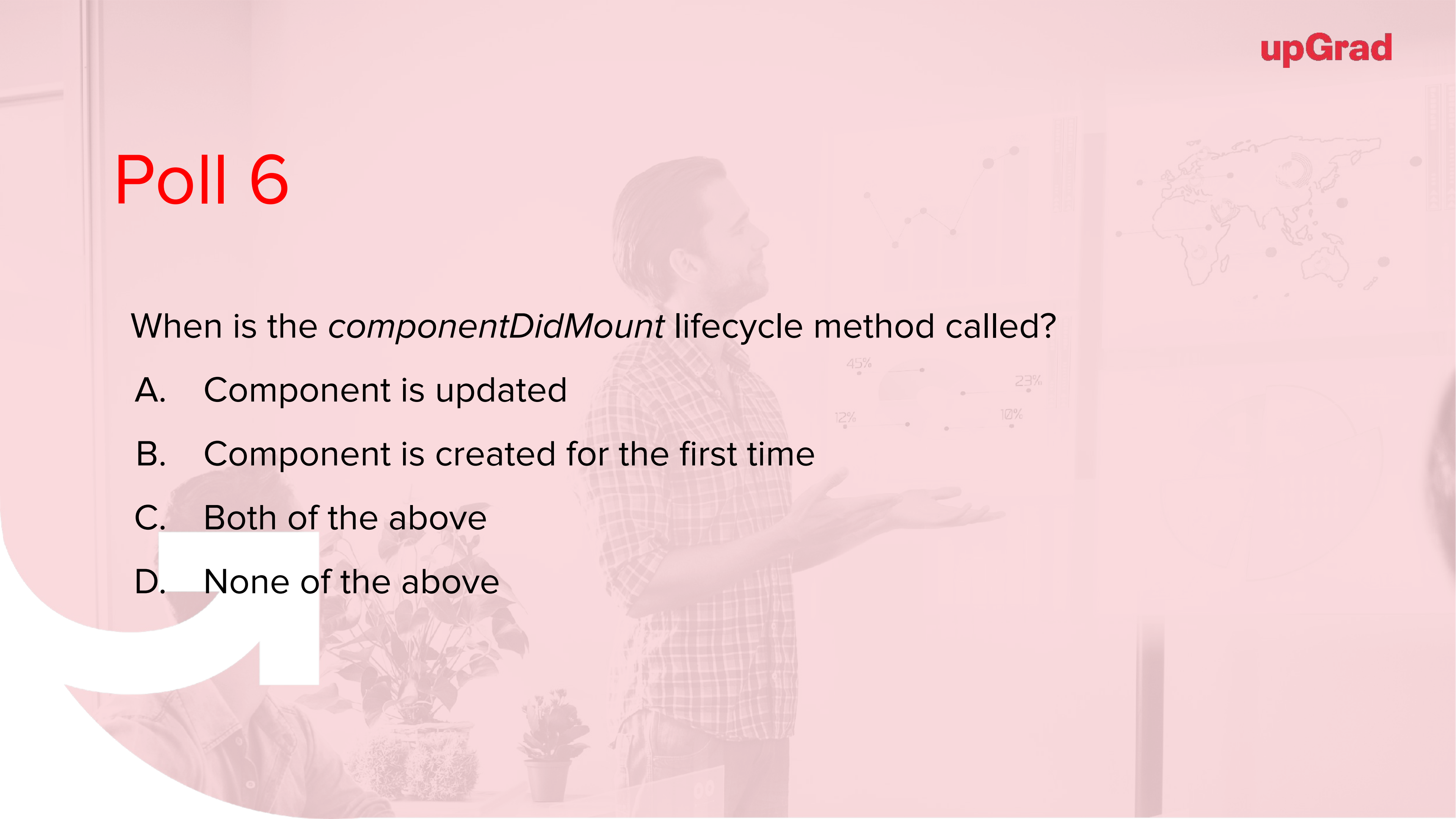
**Note:** There are times when you do not want the component's output to be affected due to any change in props or state. In such cases, the *shouldComponentUpdate()* method is used. It is invoked while receiving new props or state i.e. before rendering.

## Component Lifecycle

upGrad

### Unmounting

- In an unmounting process, **componentWillUnmount()** method is called when a component is being removed from the DOM
- Once a component instance is unmounted, you can never mount it again. Hence, you should never call the *setState()* method inside the *componentWillUnmount()* method since the component will never be re-rendered



## Poll 6

When is the *componentDidMount* lifecycle method called?

A. Component is updated

B. Component is created for the first time

C. Both of the above

D. None of the above

# Poll 6 (Answer)

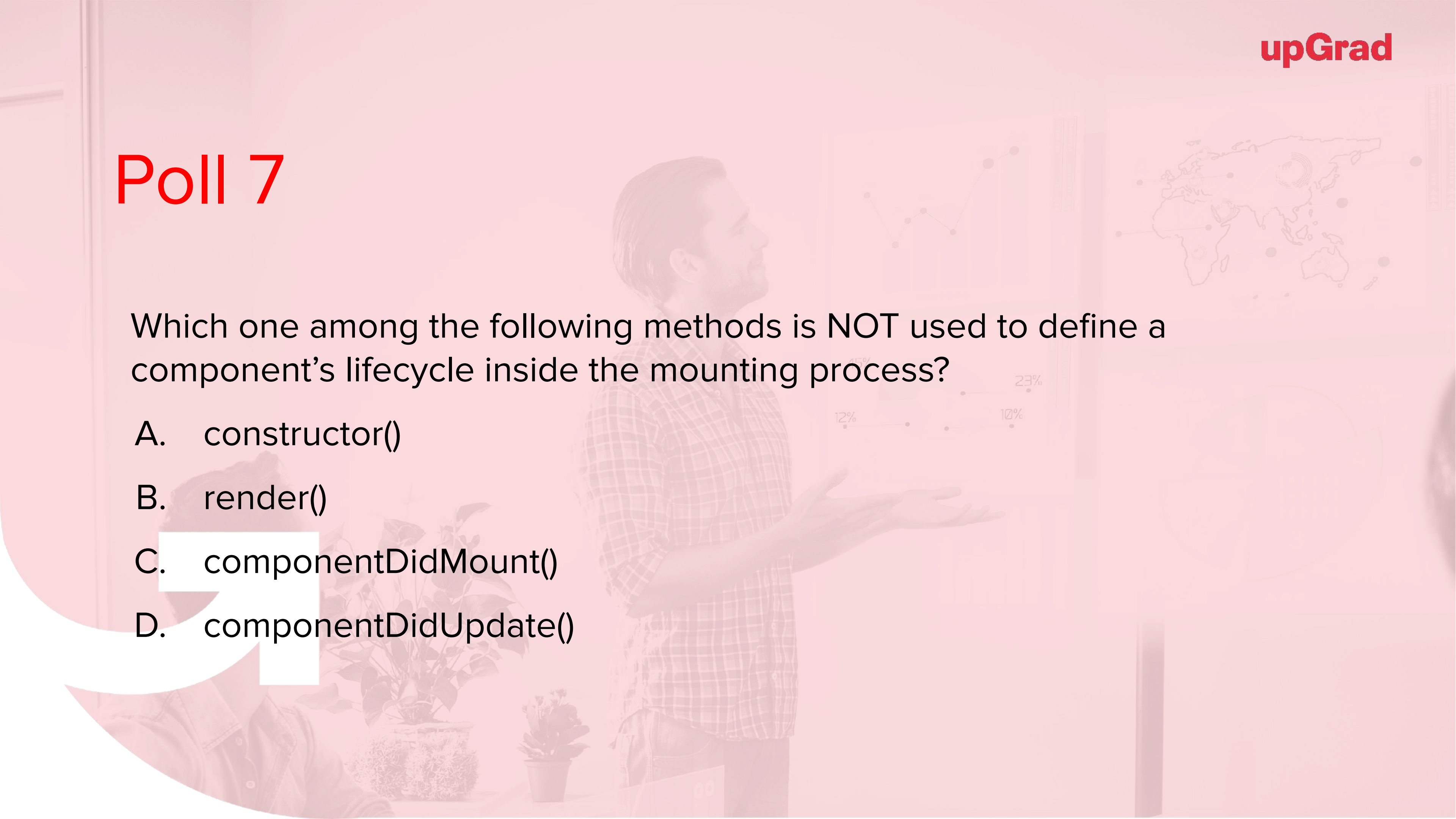
When is the *componentDidMount* lifecycle method called?

A. Component is updated

**B. Component is created for the first time**

C. Both of the above

D. None of the above



## Poll 7

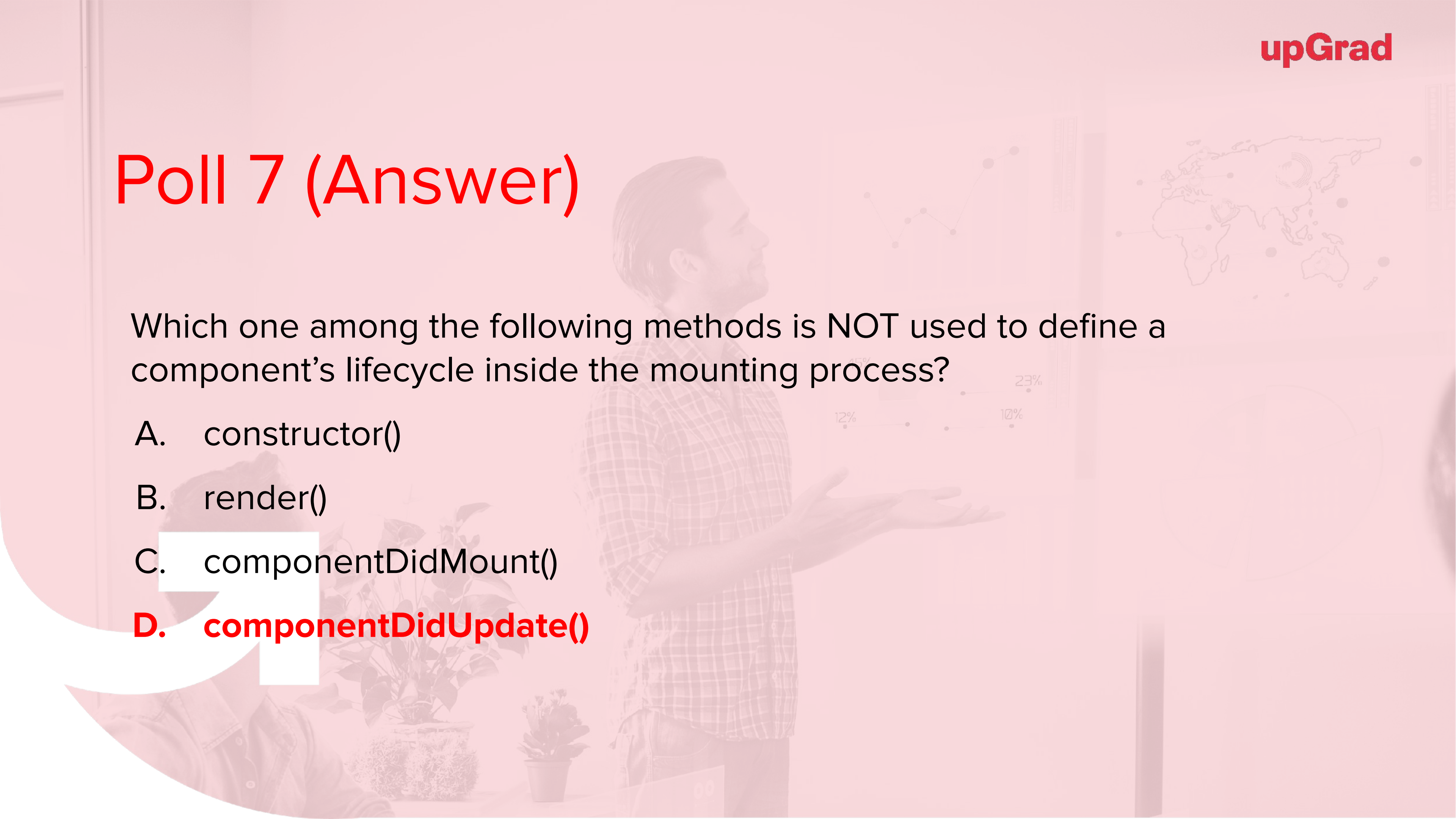
Which one among the following methods is NOT used to define a component's lifecycle inside the mounting process?

A. constructor()

B. render()

C. componentDidMount()

D. componentDidUpdate()



## Poll 7 (Answer)

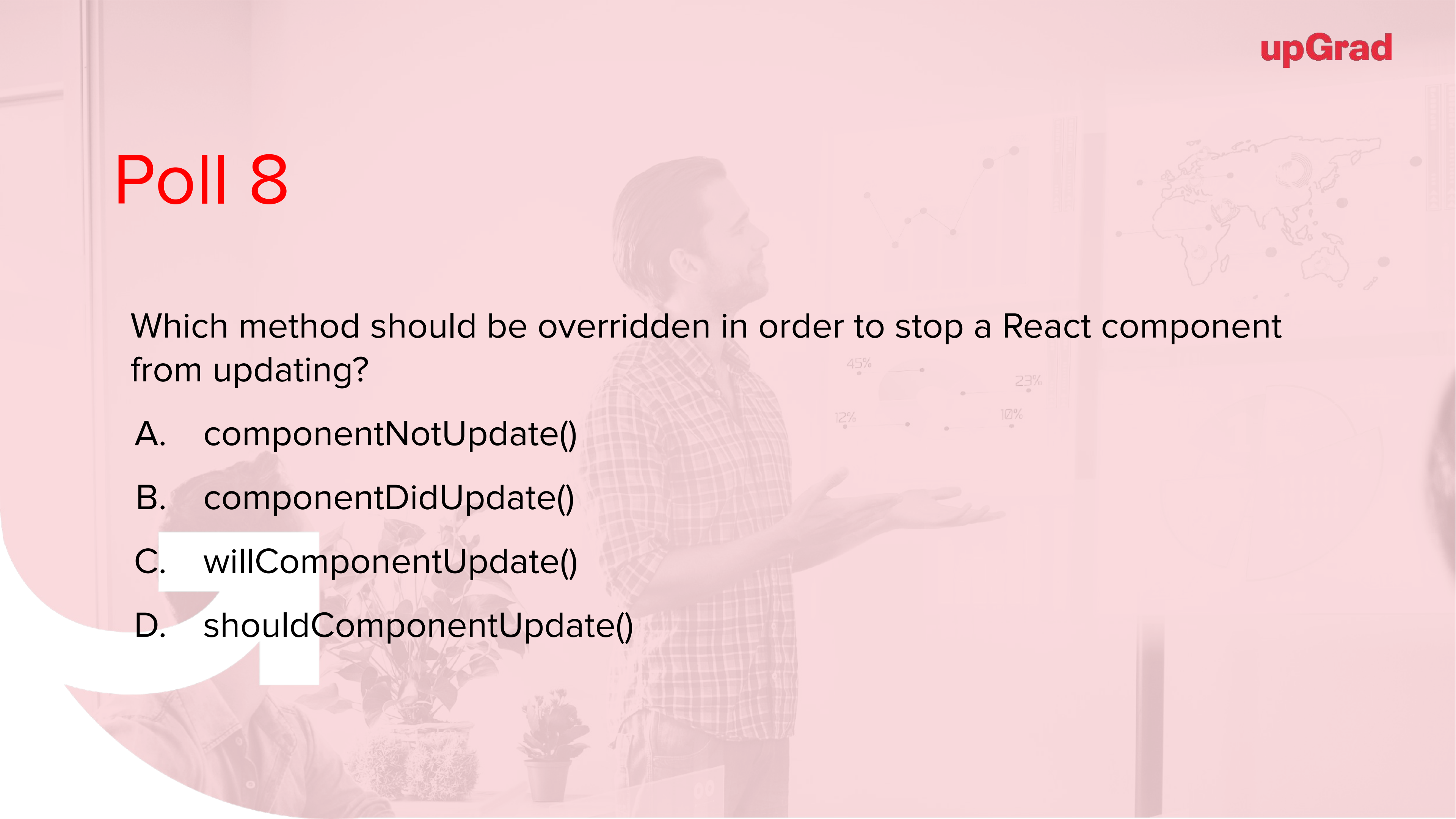
Which one among the following methods is NOT used to define a component’s lifecycle inside the mounting process?

A. constructor()

B. render()

C. componentDidMount()

**D. componentDidUpdate()**



# Poll 8

Which method should be overridden in order to stop a React component from updating?

A. componentNotUpdate()

B. componentDidUpdate()

C. willComponentUpdate()

D. shouldComponentUpdate()

upGrad

## Poll 8 (Answer)

Which method should be overridden in order to stop a React component from updating?

A. componentNotUpdate()

B. componentDidUpdate()

C. willComponentUpdate()

**D. shouldComponentUpdate()**

upGrad

All the code used in today’s session can be found in the link provided below (branch session5-demo):

https://github.com/upgrad-edu/react-class-components/tree/session5-demo

upGrad

## Doubt Clearance (5 minutes)

upGrad

## Important Questions

## Important Questions

upGrad

1. How do you update the state of a component? Why should we not do it directly?
2. Explain the lifecycle of a React component.
3. What will happen if you use the setState() method in the constructor?
4. Why should we not call the setState() method in componentWillUnmount() method?
5. What is the recommended ordering of methods in component class?

## Key Takeaways

- A state is controlled within a component unlike props that are controlled by a parent component. Also, a change in the state calls the *render()* method again
- A state can be maintained inside a class component and is always initialised inside the class constructor
- If you define the constructor of a class, you need to call the *super()* method in the first statement of the constructor definition. This method calls the constructor of the parent class
- To set the state, you must always use the *setState()* method and must never directly manipulate the application’s state. However, setState() should never be called inside the constructor

## Key Takeaways

- The component lifecycle in React varies from one process to another, and in total, the three processes are *mounting, updating* and *unmounting*
- Inside mounting process, a component's life cycle is defined by the following methods, which are called in the given order:
    - constructor()
    - render()
    - componentDidMount()

## These tasks are to be completed after today's session:

| |
|---|
| MCQs |
| Coding Questions |
| Course Project (Part A) - Checkpoint 4 |

upGrad

## In the next class, we will discuss...

1. Single-Page and Multi-Page Applications and their advantages over each other
2. Types of components: Smart and Dumb
3. Difference between smart and dumb components
4. Routing in React
5. Implementing routing in the application using a node package called ‘react-router-dom’
6. Developing a functionality for deleting a subscriber

upGrad
#RahoAmbitious

Thank You!