

## Basics of JavaScript and ECMAScript (ES6)

### Syntax

#### Internal JavaScript:

```
<html>
  <head>
    <script language = "javascript" type = "text/javascript">
      document.write("Hello World!")
      console.log("Hello!")
    </script>
  </head>
</html>
```

#### External JavaScript:

##### p1.html

```
<html>
  <head>
    <script src="p1.js"></script>
  </head>
</html>
```

##### p1.js

```
document.write("This is written in external file")
```

### Variables

JavaScript variables can be declared in **four ways**:

1. Implicit (Automatically)
2. Using **var**
3. Using **let**
4. Using **const**

The variables declared with **var** and **let** are mutable that is their value can be changed but variables declared using **const** are immutable.

#### Best Practice

- ◊ Use **let** for variables that will change.
- ◊ Use **const** by default for values that shouldn't change.
- ✗ Avoid **var** in modern JavaScript unless required.

## JavaScript: var vs let vs const

|                 | var   | let  | const   |
|-----------------|---|--|---|
| Scope           | The scope of a <code>var</code> variable is functional or global scope.   | The scope of a <code>let</code> variable is block scope.   | The scope of a <code>const</code> variable is block scope.  |
| Re-declare      | <p><b>It can</b> be re-declared in the <b>same scope</b>.</p> <pre><code>var a=10; var a=20; a=30; let a=45;</code></pre> | <p><b>It cannot</b> be re-declared in the <b>same scope</b>.</p> <pre><code>let b=20; //not allowed let b=24; var b=33;</code></pre> | <p><b>It can not</b> be re-declared in <b>same scope</b>.</p> <pre><code>const b=20; //not allowed const b=33; let b=24; var b=33;</code></pre> |
| Re-assign       | <p><b>It can</b> be updated in the <b>same scope</b>.</p> <pre><code>var a=10; a=30;</code></pre>                         | <p><b>It can</b> be in the <b>same scope</b>.</p> <pre><code>let b=20; b=45;</code></pre>  | <p><b>It can not</b> be updated in <b>same scope</b>.</p> <pre><code>const b=20; //not allowed b=45;</code></pre>                               |
| Without Init    | <b>It can</b> be declared without initialization.   | <b>It can</b> be declared without initialization.  | It cannot be declared without initialization.<br><b>No</b><br><b>Must use</b><br><code>const a=20;</code>                                       |
| Hoisting access | It can be accessed without initialization as its default value is <b>"undefined"</b> .                                    | It cannot be accessed without initialization otherwise it will give <b>'referenceError'</b> .  | It cannot be accessed without initialization, as it cannot be declared without initialization.<br><b>'referenceError'</b>                       |

## Datatypes

| Category                               | Data Type | One-Line Example  |
|--|-----------|---|
| Primitive (immutable)                  | String    | <code>let name = "ABC";</code>                                    |
|  | Number    | <code>let age = 20;</code>  |
|  | Boolean   | <code>let isValid = true;</code>                                  |
|  | Undefined | <code>let x;</code>   |
|  | Null      | <code>let data = null;</code><br><i>(*type of null is object)</i> |
| Non-Primitive (mutable)<br>(reference) | Object    | <code>let user = { name: "ABC", age: 20 };</code>                 |
|  | Array     | <code>let arr = [10, 20, 30];</code>                              |

## Conditions

Conditions help your program **make decisions** based on true/false logic.

| Condition            | WHY we use it                        | WHEN to use it   | Example  |
|----------------------|--------------------------------------|--|--|
| <b>if</b>            | To check a single condition          | One decision needed  | <pre>let isLoggedIn = true; if (isLoggedIn) {   console.log("Welcome user"); }</pre>   |
| <b>if...else</b>     | To choose between two paths          | Yes / No situations  | <pre>let age = 16; if (age &gt;= 18) {   console.log("Allowed"); } else {   console.log("Not allowed"); }</pre>  |
| <b>else if</b>       | To handle multiple conditions        | More than two outcomes   | <pre>let score = 82; if (score &gt;= 90) {   console.log("Grade A"); } else if (score &gt;= 75) {   console.log("Grade B"); } else {   console.log("Grade C"); }</pre>   |
| <b>switch</b>        | To match one value with many options | Fixed values (menu, roles, days)   | <pre>let role = "admin"; switch (role) {   case "admin":     console.log("Full access");     break;   case "editor":     console.log("Edit access");     break;   default:     console.log("No access"); }</pre> |
| <b>ternary (?) :</b> | Short decision making                | Simple conditions<br><b>Syntax:</b><br>condition ?<br>expression_if_true :<br>expression_if_false; | <pre>let age = 20; let result = age &gt;= 18 ? "Adult" : "Minor"; console.log(result);</pre>   |

## LOOPS

| Loop              | Syntax  | Used For   | Best When               |
|-------------------|---|--|-------------------------|
| <b>for</b>        | for(init; condition; step)  | Repeat code fixed number of times  | You know how many times |
| <b>while</b>      | while(condition)  | Repeat while condition is true   | Count unknown           |
| <b>do...while</b> | do { } while(condition)   | Runs at least once   | Must execute once       |
| <b>for...of</b>   | for (value of iterable)<br><b>eg.</b><br>let fruits = ["apple", "banana"]<br>for (let fruit of fruits) {<br>console.log(fruit);<br>}              | Loop through values  | Arrays, strings         |
| <b>for...in</b>   | for (key in object)<br><b>eg. (Object)</b><br>let user={name:"abc", age: 25}<br>for (let key in user) {<br>console.log(key + ": " + user[key])  } | Loop through keys/indexes<br><br>(*In a loop, use user[key] to get the value because key is a variable that stores the property name.) | Objects                 |

### Comparison of var and let (Overwrite Effect)

| Case                  | Code using var   | Output      | Code using let  | Output      |
|-----------------------|--|-------------|---|-------------|
| <b>Condition (if)</b> | var x=10;<br>if(true){ var x=3; }<br>console.log(x);                     | 3           | let x=10; if(true){ let x=3; }<br>console.log(x);                     | 10          |
| <b>Loop (for)</b>     | var i=5;<br>for(var i=0;i<2;i++){<br>console.log(i);}<br>console.log(i); | 0<br>1<br>2 | let i=5; for(let i=0;i<2;i++){<br>console.log(i);}<br>console.log(i); | 0<br>1<br>5 |

## Array Methods

| Method     | Syntax   | What it Does   | Returns          |
|------------|--|--|------------------|
| push()     | <b>array.push(item)</b><br><b>eg.</b><br><pre>let fruits = ["apple", "banana"]; fruits.push("orange"); console.log(fruits);</pre> <p><b>Output</b><br/>["apple", "banana", "orange"]</p> | Adds element(s) to the <b>end</b> of an array                              | New array length |
| pop()      | <b>array.pop()</b><br><b>eg.</b><br><pre>let numbers = [10, 20, 30]; let removed = numbers.pop(); console.log(numbers); // [10, 20] console.log(removed); // 30</pre>                    | Removes the <b>last</b> element from an array                              | Removed element  |
| map()      | <b>array.map(callback)</b><br><b>eg.</b><br><pre>let nums = [1, 2, 3, 4]; let squares = nums.map(num =&gt; num * num); console.log(squares); // [1, 4, 9, 16]</pre>                      | Creates a <b>new array</b> by transforming each element                    | New array        |
| filter()   | <b>array.filter(callback)</b><br><b>eg.</b><br><pre>let ages = [12, 18, 25, 14]; let adults = ages.filter(age =&gt; age &gt;= 18); console.log(adults); // [18, 25]</pre>                | Creates a <b>new array</b> with elements that pass a condition             | New array        |
| forEach()  | <b>array.forEach(callback)</b><br><b>eg.</b><br><pre>let names = ["abc", "pqr", "xyz"]; names.forEach(name =&gt; {   console.log("Hello " + name); });</pre>                             | Executes a function for each element (does <b>not</b> return a new array). | undefined        |
| includes() | <b>array.includes(value)</b><br><b>eg.</b><br><pre>let colors = ["red", "green", "blue"];</pre>  | Checks if array contains a value   | true or false    |

|        |   |   |                      |
|--------|---|---|----------------------|
|        | <pre>console.log(colors.includes("green")); //<br/> <b>true</b><br/> console.log(colors.includes("tomato")); //<br/> <b>false</b></pre>   |   |                      |
| find() | <p>array.find(callback)</p> <p><b>eg.</b></p> <pre>let users = [<br/>   { id: 1, name: "A"},<br/>   { id: 2, name: "B"},<br/>   { id: 3, name: "C"}<br/> ];<br/> let user = users.find(u =&gt; u.id === 2);<br/> console.log(user);<br/> // { id: 2, name: "B"}</pre> | Finds the <b>first element</b> that matches a condition | Element or undefined |

## **++ and -- Operators in JavaScript**

| Operator   | Name           | Meaning                                     | When value is updated | Example  |
|------------|----------------|---|-----------------------|--|
| <b>++a</b> | Pre-increment  | Increases value by 1 <b>before</b> using it | Immediately           | let a = 5;<br>let b = ++a;<br><b>Result:</b><br>a = 6<br>b = 6 |
| <b>a++</b> | Post-increment | Uses current value, then increases by 1     | After use             | let a = 5;<br>let b = a++;<br><b>Result:</b><br>a = 6<br>b = 5 |
| <b>--a</b> | Pre-decrement  | Decreases value by 1 <b>before</b> using it | Immediately           | let a = 5;<br>let b = --a;<br><b>Result:</b><br>a = 4<br>b = 4 |
| <b>a--</b> | Post-decrement | Uses current value, then decreases by 1     | After use             | let a = 5;<br>let b = a--;<br><b>Result:</b><br>a = 4<br>b = 5 |

### Combined Example

```
let x = 3;
let y = x++ + ++x;
console.log(x, y);
```

#### Output:

5 8

#### Explanation:

- x++ → uses 3, then x becomes 4
- ++x → increases to 5, then uses 5
- y = 3 + 5 = 8

## Functions

### Ways to Invoke (Call) a Function

| Type                 | How it Executes             | Example                 |
|----------------------|-----------------------------|-------------------------|
| Event-Based Call     | Runs on user action (click) | onclick="showMessage()" |
| Explicit Call        | Called directly in JS       | add(5, 10);             |
| Self-Invoking (IIFE) | Runs automatically          | (function(){ })();      |

### Function with / without Parameters

| Type                      | Example                                    |
|---------------------------|--|
| <b>Without Parameters</b> | function fun(){ document.write("Hello"); } |
| <b>With Parameters</b>    | function fun(a,b){ document.write(a+b); }  |

### Always Call:

```
fun();
fun(10, 20);
```

### Function Expression

```
var sum = function(a,b){
    return a+b
};
sum(3,5);
```

### One Function Calling Another

```
function info(f,l){
    return f + " " + l
}
function hello(){
    document.write(info("abc","xyz"))
}
```

## Arrow Functions (ES6)

Arrow functions, introduced in ES6 (ECMAScript 2015), provide a more concise syntax for writing functions in JavaScript.

### Syntax

```
function name = (param1, param2, ..., paramN) => expression
```

| Type                             | Syntax Example  | Key Point                                    |
|----------------------------------|---|--|
| <b>Multiple Parameters</b>       | const add = (a, b) => a + b;  | Parentheses required for multiple parameters |
| <b>Single Parameter</b>          | const square = x => x * x;<br>or<br>const square = (x) => x * x;              | Parentheses optional for one parameter       |
| <b>No Parameters</b>             | const hi = () => "Hello!";  | Empty parentheses mandatory                  |
| <b>Block Body Arrow Function</b> | const multiply = (a, b) => {<br>const result = a * b;<br>return result;<br>}; | { } used → return required                   |
| <b>Implicit Return</b>           | const fun1 = () => "Hello World!";  | No {} → value returned automatically         |

**Implicit** means the return happens automatically without writing `return`, while **explicit** means the return value is clearly specified using the `return` keyword.

**In arrow functions, when curly braces {} are used, the `return` keyword is required;  
when curly braces are not used, the expression value is returned automatically.**

⚠️ {} used → return required

⚠️ No {} → implicit return

## TEMPLATE LITERALS

Template Literals are a modern way to create strings in JavaScript. They were introduced in ES6 and provide more flexibility than normal strings. Template literals make it easier to work with variables, expressions, and multiline text.

| Feature                     | Normal Strings   | Template Literals  |
|-----------------------------|--|--|
| Quotes Used<br>Basic Syntax | Single '' or Double " " quotes<br><b>eg.</b><br><code>let msg = "Hello World";</code>                              | Backticks ``<br><b>eg.</b><br><code>let msg = `Hello World`;</code>  |
| Variable<br>Interpolation   | Uses + operator<br><b>eg.</b><br><code>let text = "Hello " + name + "!";</code>                                    | Uses \${}<br><b>eg.</b><br><code>let text = `Hello \${name}!`;</code>  |
| Using<br>Expressions        | Requires + and brackets<br><b>eg.</b><br><code>let result = "Sum is " + (a + b)</code>                             | Expressions allowed inside \${}<br><b>eg.</b><br><code>let result = `Sum is \${a + b}`;</code>               |
| Multiline Strings           | Uses \n for new line<br><b>eg.</b><br><code>let text = "Hello\nHow are<br/>you?\nGoodbye";</code>                  | Multiline supported naturally<br><b>eg.</b><br><code>let text = `Hello<br/>How are you?<br/>Goodbye`;</code> |
| HTML Creation               | Heavy string concatenation<br><b>eg.</b><br><code>"&lt;div&gt;"+"&lt;h1&gt;Title&lt;/h1&gt;"+"&lt;/div&gt;"</code> | Clean and readable<br><b>eg.</b><br><code>&lt;div&gt;&lt;h1&gt;Title&lt;/h1&gt;&lt;/div&gt;</code>           |

## Rest/Spread/Default parameter

| Method            | Syntax                 | What it Does  | Returns | Example                                       | Output    |
|-------------------|------------------------|---|---------|---|-----------|
| Rest Parameter    | function fn(...para)   | Collects multiple arguments into one array                  | Array   | function f(...a){ console.log(a); } f(1,2,3); | [1, 2, 3] |
| Spread Operator   | fn(...arr) / [...arr]  | Expands array values into individual elements               | Values  | let a=[1,2,3]; console.log(...a);             | 1 2 3     |
| Default Parameter | function fn(a = value) | Assigns a default value if argument is missing or undefined | Value   | function f(a=5){ console.log(a); } f();       | 5         |

## Pop up Boxes: Alert, Confirm, Prompt

| Popup Box | Method / Syntax          | What it Does                              | Buttons Shown         | Returns   | Example                                 |
|-----------|--------------------------|---|-----------------------|---|---|
| Alert     | alert ("message")        | Displays a warning or information message | OK                    | Nothing (undefined)                                       | alert("This is a warning");             |
| Confirm   | confirm ("message")      | Asks user for confirmation                | OK, Cancel            | <b>OK</b> → true<br><b>Cancel</b> → false                 | let res = confirm("Continue?");         |
| Prompt    | prompt("msg", "default") | Takes input from the user                 | OK, Cancel + text box | <b>OK</b> → entered text (string)<br><b>Cancel</b> → null | let name = prompt("Enter name", "abc"); |

## Common JavaScript Methods and Their Usage

| Method              | Purpose                                | Syntax            | Return Type | Example   | Output     |
|---------------------|--|-------------------|-------------|---|------------|
| <b>toFixed()</b>    | Formats number to fixed decimal places | num.toFixed(n)    | String      | (12.345).toFixed(2)   | 12.35      |
| <b>isNaN()</b>      | Checks if value is Not-a-Number        | isNaN(value)      | Boolean     | isNaN("abc")  | true       |
| <b>parseInt()</b>   | Converts string to integer             | parseInt(value)   | Number      | parseInt("45.6")  | 45         |
| <b>parseFloat()</b> | Converts string to decimal number      | parseFloat(value) | Number      | parseFloat("45.6")  | 45.6       |
| <b>entries()</b>    | Returns index-value pairs of array     | array.entries()   | Iterator    | <pre>let arr = ['a', 'b']; for (let [index, value] of arr.entries()) {   console.log(index, value); }</pre> | 0 a<br>1 b |