React js Interview questions

**What is React?**

React is a open-source front end JavaScript library developed by Facebook in 2011. It facilitate the creation of interactive, stateful & reusable UI components. React is using for create large interactive web applications and mobile application. It change over time without reloading the page. It is covering for ‘View’ in MVC (Model-View-Controller).

* Simple and easy to learn:
* Code reusability and data binding: it use one way data binding
* Performance and testing:
* Use JSX : simple use html inside render function and convert into the react

**What are the ReactJS Features?**

* It uses the virtual DOM instead of the real DOM.
* It uses server-side rendering.
* It follows uni-directional data flow or data binding.
* JSX (JavaScript XML)
* React Native (It uses in mobile application)

**What is Virtual DOM ?**Virtual DOM is in-memory representation of Real DOM. It is lightweight JavaScript object which is copy of Real DOM that is know as Virtual DOM.

The virtual DOM (VDOM) is a programming concept where an ideal, or “virtual”, representation of a UI is kept in memory and synced with the “real” DOM by a library such as ReactDOM. This process is called [**reconciliation**](https://reactjs.org/docs/reconciliation.html).

**How Does react Virtual DOM Work?**In react Virtual DOM is a node tree that have lists of elements and their attributes, content as object and their properties. React js methods create node tree react components. It updates node tree in the response to mutation in the data model.

Virtual DOM representation the user interface has created.If you are thinking about how many phase in update the browser’s DOM.

<https://www.youtube.com/watch?v=bzJDdH5FMYU> server side rendering

**Why virtual dom is faster ?**

Virtual dom is faster because it uses.

1. Efficient diff algorithm

2. Batched update operations

3. Efficient update of sub tree only

4. Uses observable instead of dirty checking to detect change

Note : ReactJS maintains two virtual DOM, one of the updated state Virtual DOM and another with the prior state Virtual DOM.  
What is Declarative as react js  
Declarative views make your code more predictable and easier to debug. React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes.

### Different between Real DOM and Virtual DOM.

**Real DOM**

1. It updates slow.

2. It Can directly update HTML.

3. Too much of memory wastage.

4. Creates a new DOM if element updates.

**Virtual DOM**

1. It updates faster.

2. It Can’t directly update HTML.

5. No memory wastage.

3. Updates the JSX if element updates

1. **Major Disadvantages of using React.**  
   It is only covering for ‘View’ layer in MVC (Model-View-Controller).React is just a JavaScript library, Not a framework.Its library is very large and takes time to understand.it uses inline templating and JSX.

**What is state in react js ?**it holds the information about the component and state is using only inside the component. State is mutable. By default component has no state. the State of a component is an object that holds some information. that control the behavior of the component.

**What happen if we use any props inside constructor?**

Constructor will call only once so if we are setting any state based on props. That state value will not reflect in the component. So we can use props outside of constructor or inside render function

**What is props in react js ?**Props means properties, It’s a way of passing data from parent to child and we can say that props are just a communication channel between components. Always moving from parent to child component. It’s a immutable and read-only.

**What is different b/w super() and super(props)?**Only one reasonm for using props parameter in super keyword. When you want to access this.props inside the constructor then you need to pass the props parameter in super keyword and no difference outside constructor.

**What is super keyword in react ?**Super keyword is use to implement parent constructor. In current component. In [JavaScript classes](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes), you need to always call super when defining the constructor of a subclass. All React component classes that have a constructor should start it with a super(props) call.

**Why can’t browsers read JSX?**Browsers can only read JavaScript objects but JSX is not a regular JavaScript object. That’s why browsers can’t read JSX. Read JSX on browsers for first of all we need to transform JSX file into a JavaScript object using JSX transformers like Babel and then pass it to the browser.

**What is different b/w React and Angular?**REACT

* Only the View of MVC
* Server side rendering
* Uses virtual DOM
* One-way data binding
* Compile time debugging
* Facebook

ANGULAR

* Complete MVC
* Client side rendering
* Uses real DOM
* Two-way data binding
* Run time debugging
* Google

**Differentiate between states and props.**In **state** parent component can not change the value. in **props** parent component can change.  
Use **state** to store the data your current page needs in your controller-view, should not be accessed from child components, The state is a data structure that starts with a default value when a Component mounts.  
Use **props** to pass data & event handlers down to your child components, Props (short for properties) are a Component's configuration.  
  
**Differentiate between stateful and stateless components ?**StateFull : Can change state, update state, set new state, stateless : can not:

**What is Higher Order Components HOC in react?**

A higher-order component (HOC) is an advanced technique in React for reusing component logic. It is a function (or wrapping around normal function) that take a component and provide additional data input and return as a new component. HOCs are not part of the React API, per se. They are a pattern that emerges from React’s compositional nature. HOC is a pure function doesn’t have any side effects  
**Note** : While the convention for higher-order components is to pass through all props to the wrapped component, this does not work for refs.React.forwardRef.connect(),withRouter(),withStyle()  
**Alternatively, if state is unnecessary, it’s recommended to use a functional HOC:**Live example : creating a tax and intrest calculation hoc.having all intrest calculation function and tax. Second example is creating a hoc for getting and setting the value in local storage

import React from 'react';

const Withcal = WrappedComponent => {

    const intrest = (p,r,t)=>((p\*r\*t)/100);

    return props => <WrappedComponent name="vinay" intrest={intrest} {...props} />;

  };

  export default Withcal;

import React from 'react';

const Withcal = WrappedComponent => {

  return class Hoc extends React.Component {

      constructor(props) {

        super(props);

        this.intrest=this.intrest.bind(this);

        this.state = {

            intrest: this.intrest,

            name:'vinay'

        };

      }

      intrest = (p,r,t)=>((p\*r\*t)/100);

      render() {

        return <WrappedComponent {...this.state} {...this.props} />;

      }

    }

  // or return Hoc

  };

  export default Withcal;

import React from 'react';

import Withcal from './Hoc';

class Users extends React.Component{

    constructor(props){

        super(props);

    }

 render(){

 return(

    <>

       Hi {this.props.name} <br />

       Intrest calculation(p=1000,r=5%,t=3years ) ={this.props.intrest(1000,5,3)}

    </>

 )

 }

}

 export default Withcal(Users);

Note : we can combine multiple hoc   
import { compose } from 'redux';

export default compose(

withStyles(styles),

withRouter,

withUser

)(UserPage);

**Differentiate between controlled and uncontrolled components?  
Controlled Components**A controlled component is a react component that controls the values of input elements in a form using setState()  
1. They do not maintain their own state  
2. Data is controlled by the parent component  
3. They take in the current values through props and then notify the changes via callbacks

**Uncontrolled Components**A controlled component is a react component that controls the values of input elements in a form using refit is alternative of control component when data is not control by its own state, control by its dom or ref  
1. They maintain their own state  
2. Data is controlled by the DOM  
3. Refs are used to get their current values

**What are Pure Components?**Pure components don’t have functionality of re-render. It is simplest and fastest components only have render() that is called pure components.  
Class TestComponent extends React.pureComponent{ }

**What is React Router?**Reactjs router most powerful routing library, It helps in adding new screens and flows of application. It provides a sync URL with data that’s be displayed on the web page. It maintains standard structure and behavior and It is used for developing single page web applications.  
The Router component functions like a control center, and connects the route path (link) with the React component that should appear on the page.

the **Router** component needs to be the highest parent. This just let’s all of the component use the power of Router, because as a parent, it passes down all of its props to its children, and thus the entire application.

Using <**Link**> functions similarly to using an <a> tag, but, as mentioned above, prevents a page refresh, and looks like a React component  
**History API lets you interact with the browser history, trigger the browser navigation methods and change the address bar content**.

How to work : browser router is a central control component. Route connect with highest parent component. Link prevent the refresh or loading the page and synch with parent component. Parent component passes props to its child component

import { Route, Link, BrowserRouter as Router } from 'react-router-dom'

const routing = (

  <Router>

    <div>

      <ul>

        <li>

          <Link to="/">Home</Link>

        </li>

      </ul>

      <Switch>

        <Route exact path="/" component={App} />

        <Route path="/users/:id" component={User} />

        <Route path="/contact" component={Contact} />

        <Route path="/(about|who)/" component={Dashboard} />

        <Route component={Notfound} />

      </Switch>

    </div>

  </Router>

)

ReactDOM.render(routing, document.getElementById('root'))

**Inline rendering**

<Route

  path="/(about|who)/"

  render={() => (

    <div>

      <h2>About</h2> Note : this.props.match.params.id \*\*\*\*\* for dynamic getting params

      ...

    </div>

  )}

/>

**xzswqdr**

export const AuthRoute = ({ loggedIn, path, component: Component }) => (

    <Route

      path={path}

      render={props => (

        loggedIn ?

        <Redirect to='/dashboard' /> :

        <Component {...props} />

      )}

    />

  )

  export  const ProtectedRoute = ({ loggedIn, path, component: Component }) => (

    <Route

      path={path}

      render={props => (

        loggedIn ?

        <Component {...props} /> :

        <Redirect to='/login' />

      )}

    />

  );

**What is Switch?**Switch component helps us to render the components only when path matches otherwise it fallbacks to the not found component.

**How is React Router different from conventional routing?  
Conventional Routing**Each view corresponds to a new file  
A HTTP request is sent to a server and corresponding HTML page is received  
User actually navigates across different pages for each view

**React Routing**Only single HTML page is involved  
Only the History attribute is changed  
User is duped thinking he is navigating across different pages.

**\* Explain life Cycle of React JS Component ?**<https://codeburst.io/how-to-use-react-lifecycle-methods-ddc79699b34e> for more about old lifecycle methods  
<https://www.w3schools.com/react/react_lifecycle.asp> for 16.+++

**Which life cycle methods got depreciated in react 16.++**componentWillMount // deprecated  
componentWillRecieveProps handled by **getDerivedStateFromProps**

**why componentwillmount deprecated**many people use **componentWillMount** to asynchronously fetch data from other servers. But asynchronous data fetches won’t return before the component renders, and that means that the component will re-render more than once.

**Mounting**Mounting means putting elements into the DOM.  
React has four built-in methods that gets called, in this order, when mounting a component:  
componentWillMount() is depreciated in 166.++  
1 constructor()  
2 getDerivedStateFromProps()  
3 render()  
4 componentDidMount()

The **render()** method is required and will always be called, the others are optional and will be called if you define them.

The **constructor()** method is called before anything else, when the component is initiated, and it is the natural place to set up the initial state and other initial values.  
The **getDerivedStateFromProps()** This method is used when the state of a component depends on changes of props. getDerivedStateFromProps(props, state) is a static method that is called just before render() method in both mounting and updating phase in React. It takes updated props and the current state as arguments.method is called right before rendering the element(s) in the DOM.

**Static** **getDerivedStateFromProps :** this is a static method because it avoid to make a mistake. This is class method not a instance method. In static method we can not use this keyword.  
The **componentDidMount()** method is called after the component is rendered.

**Updating**

Schedule

The next phase in the lifecycle is when a component is updated.  
A component is updated whenever there is a change in the component's state or props and parent component will render. componentWillReceiveProps() componentWillUpdate() have been depreciated 16.++  
React has five built-in methods that gets called, in this order, when a component is updated:

1. getDerivedStateFromProps()  
2. shouldComponentUpdate()  
3. render()  
4. getSnapshotBeforeUpdate()  
5. componentDidUpdate()

static **getDerivedStateFromProps**(nextProps, prevState){

if(nextProps.someValue!==prevState.someValue){

return { someState: nextProps.someValue};

}

else return null;

}

In the shouldComponentUpdate() method you can return a Boolean value that specifies whether React should continue with the rendering or not.The example below shows what happens when the shouldComponentUpdate() method returns false:

The **componentDidUpdate** method is called after the component is updated in the DOM.

shouldComponentUpdate(nextProps, nextState) {

    return this.state.value != nextState.value; // if previous and next state will not same

  }

**getSnapshotBeforeUpdate(prevProps, prevState)** : is invoked right before the most recently rendered output is committed to e.g. the DOM. It enables your component to capture some information from the DOM (e.g. scroll position) before it is potentially changed. Any value returned by this lifecycle will be passed as a parameter to

|  |
| --- |
| this.listRef = React.createRef();  getSnapshotBeforeUpdate(prevProps, prevState) {  // Are we adding new items to the list?  // Capture the scroll position so we can adjust scroll later.  if (prevProps.list.length < this.props.list.length) {  const list = this.listRef.current;  return list.scrollHeight - list.scrollTop;  }  return null;  }  componentDidUpdate(prevProps, prevState, snapshot) {  // If we have a snapshot value, we've just added new items.  // Adjust scroll so these new items don't push the old ones out of view.  // (snapshot here is the value returned from getSnapshotBeforeUpdate)  if (snapshot !== null) {  const list = this.listRef.current;  list.scrollTop = list.scrollHeight - snapshot;  }  }  render() {  return (  <div ref={this.listRef}>{/\* ...contents... \*/}</div>  );  }  } |

**Unmounting**The next phase in the lifecycle is when a component is removed from the DOM, or unmounting as React likes to call it. React has only one built-in method that gets called when a component is unmounted:  
The componentWillUnmount method is called when the component is about to be removed from the DOM.suppose any component call based on state true or false.

**• componentWillUnmount()**

**What is Babel and Why do we need Babel  
Babel is a JavaScript transpiler that converts edge JavaScript into plain old ES5. that can run in any browser (even the old ones)**  
Browser can understand only JavaScript. Our browsers do not understand most of the syntax and this is where we need Babel’s help. It is responsible for converting the ES5 and ES6 code to browser understandable code, basically backward compatibility.

Babel is a tool that helps you write code in the latest version of JavaScript. When your supported environments don't support certain features natively, Babel will help you compile those features down to a supported version.  
***babel-core***: Well as the name suggests the main engine of babel plugin for its dependents to work.  
***babel-preset-env***: This is the ES5, ES6 supporting part  
***babel-preset-react***: Babel can be used in any framework that needs latest JS syntax support, in our case its “React”, hence this preset.  
***babel-loader***: Consider this as a bridge of communication between Webpack and Babel

**What is webpack**Webpack is a popular module bundling system or Webpack is a bundler. It can handle not only combination and minification of JavaScript and CSS files, but also other assets such as image files (spriting) through the use of plugins.Webpack is a build tool that puts all of your assets, including Javascript, images, fonts, and CSS, in a dependency graph.Webpack is most widely used and an accepted module bundler

**Dead asset elimination.**  
You only build the images and CSS into your dist/ folder that your application actually needs.  
**Easier code splitting.**  
Homepage.js only requires specific CSS files, Webpack could easily build a homepage.css file to greatly reduce initial file size.  
**Stable production deploys.**  
You can't accidentally deploy code with images missing, or outdated styles.

***pack***: The main webpack plugin as an engine for its dependents.  
***webpack-cli***: To access some webpack commands through CLI like starting dev server, creating production build, etc.  
***webpack-dev-server***: A minimal server for client-side development purpose only.  
***html-webpack-plugin***: Will help in creating HTML templates for our application.

**\*\*\*\*\*\*\* Webpack configuration \*\*\*\*\*\*\***

An **entry point** indicates which module webpack should use to begin building out its internal [dependency graph](https://webpack.js.org/concepts/dependency-graph/). webpack will figure out which other modules and libraries that entry point depends on (directly and indirectly).

module.exports = {

entry: './path/to/my/entry/file.js'

};

The **output** property tells webpack where to emit the bundles it creates and how to name these files. It defaults to ./dist/main.js for the main output file and to the ./dist folder for any other generated file.

const path = require('path');

module.exports = {

entry: './path/to/my/entry/file.js',

output: {

path: path.resolve(\_\_dirname, 'dist'),

filename: 'my-first-webpack.bundle.js'

}

};

At a high level, **loaders** have two properties in your webpack configuration:

1. The test property identifies which file or files should be transformed.
2. The use property indicates which loader should be used to do the transforming.

const path = require('path');

module.exports = {

output: {

filename: 'my-first-webpack.bundle.js'

},

module: {

rules: [  
 { test: /\.js$/, use: "babel-loader", exclude: /node\_modules/, },

{ test: /\.txt$/, use: 'raw-loader' }

]

}

};

**Plugins**While loaders are used to transform certain types of modules, plugins can be leveraged to perform a wider range of tasks like bundle optimization, asset management and injection of environment variables.In order to use a plugin, you need to require() it and add it to the plugins array. Most plugins are customizable through options. Since you can use a plugin multiple times in a config for different purposes, you need to create an instance of it by calling it with the new operator.

const HtmlWebpackPlugin = require('html-webpack-plugin'); //installed via npm

const webpack = require('webpack'); //to access built-in plugins

module.exports = {

module: {

rules: [

{ test: /\.txt$/, use: 'raw-loader' }

]

},

plugins: [

new HtmlWebpackPlugin({template: './src/index.html'})

]

};

## Mode

module.exports = {

mode: 'production'

};

**HtmlWebpackPlugin** The HtmlWebpackPlugin simplifies creation of HTML files to serve your webpack bundles.

**DefinePlugin**The DefinePlugin allows you to create global constants which can be configured at compile time. This can be useful for allowing different behavior between development builds and production builds.

new webpack.DefinePlugin({

// Definitions...  
'process.env.NODE\_ENV': JSON.stringify(process.env.NODE\_ENV)

});

**ContextReplacementPlugin**  
Context refers to a require with an expression such as require('./locale/' + name + '.json'). When encountering such an expression, webpack infers the directory ('./locale/') and a regular expression (/^.\*\.json$/). Since the name is not known at compile time, webpack includes every file as module in the bundle.

**Source map is what**Since webpack bundles the code, source maps are mandatory to get a reference to the original file that raised an error. For example, if you bundle three source files (a.js, b.js, and c.js) into one bundle (bundle.js) and one of the source files contains an error, the stack trace will simply point to bundle.js

module.exports = {

    devtool: 'inline-source-map',

    // ... the rest of the config

  };

**What is different between webpack, grunt, gulf and Browserify**Grunt and Gulp are only tools for working with files, and have no concept of a dependency graph.  
Browserify is mainly a tool to transform require() calls that work in Node.js into calls that work in the browser. It's a dependency graph for your source code only. Plugins like Parcelify can manage some static assets, but you have go to out of your way to make it work.

**how to call prod configuration and dev configuration file in react**use package.json file in script section –config webpack.config name

**How can I use multiple script in package.json**Install npm module .. concurrently. "dev": "concurrently \"npm run client\" \"npm run server\""

**how can create two build in webpack**create two different webpack config file with different output directory. Then run with concurrently for run multiple build

**dependency and devdependcy**devdependency package is used at time of development but dependency package is used at time of run time of application  
  
**Difference between production and development build in ReactJS**The **development** build is used - as the name suggests - for development reasons. You have Source Maps, debugging and often times hot reloading ability in those builds.

The **production** build, on the other hand, runs in production mode which means this is the code running on your client's machine. The production build runs uglify and builds your source files into one or multiple minimized files. It also extracts CSS and images and of course any other sources you're loading with Webpack. There's also no source maps or hot reloading included.

How to configuration uglifyJsPlugin in prod evn.

new webpack.optimize.UglifyJsPlugin({

  compress:{

    screw\_ie8:true,

    warning:false,

  },

  output:{

    comments:false,

  },

  sourceMap:true

})

**\* Middleware redux thunk**Redux Thunk [middleware](https://redux.js.org/advanced/middleware) allows you to write action creators that return a function instead of an action. Redux itself is synchronous, so how the async operations like network request work with Redux? Here middlewares come handy. how much time it is taking or logging the state of the app before and after the action is dispatched. In this case, Redux middleware function provides a medium to interact with dispatched action before they reach the reducer

**What is the difference between Redux Thunk and Redux Saga?**Both Redux Thunk and Redux Saga take care of dealing with side effects. In very simple terms, applied to the most common scenario (async functions, specifically AJAX calls) Thunk allows Promises" to deal with them, Saga uses Generators. Thunk is simple to use and Promises are familiar to many developers, Saga/Generators are more powerful but you will need to learn them. When Promises are just good enough, so is Thunk, when you deal with more complex cases on a regular basis, Saga gives you better tools.

**Explain Flux.**

Flux is an architectural pattern which enforces the uni-directional data flow. It controls derived data and enables communication between multiple components using a central Store which has authority for all data. Any update in data throughout the application must occur here only. Flux provides stability to the application and reduces run-time errors.

**What are the three principles that Redux follows?**

**Single source of truth**: The state of the entire application is stored in an object/ state tree within a single store. The single state tree makes it easier to keep track of changes over time and debug or inspect the application.

**State is read-only**: The only way to change the state is to trigger an action. An action is a plain JS object describing the change. Just like state is the minimal representation of data, the action is the minimal representation of the change to that data. **Changes are made with pure functions**: In order to specify how the state tree is transformed by actions, you need pure functions. Pure functions are those whose return value depends solely on the values of their arguments.

**List down the components of Redux.**

1. Redux is composed of the following components: Redux is a predictable state container for JavaScript apps. Redux is a state management tool for JavaScript applications.It helps you write applications that behave consistently, run in different environments (client, server, and native), and are easy to test.

we use in react because react data flow top to bottom so this helps us to call props from one central place and update them

**Actions :**  are payloads of information that send data from your application to your store. They are the only source of information for the store. You send them to the store using [store.dispatch()](https://redux.js.org/api/store#dispatchaction). It’s an object that describes what happened. An action is a plain JS object describing the change.

const receiveProduct = product => ({ type: t.PRODUCT\_RECEIVE, product });  
export const fetchProducts = currentPage => async (dispatch, getState) => {  
 dispatch(receiveProducts(products));  
}  
  
An **action creator** is a function that returns an action, like:

**Reducers** : specify how the application's state changes in response to [actions](https://redux.js.org/basics/actions) sent to the store. Remember that actions only describe what happened, but don't describe how the application's state changes.It is a place to determine how the state will change.

const appReducer = (state = initialState, action) => {  
switch (action.type) {  
 case t.PRODUCT\_RECEIVE:  
 return Object.assign({}, state, {  
 productDetails: action.product,  
 recommendationProducts: []  
});

**Store** – State/ Object tree of the entire application is saved in the Store.

* Holds application state;
* Allows access to state via [getState()](https://redux.js.org/api/store#getState);
* Allows state to be updated via [dispatch(action)](https://redux.js.org/api/store#dispatchaction);
* Registers listeners via [subscribe(listener)](https://redux.js.org/api/store#subscribelistener);
* Handles unregistering of listeners via the function returned by [subscribe(listener)](https://redux.js.org/api/store#subscribelistener).

It's important to note that you'll only have a single store in a Redux application. When you want to split your data handling logic, you'll use [reducer composition](https://redux.js.org/basics/reducers#splitting-reducers) instead of many stores.

It's easy to create a store if you have a reducer. In the [previous section](https://redux.js.org/basics/reducers), we used [combineReducers()](https://redux.js.org/api/combinereducers) to combine several reducers into one. We will now import it, and pass it to [createStore()](https://redux.js.org/api/createstore).

import { createStore } from 'redux'  
const store = createStore(todoApp, window.STATE\_FROM\_SERVER)  
const store = createStore(todoApp, applyMiddleWare(thunk))

**View** – Simply displays the data provided by the Store.

**How is Redux different from Flux?**

**Flux**1. The Store contains state and change logic  
2. There are multiple stores  
3. All the stores are disconnected and flat  
4. Has singleton dispatcher  
5. React components subscribe to the store  
6. State is mutable

**Redux**1. Store and change logic are separate  
2. There is only one store  
3. Single store with hierarchical reducers  
4. No concept of dispatcher  
5. Container components utilize connect  
6. State is immutable

**What is different b/w React ES6 and ES5 ?**

* ES5 = require vs ES6 = mport
* ES5 exports vs ES6 export
* **component and function**
* [Arrow Functions](https://codetower.github.io/es6-features/#Arrow%20Functions)
* [Classes](https://codetower.github.io/es6-features/#Classes)
* [Destructuring](https://codetower.github.io/es6-features/#Destructuring)
* [Spread](https://codetower.github.io/es6-features/#Spread)
* [Let](https://codetower.github.io/es6-features/#Let)
* [Const](https://codetower.github.io/es6-features/#Const)

// ES5

var MyComponent = React.createClass({  
render: function() {

return <h3>Hello Edureka!</h3>;

}

});

// ES6

class MyComponent extends React.Component {

render() {

return <h3>Hello Edureka!</h3>;

}

}

* **props**

// ES5

var App = React.createClass({

propTypes: { name: React.PropTypes.string },

render: function() {

return <h3>Hello, {this.props.name}!</h3>;

}

});

// ES6

class App extends React.Component {

render() {

return <h3>Hello, {this.props.name}!</h3>;

}

}

**Does ReactJS use HTML?**No, It uses JSX which is simiar to XML.  
  
**What is difference b/w Class Component or Functional Component**?  
If you use state or a lifecycle methods of react component that is class component. If you don’t use state or lifecycle methods that is Functional component.

**What’s the difference b/w an Element and a Component in React?**Basically, a React component describes what you need to see on the screen. Not all that basically, a React element is a protest portrayal of some UI.  
**\* A React Component is a template. A blueprint. A global definition. This can be either a function or a class (with a render function).**

**\* A React Element is what gets returned from components. It’s an object that virtually describes the DOM nodes that a component represents.**

**What is the difference between createElement and cloneElement?**createElement is the thing that JSX gets transpiled to and is the thing that React uses to make React Elements (protest representations of some UI). cloneElement is utilized as a part of request to clone a component and pass it new props.

**36. What is arrow function in React? How is it used?**Arrow functions are more of brief syntax for writing the function expression. That is also called ‘fat arrow‘ (=>) the functions.  
**First**, in the arrow function, the [this](https://www.javascripttutorial.net/javascript-this/), arguments, super, [new.target](https://www.javascripttutorial.net/es6/javascript-new-target/) are lexical. It means that the arrow function uses these variables (or constructs) from the enclosing lexical scope.  
**Second**, an arrow function cannot be used as a function constructor. If you use the new keyword to create a new object from an arrow function, you will get an error.  
an arrow function does not have its own this, it takes this from the outer function where it is defined.

These functions allow to bind the context of the components properly since in ES6 auto binding is not available by default. Arrow functions are mostly useful while working with the higher order functions.

function Car(speed) {

    this.speed = speed;

//console.log(this); // Car {speed: 50}

  setTimeout(function() {

            console.log(this.speed); // undefined

//console.log(this); // Window {parent: global, opener: null, top: global,..}

        }, 1000);

}

new Car(50);

Note: above result is undefined. Because Inside the anonymous function of the **setTimeOut**() “ this “ its own reference.it try to find speed inside speed in own “this”

function Car(speed) {

    this.speed = speed;

//console.log(this); // Car {speed: 50}

  setTimeout(() => {

            console.log(this.speed); // 50

//console.log(this); // Car {speed: 50}

        }, 1000);

}

new Car(50);

in this arrow function have not own this. It calls by parent of car function.so it get the value 50. The value of “this” will be same in car function and setTimeout. But above function it not same.

**what is Yield keyword used ?**The yield keyword is used to pause and resume a generator function. That is known as yield keyword. This is a example of lexical use of this keyword. They automatic bind live value of parent to parent function value.we can not use like this in common function definition.  
  
**What do you understand from “In React, everything is a component.”**Components are the building blocks of a Reac t application’s UI. These components split up the entire UI into small independent and reusable pieces. Then it renders each of these components independent of each other without affecting the rest of the UI.

**Explain 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>,<div> etc  
**How do you modularize code in React?**We can modularize code by using the export and import properties. They help in writing the components separately in different files.

**What is server side rendering and client side rendering**

**SSR** : the server compiled everything, included the data, and delivered a fully populated HTML page to the client. It was fast and effective.But… every time you navigated to another route, the server had to do the work all over again  
CLIENT SIDE: In Client-side rendering, your browser downloads a minimal HTML page. It renders the JavaScript and fills the content into it.   
hydrate()

Same as [render()](https://reactjs.org/docs/react-dom.html#render), but is used to hydrate a container whose HTML contents were rendered by [ReactDOMServer](https://reactjs.org/docs/react-dom-server.html). React will attempt to attach event listeners to the existing markup.

**react-router Redirect vs history.push**Rendering a <Redirect> will navigate to a new location. The new location will override the current location in the history stackhisroty.push // Pushes a new entry onto the history stack

**why use bind in react**  
make sure that attribute this.state and this.props is available inside the function. There for bind function inside the const  
  
**Composing Components**

components can refer to other components in their output.

function App() {

  return (

    <div>

      <Welcome name="Sara" />

      <Welcome name="Cahal" />

      <Welcome name="Edite" />

    </div>

  );

**Refs** The ref is used to return a reference to the element. Refs should be avoided in most cases, however, they can be useful when we need DOM measurements or to add methods to the components.