













let numbersFirstAlphabetsSecond = (arr) => {

let numbers = [];

let alphabets = [];

let finalArr = [];

numbers = arr.filter((val) => typeof val === "number");

alphabets = arr.filter((val) => typeof val === "string");

finalArr = [...numbers, ...alphabets];

return finalArr;

};

let arr = [2, "b", 4, "d", 3, "a", "c", "e", 5, 1];

console.log(numbersFirstAlphabetsSecond(arr));

**What is useState, useContext, useEffect?**

`useState`, `useContext`, and `useEffect` are all hooks provided by the React library, which is a popular JavaScript library used for building user interfaces. These hooks are essential tools for managing state, side effects, and context within a React component-based architecture.

1. \*\*useState:\*\*

`useState` is a React hook that allows functional components to manage their internal state. Prior to hooks, state management was primarily handled by class components. With `useState`, you can define state variables and their initial values, and React takes care of updating and rendering the component whenever the state changes.

Example usage:

```jsx

import React, { useState } from 'react';

function Counter() {

const [count, setCount] = useState(0);

return (

<div>

<p>Count: {count}</p>

<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

);

}

```

2. \*\*useContext:\*\*

`useContext` is another React hook that provides a way to access the context (global) state within components, without the need for prop drilling. Context is used when you want to share certain state or functionality across many components in your component tree.

Example usage:

```jsx

import React, { useContext } from 'react';

const ThemeContext = React.createContext();

function App() {

return (

<ThemeContext.Provider value="dark">

<Toolbar />

</ThemeContext.Provider>

);

}

function Toolbar() {

const theme = useContext(ThemeContext);

return (

<div>

<p>Current theme: {theme}</p>

</div>

);

}

```

3. \*\*useEffect:\*\*

`useEffect` is a hook used to manage side effects in functional components. Side effects can include data fetching, DOM manipulation, subscriptions, and more. This hook runs after the component renders and can be used to perform actions in response to changes in props or state.

Example usage:

```jsx

import React, { useState, useEffect } from 'react';

function Timer() {

const [seconds, setSeconds] = useState(0);

useEffect(() => {

const intervalId = setInterval(() => {

setSeconds(seconds + 1);

}, 1000);

return () => clearInterval(intervalId); // Clean-up on unmount

}, [seconds]); // Dependency array to control when the effect runs

return (

<div>

<p>Seconds elapsed: {seconds}</p>

</div>

);

}

```

In summary, these hooks are powerful tools in React development for managing component state, accessing context, and handling side effects in a more concise and readable manner compared to traditional class components.