lec 2 summary:

🎯 JavaScript Functions and Language Objects

This summary consolidates the concepts from the provided files, focusing on **Functions** (Types, Hoisting, Parameters) and **Language Objects** (Data Types and String).

---

1. ⚙️ Function Types and Concepts

JavaScript supports several ways to define functions, each with unique characteristics regarding hoisting and scope.

Function Types

1. **Function Declaration:**
   - **Syntax:** `function getName(...) { ... }`
   - **Hoisting:** Fully hoisted (the function can be called *before* its definition in the code).
2. **Function Expression:**
   - **Syntax:** `var getName = function(...) { ... };` or `const getName = function(...) { ... };`
   - **Hoisting:** Only the **variable declaration** is hoisted (`var` is initialized to `undefined`), but the function definition is **not**. Calling the function before assignment results in an error.
3. **Arrow Function (ES6):**
   - **Syntax:** `const getName = (...) => { ... };`
   - **Hoisting:** Must be declared as a Function Expression (usually with `const`).
   - **Key Differences:** Does **not** have its own `arguments` array-like object and does **not** bind its own `this` keyword.
4. **Callback Functions & Anonymous Functions:**
   - An **Anonymous Function** is a function without a name (often used as an argument).

- A **Callback Function** is a function passed into another function to be executed later.

5. **IIFE (Immediately Invoked Function Expression):**
    - **Syntax:** `(function() { ... })()`.
    - **Purpose:** Creates a **private scope** for variables (like `count` in the example) to prevent global pollution and achieve basic module-like behavior.

Parameters and Arguments

- **Default Parameters (ES6):** Allow parameters to be initialized with default values if none is provided in the call (e.g., `function getStudentDetails(id=0, name="")`).
- `arguments` **object:** An **array-like object** available inside all traditional functions (declarations and expressions). It contains the values of all arguments passed to the function, regardless of how many parameters are defined. This is why JS accepts many parameters (`sum(1, 2, 5)`).
- **Functions as Variables:** Functions are first-class citizens in JavaScript and can be treated as variables (assigned, passed as arguments, etc.).

---

2. 📚 Data Types and Language Objects

JavaScript data types are split into **Primitive** and **Object** types. **Language Objects** are built-in object wrappers for primitives.

String Objects

- **Primitive String:** `let instrcutorName = "eman ITI instructor";` (`typeof` is `"string"`).
- **String Object:** `let userName = new String("noha ahmed ali");` (`typeof` is `"object"`).
- **Why both exist:** When you call a method on a primitive string (like `instrcutorName.toUpperCase()`), JavaScript performs **automatic wrapping**:
    1. It temporarily creates a `new String(instrcutorName)`.
    2. It calls the method (`.toUpperCase()`).
    3. It destroys the temporary object.

Equality with Objects

When comparing `String` objects, strict rules apply:

Comparison	Example	Result	Reason
Object vs. Object	`userName == clientName`	`false`	Compares **references**. They are two different objects in memory.
Object vs. Primitive	`userName == "value"`	`true`	Coercion occurs. The object is converted to its primitive value for comparison.
Object vs. Primitive (Strict)	`userName === "value"`	`false`	No coercion. They are different types (`object` vs. `string`).

String Task

The utility function `stringSwitchingCase` iterates through a string and checks if each character is equal to its lowercase version. If true, it converts the character to uppercase; otherwise, it converts it to lowercase, effectively switching the case of the input string.

---

3. 🛡️ IIFE for Module Scoping

The **IIFE (Immediately Invoked Function Expression)** pattern is used to create an isolated, private scope.

- In `file1.js` and `file2.js`, the IIFEs ensure that variables declared inside them (like `let count`) do not pollute the global scope.
- **Data Exposure:** `file1Data` is assigned the *return value* of the IIFE (`projectName`, `getInstructorData`), allowing selective data and functions to be exposed outside the private scope, mimicking a simple module structure.