

UNIT 1

Client-Side Scripting (JavaScript)

(15 Hrs) — Scripting Language (CACS254)

⚠ Important Note

These notes may contain errors, typos or omissions. Please double-check definitions, syntax and code examples against your textbook, class notes and the official TU syllabus before relying on them for exams. Test all code yourself before submitting it as practical work.

1.1 Introduction to JavaScript

JavaScript (JS) is a lightweight, interpreted, object-based scripting language that runs inside the web browser. It is used to make web pages **interactive and dynamic**. HTML gives a page its structure, CSS gives it style, and JavaScript gives it behaviour. JavaScript was created by Brendan Eich at Netscape in 1995 and is standardised as **ECMAScript (ES)**.

- **Interpreted:** the browser's JavaScript engine (e.g. V8 in Chrome) executes the code line by line, no separate compile step.
- **Case-sensitive:** `myVar` and `myvar` are different identifiers.
- **Dynamically typed:** a variable can hold any type of value and its type can change at run time.
- **Event-driven:** code reacts to user actions such as clicks, key presses and form submission.

Need of Client-Side Scripting Language

- **Faster response:** simple work (validation, calculation) is done in the browser without contacting the server.
- **Reduced server load and bandwidth:** fewer requests are sent to the server.
- **Rich user interface:** menus, sliders, animations, pop-ups, image galleries.
- **Instant feedback:** error messages appear while the user is still typing in the form.
- **Dynamic content:** page content can change without reloading (DOM manipulation, AJAX).

Client-side scripting	Server-side scripting
Runs in the user's browser	Runs on the web server

Client-side scripting	Server-side scripting
Code is visible to the user (View Source)	Code is hidden; only output (HTML) is sent
Cannot access server database directly	Can access databases and server files
Examples: JavaScript, VBScript	Examples: PHP, ASP.NET, JSP, Node.js
Less secure (user can alter it)	More secure (code stays on server)

1.2 Formatting and Coding Conventions

- Statements end with a semicolon `;` (recommended, though optional in many cases).
- Use meaningful names in **camelCase** for variables and functions, e.g. `totalMarks`.
- Identifiers start with a letter, `_` or `$`; they cannot start with a digit and cannot be reserved words.
- Indent code blocks consistently (2 or 4 spaces) and use `{ }` even for one-line blocks.
- Constants are usually written in UPPERCASE, e.g. `MAX_SIZE`.
- Declare variables with `let`, `const` (modern) or `var` (older).

Keyword	Scope	Re-assign?	Re-declare?
<code>var</code>	Function scope	Yes	Yes
<code>let</code>	Block scope	Yes	No
<code>const</code>	Block scope	No	No

1.3 JavaScript Files, Comments and Embedding

Comments

Code:

```
// This is a single-line comment
/* This is a
   multi-line comment */
```

Embedding JavaScript in HTML using the script tag

JavaScript can be placed in three ways: **inline** (inside an event attribute), **internal** (inside a `<script>` tag in the page) and **external** (in a separate `.js` file linked by `src`). External files are preferred because they can be cached and reused by many pages.

Code:

```
<!-- 1. Internal -->
<script>
  document.write("Hello, World!");
</script>
<!-- 2. External file (saved as myscript.js) -->
<script src="myscript.js"></script>
<!-- 3. Inline -->
<button onclick="alert('Clicked!')">Click me</button>
```

The noscript tag

The `<noscript>` tag holds content displayed only when the browser has JavaScript disabled or does not support scripts.

Code:

```
<script>document.write("JavaScript is enabled");</script>
<noscript>Please enable JavaScript to use this page.</noscript>
```

Note: Scripts may be placed in `<head>` or `<body>`. Placing them at the end of `<body>` lets the page load first and avoids errors caused by accessing elements that do not yet exist.

1.4 Operators

Type	Operators	Example
Arithmetic	+ - * / % ++ -- **	10 % 3 gives 1
Assignment	= += -= *= /= %=	x += 5
Comparison	== === != !== > < >= <=	5 == '5' is true, 5 === '5' is false
Logical	&& !	(a > 1) && (b < 5)
String	+ (concatenation)	'Hi ' + 'Ram'
Ternary	condition ? a : b	age >= 18 ? 'Adult' : 'Minor'
typeof	typeof x	typeof 12 gives 'number'

Note: == compares values after type conversion, while === compares both value and type. Prefer ===

1.5 Control Structures

Conditional statements

Code:

```
let marks = 72;
```

```

if (marks >= 80) {
    console.log("Distinction");
} else if (marks >= 60) {
    console.log("First Division");
} else {
    console.log("Pass or Fail");
}

switch (day) {
    case 0: name = "Sunday"; break;
    case 6: name = "Saturday"; break;
    default: name = "Weekday";
}

```

Loops

Code:

```

for (let i = 1; i <= 5; i++) { document.write(i + " "); }
let n = 1;
while (n <= 5) { document.write(n + " "); n++; }
let k = 1;
do { document.write(k + " "); k++; } while (k <= 5);

```

Output:

```
1 2 3 4 5
```

break exits a loop immediately and **continue** skips the rest of the current iteration.

1.6 Arrays and the for...each Loop

An **array** stores multiple values in a single variable; indexes start from 0.

Code:

```

let fruits = ["Apple", "Banana", "Mango"];
fruits.push("Orange");           // add at end
fruits.pop();                    // remove last
fruits.unshift("Kiwi");          // add at start
fruits.shift();                  // remove first
console.log(fruits.length);      // 3
console.log(fruits.indexOf("Banana")); // 1
// for...of and forEach
for (let f of fruits) { console.log(f); }
fruits.forEach(function (item, index) {
    console.log(index + ": " + item);
});

```

Method	Purpose
push() / pop()	Add / remove last element

Method	Purpose
shift() / unshift()	Remove / add first element
slice(s, e)	Returns a part of the array (original unchanged)
splice(i, n)	Removes/inserts elements at index i
join(sep)	Joins elements into a string
sort() / reverse()	Sorts / reverses the array
concat()	Joins two or more arrays

1.7 Defining and Invoking Functions

A **function** is a named block of reusable code that runs when it is called. It may take **parameters** and may return a value using **return**.

Code:

```
function add(a, b) {           // definition
  return a + b;
}
let result = add(10, 20);      // invocation
console.log(result);           // 30
const square = function (x) { return x * x; }; // function expression
const cube = (x) => x * x * x;   // arrow function
```

Output:

30

- Variables declared inside a function are **local**; those declared outside are **global**.
- Functions can be called from events, e.g. `<button onclick="add(1,2)">`.

1.8 Built-in Objects

JavaScript provides ready-made objects such as **String**, **Number**, **Math**, **Array**, **Date** and **RegExp**.

Object	Common members
String	length, toUpperCase(), toLowerCase(), charAt(), indexOf(), substring(), replace(), split(), trim()
Math	Math.PI, round(), floor(), ceil(), max(), min(), pow(), sqrt(), random(), abs()
Number	toFixed(), parseInt(), parseFloat(), isNaN()

Code:

```
let s = "Nepal";
console.log(s.length);           // 5
console.log(s.toUpperCase());    // NEPAL
console.log(Math.max(4, 9, 2));  // 9
console.log(Math.floor(Math.random() * 10)); // random 0-9
```

Date Object

Code:

```
let d = new Date();           // current date and time
console.log(d.getFullYear()); // e.g. 2026
console.log(d.getMonth());    // 0-11 (0 = January)
console.log(d.getDate());     // 1-31
console.log(d.getDay());      // 0-6 (0 = Sunday)
console.log(d.getHours() + ":" + d.getMinutes() + ":" + d.getSeconds());
let d2 = new Date(2026, 3, 14); // 14 April 2026
```

1.9 Interacting with the Browser: Windows and Frames

The **window** object is the top-level object of the Browser Object Model (BOM). It represents the browser window and all global variables/functions are its members.

Object	Description / example
window	alert(), confirm(), prompt(), open(), close(), setTimeout(), setInterval()
location	location.href gets/sets the URL; location.reload()
history	history.back(), history.forward(), history.length
navigator	Browser information: navigator.userAgent, navigator.language
screen	screen.width, screen.height

Code:

```
alert("Hello");           // message box
let ok = confirm("Are you sure?"); // OK / Cancel -> true / false
let name = prompt("Your name?", "Ram");// input box
let w = window.open("https://example.com", "_blank", "width=400,height=300");
w.close();
setTimeout(function () { alert("3 seconds passed"); }, 3000);
let t = setInterval(showTime, 1000); // repeat every second
clearInterval(t);
```

Frames

Frames divide a browser window into separate areas, each loading a different document. Modern pages use the `<iframe>` tag (the old `<frameset>` is obsolete). Scripts can reach frames through `window.frames[]`, and a page inside a frame can reach its parent using `window.parent` or `window.top`.

Code:

```
<iframe name="box" src="page.html" width="300" height="200"></iframe>
<script>
  window.frames["box"].location = "other.html";    // change frame content
</script>
```

1.10 Document Object Model (DOM)

The **DOM** is a tree-structured representation of an HTML document. Each tag becomes an **element node**, text becomes a **text node**, and attributes become **attribute nodes**. JavaScript uses the DOM to read and change content, structure and style of the page.

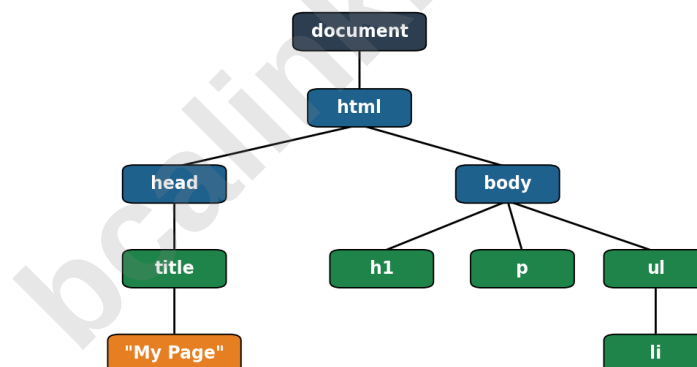


Figure 1.1: DOM tree of a simple HTML page

Code:

```
document.getElementById("demo").innerHTML = "New text";
document.getElementsByClassName("box");
document.getElementsByTagName("p");
document.querySelector("#demo");           // first match (CSS selector)
document.querySelectorAll(".item");        // all matches
let el = document.getElementById("demo");
el.style.color = "red";
el.setAttribute("title", "Hello");
let p = document.createElement("p");      // create new element
p.textContent = "Added by JS";
document.body.appendChild(p);             // add to page
el.remove();                              // delete element
```

1.11 Event Handling

An **event** is an action that occurs in the browser, such as a click or key press. An **event handler** is the code that runs in response.

Event	Occurs when
onclick	Element is clicked
onload	Page (or image) has finished loading
onchange	Value of an input/select changes
onsubmit	A form is submitted
onmouseover / onmouseout	Mouse pointer enters / leaves an element
onkeydown / onkeyup	A key is pressed / released
onfocus / onblur	Element gains / loses focus

Code:

```
<button id="b1" onclick="sayHi()">Inline</button>
<button id="b2">Using addEventListener</button>
<script>
  function sayHi() { alert("Hi!"); }
  document.getElementById("b2").addEventListener("click", function (e) {
    alert("Clicked: " + e.target.id);
  });
</script>
```

1.12 Forms

Form elements are accessed through the DOM using their `name` or `id`. The values of text boxes are read with `.value`, check boxes/radios with `.checked`, and lists with `.selectedIndex`.

Code:

```
<form name="f1" onsubmit="return check()">
  Name: <input type="text" name="uname" id="uname">
  Gender: <input type="radio" name="g" value="M"> M
         <input type="radio" name="g" value="F"> F
  <select name="city">
    <option>Kathmandu</option><option>Pokhara</option>
  </select>
  <input type="submit" value="Send">
</form>
<script>
  function check() {
    let n = document.f1.uname.value;
    alert("Hello " + n);
    return true; // false would stop submission
  }
```

```
}  
</script>
```

1.13 Cookies

A **cookie** is a small piece of data (about 4 KB) stored by the browser and sent back to the server with each request. Cookies remember users, preferences and login state. In JavaScript they are accessed through `document.cookie`.

Code:

```
function setCookie(name, value, days) {  
    let d = new Date();  
    d.setTime(d.getTime() + days * 24 * 60 * 60 * 1000);  
    document.cookie = name + "=" + value + "; expires=" + d.toUTCString()  
        + "; path="/;  
}  
function getCookie(name) {  
    let parts = document.cookie.split("; ");  
    for (let p of parts) {  
        let kv = p.split("=");  
        if (kv[0] === name) return kv[1];  
    }  
    return "";  
}  
setCookie("user", "Sita", 7);  
alert(getCookie("user")); // Sita  
setCookie("user", "", -1); // delete by setting a past date
```

1.14 Handling Regular Expressions

A **regular expression (regex)** is a pattern used to match character combinations in strings. In JavaScript it is written between slashes, e.g. `/abc/i`, or created with `new RegExp()`.

Pattern	Meaning
<code>^</code> and <code>\$</code>	Start / end of string
<code>.</code>	Any single character
<code>\d \w \s</code>	Digit / word character / whitespace
<code>*</code> <code>+</code> <code>?</code>	0 or more / 1 or more / 0 or 1
<code>{n,m}</code>	Between n and m repetitions
<code>[abc]</code> <code>[^abc]</code>	Any one of / none of these characters
<code>()</code> <code> </code>	Grouping / alternation (OR)

Flags: **i** = ignore case, **g** = global (all matches), **m** = multi-line.

Code:

```
let re = /^[A-Za-z]+$;/
console.log(re.test("Nepal"));           // true
console.log(re.test("Nepal123"));        // false
"a1b22c333".match(/\d+/g);              // ["1", "22", "333"]
"2026-04-14".replace(/-/g, "/");        // "2026/04/14"
"hello world".search(/world/);          // 6
```

1.15 Client-Side Validation

Validation checks that data entered in a form is correct and complete before it is sent to the server. Client-side validation gives instant feedback, but it must **always be repeated on the server** because users can bypass JavaScript.

Example: Validate a registration form

Requirements: name must contain letters only, e-mail must be valid, password must have at least 6 characters, and both passwords must match.

Code:

```
<form name="reg" onsubmit="return validate()">
  Name: <input type="text" name="nm"><br>
  Email: <input type="text" name="em"><br>
  Password: <input type="password" name="p1"><br>
  Confirm: <input type="password" name="p2"><br>
  <input type="submit" value="Register">
</form>
<script>
function validate() {
  let f = document.reg;
  if (!/^[A-Za-z ]+$/.test(f.nm.value)) {
    alert("Name must contain letters only"); f.nm.focus(); return false;
  }
  if (!/^[^\s@]+@[^\s@]+\.[a-z]{2,}$/i.test(f.em.value)) {
    alert("Enter a valid e-mail"); f.em.focus(); return false;
  }
  if (f.p1.value.length < 6) {
    alert("Password must be at least 6 characters"); return false;
  }
  if (f.p1.value !== f.p2.value) {
    alert("Passwords do not match"); return false;
  }
  return true;
}
</script>
```

Explanation

The **onsubmit** handler returns **false** to cancel submission when any check fails; returning **true** lets the form be sent. Each check uses a regular expression or a length/equality comparison, and `focus()` moves the cursor to the faulty field.

Key Takeaways

- ✓ JavaScript is an interpreted, event-driven, case-sensitive client-side language embedded with the `<script>` tag.
- ✓ Use `let/const`, camelCase names, and prefer `===` for comparison.
- ✓ Arrays (`push`, `pop`, `forEach`), functions (normal, expression, arrow) and built-in objects (`Math`, `String`, `Date`) are the basic tools.
- ✓ The BOM (`window`, `location`, `history`, `navigator`) controls the browser; the DOM controls the document.
- ✓ Events + DOM methods drive interactivity; cookies store small data in the browser.
- ✓ Regular expressions power validation, and client-side validation must be repeated on the server.

— *End of Unit 1* —