IIFE (Immediately Invoked Function Expressions)

Great! Let's dive deep into **IIFE (Immediately Invoked Function Expressions)** — a handy JavaScript pattern often used to execute code immediately and safely.

⚡ What is an IIFE?

IIFE stands for **Immediately Invoked Function Expression**.

It's a **function that runs as soon as it is defined**.

📌 Syntax of IIFE

```
(function () {
  // code inside
})();
```

OR

```
(() => {
  // arrow function IIFE
})();
```

- It's **wrapped in parentheses** to turn the function declaration into a **function expression**.
- Then it's **immediately invoked** using `()` at the end.

🔧 Example

```
(function () {
  console.log("IIFE executed!");
```

```
})();
```

Output:

```
IIFE executed!
```

✅ Why Use an IIFE?

1. Avoid Polluting the Global Scope

Variables declared inside an IIFE are **not accessible** from the outside.

```
(function () {
  const secret = "This is private";
  console.log(secret); // Works
})();

console.log(secret); // ❌ ReferenceError: secret is not defined
```

> IIFE helps create a private scope.

---

2. Execute Code Immediately

You can use an IIFE to **run setup logic** right away.

```
(function init() {
  console.log("App started!");
})();
```

---

3. Use with Block Scoping Before `let` / `const`

Before `let` and `const`, JavaScript only had **function scope** (not block scope), so IIFEs were used to simulate block scoping.

```
for (var i = 0; i < 3; i++) {
  (function (j) {
    setTimeout(() => {
      console.log(j);
    }, 1000);
  })(i);
}
```

Output:

```
0
1
2
```

> Without the IIFE, all would log 3.

⚙️ Named vs Anonymous IIFE

✅ Anonymous IIFE

```
(function () {
  console.log("Anonymous IIFE");
})();
```

✅ Named IIFE

```
(function myIIFE() {
  console.log("Named IIFE");
})();
```

Note: The name (`myIIFE`) is only accessible **inside** the function itself (for recursion or debugging).

🧠 IIFE with Arrow Functions

```
(() => {
  console.log("Arrow function IIFE!");
})();
```

✅ Cleaner and more concise for simple logic.

💡 Advanced IIFE Use Case — Module Pattern

```
const counter = (function () {
  let count = 0;

  return {
    increment() {
      count++;
      return count;
    },
    decrement() {
      count--;
      return count;
    }
  };
})();

console.log(counter.increment()); // 1
console.log(counter.increment()); // 2
console.log(counter.decrement()); // 1
```

> The IIFE encapsulates the private count variable and exposes only what's needed.

⚠️ Common Mistakes

❌ Missing Parentheses

```
function () { ... }(); // SyntaxError
```

✅ Fix: Wrap it in parentheses

```
(function () { ... })();
```

📝 Summary

Concept	Description
IIFE	Function that runs immediately after it's defined
Purpose	Create private scope, prevent global pollution
Syntax	`(function() {})()` or `(() => {})()`
Use Cases	Module pattern, setup code, async loops, one-time execution

---

Would you like to move on to **Function Currying and Partial Application** next, or go deeper into real-world IIFE use cases?