**Study Guide**

Q: What are the differences between ‘Let’, ‘Var’ and ‘Const?

**A: "Var" declarations are function-scoped or globally-scoped and can be redeclared and reassigned. "Let" declarations are block-scoped and can be reassigned but not redeclared. "Const" is block-scoped like "let" but cannot be reassigned nor redeclared.**

Q: What is hoisting and why does it exist?  
  
**A: Hoisting is a JavaScript mechanism where variable and function declarations are moved to the top of their containing scope during the compile phase. It's a way for the interpreter to handle variable and function declarations before execution, thus enabling calling functions before their physical placement in code.**

Q: What is Object.prototype & **proto**, and example of getPrototypeOf, setPrototypeOf.  
  
**A: *Object.prototype* is the base prototype that gets assigned when new objects are created. *proto*****is an object's actual prototype. *Object.getPrototypeOf(obj)* gets the prototype (proto) of obj. *Object.setPrototypeOf(obj, prototype)* sets the prototype (proto) of obj to a new object.**

Q: What is the difference between a class and an object?

**A: A class is a blueprint for creating objects. It defines properties and behaviors that each object created from the class will have. An object is an instance of a class - it's a tangible representation that embodies the properties and behaviors defined by the class.**

Q: -What is the 'this' keyword and when do we use it?

**A: The 'this' keyword in JavaScript refers to the object it belongs to. It's used in object methods to refer to the object the method is called upon. It has different values depending on where and how it's used - in a method, 'this' refers to the owner object, in a function, 'this' refers to the global object.**

Q: What is the difference between a normal function and an arrow function?

A:

* **this** Keyword: In a regular function, the value of **this** is set based on how the function is called (could be global, the object calling the function, etc.). In an arrow function, the value of **this** is lexically bound, meaning it's set to the **this** value of the enclosing execution context, or where the arrow function is defined.
* Syntax: Arrow functions have a shorter syntax compared to regular functions. For example, **const square = x => x \* x;** versus **function square(x) { return x \* x; }**.
* Arguments Object: Regular functions have an **arguments** object which allows access to all arguments, regardless of function's declared parameters. Arrow functions do not have an **arguments** object. To access arguments in an arrow function, you'd use rest parameters instead.
* Constructor: Regular functions, if not an ES6 class constructor, can be used as a constructor (i.e., called with the **new** keyword). Arrow functions cannot be used as constructors and will throw an error when used with **new**.
* Method Functions: If a function is likely to be used as a method, a regular function is preferred because it'll have the correct **this** context for the object method call. Arrow functions, with their lexical **this**, could lead to unexpected results if not handled carefully.

Q: What is a promise?

**A: A Promise in JavaScript is an object representing the eventual completion or failure of an asynchronous operation. It returns a single result and is in one of three states: pending, fulfilled, or rejected.**

Q: How do we use promises outside of async/await?

**A: Promises are often used with .then() and .catch() methods to handle fulfilled or rejected states, respectively.**

Q: What does it mean for something to be mutable and immutable? Which types of variables are mutable, and which are immutable?

**A: Mutable objects can be changed after they're created, while immutable objects cannot. In JavaScript, objects and arrays are mutable. Primitive types (Number, String, Boolean, undefined, null, BigInt, Symbol) are immutable.**

Q: What is closure and give an example.  
  
**A: A closure is a function that has access to its own scope, the outer function’s scope, and the global scope. Example:**  
  
**A screen shot of a computer screen

Description automatically generated**

Q: Why are block-scoped variables not available after we’ve exited a function?  
  
**A: Block-scoped variables (declared with let and const) are only accessible within the block they were declared in, not outside of it. Once the block of code has been executed, the block scope is destroyed, and the variables are no longer accessible.**

Q: Everything about Front-end Caching:

**A: Front-end caching involves storing copies of files or data client-side, reducing the need for network requests. This can be accomplished with service workers, the Cache API, HTTP caching headers, and IndexedDB. It helps improve performance and reduce latency.**

Q: What is thedifference between Local Storage and Session Storage, and what is Redis?  
  
**A: Local storage persists data between sessions and has no expiration time, while session storage only persists data for the duration of the page session. Redis is an open-source, in-memory data structure store used as a database, cache, and message broker. It's not a client-side storage mechanism like local or session storage.**

Q: What are cookies and what do we use them for?

**A: Cookies are small pieces of data stored on the user's browser by a website. They're often used for maintaining sessions, tracking user activity, and storing user preferences.**

Q: What is HTTP and what is REST?

**A: HTTP (Hypertext Transfer Protocol) is the protocol used for transmitting hypertext requests and information on the World Wide Web. REST (Representational State Transfer) is an architectural style for creating web services, often used in APIs. REST uses standard HTTP methods and is stateless.**

Q: Which HTTP verbs are there and when are they used?

**A: The primary or most-used HTTP verbs (also known as methods) are GET, POST, PUT, DELETE, and PATCH. GET retrieves data, POST sends data to create a new resource, PUT updates an existing resource, DELETE removes a resource, and PATCH partially updates a resource.**

Q: Explain the life cycle of a react component (when they are mounted, rerendered, unmounted).

**A: In React, a component's lifecycle can be broken down into three main phases: Mounting, Updating, and Unmounting. Mounting is when the component is being created and inserted into the DOM. Updating is when a component is being re-rendered as a result of changes to either its props or state. Unmounting is when the component is being removed from the DOM.**

Q: Explain useEffect.

**A: useEffect is a hook in React that's used to perform side effects in function components. Side effects could be data fetching, subscriptions, or manually changing the DOM. useEffect runs after every render by default, but it can be configured to run only when certain values have changed.**

Q; How do class components differ from functional components (not syntactically, in relation to state management)?

A: In class components, state is managed within the component using **this.state** and is modified with **this.setState()**. In functional components, state is managed using the **useState** hook which returns the current state and a function to update it.

Q: How do we make a react component observable?

**A: In libraries such as MobX, we make a React component observable by wrapping it with an observer function or decorator, which allows the component to react to changes in observable state.**

Q: What are computed values?

**A: Computed values are values that are derived from the state. In libraries like MobX, a computed value is a value that will be derived from the state automatically and cached for future reference.**

Q: What are things to look out for when using MobX and how do we improve its performance?

**A: With MobX, one should be mindful of not making too many observables and unnecessarily causing re-renders. Also, avoid complex computations in computed values or reactions. To improve performance, ensure components observe the smallest set of observables, use computed values for derived state, and use React.memo for functional components where appropriate.**