ITD523

Introduction to JavaScript fundamentals

Prepared by:

Assoc. Prof. Dr. Islam A.T.F. Taj-Eddin

Learning Objectives

- Understand the role of variables as data containers.
- Master variable declaration using let and const.
- Navigate JavaScript’s Primitive and Complex Data Types.
- Manage Data Type conversion (Casting).
- Implement effective code documentation using Comments.

Variables in JavaScript

- Variables act as containers for storing data.
- They store intermediate results and values used later in the program.
- Example: let steps = 100;

Naming Variables

- Variable names identify containers storing data.
- Good names reflect stored values.
- Examples: height, color, stepCounter.

Rules for Naming Variables

- Letters, digits, _ and $
- Cannot start with digits
- Case sensitive
- Example: test, Test, TEST are different.

Variable Shadowing (scope later) :

- Occurs when a variable declared in an inner scope has the same name as one in an outer scope.
- The inner variable "shadows" or hides the outer one within that specific block.

Declaring Variables

- Declaration reserves a name and memory for storing values.
- Keywords:
  - var
  - let
  - const

Example of Declaration

- var height;
- console.log(height); // undefined
- console.log(weight); // ReferenceError

var vs let

- var: older keyword
- let: modern recommended keyword
- let prevents redeclaration errors.

Redeclaration Example

- var height;
- var height; // allowed

- let height;
- let height; // error

Initializing Variables

- Initialization assigns the first value to a variable.
- Uses assignment operator =

Initialization Example

- let height = 180;
- let anotherHeight = height;
- let weight;
- weight = 70;

Printing Variables

- console.log(height); // prints value
- console.log("height"); // prints text

Strict Mode

- "use strict" enforces modern JavaScript rules.
- Prevents using variables without declaration.

Strict Mode Example

- "use strict";
- height = 180; // error because variable not declared

Changing Variable Values

- Variables can change values during execution.
- let steps = 100;
- steps = 120;
- steps = steps + 200;

Dynamic Typing

- JavaScript variables are dynamically typed.
- A variable can store values of different types.

Dynamic Typing Example

- let greeting = "Hello";
- greeting = 1;

Implicit Type Conversion

- JavaScript automatically converts types when needed.
- Example:
- "Hello" + 100 -> "Hello100"

Constants

- Constants store values that cannot change.
- Declared using const.
- const greeting = "Hello"; //declaration & initialization at the same time
- greeting = "Hi"; // error

Why Use Constants

- Prevent accidental changes
- Improve code safety
- Allow performance optimization

Scope in JavaScript

- Scope defines where variables are accessible.
- Depends on where they are declared.

Program Blocks

- Blocks are sections of code inside { }.
- Often used in loops, conditions, or functions.

Block Example

- let counter;
- {
- counter = 1;
- console.log(counter);
- }

Nested Blocks

- Blocks can exist inside other blocks.
- This improves program structure and readability.

Indentation

- Indentation improves code readability.
- Editors usually format indentation automatically.

Global Scope

- Variables declared outside blocks are global.
- Accessible throughout the program.

Local Scope

- Variables declared inside blocks using let or const are local.

Scope Example

- let height = 180;
- {
- let weight = 70;
- }
- console.log(weight); // error

Nested Scope Example

```
let height = 200;
{
 let weight = 100;
 {
 let info = "tall";
 }
}
```

var Scope

- Variables declared with var ignore block scope.
- They usually become global.

var Example

- var height = 180;
- {
- var weight = 70;
- }
- console.log(weight); // accessible

Data Types (The Primitives)

- JavaScript is **dynamically typed**, but values have specific types:
- **Boolean:** true or false.
- **Number:** Integers and floats (e.g., 42, 3.14).
- **BigInt:** For numbers larger than 2^{53}-1.
- **String:** Textual data (e.g., "Hello").
- **Undefined:** A variable that has been declared but not assigned a value.
- **Null:** Intentional absence of any object value.

Number Data Type

- Represents numeric values
- Can store integers and decimal numbers
- Example:
- let year = 2025;
- let price = 99.99;

String Data Type

- Represents text
- Written inside quotes
- Example:
- let name = 'Alice';
- let message = "Hello World";

Boolean Data Type

- Logical data type
- Only two values: true or false
- Example:
- let isLoggedIn = true;

Undefined and Null

- Undefined:
  - Default value for uninitialized variables

- Null:
  - Represents intentional absence of value
- Example:
- let data;
- data = null;

The typeof Operator

- Used to determine the type of a variable.
- Example:
- let x = 42;
- console.log(typeof x); // number

Type Casting (Conversion)

- **Implicit Conversion (Coercion):** JavaScript automatically converts types (e.g., 10 + "5" results in "105").

- **Explicit Conversion:** Manually changing the type for precision.
  - String(): Converts to text.
  - Number(): Converts to a number.
  - Boolean(): Converts to truthy/falsy values.

Implicit Type Conversion

- JavaScript automatically converts types.
- Example:
- let result = 'Hello' + 100;
- // Output: Hello100

Explicit Type Conversion

- Manual conversion using functions.
- Examples:
- Number('123')
- String(100)
- Boolean(0)

Complex Data Types (Objects)

- **Objects:** Collections of related data or functional entities.

- **Arrays:** A special type of object used for ordered lists.
  - *Example:* let colors = ["red", "green", "blue"];

Complex Data Types (Objects)

- **Structure:** Key-value pairs.

```
let user = {name: "Jack", age: 30 };


//change value to a feild
console.log(user.age); // -> 30
user.age = 35;
console.log(user.age); // -> 35


//add and delete a feild
console.log(user.phone); // -> undefined
user.phone = "904-399-7557";
console.log(user.phone); // -> 904-399-7557
delete user.phone;
console.log(user.phone); // -> undefined
```

Functions (Introduction)

- Functions are reusable blocks of code.
- Used when a task must be repeated many times.

Why Functions Are Useful

- Reduce code duplication
- Make programs easier to maintain
- Improve readability

Comments in JavaScript

- Comments explain code and improve readability.
- Ignored by the JavaScript interpreter.

Single-Line Comments

- Start with //
- Example:
- // This is a comment
- let x = 10; // variable declaration

Multi-Line Comments

- Use /* and */
- Example:
- /* This is a
- multi-line comment */

Why Comments Are Important

- Improve code readability
- Explain complex logic
- Help debugging
- Assist teamwork in large projects

Module Summary

- Variables store data
- let and const are modern declarations
- Strict mode improves safety
- Constants protect values
- Scope controls visibility
- Functions enable reusable code
- JavaScript supports multiple data types
- Type conversion allows changing data types
- Comments help explain and maintain code