🎬 Episode 11: `setTimeouts` + Closures 🔥

Topic: Interview Questions in JavaScript

🕰️ "Time, tide, and JavaScript wait for none."

Let's understand this through an example:

```
function x() {
  var i = 1;
  setTimeout(function() {
    console.log(i);
  }, 3000);
  console.log("Namaste JavaScript");
}
x();
```

🧠 Expected Output:

```
(Naively)
👉 Wait for 3 seconds
👉 Then print 1
👉 Then print "Namaste JavaScript"
```

📜 Actual Output:

```
Namaste JavaScript
1   (after waiting 3 seconds ⏳)
```

❓ Why does this happen?

- The `setTimeout` **callback function** forms a **closure** 🌀 — it remembers the **reference** to `i`, not its **value** at that moment.
- `setTimeout` **attaches a timer** (3000ms) to the callback and **moves on** without waiting.
- Meanwhile, JavaScript **executes the next line immediately**, printing `"Namaste JavaScript"`.
- After 3000ms, JavaScript **pushes the callback function onto the call stack** and then executes it.

🚀 Interview Challenge:

Print 1 after 1 second, 2 after 2 seconds, ..., 5 after 5 seconds

Here's the **initial (buggy)** attempt:

```
function x() {
  for (var i = 1; i <= 5; i++) {
    setTimeout(function() {
      console.log(i);
    }, i * 1000);
  }
  console.log("Namaste JavaScript");
}
x();
```

📜 Output:

```
Namaste JavaScript
6
6
6
6
6
```

❗ Reason behind the output:

- Again, because of **closures** 🌀:
  All the scheduled functions **share the same reference** to `i`.
- When the callbacks are finally executed, the loop is already over, and `i` has become `6`.
- Hence, each timeout prints `6`.

☑️ Solution 1: Use `let` instead of `var` ✨

```
function x() {
  for (let i = 1; i <= 5; i++) {
    setTimeout(function() {
      console.log(i);
    }, i * 1000);
  }
  console.log("Namaste JavaScript");
}
x();
```

🔥 Why it works?

- `let` has **block scope** ✍️.
- Each iteration creates a **new copy** of `i`.
- Every closure remembers its **own individual** `i`.

🤔 But what if the interviewer asks you to solve it using `var` only?

👉 No problem! Here's how:

```
function x() {
  for (var i = 1; i <= 5; i++) {
    (function close(i) {
      setTimeout(function() {
        console.log(i);
      }, i * 1000);
    })(i); // Immediately Invoked Function Expression (IIFE) 🔥
  }
  console.log("Namaste JavaScript");
}
x();
```

📚 Explanation:

- We wrap `setTimeout` inside another function `close(i)`.
- Every time `close(i)` is called, it **creates a new copy** of `i` specific to that closure, even though we are using `var`.

🌟 Key Takeaways:

- `setTimeout` + `closures` are a common trap in interviews! 🎯
- `var` is function-scoped 📃, while `let` is block-scoped 📦.
- Understand closures deeply 🧠 — it's one of JavaScript's most powerful concepts!
