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

Datatypes

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

Conditions

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

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

LOOPS

Loop	Syntax	Used For	Best When
for	for(init; condition; step)	Repeat code fixed number of times	You know how many times
while	while(condition)	Repeat while condition is true	Count unknown
do...while	do { } while(condition)	Runs at least once	Must execute once
for...of	for (value of iterable) **eg.** let fruits = ["apple", "banana"] for (let fruit of fruits) { console.log(fruit); } **Output** apple banana	Loop through values	Arrays, strings
for...in	for (key in object) **eg. (Object)** let user={name:"abc", age: 25} for (let key in user) { console.log(key + ": " + user[key]) } **Output** name: abc age: 25 [Array] Let a=[1,2,3] for (let i in a) { console.log(i+ ": " + a[i]) }	Loop through keys/indexes (*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
Condition (if)	var x=10; if(true){ var x=3; } console.log(x);	3	let x=10; if(true){ let x=3; } console.log(x);	10
Loop (for)	var i=5; for(var i=0;i<2;i++){ console.log(i);} console.log(i);	0 1 2	let i=5; for(let i=0;i<2;i++){ console.log(i);} console.log(i);	0 1 5

Array Methods

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

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

++ and -- Operators in JavaScript

Operator	Name	Meaning	When value is updated	Example
++a	Pre-increment	Increases value by 1 **before** using it	Immediately	let a = 5; let b = ++a; **Result:** a = 6 b = 6
a++	Post-increment	Uses current value, then increases by 1	After use	let a = 5; let b = a++; **Result:** a = 6 b = 5
--a	Pre-decrement	Decreases value by 1 **before** using it	Immediately	let a = 5; let b = --a; **Result:** a = 4 b = 4
a--	Post-decrement	Uses current value, then decreases by 1	After use	let a = 5; let b = a--; **Result:** a = 4 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
Without Parameters	function fun(){ document.write("Hello"); }
With Parameters	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
Multiple Parameters	const add = (a, b) => a + b;	Parentheses required for multiple parameters
Single Parameter	const square = x => x * x; or const square = (x) => x * x;	Parentheses optional for one parameter
No Parameters	const hi = () => "Hello!";	Empty parentheses mandatory
Block Body Arrow Function	const multiply = (a, b) => { const result = a * b; return result; };	{} used → return required
Implicit Return	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 Basic Syntax	Single ' ' or Double " " quotes **eg.** let msg = "Hello World";	Backticks ` ` **eg.** let msg = `Hello World`;
Variable Interpolation	Uses + operator **eg.** let text = "Hello " + name + "!";	Uses ${} **eg.** let text = `Hello ${name}!`;
Using Expressions	Requires + and brackets **eg.** let result = "Sum is " + (a + b)	Expressions allowed inside ${} **eg.** let result = `Sum is ${a + b}`;
Multiline Strings	Uses \n for new line **eg.** let text = "Hello\nHow are you?\nGoodbye";	Multiline supported naturally **eg.** let text = `Hello How are you? Goodbye`;
HTML Creation	Heavy string concatenation **eg.** " "+" Title "+" "	Clean and readable **eg.** Title

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	**OK** → true **Cancel** → false	let res = confirm("Continue?");
Prompt	prompt("msg", "default")	Takes input from the user	OK, Cancel + text box	**OK** → entered text (string) **Cancel** → null	let name = prompt("Enter name","abc");

Common JavaScript Methods and Their Usage

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