🎬 Episode 10: Closures in JavaScript 🔥

🧠 What is a Closure?

A **closure** is simply a **function bundled together with its lexical scope**.
In JavaScript, we operate within a **lexical scope environment**. 📚

When a function tries to access a variable:

- It first looks into its **own local memory**. 🏠
- If it **doesn't find it**, it moves up to the **memory of its lexical parent**. 👪

Take a look at this example 👇:

```
function x() {
  var a = 7;
  function y() {
    console.log(a);
  }
  return y;
}

var z = x();
console.log(z);  // 👀 value of z is the entire code of function y
```

In the above code:

- `y()` is **returned** from `x()`, but not just the function `y`.
- The **entire closure** (function `y` + its **lexical scope** from `x()`) is returned and stored inside `z`. 📦

☑️ **Important**:
Even after `x()` has finished executing, `z` still **remembers** `var a` because of **closure** magic. ✨

📜 Simple Definition:

> A **closure** is a function that has access to its **outer function's scope** even **after the outer function has returned**!
>
> ☑️ It can **remember and access** variables and arguments of its parent function **even when called elsewhere**.

🌟 Advantages of Closures:

- 🧩 **Module Design Pattern** (organizing code into reusable chunks)
- 🍛 **Currying** (breaking functions into smaller pieces)

- 🧠 **Memoization** (caching computed results)
- 🌓 **Data hiding and encapsulation** (private variables)
- ⏰ **setTimeouts and asynchronous programming**

⚠️ Disadvantages of Closures:

- 🔴 **Over-consumption of memory** (variables stay in memory longer)
- 🐌 **Potential memory leaks** (if not managed well)
- 🧊 **Browser freezes** (in extreme cases with heavy closures)
