In react applications, particularly when using React Strict mode, console logs often appear twice in the development environment. This happens because React Strict Mode intentionally renders components twice (in development mode only) to help detect potential bugs or side effects in the code.

React does it by:

* Mounting and unmounting the component
* Re-rendering it immediately

If the component contains side effects like console.log it will execute both during the first render and the second render, causing it to log twice.

For example:

import React from 'react';

function App() {

console.log("Component rendered");

return <div>Hello World</div>;

}

export default App;

Output in the console:

Component rendered

Component rendered

How to fix or avoid this:

* Understand it’s a development-only feature: This doesn’t happen in the production build.
* Avoid side effects int eh render phase: Ensure side effects like console.log, API calls, or state changes occur only in proper lifecycle methods like useEffect.

A screenshot of a computer

Description automatically generatedSo this code is showing Tea cup for guest #2 and Tea cup for guest #4. How is it incrementing it by 2?

The behavior you're observing is because of React Strict Mode, which is enabled by default in the development environment. In Strict Mode, React renders components twice during development to help detect side effects or issues in your code, like directly mutating variables outside the component.

1. The guest variable is a global variable initialized to 0.
2. When React renders <Cup />, the function runs, and guest is incremented by 1. However, because Strict Mode re-renders components to check for unintended side effects, the Cup component is rendered again, and guest is incremented a second time.
3. This double-rendering occurs for both <Cup /> components.

A screenshot of a computer program

Description automatically generated

A black rectangular object with a black stripe

Description automatically generated

To avoid this behavior and make the component work correctly, you should not mutate external variables like guest directly. Instead, you can manage state locally using React’s useState:

A computer screen shot of a program code

Description automatically generated

Here, each Cup gets its guest number as a prop from TeaSet, and there's no reliance on a mutable external variable. This makes the code predictable and avoids side effects.

So how come passing props to this component prevents this behavior?

Passing props to a component prevents this behavior because props are immutable and do not rely on external mutable variables, unlike your original implementation, which uses a global variable (guest) that is being directly mutated.

Props are passed from parent to child and are not directly mutated by the child component. When you use props:

<Cup guest={guest} />

* The parent component (TeaSet) generates the value of guest and passes it as a prop.
* The child component (Cup) simply reads the value without modifying it.
* There is no mutable external state beign shared or mutated between renders.