

Perfect! 👍 Let's make a **DOM revision sheet** that covers the basics you mentioned: traversing, styling, attributes, and classList.

DOM Basics Revision

1. Accessing Elements

// By ID

```
document.getElementById("myId");
```

// By class (returns HTMLCollection)

```
document.getElementsByClassName("myClass");
```

// By tag (returns HTMLCollection)

```
document.getElementsByTagName("div");
```

// By CSS selectors

```
document.querySelector(".myClass"); // first match
```

```
document.querySelectorAll(".myClass"); // NodeList (all matches)
```

2. Traversing the DOM

```
let el = document.querySelector(".item");
```

// Parent

```
el.parentElement
```

// Children

```
el.children // HTMLCollection
```

```
el.firstElementChild
```

```
el.lastElementChild
```

// Siblings

```
el.previousElementSibling
```

el.nextElementSibling

3. Styling Elements

```
let box = document.querySelector(".box");
```

```
// Inline styling
```

```
box.style.backgroundColor = "lightblue";
```

```
box.style.fontSize = "20px";
```

4. Attributes

```
let link = document.querySelector("a");
```

```
// Get attribute
```

```
link.getAttribute("href");
```

```
// Set attribute
```

```
link.setAttribute("href", "https://google.com");
```

```
// Remove attribute
```

```
link.removeAttribute("target");
```

```
// Check attribute
```

```
link.hasAttribute("href"); // true/false
```

5. classList (Managing Classes)

```
let div = document.querySelector("div");
```

```
// Add class
```

```
div.classList.add("active");
```

```
// Remove class
```

```
div.classList.remove("active");
```

```
// Toggle class
```

```
div.classList.toggle("dark-mode");
```

```
// Check if class exists
```

```
div.classList.contains("active"); // true/false
```

6. Content

```
let p = document.querySelector("p");
```

```
p.innerText = "Hello World"; // visible text
```

```
p.textContent = "Hi Again"; // all text (even hidden)
```

```
p.innerHTML = "<b>Bold Text</b>"; // parses HTML
```

✅ This covers the **core DOM basics you need for revision**.

Would you like me to also make a **mini practice sheet with small tasks** (like "change the color of all paragraphs", "toggle a dark mode class") so you can test yourself quickly?

You said:

go detailed in each topic

ChatGPT said:

Thought for 45s

Nice — let's **deep dive**. I'll expand each topic (accessing, traversing, styling, attributes, and classList) with APIs, examples, pitfalls, performance tips and small recipes you can copy-paste. Ready? Here we go.

Accessing elements (selection)

Core APIs

```
// Single element
```

```
document.getElementById("id"); // HTML element (fast)
```

```
document.querySelector(".class"); // first match (any CSS selector)
```

```
// Collections / lists
```

```
document.getElementsByClassName("c"); // HTMLCollection (live)
document.getElementsByTagName("div"); // HTMLCollection (live)
document.querySelectorAll(".c"); // NodeList (static)
```

Notes & differences

- **HTMLCollection (getElementsBy*) is *live*** — it updates when the DOM changes.
- **NodeList from querySelectorAll is static** (no auto-update). NodeList supports forEach.
- getElementById is typically the fastest for single element.
- If you need array methods on an HTMLCollection: Array.from(collection) or [...].

Useful helpers

```
// find closest ancestor matching selector (incl. self)
el.closest(".card");
```

```
// check if el matches selector
el.matches("button[type=submit]");
```

Traversing the DOM

Properties & methods

```
// From an element `el`:

el.parentNode // may be Node (could be document)
el.parentElement // Element or null
el.children // HTMLCollection of element children (no text nodes)
el.childNodes // NodeList (includes text nodes)
el.firstChild
el.lastElementChild
el.firstChild // could be a text node
el.lastChild

el.nextSibling // may be text/comment
el.previousSibling
el.nextElementSibling
el.previousElementSibling
```

Node vs Element

- Node includes elements, text nodes, comments, document, etc.
- Element is a type of node that represents HTML elements only.
- Node type constants: Node.ELEMENT_NODE === 1, Node.TEXT_NODE === 3.

Insert / remove / replace

// create and insert

```
const p = document.createElement("p");
```

```
p.textContent = "Hello";
```

```
el.appendChild(p);          // adds at end
```

```
el.insertBefore(node, reference); // insert before reference
```

// modern helpers

```
el.append(nodeOrString);    // accepts Node or DOMString (string -> TextNode)
```

```
el.prepend(nodeOrString);
```

```
el.replaceWith(newNode);
```

```
el.remove();                // remove element
```

// string insertion (fast for small HTML)

```
el.insertAdjacentHTML("beforeend", "<div>...</div>");
```

// positions: beforebegin, afterbegin, beforeend, afterend

Cloning

```
const clone = el.cloneNode(true); // true => deep clone (children). Note: event listeners are NOT cloned.
```

Styling elements (inline, computed, CSS variables)

Inline style — element.style

```
el.style.backgroundColor = "red";    // camelCase property
```

```
el.style.setProperty("background-color", "red"); // use exact CSS name
```

```
el.style.removeProperty("background-color");
```

- element.style is a CSSStyleDeclaration representing **inline styles only**.

Computed styles — getComputedStyle

```
const cs = window.getComputedStyle(el);  
console.log(cs.width); // computed value (string, e.g., "100px")  
console.log(cs.getPropertyValue("padding-left"));
```

- `getComputedStyle` returns the *applied* style (after CSS selectors, inheritance, defaults).
- Values are strings (often px or rgb(...)).

CSS variables (custom properties)

// set on root so CSS rules can use it

```
document.documentElement.style.setProperty("--main", "#0a0");  
const value = getComputedStyle(document.documentElement).getPropertyValue("--main");
```

Best practices & performance

- Prefer toggling a CSS class (via `.classList`) instead of many inline style changes.
- **Batched updates:** change classes or accumulate updates, then apply — avoid many separate DOM writes.
- Reading layout properties (like `offsetWidth`, `getComputedStyle`) after writes forces reflow. Batch reads then writes.
- For many insertions, use `DocumentFragment` or `innerHTML` once rather than repeated `appendChild`.

Attributes vs Properties (set/get)

Key difference

- **Attributes** are the markup-level variables in HTML (`<input value="a">`).
- **Properties** are the JavaScript object representation on element objects (`el.value`).
- Some attributes reflect to properties and vice-versa, but not always. Example:
 - `` → `el.getAttribute('href') === "/x"` while `el.href` returns an absolute URL (browser-resolved).
 - `<input value="initial">` → `input.getAttribute('value')` returns "initial", while `input.value` is the **current** value.

API

```
el.getAttribute("name")  
el.setAttribute("name", "value")  
el.removeAttribute("name")  
el.hasAttribute("name")
```

```
// iterate attributes
for (let i=0; i < el.attributes.length; i++) {
  const attr = el.attributes[i]; // NamedNodeMap
}
```

data-* attributes (dataset)

```
<div id="card" data-user-id="123" data-flag="true"></div>
const d = document.getElementById("card").dataset;
console.log(d.userId); // "123" (note camelCase mapping)
d.userId = "456"; // updates attribute data-user-id
```

Boolean attributes

- disabled, checked, hidden — presence of attribute means true.
- Use `el.disabled` (property) for boolean behavior; `el.hasAttribute('disabled')` checks presence.

ARIA, custom attributes

- Put accessibility attributes with `setAttribute('aria-label', '...')`.
 - Avoid creating non-standard attributes with the same names as standard ones.
-

classList (managing classes)

Methods

```
el.classList.add("a", "b");
el.classList.remove("a");
el.classList.toggle("hidden"); // toggles
el.classList.toggle("hidden", true); // force add
el.classList.contains("active");
```

```
el.classList.replace("old", "new"); // replace old class with new class
```

- `classList` is a `DOMTokenList` and is safer than `className` string manipulation.
- `el.className = "a b c"` sets the whole class attribute (overwrites).

Best uses

- Use `classList` for show/hide, state toggles, theming (dark-mode), and CSS-driven UI updates.
 - Use `toggle(name, condition)` to explicitly set a state.
-

Useful small recipes (copy-paste)

Toggle dark mode (class + CSS)

```
document.getElementById("toggle").addEventListener("click", () => {  
  document.documentElement.classList.toggle("dark");  
});
```

Change color of all paragraphs

```
document.querySelectorAll("p").forEach(p => p.style.color = "green");
```

Event delegation (efficient)

```
document.querySelector(".list").addEventListener("click", (e) => {  
  const item = e.target.closest(".item");  
  if (!item) return;  
  // handle click on .item even if .item was added later  
});
```

Insert many items efficiently

```
const frag = document.createDocumentFragment();  
for (let i=0; i<1000; i++){  
  const li = document.createElement("li");  
  li.textContent = `Item ${i}`;  
  frag.appendChild(li);  
}  
document.querySelector("ul").appendChild(frag);
```

Safe text insertion (avoid XSS)

```
node.textContent = userInput; // safe  
// avoid: node.innerHTML = userInput; // unsafe if user input not sanitized
```

Common pitfalls & gotchas

- innerHTML executes HTML — **XSS risk**. Use textContent for user strings.
- querySelectorAll returns static list — if you expect live updates, use other APIs or re-query.
- cloneNode(true) does not clone event listeners (attached via addEventListener).
- Changing el.style often causes reflow; minimize repeated writes.

- `getComputedStyle` returns computed values (strings) and may trigger style calculation — avoid heavy use in loops.
 - Difference between `el.parentNode` and `el.parentElement` (`parentElement` ignores Document fragments).
-

Quick cheatsheet (one-line)

- Select: `qS = document.querySelector`, `qSA = querySelectorAll`, `gEBI = getElementById`
 - Create: `document.createElement("div")`
 - Insert: `parent.append(child)`, `parent.prepend(child)`, `ref.insertBefore(node, refNode)`
 - Remove: `el.remove()`
 - Clone: `el.cloneNode(true)`
 - Attribute: `el.getAttribute("x")`, `el.setAttribute("x","y")`
 - Property: `el.value`, `el.checked`
 - Class: `el.classList.add("a")`, `el.classList.toggle("a", cond)`
 - Style inline: `el.style.backgroundColor = "red"`
 - Computed: `getComputedStyle(el).width`
-

Mini practice tasks (try these)

1. Toggle selected class on clicked item using event delegation.
2. Set all image alt attributes that are missing to "image".
3. Add 100 `div.card` elements efficiently (use `fragment`).
4. Read an element's `--main-color` CSS variable and set it to a new hex value.
5. Show the difference: set an `<input>`'s `value` attribute and then change the `.value` property — observe `getAttribute("value")` vs `.value`.

If you want, I can give solutions for any of these right away (I can include code + short explanation).

If you want, I'll:

- convert this into a **one-page printable cheat-sheet**, or
- create a **short interactive practice file** (HTML + JS) that includes the exercises with "show answer" toggles.

Which of those (or which exercise solution) should I give next?