Lecture 1: Intro to JavaScript

Features of JavaScript

- **Lightweight and Interpreted –** Runs directly in the browser, without needing compilation.
- **Cross-Platform –** Works across all major browsers and operating systems
- **Event Driven –** Reacts to user actions, like clicks, and mouse movements
- **Dynamic Typing –** No need to explicitly define data types. It infers them automatically.
- **Prototype Based –** Supports object-oriented programming with prototypes
- **Asynchronous Programming –** Built-in support to handle asynchronous tasks
- **Wide Ecosystem –** Includes libraries, frameworks, and tools that expand its capabilities for front-end and back-end development

Why Use JavaScript?

Widely Supported

Runs in all modern browsers without additional plugins

Interactive User Experience

Adds dynamic content and real-time interactivity to web pages

Versatile

Used for both front-end and back-end development

Fast and Efficient

Asynchronous features improve performance without blocking tasks

Large Ecosystem

Extensive libraries and frameworks simplify development

In-Demand Skill

Essential for web development, offering strong career opportunities

Basic Types

Boolean

Represents true/false values, often used in conditional statements

Null

Means “nothing” or “empty”. A variable doesn’t have a value on purpose

Undefined

A variable that has been declared but not yet assigned a value

Number

Represents both integer and floating-point numbers

String

A sequence of characters in quotes

Object

A complex data type used to store collections of key-value pairs.

JavaScript typeof Operator

typeof is used to check the type of a variable or value

It returns the data type as a **string.**

```
typeof 42;                  // "number"
typeof "Hello“;             // "string"
typeof true;                // "boolean"
typeof undefined;           // "undefined"
typeof {name: "Alice"};     // "object"
```

What is a Variable

- A **variable** is a container for storing data values
- It allows you to label and store information that can be reused or modified later
- Think of it as a box where you keep different types of data

Rules for Creating Variables

- Variable names must start with:
    - A letter (e.g. name)
    - An underscore _ (e.g. _value)
    - A dollar sign $ (e.g. $price)

- Variable names cannot start with a number
- No spaces allowed in variable names
- Case-sensitive: **myVar** and **myvar** are different variables
- Avoid reserved keywords

Hoisting

Hoisting is a behavior in JavaScript where **variable and function declarations** are automatically moved to the top of the code, before it runs:

- This means you can use variables and functions **before** they appear in your code
- However, only the **declaration** is moved to the top, not the actual value or the function’s code.

Declaration at the Block Level

Block: A block is any code wrapped inside **{ }** curly braces, such as **if, for,** or **while** loops, and functions.

Block-level scope: Variables declared inside a block are only accessible **inside that block**. Once the block ends, the variable disappears and can't be used outside.

Declaration with let

let is used to declare variables that are **limited to the block** in which they are declared.

Variables declared with **let** can be **updated** but **not re-declared** in the same scope.

Declaration with const

const is used to declare variables whose value **cannot be changed** after they are assigned.

Like **let, const** variables are **block-scoped**, meaning they only exist inside the block they are declared.

Temporary Dead Zone (TDZ)

The **Temporary Dead Zone** is the time between when a variable is **hoisted** (moved to the top of its scope) an when it is **initialized** (assigned a value).

Even though **let** and **const** are hoisted, you cannot use them before their declaration line, and trying to do so will throw an error

Block Binding In Loops

- When you use **let** and **const** inside loops, a **new variable** is created for each iteration (loop cycle).
- This is different from **var**, where the same variable is used for every loop cycle.
- This behavior helps avoid issues where loop variables get overwritten unexpectedly