1. React functional component lifecycle: mounting > updating > unmounting
   1. Only called when component is mounted:   
      useEffect(() {  
      // logic  
      }, [])
   2. Only called when component is mounted and updated:   
      useEffect(() {  
      // logic  
      })
   3. Called when component is mounted and unmounted(return statement):   
      useEffect(() {  
      // logic  
      return() => {  
      // logic  
      }  
      }, [])
2. **Controlled vs uncontrolled components**
   1. A [Controlled Component](https://facebook.github.io/react/docs/forms.html#controlled-components) is one that takes its current value through props and notifies changes through callbacks like onChange. A parent component "controls" it by handling the callback and managing its own state and passing the new values as props to the controlled component. You could also call this a "dumb component".
   2. A [Uncontrolled Component](https://facebook.github.io/react/docs/uncontrolled-components.html) is one that stores its own state internally, and you query the DOM using a ref to find its current value when you need it. This is a bit more like traditional HTML.
3. **Props vs state**
   1. **Props** get passed to the component similar to function parameters
   2. **State** is managed within the component similar to variables declared within a function
4. **Phases of reactJS lifecycle**
   1. **Initialization:** In this phase react component prepares setting up the initial state and default props.
   2. **Mounting:** The react component is ready to mount in the browser DOM. This phase covers **componentWillMount** and **componentDidMount** lifecycle methods.
   3. **Updating:** In this phase, the component get updated in two ways, sending the new props and updating the state. This phase covers **shouldComponentUpdate, componentWillUpdate and componentDidUpdate** lifecycle methods.
   4. **Unmounting:** In this last phase, the component is not needed and get unmounted from the browser DOM. This phase include **componentWillUnmount** lifecycle method.
5. **Lifecycle methods of react (Class based Components)**
   1. **componentWillMount:** Executed before rendering and is used for App level configuration in your root component.
   2. **componentDidMount:** Executed after first rendering and here all AJAX requests, DOM or state updates, and set up eventListeners should occur.
   3. **componentWillReceiveProps:** Executed when particular prop updates to trigger state transitions.
   4. **shouldComponentUpdate:** Determines if the component will be updated or not. By default it returns true. If you are sure that the component doesn't need to render after state or props are updated, you can return false value. It is a great place to improve performance as it allows you to prevent a rerender if component receives new prop.
   5. **componentWillUpdate:** Executed before re-rendering the component when there are pros & state changes confirmed by shouldComponentUpdate which returns true.
   6. **componentDidUpdate:** Mostly it is used to update the DOM in response to prop or state changes. Takes in two input parameters (**prevProps**, **prevState**)
   7. **componentWillUnmount:** It will be used to cancel any outgoing network requests, or remove all event listeners associated with the component.
6. DOM, Virtual DOM, Shadow DOM
   1. DOM: API for HTML and XML documents. Creates a logical structure which can be accessed and manipulated by browsers
   2. Virtual DOM: Creates copy of entire DOM and stores in memory. On change of component, it updates Virtual DOM. Virtual DOM is compared with DOM, if any changes they are reflected in DOM. To see in action > Go to Console > click 3 dots > Rendering > **Paint Flashing**
   3. Shadow DOM: Creates small pieces of DOM which has their own isolated scope for the element they represent. Eg: video tag is html. DOM doesn’t shows volume or play/pause button
7. When component renders
   1. When the state changes of component
   2. When the value of props which the component receives changes
   3. When parent re-renders, by default the child also renders
8. Memoization
   1. Speed optimization technique where given a functions, it returns the cached version of output for same inputs
   2. It remembers output for the given set of inputs
   3. It does a shallow comparison (reference check for non-primitive(object, arrays or functions) that memory location is still same or not) of props, if value changed, it re-renders
9. useMemo, memo and useCallback
   1. React.memo – similar to React.purecomponent() for class based components. It will compare all props passed to the component by referential equality. If these props are unchanged, React.memo will reuse the last rendered result, therefore, it prevents the component from being re-rendered.
   2. useMemo and useCallback: Both React.useMemo and React.useCallback receives a function as its first argument and a dependencies array as the second one. The hook will return a new value only when one of the dependencies value changes (referential equality). The main difference is that React.useMemo will call the function which is received and return its result while React.useCallback will return the received function without calling it.
   3. useMemo – memorizes the result of the function that is called. use React.useMemo when we compute expensive value that we don't want to compute it again and again when the component is rerendered
   4. useCallback – memorizes a function which can later be called
10. React vs Node Js: <https://www.geeksforgeeks.org/difference-between-node-js-and-react-js/>
11. Redux vs Context API : <https://dev.to/ruppysuppy/redux-vs-context-api-when-to-use-them-4k3p>
12. React Hooks :
    1. useState - useState is a Hook that lets you add React state to function components
    2. useEffect – single hook to take care of **componentDidMount and componentDidUpdate. R** **uns both after the first render and after every update**
    3. useReducer - To help separate the concerns (rendering and state management) React provides the hook useReducer(). The hook does so by extracting the state management out of the component.
    4. useRef -
    5. createContext – useContext  
         
       <https://youtu.be/o-alRbk_zP0>
    6. useCallback - Pass an inline callback and an array of dependencies. useCallback will return a memoized version of the callback that only changes if one of the dependencies has changed. This is useful when passing callbacks to optimized child components that rely on reference equality to prevent unnecessary renders
    7. useMemo - useMemo will only recompute the memoized value when one of the dependencies has changed. This optimization helps to avoid expensive calculations on every render.
    8. React.memo() - it does a shallow comparison. It works good with strings and numbers. Doesn’t works when passed in objects or functions (useCallback/useMemo helps) . It will compare all props passed to the component by referential equality. If these props are unchanged, React.memo will reuse the last rendered result, therefore, it prevents the component from being re-rendered
13. Stopwatch in react
14. List in react - <https://www.javatpoint.com/react-lists>
15. Dynamic form - <https://www.freecodecamp.org/news/build-dynamic-forms-in-react/>
16. Higher order Component
17. React Routers – Browser, Memory, Hash (<https://learnwithparam.com/blog/different-types-of-router-in-react-router/>) <https://codesandbox.io/s/objective-nash-illdyn?file=/src/App.js>
18. React Pure Component – A React component is said to be pure if it renders the same output for the same state and props. For React **pure class components**, React provides the **PureComponent** base class. Class components that extend the React.PureComponent classes are treated as pure components. To create a pure functional component in React, React provides a React.memo() API. Using the React.memo() API, the React functional component can be wrapped as follows to get React Pure Functional Component.
19. <React.StrictMode/> -
    1. Identifying components with unsafe lifecycles - when we are still using class based components and not using the lifecycle methods in correct order
    2. Warning about legacy string ref API usage
    3. Warning about deprecated findDOMNode usage
    4. Detecting unexpected side effects
    5. Detecting legacy context API
20. React Custom Hooks – Javascript function which starts with “use” keyword and can make use of inbuilt hooks. It is used for sharing the code.
21. Reconciliation: Reconciliation is the process through which React updates the Browser DOM. It uses virtual DOM approach. Refer to point 6
22. **Babel** - Babel is a transpiler i.e. it converts the JSX to vanilla JavaScript. You can view babel as an intermediate step between your code and "executable" code.
23. **Prop drilling** –
24. **React Flux Architecture - https://www.freecodecamp.org/news/how-to-use-flux-in-react-example/**
25. **State lifting**
26. **Cache headers**
    1. **Expires – Date Obj**
    2. **Cache-control – multi value header (**
       1. **private (only in browser cache) ,**
       2. **public (browser as well as proxy server cache),**
       3. **no-store (no cache stored, always fetch from server),**
       4. **no cache (can be cached, but to reuse, it needs to be revalidated from server),**
       5. **max-age (cached only for a particular time) )**
27. **Increase performance of a website:**
    1. Caching & Content Delivery Networks
    2. Minimize HTTP Requests
    3. Reduce Redirects
    4. Enable compression of files – html, css, javascript
    5. Optimise Database - Get rid of all these garbage data and useless content
    6. Keep scripts at the bottom
    7. Optimize images
    8. Reduce number of custom fonts
28. Service Worker (https://www.youtube.com/watch?v=ksXwaWHCW6k)
    1. JS script which is registered with browser
    2. Virtual proxy between the browser and the network
    3. Stays registered even when there is no internet connection
    4. Load content even if offline
    5. They don’t have access to the DOM
    6. Works only in HTTPs
    7. Eg: Push notifications and notification API
    8. Register a service broker with navigator.servicebroker.register (it is a promise)
    9. Major event listeners:
       1. **Install** - In the install listener, we can initialize the cache and add files to it for offline use
       2. **Activate** - This ensures we have only the files we need in the cache, so we don't leave any garbage behind; the available cache space in the browser is limited, so it is a good idea to clean up after ourselves.
       3. **Fetch -** fires every time an HTTP request is fired off from our app. Here, we respond to the fetch event with a function that tries to find the resource in the cache and return the response if it's there. If not, we use another fetch request to fetch it from the network, then store the response in the cache so it will be available there next time it is requested.
29. Infinite Scrolling (Used to more user interactive applications – Facebook, instagram)
    1. Pros
       1. More user engagement -
       2. Better User experience – especially for mobile devices
       3. Faster than clicking – less number of user clicks
    2. Cons
       1. More loading speed
       2. Unable to bookmark
       3. No footers
30. Pagination (Used for e-commerce applications – Amazon, flipkart, aliexpress)
    1. Pros
       1. More user control
       2. Scanning of item is much better than infinite scroll
    2. Cons
       1. More user clicks
       2. Limited content
31. Lazy loading – identify resource which are non-critical and load only when needed. Load only the critical resource which loads when the page is visited for the first time
    1. Lazily load images and videos
32. Event bubbling – Event propagation happens in bottom-up approach. The process in which the lowest element is executed first and it goes upto the body.
33. Event capturing/event trickling – Event propagation happens in top-down approach. reverse of event bubbling. This is the first phase
34. Event delegation - instead of assigning a handler to each of them – we put a single handler on their common ancestor
35. Webpack, webpack-dev-server
36. PWA – Progressive Web Apps (https://web.dev/pwa-checklist/)
    1. Completely HTML5 based website meant for offline support which supports in all browsers
    2. Make use of service workers for cache support and interact with server
    3. Use web workers for background processing
    4. Low file size compared to application
37. Redux Thunk
38. Redux Saga
39. Check for cross browser issues
    1. Check for javascript functions which are compatible with older browsers and different versions
    2. Check online tools like – caniuse
    3. Use transpiling using babel
    4. Test the functionality on real devices
    5. Use polyfills
    6. Use respective browser debugger tools for further analysis
40. Micro frontend - <https://single-spa.js.org/docs/microfrontends-concept>
    1. Easier to test
    2. Easier to develop and deploy – separate CI/CD
    3. No shared code
    4. Can change architecture of one without touching another
    5. Implement micro-frontend
       1. <https://medium.com/bb-tutorials-and-thoughts/how-to-implement-micro-frontend-architecture-with-react-5ab172a0fec7>
41. Cache vs Cookie –
    1. Cookie is used to store information to track different characteristics related to user, while cache is used to make the loading of web pages faster.
    2. Cookies stores information such as user preferences, while cache will keep resource files such as audio, video or flash files.
    3. cookies expire after some time, but cache is kept in the client’s machine until they are removed manually by the user.
42. Synthetic Events
    1. A wrapper which registers all the different names of same event
    2. It gives cross browser functionality and makes sure that react applications are working across browsers
    3. React uses the event object to improve the performance
43. Semantic elements
    1. Tags which clearly tell the meaning – form, table, article, header, footer etc
    2. Straightforward to understand and modify
    3. Improves accessibility
44. Block, inline, inline-block
    1. Block – take the entire width of the page , always start in a new line. Eg - Div
    2. Inline – only occupy the width of the content, starts in the same line. It doesn’t takes height and width properties. Eg – span tag
    3. Inline-block - use inline elements and adjust the height and width
45. Progressive Hydration
46. ReactDOM.renderToString – Render bunch of components one at a time and produces string as a resulting HTML
47. ReactDOM.hydrate
48. oAuth 2.0
49. React callback way - <https://codesandbox.io/s/quirky-elion-ik9ifp?file=/src/App.js>
50. web worker – offload the work in a separate thread
    1. used to compute expensive computations
    2. helps to free up the main thread
51. StopWatch - <https://codesandbox.io/s/modest-sea-3ddc92?file=/src/App.js>
52. React Lazy - <https://codesandbox.io/s/react-lazy-gumcwg?file=/src/index.js>
53. Toast Message - <https://codesandbox.io/s/cranky-browser-xclovw?file=/src/App.js:825-1157>
54. Why is key needed when iterating over list and creating components:  
    Keys help react identify which elements have changed, have been deleted or added. It gives the elements in the array a stable identity
55. Preload vs Prefetch
    1. **Preloaded** chunks will be loaded with higher priority in parallel to its parent chunk. Mark chunks to be preloaded only if you are confident that the user will interact with them immediately. This can be a dropdown or the contents of a tab.
    2. **Prefetched** chunks have lower priority and will be loaded in the browser’s idle time. In other words, mark chunks to be prefetched if the user may need them at some point. This can be the next page he is most likely to visit. He won’t request it immediately but you want it to be there when he does.
56. Code Splitting - <https://hackernoon.com/lessons-learned-code-splitting-with-webpack-and-react-f012a989113>
57. React createElement - <https://codesandbox.io/s/sleepy-fire-il0gv0?file=/src/App.js>
58. React questions: <https://www.freecodecamp.org/news/react-interview-questions-to-know/>
59. **React vs ReactDOM**: As the name implies, ReactDOM is the glue between React and the DOM. For everything else, there’s React. You use React to define and create your elements, for lifecycle hooks, etc. i.e. the guts of a React application.
    1. **React**: a javascript library, designed for building user interfaces
    2. **React-DOM**: a complimentary library to React which glues React to the browser DOM
60. Routing - <https://codesandbox.io/s/ecstatic-panini-3lg7xv?file=/src/App.js>
61. PWA - <https://www.youtube.com/watch?v=IaJqMcOMuDM>
62. React Event Propagation: <https://codesandbox.io/s/intelligent-borg-7qzbxr?file=/src/App.js>
63. Server Side Rendering Example - <https://www.youtube.com/c/BeALearnerofficial/videos>
64. Web worker - <https://codesandbox.io/s/web-worker-of8tk4?file=/src/worker.js>
65. Accordion (Expand-Collapse) - <https://codesandbox.io/s/expand-collapse-mcohji?file=/src/App.js>
66. Carousel - <https://codesandbox.io/s/carousel-z4gwg0?file=/src/styles.css>
67. Calculator: <https://codesandbox.io/s/js-calculator-xn7smz?file=/index.html>
68. Event bubbling/capturing example - <https://jsfiddle.net/nte61bfh/25/>
69. Promise Types - <https://www.youtube.com/watch?v=vQtgi8uElmk>
    1. Promise.all – Resolves all the promises or rejects as soon as any one of them is rejected. It preserves the order in which they are called
    2. Promise.allSettled – Resolves and rejects all the promises and gives data
    3. Promise.any – Whichever promise is resolved first, it returns. If all promises are rejected, it returns aggregate error
    4. Promise.race - Whichever promise is resolved or rejected first, it returns that.
70. Optimization of webpage: <https://web.dev/fast/#introduction>
71. Strict Mode in JS:
    1. Impossible to create accidental global variables
    2. Avoid unexpected assignments (let undefined = 5, let NaN = 20;)
    3. Avoids deleting undeletable properties
    4. Function parameters cannot be same
72. Difference between <script type=module〉and〈script〉
    1. <https://usefulangle.com/post/256/script-type-module-vs-script-javascript#:~:text=The%20%3Cscript%20type%3D%22module,executes%20Javascript%20in%20strict%20mode.&text=A%20normal%20tag%20is,to%20explicitly%20mention%20use%20strict>.
73. JWT(Json Web Token) - <https://jwt.io/introduction/>
74. For vs forEach() vs for/in vs for/of in JavaScript - <https://thecodebarbarian.com/for-vs-for-each-vs-for-in-vs-for-of-in-javascript.html>
75. Object.getOwnPropertyNames() vs Object.keys() - <https://thisthat.dev/object-get-own-property-names-vs-object-keys/>
76. Array Like Objects - <https://dev.to/capscode/what-is-array-like-object-in-javascript-3f5m>
77. Pros and cons of ssr - <https://prerender.io/what-is-srr-and-why-do-you-need-to-know/>
78. New keyword in Js for function constructors: Performs 3 things
    1. Creates an empty Js object
    2. **this** point to the empty object
    3. Return the object from that function (basically returns this)
79. Typeof vs instanceof –
80. <https://docs.google.com/document/d/1_y9ZIjRt1r9qptIuE5qGSEdnaEMMUqfr_MFdFecyaeE/edit>

Interview questions links:

1. <https://www.fullstack.cafe/interview-questions/react>

2. <https://javascriptinterviewquestions.com/2020/04/100-javascript-interview-questions-to-crack-any-javascript-inteview.html>

3. <https://www.educba.com/js-interview-questions/>