**Function Declaration vs Function Expression vs Arrow Function**

**Very Important Link**

https://levelup.gitconnected.com/7-differences-between-arrow-functions-and-regular-functions-in-javascript-9152883a839f

**Why we use Arrow function?**

Arrow functions are commonly used in React JavaScript for several reasons:

1. Concise syntax: Arrow functions have a shorter and more concise syntax compared to regular function expressions. They allow you to write functions with fewer lines of code, which can improve readability and reduce boilerplate.

2. Lexical scoping of `this`: Arrow functions do not bind their own `this` value but inherit it from the surrounding context. This behavior is known as lexical scoping, and it can be particularly useful in React components. In regular functions, the value of `this` can change depending on how the function is called, which can lead to confusion and bugs. Arrow functions, on the other hand, ensure that the `this` value refers to the correct context automatically, making it easier to access component properties and methods.

3. No binding of `this`: When using regular functions as callbacks, you often need to bind the function to the component's context explicitly. This binding ensures that the correct `this` value is maintained when the function is called. With arrow functions, there is no need for explicit binding since they automatically capture the `this` value from the surrounding context.

4. Performance benefits: Arrow functions can have slight performance advantages due to their internal handling of the `this` value. Regular functions need to create a new execution context and bind the `this` value each time they are called. Arrow functions, being lexically scoped, avoid this overhead.

5. Avoiding the need for the `bind` method: The `bind` method allows you to explicitly set the `this` value of a function. However, using arrow functions eliminates the need for `bind` in many cases, simplifying your code and making it more readable.

While arrow functions offer these benefits, it's important to note that there are still scenarios where regular functions may be more appropriate, such as when you need to access the `arguments` object or when you want to define methods on a class component. The choice between arrow functions and regular functions ultimately depends on the specific context and requirements of your React application.

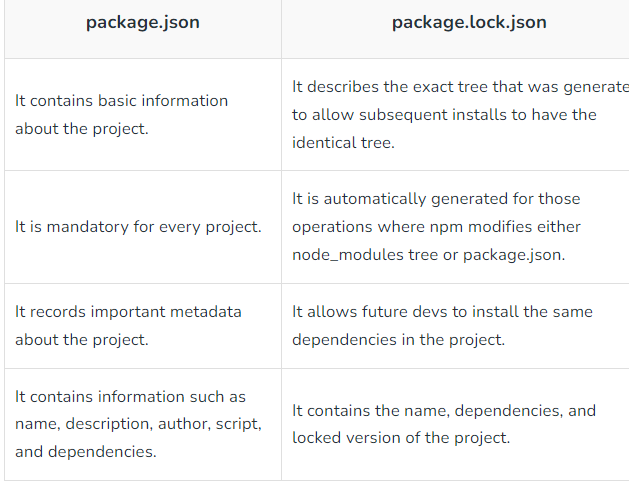
**What is the Alternative of Npm?**

Yarn is the alternative of npm

**Which file is the entry point of react application?**

Index.js is the entry point of react application

**Package.json vs Package.lock.json**



**Why we dont push node modules folder while pushing project to github?**

Ziada folder size ki waja sa.Kionky hum apna project without node modules folder ka kisi ko bhi dain usko sirf npm install likhna ha aur tamam dependencies automatically install ho jaien gi aur nodeModules folder a jay ga usky pas kionky npm install command krty hi package.json file k andr jo jo dependency likhi ha aur uska version wohi wohi wala sab ho jay ga download yani node modules ka folder bhi ban jay ga aur usky andr dependencies ka bhi wohi wohi versions a jaien ga.

The "node\_modules" folder is typically not pushed to GitHub for the following reasons:

1. Large File Size: The "node\_modules" folder contains all the dependencies of a Node.js project, which can be quite large in size. Including this folder in the repository would increase the repository size significantly, making it slower to clone, download, and manage.

2. Redundant Code: The dependencies listed in the "package.json" file are sufficient for others to install the required packages and their respective versions. Including the "node\_modules" folder would essentially be duplicating code that can be easily obtained by running the package installation command (e.g., npm install or yarn) locally.

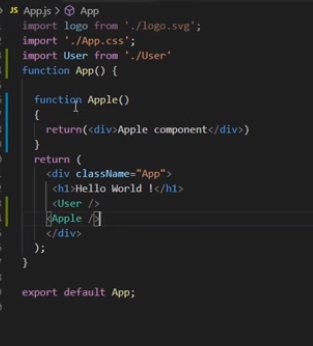
3. Package Manager Compatibility: Different developers may use different package managers (e.g., npm, yarn) or different versions of the same package manager. Including the "node\_modules" folder would not be compatible with all setups and can lead to conflicts or inconsistencies.

4. Version Control Efficiency: The primary purpose of version control systems like Git is to track changes in code over time. Since the content of the "node\_modules" folder is generated and managed by package managers, it does not need to be version controlled as it can be easily reproduced on any machine by running the package installation command.

To ensure proper collaboration and dependency management, it is common practice to include a ".gitignore" file in the project's root directory and specify the "node\_modules" folder in it. This will prevent the "node\_modules" folder from being tracked by Git and pushed to the repository. Instead, developers can include instructions for installing the required dependencies in the project's documentation or README file, allowing others to set up the project with ease.

**Can we made function component insider another functional component?**

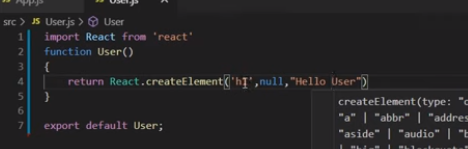
Yes

We can also write {Apple()} instead of <Apple/>

Drawback:Agr hum App() sa bahir Apple() ko kahien call krna chahty hain tu hamain pehly App() ko krna ho ga use then usky andr Apple() ko call.

**Can we use react without jsx?**

Yes,for example



**Can we use react and angular together?**

Yes because react is just a library and we can use it with angular.

**Why we use states instead of variables?**

In React, we use states instead of regular variables for several reasons:

1. Component Re-rendering: React components are designed to be reusable and responsive to changes in data. When a state value changes, React automatically triggers a re-render of the component to reflect the updated state. This ensures that the UI stays in sync with the data. On the other hand, if you use regular variables, changing their values won't trigger a re-render, and the UI won't be updated accordingly.

2. Preserving Data: React state allows you to preserve data across re-renders. When a component re-renders, the state values are retained, allowing you to maintain the previous data without the need for external variables or manual handling.

3. Controlled Updates: With React state, updates are controlled and predictable. When you modify a state value using the `setState` function, React ensures that the component's re-rendering is performed in a synchronized and efficient manner. React batches multiple state updates together, optimizing the performance.

4. React's Virtual DOM: React uses a virtual representation of the DOM to efficiently update the actual DOM when changes occur. By utilizing states, React can compare the previous and new states, determine the minimal changes required, and update only the necessary parts of the UI. This results in better performance compared to manually manipulating the DOM using regular variables.

5. React Component Lifecycle: React provides a set of lifecycle methods that allow you to hook into specific points of a component's life cycle. State changes trigger these methods, enabling you to perform additional logic or side effects when the state is updated.

By using states in React, you leverage React's declarative and efficient approach to UI rendering and data management. It helps you build dynamic and interactive components that respond to user interactions and data changes effectively.

Variables ki values change hony pr hamara component rerender nahi hota pr react aisy design ha ka states aur props ki values change hony pr component rerender hota ha.

**Why we use super() in constructor of Class component?**

In React class components, the `super()` function is used in the constructor to call the constructor of the parent class. When you define a class component that extends the `React.Component` class, you need to call `super()` before accessing `this` in the constructor.

The `super()` function is a reference to the constructor of the parent class (`React.Component`). By calling `super()`, you ensure that the parent class is properly initialized before initializing the child class.

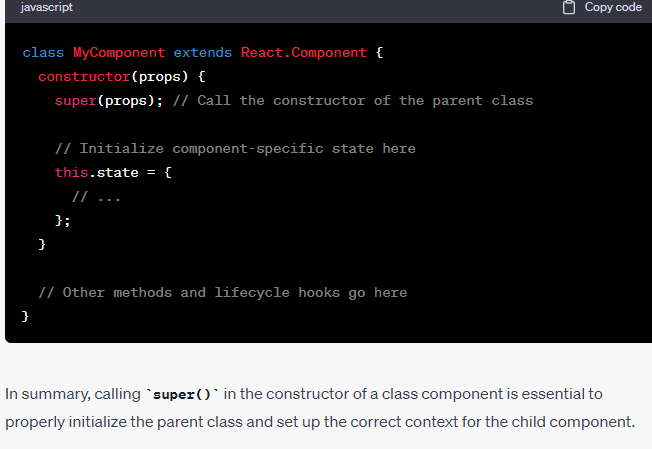
Here are a few reasons why we use `super()` in the constructor of a class component in React:

1. Accessing `this`: Before calling `super()`, the `this` keyword is not accessible in the constructor. It is necessary to call `super()` first to set up the `this` reference correctly, which allows you to access and modify class properties and methods.

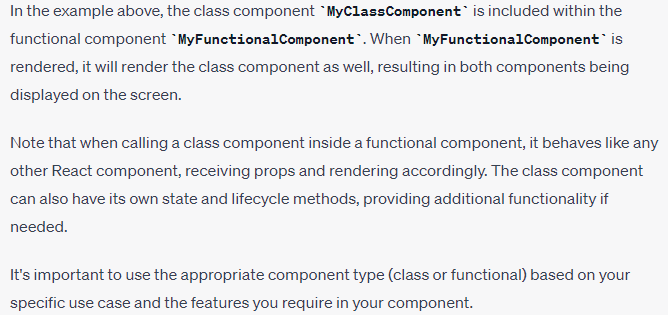
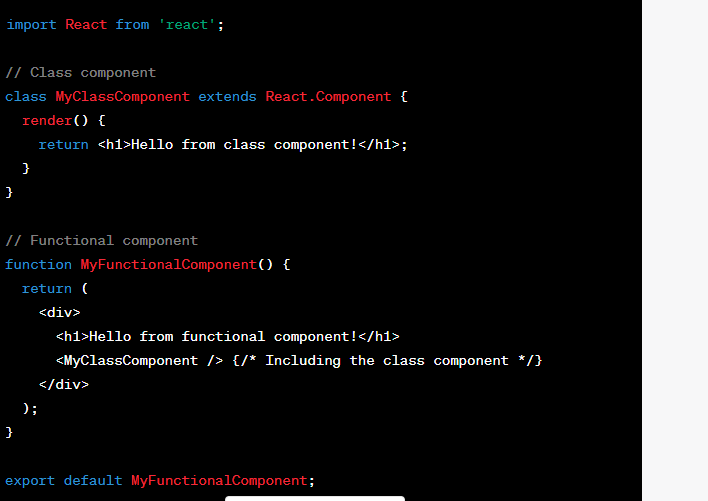
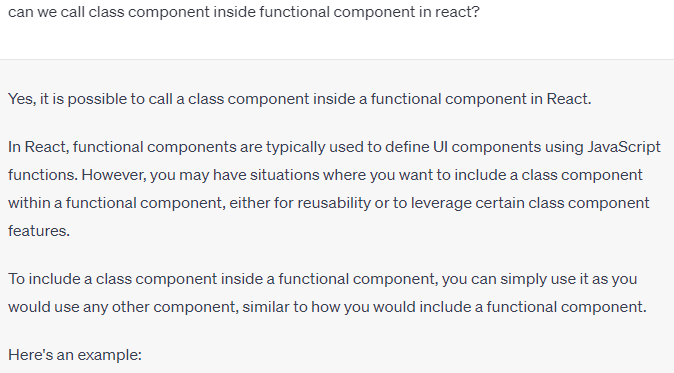
2. Inheriting Behavior: By calling `super()`, the child component inherits the behavior and properties defined in the parent class. It sets up the necessary internal state and functionality required for the component to work as expected.

3. Passing Props: The `super()` function also allows you to pass the `props` object to the parent class constructor. This ensures that the `props` are correctly initialized in the parent class, making them accessible within the child component.

Here's an example of a class component in React that demonstrates the usage of `super()`:



**Can we call class component inside functional component?**



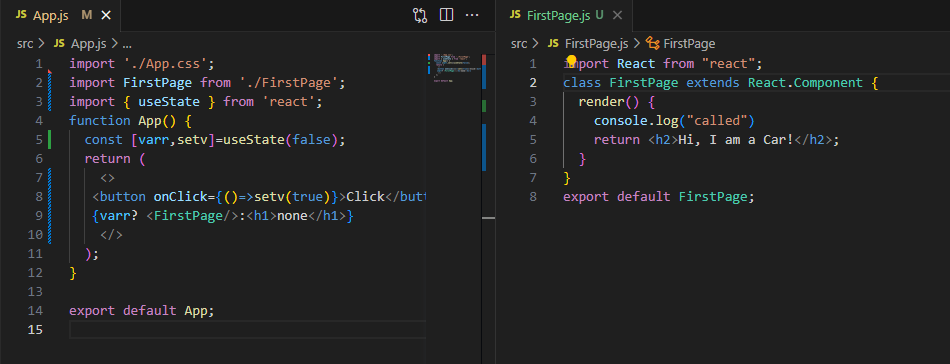
**When render() method will be called?**

In React, the **render()** method is called during the initial rendering of a component and whenever there is a state or prop change that triggers a re-render. The exact timing of when the **render()** method is called can be summarized as follows:

1. Initial Rendering: When a React component is first mounted in the DOM, the **render()** method is called to generate the initial HTML representation of the component. This happens once during the component's lifecycle.
2. State Changes: When the state of a component changes, typically by calling **setState()**, React automatically triggers a re-render of the component. During this re-render, the **render()** method is called again to generate a new HTML representation based on the updated state. React's reconciliation algorithm efficiently updates the DOM to reflect only the necessary changes.
3. Prop Changes: When a component receives new props from its parent component, React compares the new props with the previous props to determine if a re-render is necessary. If the props have changed, React calls the **render()** method to generate a new HTML representation based on the updated props.
4. Force Update: In some cases, you may need to explicitly trigger a re-render of a component even if the state or props haven't changed. You can use the **forceUpdate()** method provided by React to force the component to re-render, which will invoke the **render()** method.

It's important to note that React may optimize the rendering process by batching multiple state updates together or performing updates asynchronously. This means that even if you call **setState()** multiple times in a row, React might batch those updates and call **render()** only once.

In summary, the **render()** method in React is called during the initial rendering of a component and whenever there is a state or prop change that triggers a re-render.

Jab hamara component ban kr ready hota ha tu render() call hota ha. Msln humny 2 files lien ek app.js aur dusri file.js aur file.js ma class component call kia tu app.js ma call kia file.js ka class component tu jaisy hi file.js ka class component ready ho ga render() file.js ka andr waly class component ka call ho ga. 

Jb hum button pr click krain ga tu FirstPage component call ho tu jb ready ho ga tu tb render() method call ho ga.

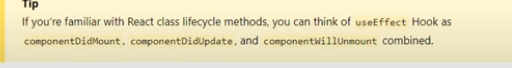
Jab hamara html bhi ready nahi hota us waqt class component ka constructor call hota ha

**shouldComponentUpdate() componentdidupdate() sa pehly call hota ha ya badmy?**

**shouldComponentUpdate() componentdidupdate() sa pehly call hota hai**

**Agr shouldComponentUpdate() true return kry ga tu componentdidupdate() call nahi ho ga vicevera.**

**Class component ka kon konsy method ka barabar akela functional component ka useEffect() ha?**



**Why we need index when we use map() in react?**

i.e Yani react ko pata lgta rehta ka is unique item ko update krna ha baki tamam list ki items ko nahi.

In React, when using the `map()` function to iterate over an array and render a list of elements, it is recommended to include a unique `key` prop for each rendered item. This `key` prop helps React efficiently identify and track each item in the list, especially when the list needs to be updated or re-rendered.

The `key` prop serves two main purposes:

1. Efficient Reconciliation: React uses the `key` prop to perform a process called reconciliation, which is the algorithm that determines the minimal set of changes needed to update the DOM. The `key` prop allows React to differentiate between different items in the list and efficiently update, re-order, or delete specific items without affecting the entire list.

2. Performance Optimization: Assigning a unique `key` to each item in the list helps improve the rendering performance. When React re-renders a list, it can quickly identify the items that have changed or need to be updated based on the `key` prop. Without a unique `key`, React might have to resort to less efficient methods, such as comparing the entire content of each item, to determine if any changes have occurred.

The `key` prop should be a unique identifier for each item in the list. It is recommended to use a stable identifier that remains consistent across re-renders, such as an ID or a unique attribute of the item itself. Avoid using the array index as the `key`, as it can lead to issues when the list is modified, re-ordered, or filtered.

Here's an example of using the `map()` function with the `key` prop in

