# Chapter 2

# Values, Types, Variables, and Operators

# Objectives

- Understand and Work with Core Data Types and values
- Declare and Manage Variables
- Initialize and Inspect Variables
- Work with Operators and Expressions
- the use of typeOf operator
- Increment and decrement operators
- Implicit and explicit type conversions
- Template literals In modern JavaScript

# 2.1
# Data type and values

# Data type and values

- A **data type** defines the kind or category of a value.
- A **value** is the actual piece of data stored or used in a program.
- Data types in JavaScript can be divided into two main categories:
- Primitive data type and Non-Primitive data type

# Primitive VS Non-Primitive

- Primitive data types:
- These are the most basic, built-in data types. They store **single, simple values**
  - String → text ("Hello")
  - Number → numbers (int 42,  decimal 3.14)
  - BigInt → very large numbers (123n)
  - Boolean → true/false (true, false)
  - Undefined → a variable that has been declared but not assigned a value
  - Null → intentional empty value
  - Symbol → unique and immutable identifier

# Primitive VS Non-Primitive

- Non-Primitive Data Types (Reference Types)
- These are **complex data structures** that can hold multiple values.
  - Arrays ([1, 2, 3])
  - Functions (function() {})
  - Dates (new Date())
  - Other objects created using classes

# 2.1
# What is a Variable?

# Variables

- **Variables** are used to store and manage data in JavaScript. You can declare variables using the **let**, **const**, or **var** keywords.
- JavaScript Variables can be declared in **4 ways**:
- Automatically
  - `x = 10;`
- Declaring a variable using let
  - `let x = 10;`
- Declaring a constant variable using const
  - `const PI = 3.14;`
- Declaring a variable using var (less recommended)

```
var y = "Hello";
```

# Deference between Let, Var and const

- **let**: Allows you to change the value of the variable.
- **const**: Declares a constant variable whose value cannot be changed once assigned.
- **var**: Older way to declare variables; it has some scoping quirks and is less recommended in modern JavaScript.

# Variable Names: Rules

In JavaScript, variable names (also known as identifiers) must follow specific rules to be valid.

- Variable Names Must Start with a **Letter**, **Underscore (_)**, or **Dollar Sign ($).**
- Subsequent Characters Can Include **Letters**, **Digits**, **Underscores**, and **Dollar Signs**.
- Variable Names **Are Case-Sensitive**.
- Reserved Words Are Not Allowed.
- Use Descriptive and Meaningful Names.

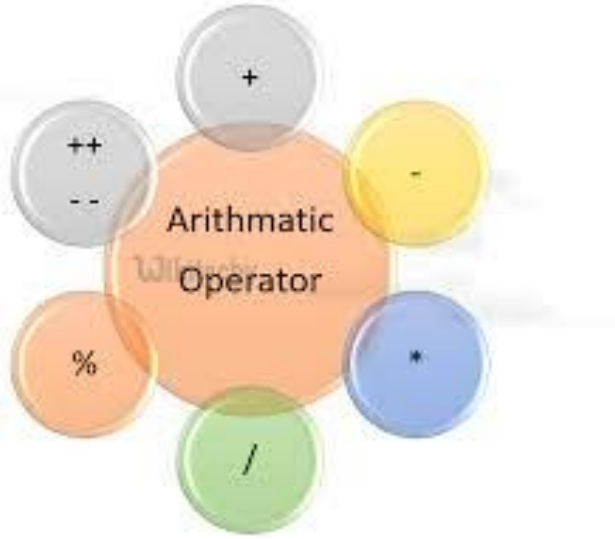
# Variable Names: Tips

- Be sure your variable name is not too long
  - You have to retype it!
  - `theFirstPlayerInTheGame` or
  - `playerOne`
- Use CamelBack or underscores for readability
  - `number1` or `number_1` or `playerOne`
- Make variable names meaningful
  - `abc` or `first_name`

# JavaScript Comments

- JavaScript comments are hints that a programmer can add to make their code easier to read and understand. They are completely ignored by JavaScript engines.
- There are two ways to add comments to code:
  - `// A single line comment`
  - ```
    /* It is a multi-line comment.
       It will not be displayed upon
       execution of this code */
    ```
    ```

# 2.3 Arithmetic Operators



# Arithmetic Operators

- Are symbols that perform operations on values.
- JavaScript includes various types of operators:

| Operator | Description |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| ** | Exponentiation |
| / | Division |
| % | Modulus (Remainder) |
| ++ | Increment |
| -- | Decrement |

# The Modulus Operator

- Written as `x % y`
- Read as `x mod y`
- Means the integer remainder after dividing `x` by `y`
- Examples:
  - `15 % 2 = 1`
  - `23 % 7 = 2`
  - `18 % 3 = 0`

# The Hierarchy of Operations or The Order of Precedence

1. Parentheses
2. Exponents
3. Multiplication, division, and modulus in order from left to right
4. Addition and subtraction in order from left to right

# The Concatenation Operator

- Joins strings of text
- Is represented by + sign
- Example:

```
let username = "lizzy";
let school = "myschool";
let domain = "edu";
let email = username + "@" + school + "." + domain;
console.log(email);
```

Output is: `lizzy@myschool.edu`

# 2.4 Relational Operators

# Relational Operators

- `>` greater than
- `<` less than
- `>=` greater than or equal to
- `<=` less than or equal to
- `==` is the same as value
- `!=` is not the same as value
- `===` equal value and equal type (checks both value and type)
- !== is not equal as value and as type

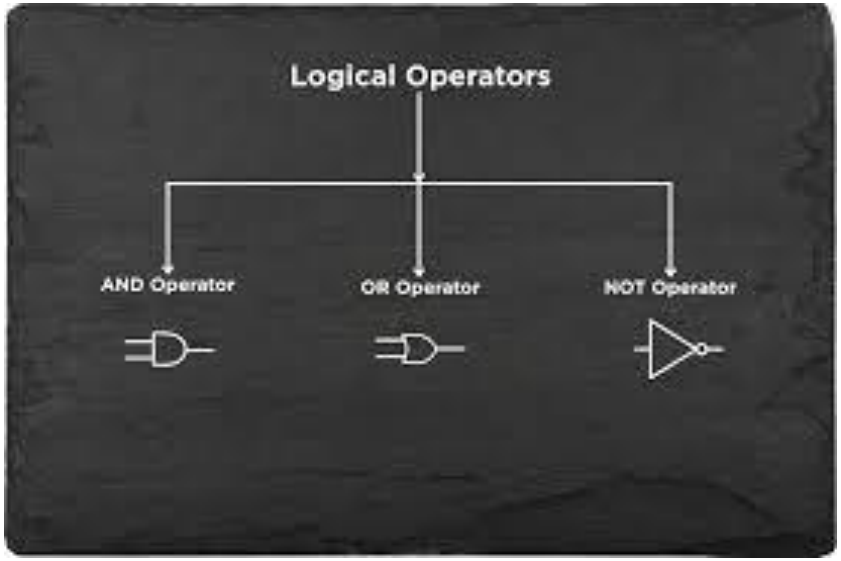
Result of using relational operators is always either `true` or `false`

# Relational Operators

- `console.log(5 == "5"); // true (same value)`
- `console.log(5 === "5");// false (different type)`
- `console.log(10 != "10");// false(same value)`
- `console.log(10 !== "10"); //true (different type)`
- `console.log(7 > 3);  // true`
- `console.log(2 < 5);  // true`
- `console.log(6 >= 6);  // true`
- `console.log(4 <= 2);  // false`

# 2.5

# Logical Operators and the Conditional Operator



# Logical Operators

- The result of an expression with logical operators is always either `true` or `false`
- The `AND` operator is written as `&&`
  - Always results in `false` unless both sides of the expression are `true`
- The `OR` operators is written as `||`
  - Always results in `true` unless both sides of the expression are `false`
- The `NOT` operator is written as `!`
  - It is `true` if the expression is `false` and `false` if the expression is `true`

# Truth Table for Not Operator

| x | !x | Example (assume age = 24, weight = 140) |
|---|---|---|
| **true** | false | !(age > 18) is false, because (age > 18) is true. |
| **false** | true | !(weight == 150) is true, because (weight == 150) is false. |

# Truth Table for AND Operator

| $X_1$ | $X_2$ | $X_1$ && $X_2$ | Example (assume age = 24, weight = 140) |
|---|---|---|---|
| **false** | false | false | (age <= 18) && (weight < 140) is false, because both conditions are both false. |
| **false** | true | false | |
| **true** | false | false | (age > 18) && (weight > 140) is false, because (weight > 140) is false. |
| **true** | true | true | (age > 18) && (weight >= 140) is true, because both (age > 18) and (weight >= 140) are true. |

# Truth Table for Not Operator

| $X_1$ | $X_2$ | $X_1$ \|\| $X_2$ | **Example (assume age = 24, weihgt = 140)** |
|---|---|---|---|
| **false** | false | false | |
| **false** | true | true | (age > 34) \|\| (weight <= 140) is true, because (age > 34) is false, but (weight <= 140) is true. |
| **true** | false | true | (age > 14) \|\| (weight >= 150) is True, because (age > 14) is true. |
| **true** | true | true | |

# Order of Operations

| | Description | Symbol |
|---|---|---|
| Arithmetic Operators are evaluated first in the order listed | | |
| | 1st: Parentheses | ( ) |
| | 2nd: Exponents | ^ |
| | 3rd: Multiplication / Division / Modulus | *, /, % |
| | 4th: Addition / Subtraction | + – |
| Relational Operators are evaluated second and all relational operators have the same precedence | | |
| | Less than | < |
| | Less than or equal to | <= |
| | Greater than | > |
| | Greater than or equal to | >= |
| | The same as, equal to | == |
| | Not the same as | != |
| Logical Operators are evaluated last in the order listed | | |
| | 1st: NOT | ! |
| | 2nd: AND | && |
| | 3rd: OR | \|\| |

# The Conditional Operator

This asks: Is one thing the same as another thing? If yes, do something. If not, do something else or do nothing.

Written like this:

```
varName = (condition) ? value_1 : value_2;
```

# The Conditional Operator: Examples

1. 
```
let password = "JUST";
let userpw = prompt("enter password:");
let message = (password == userpw) ? "Correct password!" : "incorrect password.";
console.log(message);
```

2. 
```
x = 3; y = 15;  z = 5;
answer = prompt("What is " + y + " divided by " + x + "?");
answer = parseFloat(answer);
message = (y/x == answer) ? "Correct" : "Wrong";
console.log(message);
```

# 2.5
# Increment and Decrement Operators

# Increment Operator (++)

- **Definition**: Increases the value of a variable by **1**.
- **Types**:
  - **Pre-increment (++x)** → Value increases **before** it is used.
  - **Post-increment (x++)** → Value increases **after** it is used.

```
let a = 5;
console.log(++a); // Pre-increment → 6 (a becomes 6, then used)
console.log(a++); // Post-increment → 6 (a used first, then becomes 7)
console.log(a);   // Now a = 7
```

# Decrement Operator (--)

- **Definition**: Decreases the value of a variable by **1**.
- **Types**:
  - **Pre-decrement (--x)** → Value decreases **before** it is used.
  - **Post-decrement (x--)** → Value decreases **after** it is used.

```
let b = 5;
console.log(--b); // Pre-decrement → 4 (b becomes 4, then used)
console.log(b--); // Post-decrement → 4 (b used first, then becomes 3)
console.log(b);   // Now b = 3
```

# typeof Operator

- A **unary operator** in JavaScript that returns the **data type** of a value or variable.
  - `console.log(typeof 42);` `// "number"`
  - `console.log(typeof 3.14);` `// "number"`
  - `console.log(typeof "Hello");` `// "string"`
  - `console.log(typeof true);` `// "boolean"`
  - `console.log(typeof undefined);` `// "undefined"`
  - `console.log(typeof null);` `// "object"` `(special case/quirk in JS)`
  - `console.log(typeof {name: "Ali"});` `// "object"`
  - `console.log(typeof [1,2,3]);` `// "object"` `(arrays are objects in JS)`
  - `console.log(typeof function(){});` `// "function"`

# Automatic/Implicit type conversion

JavaScript employs automatic type conversion, also known as type coercion, to convert values from one data type to another in certain situations. This can happen implicitly without the need for explicit type conversion functions or operations. Understanding how automatic type conversion works is essential for writing robust JavaScript code. Here are some common scenarios of automatic type conversion:

**String Concatenation implicit conversion**

```
let age = 25;
let message = "I am " + age + " years old."; // Implicitly converts age to a string
```

**Numeric Operations implicit conversion**

```
let numStr = "42";
let result = numStr * 2; // Implicitly converts numStr to a number: result is 84
```

# Cont…

- **Comparison Operators implicit conversion**
- `let num = 10;`
- `let strNum = "5";`
- `console.log(num > strNum);`
- // Implicitly converts strNum to a number for comparison: true

A table listing some commonly used JavaScript built-in functions and methods, including their descriptions:

| Function/Method | Description |
| --- | --- |
| **alert(message)** | Displays a dialog box with a specified message. |
| **prompt(message)** | Displays a dialog box with a prompt for user input. |
| **confirm(message)** | Displays a dialog box with a confirmation message. |
| **console.log(data)** | Outputs data to the browser's console for debugging. |
| **document.write(text)** | Writes HTML content directly to the document. |

# Escape character

| Escape Sequence | Character Represented | Description |
|---|---|---|
| \\n | Newline | Inserts a newline character. |
| \\t | Tab | Inserts a tab character. |
| \' | Single Quote | Escapes a single quote within a string. |
| \" | Double Quote | Escapes a double quote within a string. |
| \\ | Backslash | Escapes a backslash within a string. |