02-Variables and DataTypes

Variables:

In JavaScript, a variable is a container that holds a value. Think of it like a labeled box where you can store data.

Basic Syntax

```
// Declaring a variable
let name = "Moiz";

// Using a variable
console.log(name); // Outputs: Moiz

// Changing a variable's value
name = "Ali";
console.log(name); // Outputs: Ali
```

Three Ways to Declare Variables

- **let**: Modern way to declare variables that can be reassigned.
- **const**: For values that won't change (constants).
- **var**: Older way to declare variables (less used now).

Real-World Example

```
// Shopping cart
let itemCount = 0;
let totalPrice = 0;

// Adding an item
function addItem(price) {
  itemCount = itemCount + 1; // or itemCount++
  totalPrice = totalPrice + price;
  console.log(`Cart: ${itemCount} items, $${totalPrice}`);
}
```

```
// Use the function
addItem(25); // Cart: 1 items, $25
addItem(10); // Cart: 2 items, $35
```

Variables allow your code to store, track, and manipulate data throughout your program.

Comparison of Variable Declaration Methods

Feature	let	const	var
Reassignment	Yes	No	Yes
Scope	Block scope	Block scope	Function scope
Hoisting	Hoisted but not initialized	Hoisted but not initialized	Hoisted and initialized as undefined
Redeclaration	Not allowed in same scope	Not allowed in same scope	Allowed in same scope
Temporal Dead Zone	Yes	Yes	No

Data Types in JS:

1. Premitive DataTypes
    a. number
    b. string
    c. boolean
    d. undefined
    e. null
    f. bigint
    g. symbol
2. Non-Premitive DataTypes
    a. array
    b. object

c. function

1. Number DataType:-

In JavaScript, a number is exactly what you think it is - it's a numeric value. You can use numbers to perform all sorts of calculations in your code.

Examples of Number Data Type:

```
// Basic numbers
let age = 25;
let price = 19.99;
let temperature = -5;

// Special number values
let infinity = Infinity;  // Represents infinity
let notANumber = NaN;     // "Not a Number"

// Mathematical operations
let sum = 10 + 5;        // 15
let product = 6 * 7;      // 42
let division = 20 / 4;    // 5
```

Important Reminders:

- JavaScript has only one number type for both integers and decimals
- The maximum safe integer is: 9007199254740991 (use BigInt for larger numbers)
- Watch out for precision issues with decimal calculations (e.g., 0.1 + 0.2 ≠ 0.3 exactly)
- Division by zero gives Infinity, not an error

2. String DataType:-

A string is simply text. It's any sequence of characters enclosed in quotes. Think of it as a word, sentence, or even a whole paragraph.

Examples of String Data Type:

```
// Creating strings
let name = "Moiz";
let message = 'Hello there!';
let phrase = `Welcome, ${name}!`;  // Template string with variable

// String operations
let combined = "Hello" + " " + "World";  // "Hello World"
let length = name.length;              // 4
let uppercase = name.toUpperCase();      // "MOIZ"
```

Important Reminders:

- Strings can use single quotes, double quotes, or backticks
- Backticks allow embedding variables with ${variable}
- Strings are immutable - once created, they cannot be changed
- Use special characters with backslash: \n (new line), \t (tab), etc.

3. Boolean DataType:-

A boolean is like a light switch - it can only be in one of two states: true or false. Booleans are perfect for representing yes/no or on/off conditions.

Examples of Boolean Data Type:

```
// Creating booleans
let isLoggedIn = true;
let hasPermission = false;

// Boolean from comparisons
let isAdult = age >= 18;        // true if age is 18 or more
let isValidName = name !== "";  // true if name is not empty

// Boolean logic
let canAccess = isLoggedIn && hasPermission;  // AND - true only if both ar
e true
let needsAttention = !isLoggedIn || !hasPermission;  // OR with NOT
```

Important Reminders:

- Booleans are essential for conditional logic (if statements, loops)
- Values that convert to false: 0, "", null, undefined, NaN (these are "falsy")
- All other values convert to true (these are "truthy")
- Use === (strict equality) instead of == for more predictable comparisons

4. Undefined DataType:-

Undefined means "no value has been assigned yet." When you create a variable but don't give it a value, it's undefined.

Examples of Undefined:

```
// Variable declared but not initialized
let username;
console.log(username);  // undefined

// Function with no return statement
function sayHi() {
  console.log("Hi!");
  // No return statement
}
let result = sayHi();  // function logs "Hi!" but returns undefined
console.log(result);   // undefined
```

Important Reminders:

- Undefined is the default value for variables that are declared but not initialized
- Function parameters that aren't provided are undefined
- It's different from not defined (which means the variable doesn't exist)
- Best practice: don't explicitly assign undefined - use null to represent "no value" intentionally

5. Null DataType:-

Null means "intentionally no value" or "nothing." Unlike undefined, null is used when you want to explicitly say that a variable has no value.

Examples of Null:

```
// Setting a variable to null
let user = null;  // We know there's no user yet

// Checking for null
if (user === null) {
  console.log("No user logged in");
}

// Common use case: initial state before data is loaded
let userData = null;
// Later, after fetching data:
userData = { name: "Moiz", age: 25 };
```

Important Reminders:

- Null represents an intentional absence of value (unlike undefined)
- typeof null returns "object" - this is actually a JavaScript bug/quirk
- Always use === (strict equality) when checking for null
- In JSON, null is a valid value, but undefined is not

6. BigInt DataType:-

BigInt is a special data type for really, really big numbers - bigger than what regular numbers can handle.

Examples of BigInt:

```
// Creating BigInt values
let bigNumber = 9007199254740991n;  // Add 'n' at the end
let anotherBig = BigInt("9007199254740991");  // Another way to create

// Operations with BigInts
let sum = bigNumber + 1n;  // 9007199254740992n
let product = bigNumber * 2n;  // 18014398509481982n

// Cannot mix with regular numbers
// This will cause an error: let mixed = bigNumber + 1;
```

Important Reminders:

- Use BigInt for integers larger than 9007199254740991 (2^53-1)
- Cannot mix BigInt with regular numbers in calculations
- Append 'n' to number literals to make them BigInt
- Relatively new feature - check browser compatibility if needed

7. Symbol DataType:-

Symbols are unique identifiers. Each Symbol value is guaranteed to be different from all other values, even if they have the same description.

Examples of Symbol:

```
// Creating symbols
let id = Symbol("id");  // Description is optional and for debugging
let key = Symbol("key");

// Even with same description, symbols are different
let sym1 = Symbol("hello");
let sym2 = Symbol("hello");
console.log(sym1 === sym2);  // false - they're different!

// Using symbols as object properties
let user = {
  name: "Moiz",
  [id]: 123  // Symbol as property key
};
console.log(user[id]);  // 123
```

Important Reminders:

- Symbols are always unique, even with the same description

- They're mainly used as unique object property keys
- Symbol properties are not included in Object.keys() or for...in loops
- They help avoid name collisions in objects

8. Array (Non-Primitive):-

An array is like a list or collection where you can store multiple items in a single variable. Think of it as a row of boxes, each containing something.

Examples of Arrays:

```
// Creating arrays
let fruits = ["Apple", "Banana", "Orange"];
let mixed = [1, "hello", true, null];  // Can hold different types
let empty = [];  // Empty array

// Accessing array elements (0-based index)
console.log(fruits[0]);  // "Apple"
console.log(fruits[2]);  // "Orange"

// Modifying arrays
fruits.push("Mango");  // Add to end
fruits.pop();  // Remove from end
console.log(fruits.length);  // Number of items

// Looping through arrays
for (let i = 0; i < fruits.length; i++) {
  console.log(fruits[i]);
}
```

Important Reminders:

- Arrays are zero-indexed (first item is at position 0)
- Arrays can hold items of any type, including other arrays
- Use array methods like push(), pop(), splice(), map(), filter() for common operations
- Check if something is an array with Array.isArray()

9. Object (Non-Primitive):-

Objects are collections of key-value pairs. Think of them like a dictionary, where you can look up values by their names (keys).

Examples of Objects:

```
// Creating objects
let person = {
  name: "Moiz",
  age: 25,
  isStudent: true,
  greet: function() {
    return "Hello, I'm " + this.name;
  }
};

// Accessing object properties
console.log(person.name);  // "Moiz"
console.log(person["age"]);  // 25 (alternative syntax)

// Modifying objects
person.location = "Karachi";  // Add new property
delete person.isStudent;  // Remove property
person.age = 26;  // Change value

// Using methods
console.log(person.greet());  // "Hello, I'm Moiz"
```

Important Reminders:

- Objects store data in key-value pairs, where keys are strings or symbols
- Access properties with dot notation (obj.prop) or bracket notation (obj["prop"])
- Objects can contain methods (functions as properties)
- Objects are passed by reference, not by value

10. Function (Non-Primitive):-

Functions are reusable blocks of code that perform specific tasks. They're like little machines that take input, do something with it, and give you back a result.

Examples of Functions:

```
// Function declaration
function add(a, b) {
  return a + b;
}

// Function expression
const multiply = function(a, b) {
  return a * b;
};

// Arrow function
const square = (num) => num * num;

// Using functions
console.log(add(5, 3));      // 8
console.log(multiply(4, 2)); // 8
console.log(square(4));      // 16

// Functions are objects and can have properties
add.description = "Adds two numbers";
console.log(add.description); // "Adds two numbers"
```

Important Reminders:

- Functions are first-class objects in JavaScript - they can be passed around like values
- Functions can be assigned to variables, passed as arguments, and returned from other functions
- Every function returns a value (if no explicit return, it returns undefined)
- Functions create their own scope for variables declared inside them

Checking Data Types in JavaScript:

```javascript
// Using typeof operator
console.log(typeof 42);          // "number"
console.log(typeof "hello");      // "string"
console.log(typeof true);        // "boolean"
console.log(typeof undefined);    // "undefined"
console.log(typeof null);        // "object" (this is a historical bug)
console.log(typeof 42n);         // "bigint"
console.log(typeof Symbol());     // "symbol"
console.log(typeof []);          // "object"
console.log(typeof {});          // "object"
console.log(typeof function(){});  // "function"

// Better way to check for arrays
console.log(Array.isArray([]));   // true
console.log(Array.isArray({}));   // false
```

Understanding data types is the foundation of programming. They help JavaScript know what operations can be performed on your data and how to store it in memory. As you continue learning, you'll get more comfortable with how these types work together in your programs!