Variables and Datatypes

Variables and Datatypes

Variables

Variables are Containers for Storing Data

JavaScript Variables can be declared in 4 ways:

- Automatically
- Using `var`
- Using `let`
- Using `const`

```
var name = "Chetan"; // not recommended
// scopre error
let name1 = "Chaitanya"; // Recommended
const pi = 3.14; // Recommended
```

```
// Automatically
x = 5;
y = 6;
z = x + y;
```

`var`

- Can be **redeclared** and **updated**.
- **Function-scoped.**

```
var name = "John";
var name = "Doe";  // Redeclaration is allowed
console.log(name);  // Output: Doe

function test() {
    if (true) {
        var x = 10;
    }
    console.log(x);  // Accessible here (not block-scoped)
}
test();
```

`let`

- Can be **updated**, but **not redeclared** in the same scope.
- **Block-scoped** → exists only inside `{}` .

```
let age = 25;
age = 30;  // Allowed

if (true) {
    let age = 35;  // This is a new variable (block-scoped)
    console.log(age);  // Output: 35
}
console.log(age);  // Output: 30
```

`const`

- **Cannot be updated or redeclared**.
- **Block-scoped and Constant**.

```
const PI = 3.1416;
// PI = 3.15;  // Error: Assignment to constant variable
```

```
const user = { name: "Alice" };
user.name = "Bob";  // Allowed (modifying properties)
console.log(user);  // Output: { name: "Bob" }
```

Variable Hoisting

- Hoisting moves declarations to the top.

```
// only initializes var with undefined.
console.log(x); // undefined (hoisted)
var x = 10;

console.log(y); // ReferenceError (hoisted)
let y = 20;
```

Temporal Dead Zone

The term to describe the state where variables are un-reachable. They are in scope, but they aren't declared.

- `let` and `const` is also host but can not be access.

`let` Temporal Dead Zone

```
console.log(myVar);  // ❌ ReferenceError: Cannot access 'myVar' before initializ

// DEAD ZONE
let myVar = 10;
console.log(myVar);  // ✅ 10
```

`const` Temporal Dead Zone

```
console.log(myConst);  // ❌ ReferenceError

// DEAD ZONE
const myConst = 42;
```

- TDZ in Function Scope

```
function testTDZ() {
    console.log(a);  // ❌ ReferenceError

    // DEAD ZONE
    let a = 5;
    console.log(a);  // ✅ 5
}
testTDZ();
```

Datatypes

1. Primitive Data Types (Immutable)
2. Non-Primitive (Reference) Data Types

Primitive Data Types

1. Number → Representing Integers and Floating-Point number.

```
let number = 10; // Number
let price = 99.99;
```

2. String → Represents a sequence of characters.

```
let text = "Hello"; // String
let templateLiteral = `My name is ${name}`;
```

3. Boolean → Represents `true` or `false`.

```
let isAdmin = true;
let isLoggedIn = false;
```

4. Undefined → A variable that has been declared but not assigned a value.

```
let undefiendVar = undefined; // Undefined
console.log(typeof undefiendVar); // undefined

let x;
console.log(x);  // undefined
```

5. Null

```
let nothing = null; // Object - (historical bug)
console.log(typeof nothing); // object
```

6. Symbol → Used to create unique identifiers.

```
let symbolVar = Symbol(); // Symbol
console.log(typeof symbolVar); // symbol

let sym1 = Symbol("id");
let sym2 = Symbol("id");
console.log(sym1 === sym2); // false (Symbols are always unique)
```

Non Primitive / Reference Data Types

1. Object → A collection of key-value pairs.

```
let person = {
  name: "Sandy",
  age: 20,
  isWorkingProfessional: true
}

console.log(typeof {});        // "object"
```

2. Array → A special type of object that holds indexed values.

```
let numbers = [1, 2, 3, 4, 5];

console.log(typeof []);        // "object"
```

3. Function → Functions in JavaScript are objects and can be assigned to variables.

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

let sayHello = function() {
   console.log("Hi!");
};

console.log(typeof function(){}); // "function"
```