**Pure & Impure Functions & What is a Side Effect**

**Pure Functions:**

1. **Definition**: A function is said to be pure if it always returns the same result given the same arguments, and it has no side effects.
2. **Characteristics**:
   * **Deterministic**: For a given set of input values, a pure function will always produce the same output. It doesn't rely on any external data or state.
   * **No Side Effects**: A pure function doesn't produce side effects. It doesn't modify any external state or data.
3. **Benefits in React**:
   * **Predictability**: Because pure functions are deterministic and have no side effects, they're predictable. This can make your React components easier to test, debug, and understand.
   * **Performance Optimizations**: React can make certain performance optimizations with pure components (like with **React.memo** and **React.PureComponent**). React can skip re-renders when the props are the same, knowing that the output will also be the same.

**Impure Functions:**

1. **Definition**: An impure function is one that does not satisfy the criteria for purity—either it doesn't always produce the same output for the same input, or it has side effects.
2. **Characteristics**:
   * **Non-deterministic**: The function's output might vary for the same input, often due to reliance on external data or state.
   * **Has Side Effects**: The function might modify external state, produce observable data operations, or interact with external systems.

**Side Effects:**

A side effect in programming refers to a situation where a function or operation changes the state outside of its local environment, or has an observable interaction with the external world, other than returning a value. Common examples include:

* Modifying any external variable or object property.
* Logging to the console.
* Writing to or reading from a file.
* Making a network request.
* Modifying the DOM.
* Using timers (like **setTimeout** or **setInterval**).

In the context of React, side effects are operations that reach outside of the local function scope. The **useEffect()** hook, for example, is specifically designed to handle side effects in functional components.

**Conclusion**: Pure functions play a foundational role in functional programming and understanding them can help in designing more predictable and maintainable systems. In React, embracing purity can lead to more predictable components that are easier to test, debug, and reason about. However, real-world applications often need to handle side effects, and React provides mechanisms like **useEffect** to manage them in a controlled and effective manner.