**React-day-05-useEffect Hook-I**

**Why is it called useEffect?**

When the core React Hooks were added to the library in 2018 (useState, useEffect, and so on), many developers were confused by the name of this hook: "useEffect".

What exactly is an "effect"?

The word effect refers to a functional programming term called a **"side effect"**.

But to really understand what a side effect is, we first have to grasp the concept of a **pure function**.

You may not know this, most React components are intended to be pure functions.

It may be strange to think about React components as functions, but they are.

It helps to see that a regular React function component is declared like a JavaScript function:

function MyReactComponent() {}

Most React components are pure functions, meaning they receive an input and produce a predictable output of JSX.

The input to a JavaScript function is arguments. What is the input to a React component, however? **Props**!

Here we have a User component that has the prop name declared on it. Within User, the prop value is displayed in a header element.

export default function App() {

return <User name="John Doe" />

}

function User(props) {

return <h1>{props.name}</h1>; // John Doe

}

This is pure because, given the same input, it will **always** return the same output.

If we pass User a name prop with value "John Doe", our output will always be John Doe.

You might be saying, "Who cares? Why do we even have a name for this?"

Pure functions have the great benefit of being **predictable,** **reliable, and easy to test.**

This is as compared to when we need to perform a side effect in our component.

# ****What are side effects in React?****

Side effects are not predictable because they are actions which are performed with the "outside world."

We perform a side effect when we need to reach outside of our React components to do something. Performing a side effect, however, will not give us a predictable result.

Think about if we were to request data (like blog posts) from a server that has failed and instead of our post data, gives us a 500 status code response.

Virtually all applications rely on side effects to work in one way or another, aside from the simplest applications.

Common side effects include:

* Making a request to an API for data from a backend server
* To interact with browser APIs (that is, to use document or window directly)
* Using unpredictable timing functions like setTimeout or setInterval

This is why useEffect exists: to provide a way to handle performing these side effects in what are otherwise pure React components.

**How do I use useEffect?**

The basic syntax of useEffect is as follows:

// 1. import useEffect

import { useEffect } from 'react';

function MyComponent() {

// 2. call it above the returned JSX

// 3. pass two arguments to it: a function and an array

useEffect(() => {}, []);

// return ...

}