Variables, functions, loops, datatypes

🟦 VARIABLES

- let → value can change
- const → value cannot change
- var → old keyword, avoid using

🟩 DATA TYPES
Primitive:

- Number
- String
- Boolean
- Undefined
- Null
- BigInt
- Symbol

Non-Primitive:

- Object
- Array

- Function

🟨 FUNCTIONS

- A block of reusable code
  Example:
  function add(a, b) { return a + b; }

- Arrow function:
  const add = (a, b) => a + b;

🟥 LOOPS

1. for loop
   for (let i = 0; i < 5; i++) {}

2. while loop
   while (condition) {}

3. for...of (for arrays)
   for (let item of array) {}

DOM MANIPULATION

---

1. DOM (Document Object Model)
- Browser converts HTML into a tree of objects called the DOM.
- Each tag (html, body, div, p, button, etc.) becomes a DOM element.
- DOM Manipulation = using JavaScript to read/change the page after it loads.

Common actions:
- Find elements

- Change text / HTML
- Change styles (color, size, etc.)
- Add/remove elements

---
2. document.getElementById(id)

- Selects ONE element by its id.
- Syntax:
  const el = document.getElementById("myId");
- Returns:
  - The element (if found)
  - null (if not found)
- id must be unique in the page.

Example:
```
<h1 id="title">Hello</h1>
const titleEl = document.getElementById("title");
titleEl.innerHTML = "New title";
```

---
3. document.querySelector(selector)

- Selects the FIRST element that matches a CSS selector.
- Syntax:
  const el = document.querySelector("selector");

Examples:
```
document.querySelector("#title");       // id=title
document.querySelector(".btn");         // class=btn
document.querySelector("p");            // first <p>
document.querySelector("div.card > p"); // <p> inside .card
```

---
4. document.querySelectorAll(selector)

- Selects ALL elements matching the CSS selector.
- Returns a NodeList (array-like).
- Syntax:
  const items = document.querySelectorAll(".menu-item");

Looping:
```
items.forEach(item => {
  item.style.color = "blue";
});
```

---
5. innerHTML

- Property that gets/sets the HTML content inside an element.

Get:
```
const box = document.getElementById("box");
console.log(box.innerHTML);
```

Set:
```
box.innerHTML = "<h2>New heading</h2><p>Some text</p>";
```

- innerHTML can include HTML tags.
- If you only need plain text (no tags), use:
```
element.textContent = "Some text";
```

---
6. Changing Styles with JavaScript

A) Using element.style
- You can modify inline CSS styles directly.
- CSS "background-color" → JS "backgroundColor" (camelCase).

Example:
```
const box = document.getElementById("box");
box.style.backgroundColor = "lightblue";
box.style.fontSize = "20px";
box.style.border = "2px solid black";
```

B) Using classList
- Better for larger style changes. Define styles in CSS, then add/remove class names.

CSS:
```
.highlight {
  background-color: yellow;
  color: red;
}
.hidden {
  display: none;
}
```

JavaScript:
```
const para = document.querySelector("p");
para.classList.add("highlight");     // apply highlight styles
para.classList.remove("highlight");  // remove them
para.classList.toggle("hidden");     // show/hide paragraph
```

classList methods:
- add("className")
- remove("className")

- toggle("className")
- contains("className")

---
7. When to use what?

- getElementById("id")
  - When you know the unique id and just want that element.
  - Simple, fast.

- querySelector("selector")
  - When you want flexibility with CSS selectors (#id, .class, tag, nested selectors).
  - Returns FIRST match.

- querySelectorAll("selector")
  - When you want ALL matching elements (loop over them).

- innerHTML
  - When you want to add/change HTML structure inside an element.
  - Be careful with user-generated content (security).

- style / classList
  - style: quick, one-off style changes.
  - classList: cleaner, re-usable CSS, better for bigger UI changes.
==========

DOM MANIPULATION — COMPLETE SUMMARY WITH ADDITIONAL METHODS

1️⃣ Selecting Elements

- document.getElementById("id")
- document.getElementsByClassName("class")
- document.getElementsByTagName("tag")
- document.querySelector("selector")
- document.querySelectorAll("selector")

2️⃣ Changing Content

- element.innerHTML = "Text";
- element.textContent = "Plain text";
- element.innerText = "Text (ignores hidden)";
- element.outerHTML = " Replaces whole element ";

3️⃣ Changing Styles and Classes

- element.style.property = "value";
- element.classList.add("className");
- element.classList.remove("className");
- element.classList.toggle("className");
- element.classList.contains("className");

4️⃣ Creating and Inserting Elements

- document.createElement("tag")
- document.createTextNode("text")
- parent.appendChild(child)
- parent.prepend(child)
- parent.insertBefore(newNode, existingNode)
- parent.append(newNode)
- element.insertAdjacentHTML(position, html)

Positions for insertAdjacentHTML:
- "beforebegin"
- "afterbegin"
- "beforeend"
- "afterend"

5️⃣ Removing Elements

- element.remove()
- parent.removeChild(child)
- element.replaceWith(newNode)

6️⃣ Attributes Manipulation

- element.getAttribute("attr")
- element.setAttribute("attr", "value")
- element.removeAttribute("attr")
- element.hasAttribute("attr")
- element.id / element.src / element.href // direct access

7️⃣ Event Handling

- element.addEventListener("event", function)
- element.removeEventListener("event", function)
- element.onclick = function() { ... } // older way

8️⃣ DOM Traversing

- element.parentElement
- element.children
- element.firstElementChild / lastElementChild
- element.nextElementSibling / previousElementSibling

- element.closest("selector")

9️⃣ Common Element Properties

- element.value // inputs
- element.src, element.alt // images
- element.href // links
- element.id, element.className
- element.disabled, element.checked // forms
==========

EVENT HANDLING: onclick, onchange, addEventListener, bubbling

1️⃣ onclick

- Event handler for "click" event.
- Runs when the element is clicked.

Two common ways:

A) Inline HTML (not ideal for big projects):
```
<button onclick="alert('Clicked!')">Click</button>
```

B) As a property in JS:
```
const btn = document.getElementById("btn");
btn.onclick = function () {
  alert("Button clicked!");
};
```

⚠ Limitation:
- Only ONE onclick function can be assigned.
- Setting btn.onclick again overwrites the previous one.

2️⃣ onchange

- Fires when the value of an element changes and is "committed".
- Common for: <input>, <select>, <textarea>.

Examples:
```
<input id="name" type="text" onchange="console.log('Changed')" />
```

```
const nameInput = document.getElementById("name");
nameInput.onchange = function () {
  console.log("New value:", nameInput.value);
};
```

Usecases:
- Validate input after user finishes typing.
- React when a dropdown selection changes.

3 addEventListener

- Modern and flexible way to listen for events.

Syntax:
```
element.addEventListener("eventName", callback);
```

Example:
```
btn.addEventListener("click", function () {
  alert("Clicked using addEventListener");
});
```

Advantages over onclick:
- Can attach multiple listeners to the same event.
- Can remove listeners with removeEventListener.
- Supports options like capture / passive / once.

Removing a listener:
```
function handler() {
  console.log("clicked");
}
btn.addEventListener("click", handler);
btn.removeEventListener("click", handler);
```

4 Event Bubbling

- When an event occurs on a nested element, it moves from:
  innermost element → its parent → up to document/window.

Example structure:
```
<div id="outer">
  <div id="inner">
    <button id="btn">Click</button>
  </div>
</div>
```

Click on btn:
- "click" fires on btn (target)
- then bubbles to #inner

- then to #outer
- then to document, etc.

This means:
- If #inner and #outer both have "click" listeners,
  they can also run when clicking the button.

Stopping bubbling:
```
element.addEventListener("click", function (event) {
  event.stopPropagation();
});
```

Why useful?
- Good for event delegation: attach one listener on a parent
  instead of many listeners on each child.

==========

STORING & RETRIEVING VALUES – JS STORAGE

🔹 Why storage?
- Normal variables are lost on page refresh or tab close.
- Browser storage lets us save data that stays longer:
  - Remember user name
  - Save theme preference
  - Save simple app data (todo, cart, etc.)

1️⃣ localStorage

- Key-value storage in the browser.
- Data persists even after closing the browser.
- Data belongs to a specific domain.

Methods:
- localStorage.setItem("key", "value");
- localStorage.getItem("key");
- localStorage.removeItem("key");
- localStorage.clear(); // removes all keys

Example:
```
localStorage.setItem("username", "Shahil");
```

```
const name = localStorage.getItem("username"); // "Shahil"
localStorage.removeItem("username");
```

Note:
- Only stores STRINGS.
- To store objects/arrays → use JSON.stringify() / JSON.parse().

2️⃣ sessionStorage

- Very similar to localStorage.
- BUT data is kept only for the current TAB/SESSION.
- When tab is closed → data is lost.
- New tab = fresh empty sessionStorage.

Methods (same as localStorage):
- sessionStorage.setItem("key", "value");
- sessionStorage.getItem("key");
- sessionStorage.removeItem("key");
- sessionStorage.clear();

Use cases:
- Temporary data while user is on the page.
- Wizard steps, temporary filters, etc.

3️⃣ Storing Objects/Arrays with JSON

Because storage only supports strings:

Store object:
```
const user = { name: "Shahil", age: 20 };
localStorage.setItem("user", JSON.stringify(user));
```

Retrieve object:
```
const str = localStorage.getItem("user");
const userObj = JSON.parse(str);
console.log(userObj.name); // "Shahil"
```

Always check for null:
```
const str = localStorage.getItem("user");
if (str !== null) {
  const obj = JSON.parse(str);
}
```

4️⃣ localStorage vs sessionStorage vs Cookies (quick)

- localStorage:
  - Persistent (until cleared)
  - Shared across tabs (same domain)

- Not sent to server

- sessionStorage:
  - Only for current tab
  - Cleared when tab closes

- Cookies:
  - Small (about 4KB)
  - Can have expiry
  - Sent automatically to server with each request

==========

COOKIES & USAGE IN BROWSER – JAVASCRIPT

🔹 What is a cookie?
- Small piece of data stored in the browser (about 4 KB).
- Stored as "name=value" pairs.
- Used to remember information between page loads and requests.

🔹 Common uses:
- Login/session tracking (sessionId).
- "Remember me" features.
- User preferences (language, theme).
- Analytics and tracking (visits, page views).

1️⃣ Cookie Format

General format:
  name=value; expires=DATE; path=/; domain=example.com; secure; SameSite=Lax

Important attributes:
- name=value → actual data
- expires → when cookie should expire
- path → which paths can access the cookie
- domain → which domain(s) can access the cookie
- secure → only send over HTTPS
- SameSite → controls cross-site behavior (Lax, Strict, None)

2️⃣ Cookies vs localStorage/sessionStorage

Cookies:
- Small (~4KB).
- Sent automatically to server with every HTTP request.
- Can have expiry.

localStorage / sessionStorage:
- Larger (~5–10MB).
- NOT sent to server.
- Used for front-end state/data.

3️⃣ Using cookies in JavaScript (document.cookie)

A) Set a cookie:
```
document.cookie = "username=Shahil";
```

With expiry and path:
```
document.cookie =
  "username=Shahil; expires=Tue, 01 Jan 2030 00:00:00 GMT; path=/";
```

B) Read cookies:
```
console.log(document.cookie);
// "username=Shahil; theme=dark; visits=5"
```

To get a specific cookie, split and search:
```
function getCookie(name) {
  const cookies = document.cookie.split(";");
  for (let c of cookies) {
    const [key, value] = c.trim().split("=");
    if (key === name) return value;
  }
  return null;
}
```

C) Delete a cookie:
```
// Set the same name with an expiry date in the past
document.cookie =
  "username=; expires=Thu, 01 Jan 1970 00:00:00 GMT; path=/";
```

4️⃣ HttpOnly cookies (server-side)

- Cookies with HttpOnly flag cannot be accessed by JavaScript.
- Used for secure session/auth tokens.
- Controlled by the server using Set-Cookie header.

==========

BROWSER DEVTOOLS: INSPECT, CONSOLE, NETWORK, CACHING

1️⃣ Inspect Element (Elements Tab)

- Built-in browser tool to view/edit HTML and CSS.
- Lets you see the structure of the webpage (DOM).
- Can modify elements, text, and styles live.
- Shortcut: Right-click → Inspect or Ctrl+Shift+I / F12.
- Highlights elements when you hover in the DOM panel.

Use cases:
- Check CSS styles.
- Test design changes.
- Debug layout or color issues.

2️⃣ Console Tab

- Place where JavaScript logs and errors appear.
- You can run JS code directly.

Common uses:
- console.log("Hello");
- Check variables and errors.
- Debug JS interactively.

Shortcut:
- Open DevTools → Console tab.

3️⃣ Network Tab

- Shows all network requests made by the page.
- Helps you check:
  - File load times
  - API calls (fetch, AJAX)
  - Response status (200, 404, 500)
  - Whether files are cached

Open:
- DevTools → Network tab → Refresh page.

4️⃣ Caching

- Browser temporarily stores files (HTML, JS, CSS, images)
  so they load faster next time.
- Stored on your computer in browser's cache folder.
- You can see cached requests in Network tab:
  - "(from disk cache)" or "(from memory cache)"

Disable cache (for testing):
- In Network tab, check "Disable cache".

Types:
- Memory cache (temporary, quick access)
- Disk cache (stored on disk, survives restart)

🧠 Summary:
- Inspect → Look at or modify HTML/CSS.
- Console → Debug & log JS.
- Network → Watch requests and responses.
- Caching → Browser's speed trick by reusing downloaded files.

==========