Day 3 — Functions, Scope & Closures

This document contains all concepts required for Day 3 with explanations and examples. You can edit anything inside this document.

---

1. Function Declarations vs Arrow Functions

Function Declaration

A regular function defined using the `function` keyword.

```
function greet(name) {
  return "Hello, " + name;
}
```

- Function declarations are **hoisted**.
- They have their own `this` depending on how they are called.

Arrow Function

A shorter way to write functions.

```
const greet = (name) => "Hello, " + name;
```

- Arrow functions **do not have their own this**.
- They use lexical `this` (from the parent scope).
- They are not hoisted like normal functions.

---

2. Lexical Scope

Lexical scope means functions access variables based on **where they are defined**, not where they are called.

```
function outer() {
  const msg = "From outer";

  function inner() {
    console.log(msg);
  }

  inner();
}
```

The inner function can access `msg` because it is defined inside `outer()`.

---

3. Closures

A closure happens when a function remembers variables from its outer scope even after the outer function finishes executing.

```
function makeCounter() {
  let count = 0;

  return function() {
    count++;
    return count;
  };
}

const counter = makeCounter();
counter(); // 1
counter(); // 2
```

The inner function remembers `count` because of closure.

---

4. Hands-On Functions

These are the required exercises for Day 3.

4.1 counter()

Creates a counter with private state.

```
function counter() {
  let value = 0;
  return {
    inc: () => ++value,
    dec: () => --value,
    get: () => value
  };
}
```

4.2 once()

Runs a function only once.

```
function once(fn) {
  let called = false;
  let result;

  return function(...args) {
    if (!called) {
      result = fn(...args);
      called = true;
    }
    return result;
  };
}
```

4.3 memoize()

Caches results of function calls for better performance.

```
function memoize(fn) {
  const cache = {};

  return function(...args) {
    const key = JSON.stringify(args);
    if (cache[key]) return cache[key];

    const result = fn(...args);
    cache[key] = result;
    return result;
  };
}
```