Core JavaScript Mastery Guide

Perfect! Let's do a **deep dive into IIFE (Immediately Invoked Function Expression)** — a very important concept in JavaScript, often combined with **closures**.

1️⃣ What is an IIFE?

An **IIFE** is a function that:

1. Is **defined**.
2. **Immediately invoked** right after creation.

Syntax:

```
js

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

- Function is **wrapped in parentheses** `(function(){})` to treat it as an **expression**, not a declaration.
- Immediately invoked using `()` at the end.

2️⃣ Basic Example

```
js

(function() {
  let name = "Anoop";
  console.log("Hello " + name);
})();
// Output: Hello Anoop

// name is not accessible outside
console.log(typeof name); // undefined
```

- Variables inside IIFE are **private**, preventing global pollution.

3️⃣ Why Use IIFE?

1. **Avoid polluting global scope**

```
js

(function(){
  let counter = 0; // private
})();
console.log(typeof counter); // undefined
```

2. **Create private variables (closure)**

```
js

const module = (function(){
  let value = 0;
  return {
    increment: function(){ value++; },
    getValue: function(){ return value; }
  };
})();
module.increment();
console.log(module.getValue()); // 1
```

3. **Run code immediately**

```
js

(function(a,b){
  console.log(a+b);
})(5,10); // 15
```

4️⃣ Different IIFE Syntax Variations

(a) Function Expression IIFE

```
js

(function(){
  console.log("Hello World");
})();
```

(b) Arrow Function IIFE

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

(c) With Parameters

```js
(function(x, y){
  console.log(x + y);
})(5, 10); // 15
```

(d) Using Unary Operator (less common)

```js
+function(){
  console.log("Unary IIFE");
}();
```

- `+` or `!` or `~` converts the function to expression → immediately invoked.

5️⃣ IIFE + Closure Pattern (Module Pattern)

```js
const counter = (function(){
  let count = 0; // private
  return {
    increment: function(){ count++; },
    decrement: function(){ count--; },
    get: function(){ return count; }
  };
})();

counter.increment();
console.log(counter.get()); // 1
```

```
counter.decrement();
console.log(counter.get()); // 0
```

- `count` is **protected** inside IIFE.
- This pattern was widely used **before ES6 modules**.

6️⃣ Common Interview Questions / Tricks

1. **Why wrap in parentheses?**

```
js

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

- JS treats `function(){}` alone as **declaration**, which cannot be invoked immediately.
- Wrapping makes it a **function expression**.

2. **IIFE with parameters**

```
js

(function(a,b){ console.log(a+b); })(2,3); // 5
```

3. **IIFE with arrow function**

```
js

(() => console.log("Hi"))(); // Hi
```

4. **IIFE for async code**

```
js

(async function(){
  const data = await fetch("https://api.example.com");
})();
```

5. **Variable isolation**

```
js

var a = 10;
(function(){
  var a = 20;
```

```
  console.log(a); // 20
})();
console.log(a); // 10
```

7️⃣ Key Takeaways

1. **IIFE = Immediately Invoked Function Expression**.
2. Prevents **global variable pollution**.
3. Creates **private scope** → great with closures.
4. Can accept **parameters**.
5. Can be **function expression** or **arrow function**.
6. Foundation for **module pattern** and **encapsulation**.

If you want, next I can **combine IIFE + closures + modules + private variables** with **real-world examples** and **tricky interview patterns**, which often appear in interviews.

Do you want me to do that next?