Part 1: DOM Selection Methods

1. `getElementById()`

```
var parag1 = document.getElementById("paraghh")
console.log(parag1);
```

Explanation:

- Selects **ONE element** by its `id` attribute
- Returns: **Element object** OR **null** (if not found)
- ID must be **unique** on the page

W3Schools Reference: Accesses an HTML element with a specific ID.

Scenarios:

```
// ✅ Element exists
var element = document.getElementById("myDiv")
// Returns: <div id="myDiv">...</div>

// ❌ Element doesn't exist
var element = document.getElementById("notExist")
// Returns: null
```

---

2. `getElementsByClassName()`

```
var elements = document.getElementsByClassName("p2")
console.log(elements);
```

Explanation:

- Selects **ALL elements** with the specified class name
- Returns: **HTMLCollection** (array-like but NOT an array)
- **Live collection** = automatically updates when DOM changes

W3Schools Reference: Returns a collection of all elements with a specified class name.

Scenarios:

```
// Multiple elements with same class
var items = document.getElementsByClassName("p2")
// Returns: HTMLCollection [<p>, <p>, <p>]

// Access by index
console.log(items[0]) // First <p> element
console.log(items[1]) // Second <p> element

// Check length
console.log(items.length) // 3

// No elements found
var notFound = document.getElementsByClassName("notExist")
// Returns: HTMLCollection [] (empty collection)
```

Accessing HTMLCollection:

```
var Eles = document.getElementsByClassName("item")

// By index
console.log(Eles[0]) // First element

// By id attribute (if element has id="test")
console.log(Eles["test"]) // Element with id="test"

// By name attribute (if element has name="myDiv")
console.log(Eles["myDiv"]) // Element with name="myDiv"
```

---

3. `getElementsByClassName()` on Specific Parent

```
var div = document.getElementById("parent").getElementsByClassName("p3")
console.log(div);
```

Explanation:

- First, get the parent element by ID
- Then search for class **ONLY inside that parent**
- More specific and faster than searching entire document

Scenarios:

```
<div id="parent">
  <p class="p3">hello world</p>      <!-- ✅ Found -->
  <p class="p3">hello PD track</p>   <!-- ✅ Found -->
</div>
<p class="p3">ddd</p>                <!-- ❌ Not found (outside parent) -->
```

```
var parent = document.getElementById("parent")
var insideParent = parent.getElementsByClassName("p3")
// Returns: HTMLCollection [<p>, <p>] (only 2 elements inside parent)

var allP3 = document.getElementsByClassName("p3")
// Returns: HTMLCollection [<p>, <p>, <p>] (all 3 elements)
```

---

4. `getElementsByTagName()`

```
var ele = document.getElementsByTagName("span")
console.log(ele);
```

Explanation:

- Selects **ALL elements** with the specified tag name
- Returns: **HTMLCollection**
- Case-insensitive: "span" = "SPAN"

W3Schools Reference: Returns a collection of all elements with the specified tag name.

Scenarios:

```
// Get all span elements
var spans = document.getElementsByTagName("span")
// Returns: HTMLCollection [<span>, <span>, <span>]

// Get all div elements
var divs = document.getElementsByTagName("div")

// Get all elements (wildcard)
var allElements = document.getElementsByTagName("*")
```

---

5. `getElementsByName()`

```
var elements = document.getElementsByName("choose")
console.log(elements);
```

Explanation:

- Selects elements by their `name` attribute
- Returns: **NodeList** (NOT HTMLCollection)
- Commonly used for **radio buttons** and **checkboxes**

W3Schools Reference: Returns a NodeList of elements with a specified name.

Scenarios:

```
<input type="radio" name="choose" value="yes" />
<input type="radio" name="choose" value="no" />
<input type="radio" name="choose" value="none" />
```

```
var inputs = document.getElementsByName("choose")
// Returns: NodeList [input, input, input]

// Loop through to find checked radio
for(var i = 0; i < inputs.length; i++){
    if(inputs[i].checked){
        console.log(inputs[i].value) // Logs selected value
    }
}
```

---

6. `querySelector()`

```
var ele = document.querySelector("#parent")
var ele2 = document.querySelector(".span1h")
console.log(ele);
console.log(ele2);
```

Explanation:

- Selects **FIRST element** matching CSS selector
- Returns: **Element** OR **null**

- More flexible than getElementById (can use any CSS selector)

W3Schools Reference: Returns the first element that matches a CSS selector.

Scenarios:

```
// By ID
var byId = document.querySelector("#parent")
// Same as: document.getElementById("parent")

// By Class
var byClass = document.querySelector(".span1h")
// Returns ONLY first element with class="span1h"

// By Tag
var byTag = document.querySelector("p")

// Complex selectors
var complex = document.querySelector("div > p.info")
var attribute = document.querySelector("input[type='radio']")
var pseudo = document.querySelector("li:first-child")
```

---

7. `querySelectorAll()`

```
var Elements = document.querySelectorAll(".span1")
console.log(Elements);
console.log(Elements[0]);
```

Explanation:

- Selects **ALL elements** matching CSS selector
- Returns: **NodeList** (static, NOT live)
- Most powerful selection method

W3Schools Reference: Returns all elements matching a CSS selector.

Scenarios:

```
// Get all elements with class="span1"
var spans = document.querySelectorAll(".span1")
// Returns: NodeList [<span>, <span>, <span>]
```

```
// Access by index
console.log(spans[0]) // First span
console.log(spans[1]) // Second span

// Can only access by index (NOT by id or name like HTMLCollection)
console.log(spans["span2"]) // undefined ❌
console.log(spans["span3"]) // undefined ❌

// Complex selectors
var divParagraphs = document.querySelectorAll("div p")
var checked = document.querySelectorAll("input:checked")
```

Part 2: HTMLCollection vs NodeList

HTMLCollection:

```
var collection = document.getElementsByClassName("item")
```

Properties:

- **Access methods:** index, id, name

```
  collection[0]        // By index
  collection["test"]   // By id attribute
  collection["myDiv"]  // By name attribute
```

- **Returns:** Only **element nodes** (tags)
- **Live:** Auto-updates when DOM changes
- **No forEach:** Must use regular for loop

Example:

```
var items = document.getElementsByClassName("item")
console.log(items.length) // 2

// Add new element with class="item"
var newDiv = document.createElement("div")
newDiv.className = "item"
document.body.appendChild(newDiv)
```

```
console.log(items.length) // 3 (automatically updated!) ✅
```

NodeList:

```
var nodeList = document.querySelectorAll(".span1")
```

Properties:

- **Access methods:** Only by index

```
  nodeList[0]  // ✅ Works
  nodeList["span2"]  // ❌ undefined
```

- **Returns:** Element nodes, text nodes, comment nodes
- **Static:** Does NOT update when DOM changes (for querySelectorAll)
- **Has forEach:** Can use forEach method

Example:

```
var items = document.querySelectorAll(".item")
console.log(items.length) // 2

// Add new element
var newDiv = document.createElement("div")
newDiv.className = "item"
document.body.appendChild(newDiv)

console.log(items.length) // Still 2 (NOT updated!) ❌

// Must query again
items = document.querySelectorAll(".item")
console.log(items.length) // Now 3 ✅
```

Part 3: Navigating DOM Tree

1. `children` vs `childNodes`

```
var ele1 = document.getElementById("parent").children
var ele2 = document.getElementById("parent").childNodes
console.log(ele1);
console.log(ele2);
```

Explanation:

`children` **(HTMLCollection):**

- Returns **only element nodes** (tags)
- Ignores text nodes and comments

`childNodes` **(NodeList):**

- Returns **all nodes:** elements, text, comments
- Includes whitespace as text nodes

Scenario:

```
<div id="parent">
  <!-- Comment -->
  Hello World
  <span>Span</span>
</div>
```

```
var parent = document.getElementById("parent")

console.log(parent.children)
// HTMLCollection [<span>] (only element)

console.log(parent.childNodes)
// NodeList [comment, text, <span>, text]
// Includes comment, whitespace text, span element, whitespace text
```

Part 4: Document Properties

```
console.log(document.body);    // <body> element
console.log(document.links);   // All <a> elements with href
console.log(document.images);  // All <img> elements
console.log(document.forms);   // All <form> elements
```

W3Schools Reference: Document object properties to access specific element collections.

Scenarios:

```
// Get first image
var firstImage = document.images[0]

// Get all links
var allLinks = document.links
for(var i = 0; i < allLinks.length; i++){
    console.log(allLinks[i].href)
}

// Access body directly
document.body.style.backgroundColor = "lightblue"
```

Part 5: First/Last Child Navigation

javascript

```
var ele = document.getElementById("parent")
console.log(ele.firstElementChild);  // First element child
console.log(ele.lastElementChild);   // Last element child
console.log(ele.firstChild);         // First child (any node)
console.log(ele.lastChild);          // Last child (any node)
```

Differences:

Property	Returns	Ignores Whitespace
`firstElementChild`	First **element** only	✅ Yes
`lastElementChild`	Last **element** only	✅ Yes
`firstChild`	First **any node**	❌ No
`lastChild`	Last **any node**	❌ No

Scenario:

html

```
<div id="parent">
  <h1>Hello world</h1>
  <p>Hello world</p>
  <span>Hello world</span>
</div>
```

javascript

```
var parent = document.getElementById("parent")

console.log(parent.firstElementChild)
// <h1>Hello world</h1> ✅

console.log(parent.firstChild)
// text node (whitespace) ⚠️

console.log(parent.lastElementChild)
// <span>Hello world</span> ✅

console.log(parent.lastChild)
// text node (whitespace) ⚠️
```

Part 6: innerHTML vs innerText vs textContent

javascript

```
var parag = document.getElementById("demo")
console.log(parag.innerHTML)
console.log(parag.innerText)
console.log(parag.textContent)
```

HTML:

html

```
<p id="demo">
  Hello <strong>World</strong>
</p>
```

Comparison Table:

Property	Returns	Includes HTML Tags	Preserves Spaces	Triggers Reflow
`innerHTML`	HTML + Text	✅ Yes	✅ Yes	✅ Yes
`innerText`	Text only	❌ No	❌ No (trims)	✅ Yes
`textContent`	Text only	❌ No	✅ Yes	❌ No

Output:

javascript

```
console.log(parag.innerHTML)
// "Hello <strong>World</strong>"

console.log(parag.innerText)
// "Hello World"

console.log(parag.textContent)
// "  Hello World  " (with extra spaces)
```

Setting Content:

javascript

```
// innerHTML - Parses HTML tags
parag.innerHTML = "<b>Hello</b>"
// Result: Hello (bold) ✅

// innerText - Treats as plain text
parag.innerText = "<b>Hello</b>"
// Result: <b>Hello</b> (shows tags as text) ✅

// textContent - Treats as plain text
parag.textContent = "<b>Hello</b>"
// Result: <b>Hello</b> (shows tags as text) ✅
```

⚠️ Security Warning: XSS (Cross-Site Scripting)

javascript

```
// DANGEROUS ❌ - Can execute malicious code
parag.innerHTML = "<button onclick=\"alert('hacked')\">Click me</button>"
// Button is created and onclick will execute!

// SAFE ✅ - Displays as text
parag.textContent = "<button onclick=\"alert('hacked')\">Click me</button>"
// Shows the HTML code as text, doesn't execute
```

Best Practice:

- Use `textContent` or `innerText` for user-generated content
- Only use `innerHTML` when you control the content

Part 7: Working with Input Values

javascript

```
var input = document.getElementById("input1")
var parag = document.getElementsByClassName("p1")[0]

function execute(){
    parag.textContent = input.value
    console.log(input.value)
}
```

HTML:

html

```
<input type="text" id="input1" placeholder="Enter your name"
oninput="execute()" />
<p class="p1"></p>
```

Explanation:

- `input.value` gets the current text in input field
- `oninput` event fires every time user types
- Updates paragraph in real-time

Scenarios:

javascript

```
// Get input value
var userInput = input.value
console.log(userInput) // Whatever user typed

// Set input value
input.value = "Default text"

// Clear input
input.value = ""

// Check if empty
if(input.value === ""){
    alert("Please enter something")
}
```

Part 8: Radio Buttons - Finding Checked Value

javascript

```
var inputs = document.getElementsByName("choose")

function showAnswer(){
    for(var i = 0; i < inputs.length; i++){
        if(inputs[i].checked){
            console.log(inputs[i].value)
        }
    }
}
```

HTML:

html

```
<input type="radio" name="choose" value="yes" />Yes
<input type="radio" name="choose" value="no" />No
<input type="radio" name="choose" value="none" />None
<button onclick="showAnswer()">Show Answer</button>
```

Explanation:

- All radio buttons with same `name` are grouped
- Only ONE can be checked at a time
- `.checked` property returns `true` or `false`

Scenarios:

javascript

```
// Find checked radio button
for(var i = 0; i < inputs.length; i++){
    if(inputs[i].checked){
        console.log(inputs[i].value) // Logs: "yes", "no", or "none"
    }
}

// Check specific radio programmatically
inputs[0].checked = true // Selects first radio

// Uncheck all
for(var i = 0; i < inputs.length; i++){
    inputs[i].checked = false
}
```

Part 9: Styling Elements with JavaScript

Method 1: Direct Style Properties

javascript

```
var para = document.getElementById("parag1")

function changeStyle(){
    para.style.backgroundColor = "red"
    para.style.color = "white"
    para.style.padding = "2rem"
}
```

Note: Use camelCase for CSS properties

- `background-color` → `backgroundColor`

- `font-size` → `fontSize`

Method 2: cssText

javascript

```
para.style.cssText = 'background-color:red; color:white; padding:2rem'
```

Advantage: Set multiple styles at once **Warning:** Overwrites all existing inline styles

---

Method 3: className

javascript

```
console.log(para.className) // "info test"

// Replace all classes
para.className = "jsClass"

// Add to existing classes
para.className += " jsClass"
```

Warning: `className` works with strings, easy to overwrite accidentally

---

Method 4: classList (BEST METHOD) ✅

javascript

```
console.log(para.classList) // DOMTokenList ["info", "test"]

// Add class
para.classList.add('jsClass')

// Remove class
para.classList.remove('test')

// Toggle class (add if not present, remove if present)
para.classList.toggle('test')
```

```
// Check if class exists
if(para.classList.contains('info')){
    console.log("Has info class")
}
```

W3Schools Reference: classList property provides methods to add/remove/toggle classes.

Scenarios:

javascript

```
// Add multiple classes
para.classList.add('class1', 'class2', 'class3')

// Remove multiple classes
para.classList.remove('class1', 'class2')

// Toggle for animations
button.addEventListener('click', function(){
    menu.classList.toggle('active') // Show/hide menu
})
```

Part 10: Working with Attributes

getAttribute() / setAttribute()

javascript

```
var img = document.images[0]

// Get attribute value
console.log(img.getAttribute("src"))
console.log(img.getAttribute("alt"))

// Set attribute value
img.setAttribute("alt", "Image")
img.setAttribute("src", "images/coffee.jpg")
```

W3Schools Reference: getAttribute() returns attribute value, setAttribute() sets it.

---

hasAttribute() / removeAttribute()

javascript

```
var a = document.links[0]

a.setAttribute("href", "https://www.google.com")
a.setAttribute("target", "_blank")

function changeBackground(){
    if(a.hasAttribute("class")){
        a.removeAttribute("class")
    }
    else{
        a.setAttribute("class", "info")
    }
}
```

Methods:

- `hasAttribute("attr")` - Checks if attribute exists → true/false
- `removeAttribute("attr")` - Removes attribute completely
- `toggleAttribute("attr")` - Adds if absent, removes if present

Scenarios:

javascript

```
// Check before removing
if(element.hasAttribute("data-id")){
    element.removeAttribute("data-id")
}

// Toggle disabled attribute
button.toggleAttribute("disabled")

// Custom data attributes
element.setAttribute("data-user-id", "12345")
console.log(element.getAttribute("data-user-id"))
```

Part 11: Creating and Manipulating Elements

createElement() and createTextNode()

javascript

```
var div1 = document.getElementById("parent")
var p1 = document.createElement("p")
var p2 = document.createElement("p")
```

Explanation:

- `createElement("tag")` creates new element in memory
- Element is NOT in DOM yet until you append it

---

appendChild() (Old Method)

javascript

```
var text = document.createTextNode("Hello SD")
p1.appendChild(text) // ✅ Works with nodes

// p1.appendChild("Hello SD") // ❌ ERROR - needs node, not string

div1.appendChild(p1) // Adds to DOM
```

W3Schools Reference: appendChild() adds a node as the last child.

Limitations:

- Only accepts **Node objects**
- Cannot pass strings directly

---

append() (Modern Method) ✅

javascript

```
p1.append("Hello Sd p1") // ✅ Works with strings
p2.append("Hello Pd p2")

div1.append(p1) // Adds to DOM
```

Advantages over appendChild:

- Accepts **strings** directly
- Can append **multiple items** at once
- More flexible

javascript

```
div.append(p1, "Some text", p2) // Multiple at once
```

Positioning Elements

javascript

```
// Remove element
p1.remove()

// Insert before element
p1.before(p2) // p2 comes before p1

// Insert after element
p1.after(p2) // p2 comes after p1
```

Part 12: insertAdjacentElement() - Advanced Positioning

javascript

```
var div1 = document.getElementById("parent")
var p3 = document.createElement("p")
var p4 = document.createElement("p")
var p5 = document.createElement("p")
var p6 = document.createElement("p")

p3.innerHTML = "AfterBegin"
p4.innerHTML = "BeforeBegin"
p5.innerHTML = "Beforeend"
p6.innerHTML = "AfterEnd"
```

```
div1.insertAdjacentElement("afterbegin", p3)
div1.insertAdjacentElement("beforebegin", p4)
div1.insertAdjacentElement("beforeend", p5)
div1.insertAdjacentElement("afterend", p6)
```

W3Schools Reference: insertAdjacentElement() inserts element at specified position.

Visual Explanation:

html

```
<!-- beforebegin (p4) -->
<div id="parent">
    <!-- afterbegin (p3) -->

    <p>Original content</p>
    <span>Original span</span>

    <!-- beforeend (p5) -->
</div>
<!-- afterend (p6) -->
```

Result:

html

```
<p>BeforeBegin</p>          <!-- p4: before parent -->
<div id="parent">
    <p>AfterBegin</p>       <!-- p3: first child -->
    <p>Original content</p>
    <span>Original span</span>
    <p>Beforeend</p>        <!-- p5: last child -->
</div>
<p>AfterEnd</p>             <!-- p6: after parent -->
```

Position Options:

Position	Location	Inside Parent?
`beforebegin`	Before opening tag	❌ No (sibling)
`afterbegin`	After opening tag	✅ Yes (first child)
`beforeend`	Before closing tag	✅ Yes (last child)
`afterend`	After closing tag	❌ No (sibling)

📊 Summary Table: All Selection Methods

Method	Returns	Live?	Access By
`getElementById()`	Element / null	N/A	Single element
`getElementsByClassName()`	HTMLCollection	✅ Yes	index, id, name
`getElementsByTagName()`	HTMLCollection	✅ Yes	index, id, name
`getElementsByName()`	NodeList	❌ No	index only
`querySelector()`	Element / null	N/A	Single element
`querySelectorAll()`	NodeList	❌ No	index only

🎯 Best Practices:

1. **Use** `querySelector/querySelectorAll` for flexibility
2. **Use** `classList` instead of `className` for classes
3. **Use** `textContent` for user input to prevent XSS
4. **Use** `append()` instead of `appendChild()` (modern)
5. **Check if element exists** before manipulating

javascript

```
    var element = document.getElementById("myId")
    if(element){
        element.style.color = "red"
    }
```

BY. Abdullah Ali

Contact : +201012613453