What are the differences between functional and class components?

Before the introduction of Hooks in React, functional components were called stateless components and were behind class components on a feature basis. After the introduction of Hooks, functional components are equivalent to class components.

Although functional components are the new trend, the react team insists on keeping class components in React. Therefore, it is important to know how these components differ.

On the following basis let's compare functional and class components:

- **Declaration**

  Functional components are nothing but JavaScript functions and therefore can be declared using an arrow function or the function keyword:

```
function card(props){
 return(
    <div className="main-container">
      <h2>Title of the card</h2>
    </div>
  )
}
const card = (props) =>{
 return(
    <div className="main-container">
      <h2>Title of the card</h2>
    </div>
  )
}
```

  Class components, on the other hand, are declared using the ES6 class:

```
class Card extends React.Component{
 constructor(props){
    super(props);
  }
   render(){
     return(
       <div className="main-container">
         <h2>Title of the card</h2>
       </div>
     )
   }
  }
```

- **Handling props**

  Let's render the following component with props and analyse how functional and class components handle props:

```
<Student Info name="Vivek" rollNumber="23" />
```

In functional components, the handling of props is pretty straightforward. Any prop provided as an argument to a functional component can be directly used inside HTML elements:

```
function StudentInfo(props){
  return(
    <div className="main">
      <h2>{props.name}</h2>
      <h4>{props.rollNumber}</h4>
    </div>
  )
}
```

In the case of class components, props are handled in a different way:

```
class StudentInfo extends React.Component{
  constructor(props){
    super(props);
   }
   render(){
     return(
       <div className="main">
         <h2>{this.props.name}</h2>
         <h4>{this.props.rollNumber}</h4>
       </div>
     )
   }
  }
```

As we can see in the code above, **this** keyword is used in the case of class components.

- **Handling state**

  Functional components use React hooks to handle state. It uses the useState hook to set the state of a variable inside the component:

```
function ClassRoom(props){
  let [studentsCount,setStudentsCount] = useState(0);
   const addStudent = () => {
     setStudentsCount(++studentsCount);
  }
   return(
     <div>
       <p>Number of students in class room: {studentsCount}</p>
       <button onClick={addStudent}>Add Student</button>
     </div>
   )
  }
```

  Since useState hook returns an array of two items, the first item contains the current state, and the second item is a function used to update the state.

In the code above, using array destructuring we have set the variable name to studentsCount with a current value of “0” and setStudentsCount is the function that is used to update the state.

For reading the state, we can see from the code above, the variable name can be directly used to read the current state of the variable.

We cannot use React Hooks inside class components, therefore state handling is done very differently in a class component:

Let’s take the same above example and convert it into a class component:

```
class ClassRoom extends React.Component{
        constructor(props){
            super(props);
            this.state = {studentsCount : 0};

            this.addStudent = this.addStudent.bind(this);
         }

            addStudent(){
            this.setState((prevState)=>{
               return {studentsCount: prevState.studentsCount++}
            });
         }

            render(){
             return(
               <div>
                 <p>Number of students in class room:
{this.state.studentsCount}</p>
                 <button onClick={this.addStudent}>Add Student</button>
               </div>
             )
           }
         }
```

In the code above, we see we are using **this.state** to add the variable studentsCount and setting the value to “0”.

For reading the state, we are using **this.state.studentsCount**.

For updating the state, we need to first bind the addStudent function to **this**. Only then, we will be able to use the **setState** function which is used to update the state.

What is the purpose of render() in React.

Each React component must have a **render()** mandatorily. It returns a single React element which is the representation of the native DOM component. If more than one HTML element needs to be rendered, then they must be grouped together inside one enclosing tag such as **<form>, <group>, ** etc. This function must be kept pure i.e., it must return the same result each time it is invoked.

Differentiate between states and props.

Conditions	State	Props
1. Receive initial value from parent component	Yes	Yes
2. Parent component can change value	No	Yes
3. Set default values inside component	Yes	Yes
4. Changes inside component	Yes	No
5. Set initial value for child components	Yes	Yes
6. Changes inside child components	No	Yes

Differentiate between stateful and stateless components.

Stateful Component	Stateless Component
1. Stores info about component's state change in memory	1. Calculates the internal state of the components
2. Have authority to change state	2. Do not have the authority to change state
3. Contains the knowledge of past, current and possible future changes in state	3. Contains no knowledge of past, current and possible future state changes

4. Stateless components notify them about the requirement of the state change, then they send down the props to them.	4. They receive the props from the Stateful components and treat them as callback functions.

What are the different phases of React component's lifecycle?

There are three different phases of React component's lifecycle:

i. *Initial Rendering Phase:* This is the phase when the component is about to start its life journey and make its way to the DOM.
ii. *Updating Phase:* Once the component gets added to the DOM, it can potentially update and re-render only when a prop or state change occurs. That happens only in this phase.
iii. *Unmounting Phase:* This is the final phase of a component's life cycle in which the component is destroyed and removed from the DOM.

Explain the lifecycle methods of React components in detail.

Some of the most important lifecycle methods are:

i. ***componentWillMount()*** – Executed just before rendering takes place both on the client as well as server-side.
ii. ***componentDidMount()*** – Executed on the client side only after the first render.
iii. ***componentWillReceiveProps()*** – Invoked as soon as the props are received from the parent class and before another render is called.
iv. ***shouldComponentUpdate()*** – Returns true or false value based on certain conditions. If you want your component to update, return **true** else return **false**. By default, it returns false.
v. ***componentWillUpdate()*** – Called just before rendering takes place in the DOM.
vi. ***componentDidUpdate()*** – Called immediately after rendering takes place.
vii. ***componentWillUnmount()*** – Called after the component is unmounted from the DOM. It is used to clear up the memory spaces.