Definitive JS/React Interview Questions [2023]

IMPORTANT UDEMY COURSES/links
JavaScript Algorithms and Data Structures Masterclass by Colt Steele - link
https://www.geeksforgeeks.org/reactjs-tutorials/

Note - (*) marked questions are asked a lot.
DSA

- **do leetcode top 100 interview questions & top 100 liked questions**
- traverse a tree/nested object with left and right keys in it answer link
- *stack - LIFO (props: size, first, last, node(val, next)) **[methods: push, pop]**
- *Queue - FIFO (props: size, first, last, node(val, next)) [**methods: enqueue, dequeue]**
- *Binary tree - has two or more child nodes at most
- *Binary search tree - a binary tree having some order in nodes
- LRU cache - leetcode question

Ans -

```
class LRUCache {
  constructor(capacity) {
    this.cache = new Map();
    this.capacity = capacity;
  }

  get(key) {
    if (!this.cache.has(key)) return -1;

    const v = this.cache.get(key);
    this.cache.delete(key);
    this.cache.set(key, v);
    return this.cache.get(key);
  };

  put(key, value) {
    if (this.cache.has(key)) {
      this.cache.delete(key);
    }
    this.cache.set(key, value);
    if (this.cache.size > this.capacity) {
      this.cache.delete(this.cache.keys().next().value);  // keys().next().value
returns first item's key
    }
  };
}
```

JAVASCRIPT

- Is let & const hoisted? if so, why reference error?

Ans - Variables declared with let or const are **hoisted WITHOUT a default initialization.** So accessing them before the line they were declared throws ReferenceError: Cannot access 'variable' before initialization.

- can you delete some property from __proto__ or prototype of an object. **NO**
- **how browser works?** [link]
- **before any request is made to server - an OPTIONS request is made to check for CORS or any other issue.**
- *Var hoisting & function hoisting. **Guess output from given code.**

```
console.log(a); //ref error

let a = 12;
console.log(a);

if(true) {
 console.log(a);

    var a = 12;  //syntax error, a already declared above
  function test () {
   console.log(a);
 }

test();
}
```

```
//Example 2

var a = 12;
console.log(a);

if(true) {
 console.log(a); -> a cant be accessed before init, block scoped let

    let a = 12;
  function test () {
   console.log(a);
 }

test();
}
//Example 3

function foo() {
  var x = 1;
}

foo();
console.log(x); // x is not defined. Var is function scoped

//Example 4
function a () {
```

```
var b  = 12;
function b () {
console.log('function');
}
return b();
}
a(); // b is not a function error



//Example 5
const a= [1,2,3];
console.log(a);
const b = a.map((v,index,array ) => v*2); // why 3rd arg arr is used?
```

- *Event Loop
- *Call vs Apply vs bind - call and apply both call the function directly with first arg as this value and second arg as **array** in case of **apply** and rest args in case of **call.** Bind creates a new function and second arg is rest args.
- *FE Design patterns? link
- Polyfill? Implement one.
- prototype chain

Ans - Ref link
Prototype in JS - Each object has a private property which holds a link to another object called its **prototype**. That prototype object has a prototype of its own, and so on until an object is reached with **null** as its prototype. By definition, **null has no prototype, and acts as the final link in this prototype chain**.
!Note! If a **non-primitive** is returned from the **constructor** function(new keyword instantate), that value will become the result of the new expression instead of this props. E.g -

```
function a() {
this.value = 1234;
}
new a(); // outputs - {value: 1234}
but
function a() {
this.value = 1234;
return {}
}
new a(); // outputs - {}
```

- Overriding prototype method or property from **either** function prototype or object __proto__ can be done. The **last override** done by any of it will actually be called. **Overriding done either on object or function prototype gets done for all objects**
- a property or method save in prototype of an object **cannot be deleted ever**
- *Shallow vs deep Copy? Implement deepcopy. **solution link (use WeakMap for creating cache & using objects as keys in it. WeakMap have a weak reference relationship. obj will be free up in the next time the garbage collection mechanism is executed.)**

- ***Deepcopy iterative link**
- *debounce vs throttle? Implement both.
- Service worker VS web worker

Ans - Ref link
Web Workers are a simple means for web content to run scripts in background threads.The worker thread can perform tasks without interfering with the user interface. **workers run in another global context that is different from the current window**. Thus, using the **window** shortcut to get the current global scope (instead of **self**) within a Worker will return an error.
3 types of web workers - Dedicated, shared & service workers
While **Service workers are a type of web workers**. And works only in https env. Works as proxy/interceptor between browser, JS and network.

- Cancel an api call? with abortsignal & abortcontroller
- Behaviour of **this** keyword in function statement(function a(){}) & expressions(const a = () => {}). Diff between function vs const function declaration.

Ans -

```
function doSomething() {}
console.log(doSomething.prototype);
// It does not matter how you declare the function; a
// function in JavaScript will always have a default
// prototype property -- with one exception: an arrow
// function doesn't have a default prototype property:
const doSomethingFromArrowFunction = () => {};
console.log(doSomethingFromArrowFunction.prototype);
```

- promises vs async await
- event.preventDefault vs event.stopPropagation
- *Event delegation, event bubbling & event capturing
- Localstorage vs session storage - give usecase example of both. (session storage - keep form values filled even after reload, has 5mb memory compared to localstorage with 10mb memory
- override console.log function

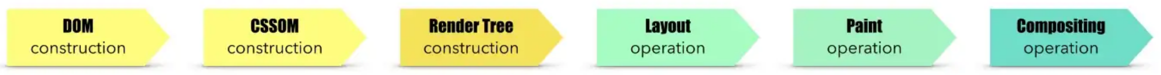
CSS

- *CSS Specificity?
- SASS/SCSS benefits over css?
- Above the fold vs below the fold?
- inline vs inline block - in inline, you can’t set height, width and top bottom padding
- display none vs opacity 0 vs visibility: hidden(same as opacity but button clicks are disabled)
- *Default position of all html elements ? position: static
- which things cant be done in css?
- pseudo selectors(:hover) vs elements(::before)
- How to make a webpage responsive without media queries? Use em, rem, flex & grid

REACT

- How is react or framework better than js?
- *How browser renders a webpage? link1 **link2**

Ans - **Critical rendering path**



Construction phase - first DOMTree is create from html tags, then CSSDom tree from all css sources(inline, external, default) created and finally **Render tree** combining both these trees is created.
Rendering phases -
layout - The first browser creates the layout of each individual Render-Tree node. The layout consists of the size of each node in pixels and where (position) it will be printed on the screen.
paint - Until now we have a list of geometries that need to be printed on the screen. Since elements (or a sub-tree) in the Render-Tree can overlap each other and they can have CSS properties that make them frequently change the look, position, or geometry (such as animations), the browser creates a layer for it.

Creating layers helps the browser efficiently perform painting operations throughout the lifecycle of a web page such as while scrolling or resizing the browser window. Having layers also help the browser correctly draw elements in the stacking order (along the z-axis) as they were intended by the developer.

Now that we have layers, we can combine them and draw them on the screen. But the browser does not draw all the layers in a single go. Each layer is drawn separately first.

compositing - Until now, we haven't drawn a single pixel on the screen. What we have are different layers (bitmap images) that should be drawn on the screen in a specific order. In compositing operations, these layers are sent to GPU to finally draw it on the screen.

Sending entire layers to draw is clearly inefficient because this has to happen every time there is a reflow (layout) or repaint. Hence, a layer is broken down into different tiles which then will be drawn on the screen. You can also visualize these tiles in Chrome's DevTool Rendering panel.

- *Role of **key** attribute in react? What happens on using index
- why super(props)? only for consistency of this.props in constructor, nothing else
- avoid this binding in class based? use arrow functions - no other way
- **componentdidmount or useeffect with [] array gets triggered after the first render**
- *React component life cycle methods - 3 phases - mounting, updating & unmounting then define methods in each of them
- *Virtual dom & how it looks like? link
  Ans - When anything new is added to the application, a virtual DOM is created and it is **represented as a tree**. Each element in the application is a node in this tree. So, whenever there is a change in the state of any element, **a new Virtual DOM tree is created**. This new Virtual DOM tree is then **compared with the previous Virtual DOM** tree and make a note of the changes. After this, it finds the best possible ways to make these changes to the real DOM. Now only the updated elements will get rendered on the page again. This process of comparing the current Virtual DOM tree with the previous one is known as **'diffing'**. Once React finds out what exactly has changed then it updates those objects only, on real DOM. React uses something called **batch updates** to update the real DOM. It just means that the changes to the real DOM are sent in batches instead of sending any update for a single change in the state of a component. We have seen that the re-rendering of the UI is the most

expensive part and React manages to do this most efficiently by ensuring that the Real DOM receives batch updates to re-render the UI. This entire process of transforming changes to the real DOM is called ***Reconciliation.**

- Synthetic event **wrapper around real dom event, done to provide consistent prop names & behaviour across all browsers**
- ssr vs client side render **todo**
- ***Is setState async or async operation?** setState is sync operation under the hood. only thing is that the updated value is received in the next rendering cycle. It “sets” the state instantly but does not also “gets” it instantly for use. React has constantly running re render cycles every 16ms for reconciliation(kind of emulates event loop).
- *nextjs vs react? benefits and features
- *error boundary? **static getDerivedStateFromError() or componentDidCatch(). Use static getDerivedStateFromError() to render a fallback UI after an error has been thrown. Use componentDidCatch() to log error information.** how to add to a component? **link**
- *redux thunk

Ans - The word "thunk" is a programming term that means "a piece of code that does some delayed work". Rather than execute some logic now, we can write a function body or code that can be used to perform the work later.

For Redux specifically, "thunks" are a pattern of writing functions with logic inside that can interact with a Redux store's dispatch and getState methods. Thunks are a standard approach for writing async logic in Redux apps, and are commonly used for data fetching. However, they can be used for a variety of tasks, and can contain both synchronous and asynchronous logic.

Why Use Thunks?

Thunks allow us to write additional Redux-related logic separate from a UI layer. This logic can include side effects, such as async requests or generating random values, as well as logic that requires dispatching multiple actions or access to the Redux store state.

Redux reducers must not contain side effects, but real applications require logic that has side effects. Some of that may live inside components, but some may need to live outside the UI layer. Thunks (and other Redux middleware) give us a place to put those side effects.

It's common to have logic directly in components, such as making an async request in a click handler or a useEffect hook and then processing the results. **However, it's often necessary to move as much of that logic as possible outside the UI layer. This may be done to improve testability of the logic, to keep the UI layer as thin and "presentational" as possible, or to improve code reuse and sharing.**

In a sense, a thunk is a loophole where you can write any code that needs to interact with the Redux store, ahead of time, without needing to know which Redux store will be used. This keeps the logic from being bound to any specific Redux store instance and keeps it reusable.

- *can we have two redux stores in redux. how? link

Ans - As with several other questions, **it is possible to create multiple distinct Redux stores in a page, but the intended pattern is to have only a single store**. Having a single store enables using the Redux DevTools, makes persisting and rehydrating data simpler, and simplifies the subscription logic. **use Provider comp twice and pass diff store.**

- *what is middleware in redux? **middleware functions** run between dispatching an action and the moment it reaches the reducer, applymiddleware can be used in createStore’s second arg to apply middlewares **link**

- **REDUX HAS 3 MAIN PRINCIPLES:**
    - *1. Single Source of Truth: "The entire application's state is stored in an object tree within a single store."*
    - *2. State is read-only: "The only way to change the state is to emit an action that describes what happened.."*
    - *3. Changes are made with pure functions: "You write reducers as pure functions to specify the concrete way the state tree is transformed by action."*
- **webpack interview questions - link | Webpack 5 self notes**
- ***webpack allows us to be free from taking care of order of script tags(rearranging multiple script tags can break the code)**
- componentdidupdate simulation with hooks.

Ans -

```
 const mounted = useRef();
useEffect(() => {
  if (!mounted.current) {
    // do componentDidMount logic
    mounted.current = true;
  } else {
    // do componentDidUpdate logic
  }
});
```

- how to setup basic redux with & without redux toolkit
- context vs redux
- *custom hook implementation. 3 rules of hooks - always top level, should not be in a condition, must start with 'use' prefix
- **usecallback vs usememo link - usememo for storing dynamic object literals, usecallback for caching functions. make sure to pass an empty array as second arg in both even if no dependencies.**
- **React profiler -** enable highlight when re render, debug when components got updated
- *usecallback, useref vs usestate - useref doesnt rerender comp but persists value
- useeffect for component life cycles
- *Microfrontend approach (module federation plugin) link
- *use graphql with and without redux (without redux, use apollo cache)

PERFORMANCE, Security/ BEST PRACTICES

- How do you approach a new design? - component inheritance, card example plus atomic principle file structure
- *ATOMIC principle/file structure
- *PureComponent/shouldcomponentupdate and react memo- give class & functional comp example. Pure component implements shouldcomponentupdate by default by shallow comparison of previous & next props.
- *Core Web vitals, performance

Ans - Largest Contentful Paint threshold recommendationsFirst Input Delay threshold recommendationsCumulative Layout Shift threshold recommendations

Largest Contentful Paint (LCP): measures loading performance. To provide a good user experience, LCP should occur within 2.5 seconds of when the page first starts loading.

First Input Delay (FID): Tells how long it takes for first interaction to complete after starting it. measures interactivity. To provide a good user experience, pages should have a FID of 100 milliseconds or less.
Cumulative Layout Shift (CLS): measures visual stability. To provide a good user experience, pages should maintain a CLS of 0.1. or less.

- How to optimise non performant webpage/ how to make sure application performance is best? IMPORTANT What tools, steps to take? - link - https://developer.chrome.com/docs/devtools/evaluate-performance/
- *Improve web page performance? Production build(minified js, css),core web vitals, react profiling, Chunk split, lighthouse - code optimise opportunity - how to fix those?
- webpack split chunks how to?
- Write a function. Tell its time complexity.
- Performance tab debug memory leak
- How to secure a React app?- no referrer noopener link on target _blank urls, **sanitize url, keep as least values in local as possible.**
- *Virtualization/Windowing in React.(react-window, react-virtualized) **Used when you need to implement infinite scroll or rendering long list of data**.

OTHERS
- *Authentication vs Authorization
- Traditional vs spa - pros and cons? **todo**
- *Test use effect/usestate in jest, react test library?
- what is cdn? - A content delivery network (CDN) refers to a geographically distributed group of servers that work together to provide fast delivery of Internet content.
- ***A/B testing (split testing)** is the process of setting up two versions of one page, and testing which page outperforms the other. A/b testing tools - optimizely, usertesting.com, google optimize
- *Mount and shallow in jest **shallow renders a comp, not its child comps, mount renders full life cycle & its child**
- *Seo mean - what basics to do for seo?
- Accessibility basic checks - semantic html, aria attributes, keyboard tabbable, use axedevtools, make sure directly user accessible pages are prerendered or return html for important stuff by default
- ***Dynamic programming** for solving code JS questions
- What is a **design system**? is a set of standards intended to manage design at scale using reusable components and patterns(style guide - typography, docs of comps in storybook).