**🌱 Basics of XPath & CSS Selectors**

**1. XPath Basics**

|  |  |  |
| --- | --- | --- |
| **Concept** | **Example** | **Meaning** |
| Absolute XPath | /html/body/div[1]/ul/li[2]/a | Starts from root, full path |
| Relative XPath | //div[@id='login'] | Starts from anywhere in DOM |
| By Attribute | //input[@id='username'] | Locate element by attribute |
| By Text | //button[text()='Login'] | Match exact text |
| Using contains() | //input[contains(@placeholder,'user')] | Partial match |
| Using starts-with() | //input[starts-with(@id,'user\_')] | Match beginning of string |
| Multiple Attributes | //input[@id='user' and @type='text'] | Match 2 conditions |
| Wildcard \* | //\*[@id='email'] | Matches any tag with attribute |
| Indexing | (//input[@type='text'])[2] | Selects 2nd matching element |
| Parent to Child | //div[@id='form']/input | Direct child |
| Ancestor | //input[@id='user']/ancestor::form | Finds parent form |
| Following-Sibling | //label[text()='User']/following-sibling::input | Next element after label |

**2. CSS Selector Basics**

|  |  |  |
| --- | --- | --- |
| **Concept** | **Example** | **Meaning** |
| By ID | #username | Element with id="username" |
| By Class | .btn-primary | Element with class "btn-primary" |
| By Attribute | input[type='text'] | Input field of type text |
| Attribute Partial Match | input[placeholder\*='user'] | Contains text |
| Attribute Starts With | input[id^='user\_'] | Begins with user\_ |
| Attribute Ends With | input[id$='\_name'] | Ends with \_name |
| Tag with Class | button.login-btn | <button class="login-btn"> |
| Descendant | div#form input | All inputs inside form div |
| Direct Child | div#form > input | Direct input child only |
| nth-of-type | ul li:nth-of-type(2) | 2nd list item |
| nth-child | table tr:nth-child(3) | 3rd row of table |
| Sibling (+) | label + input | First input after label |
| General Sibling (~) | h2 ~ p | All <p> after <h2> |

⚡️ **Tip for interviews:**

* Always **prefer CSS selectors** for speed in Selenium/Cypress (unless you need advanced functions like text() which only XPath supports).
* Practice writing **both XPath and CSS** for the same element (common interview trick).

**📌 Exercises – Part 1 (Basics)**

👉 Use this sample DOM for practice (same one we’ve been using):

<div id="main" class="container">

<h1>Welcome to Demo Page</h1>

<ul id="menu">

<li class="item active"><a href="/home">Home</a></li>

<li class="item"><a href="/about">About</a></li>

<li class="item"><a href="/services">Services</a></li>

<li class="item"><a href="/contact">Contact</a></li>

</ul>

<div class="content">

<p class="intro">This is <span>Demo</span> content.</p>

<p class="desc">XPath vs CSS Selectors</p>

<form id="loginForm">

<input type="text" name="username" placeholder="Enter Username"/>

<input type="password" name="password" placeholder="Enter Password"/>

<button type="submit" class="btn primary">Login</button>

<button type="reset" class="btn secondary">Reset</button>

</form>

</div>

</div>

**Exercise Questions (Basics)**

1. Write an **absolute XPath** to locate the <h1> element.
2. Write a **relative XPath** to locate the “About” link inside the <ul>.
3. Write a CSS selector to locate the <ul> with id menu.
4. Write XPath to select the **first <li> element** under menu.
5. Write XPath to select the **last <li> element** under menu.
6. Write CSS to locate the “Contact” <a> link.
7. Write XPath using // to select all <p> tags inside the content div.
8. Write XPath to select the **second <input> field** inside the form.
9. Write CSS to select the login <button> by class.
10. Write XPath to select <span> containing the text “Demo”.

**📌 Part 2: Intermediate XPath & CSS Functions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Syntax** | **Example** | **Usage / Notes** |
| **text()** | //tag[text()='value'] | //button[text()='Login'] | Matches exact text. Case-sensitive. |
| **contains()** | //tag[contains(@attr,'value')] | //input[contains(@id,'user')] | Handles dynamic attributes. Useful for partial matches. |
| **starts-with()** | //tag[starts-with(@attr,'value')] | //input[starts-with(@name,'emp')] | Matches elements where attribute starts with given string. |
| **normalize-space()** | //tag[normalize-space(text())='value'] | //a[normalize-space(text())='Home'] | Ignores leading/trailing spaces. Good for messy HTML. |
| **and / or** | //tag[@attr1='v1' and @attr2='v2'] | //input[@type='text' and @name='email'] | Combine multiple conditions. |
| **position()** | (//tag)[position()=n] or (//tag)[n] | (//div[@class='menu'])[2] | Select nth element from a set. |
| **last()** | (//tag)[last()] | (//input[@type='text'])[last()] | Picks last matching element. |
| **count()** | count(//tag[@attr='v']) | count(//tr[@class='row']) | Returns number of matching nodes (used in validation). |
| **not()** | //tag[not(@attr='v')] | //input[not(@type='hidden')] | Excludes elements. |
| **string()** | string(//tag) | string(//h1) | Converts node to string (text). Useful in validation. |

**🌐 CSS Equivalent Selectors**

|  |  |
| --- | --- |
| **XPath** | **CSS Equivalent** |
| //tag[@id='value'] | tag#value |
| //tag[@class='value'] | tag.value |
| //tag[contains(@class,'val')] | [class\*='val'] |
| //tag[starts-with(@id,'val')] | [id^='val'] |
| //tag[substring(@id,string-length(@id)-3)='val'] | [id$='val'] |
| (//tag)[n] | tag:nth-of-type(n) |
| //tag[not(@attr)] | tag:not([attr]) |

✅ These cover **dynamic element handling** + **common functions** → almost every Selenium / Cypress / Playwright interview includes at least 2–3 from this set.

**📌 Exercises – Part 2 (Functions & Conditions)**

👉 Use this sample DOM for practice (same one we’ve been using):

<div id="main" class="container">

<h1>Welcome to Demo Page</h1>

<ul id="menu">

<li class="item active"><a href="/home">Home</a></li>

<li class="item"><a href="/about">About</a></li>

<li class="item"><a href="/services">Services</a></li>

<li class="item"><a href="/contact">Contact</a></li>

</ul>

<div class="content">

<p class="intro">This is <span>Demo</span> content.</p>

<p class="desc">XPath vs CSS Selectors</p>

<form id="loginForm">

<input type="text" name="username" placeholder="Enter Username"/>

<input type="password" name="password" placeholder="Enter Password"/>

<button type="submit" class="btn primary">Login</button>

<button type="reset" class="btn secondary">Reset</button>

</form>

</div>

</div>

**Exercise Questions**

1. Write XPath using contains() to locate the <p> element with text containing **“XPath”**.
2. Write XPath using starts-with() to locate the <input> field where the placeholder starts with **“Enter User”**.
3. Write XPath using normalize-space() to locate the <button> with text = “Login” (handle extra spaces).
4. Write XPath using multiple conditions to locate the <button> with type='submit' **and** class='btn primary'.
5. Write XPath to select the **last <li> element** under the <ul id="menu">.
6. Write XPath using position() to select the **2nd <li>** element under the <ul id="menu">.
7. Write XPath that uses count() to verify the <ul> with id menu contains **4 <li> elements**.
8. Write XPath that selects the <input> with name='username' OR with placeholder='Enter Username'.
9. Write XPath to get the <p> element where normalize-space(text()) = “XPath vs CSS Selectors”.

**📌 Part 3: Advanced XPath Axes (with Unified DOM Example)**

**🏗 Sample DOM Snippet**

<html>

<body>

<div id="container">

<h1>Main Title</h1>

<div class="section">

<h2>Section 1</h2>

<ul>

<li id="item1">Item 1</li>

<li id="item2">Item 2</li>

<li id="item3">Item 3</li>

</ul>

</div>

<div class="section">

<h2>Section 2</h2>

<p class="desc">Description here</p>

<a href="#" id="link1">Click Me</a>

</div>

<footer>

<p>Footer Text</p>

</footer>

</div>

</body>

</html>

**📖 XPath Axes Explained with Examples**

|  |  |  |  |
| --- | --- | --- | --- |
| **Axis** | **Meaning** | **Example** | **Output (on unified DOM)** |
| **self::** | Refers to the current node itself | //h2[self::h2] | <h2>Section 1</h2> |
| **parent::** | Selects the parent of the current node | //h2/parent::div | The <div class="section"> that contains <h2> |
| **child::** | Selects direct children of a node | //div[@class='section']/child::h2 | <h2>Section 1</h2> and <h2>Section 2</h2> |
| **descendant::** | Selects all descendants (children, grandchildren, etc.) | //div[@id='container']/descendant::li | All <li> inside container |
| **ancestor::** | Selects all ancestors (parent, grandparent, etc.) | //li[@id='item2']/ancestor::div | Both <div class="section"> and <div id="container"> |
| **ancestor-or-self::** | Current node + all ancestors | //h1/ancestor-or-self::\* | <h1>, <div id="container">, <body>, <html> |
| **following-sibling::** | Siblings after the current node | //li[@id='item1']/following-sibling::li | <li id="item2">, <li id="item3"> |
| **preceding-sibling::** | Siblings before the current node | //li[@id='item3']/preceding-sibling::li | <li id="item1">, <li id="item2"> |
| **following::** | All nodes in the doc after the current node | //h1/following::a | <a id="link1">Click Me</a> |
| **preceding::** | All nodes before the current node | //footer/preceding::h2 | Both <h2> elements |
| **descendant-or-self::** | Current node + all descendants | //div[@class='section'][1]/descendant-or-self::\* | <div>, its <h2>, <ul>, <li>s |
| **namespace::** | Selects all namespace nodes of current node | Rare in HTML; used in XML | Not applicable in this DOM |

**🚀 Key Takeaways for Interviews**

* **Commonly asked axes:** parent, ancestor, child, descendant, following-sibling, preceding-sibling.
* **Tricky ones:** following, preceding (they select *all nodes in document order* not just siblings).
* **namespace** → more relevant in XML, rarely in Selenium/Web UI testing.

**📌 Exercises – Part 3 (Axes & Traversals)**

👉 Use this sample DOM for practice (same one we’ve been using):

<div id="main" class="container">

<h1>Welcome to Demo Page</h1>

<ul id="menu">

<li class="item active"><a href="/home">Home</a></li>

<li class="item"><a href="/about">About</a></li>

<li class="item"><a href="/services">Services</a></li>

<li class="item"><a href="/contact">Contact</a></li>

</ul>

<div class="content">

<p class="intro">This is <span>Demo</span> content.</p>

<p class="desc">XPath vs CSS Selectors</p>

<form id="loginForm">

<input type="text" name="username" placeholder="Enter Username"/>

<input type="password" name="password" placeholder="Enter Password"/>

<button type="submit" class="btn primary">Login</button>

<button type="reset" class="btn secondary">Reset</button>

</form>

</div>

</div>

**Exercise Questions**

1. Write XPath using the **parent axis** to select the <p> parent of <span>Demo</span>.
2. Write XPath using the **ancestor axis** to select the <div> with class container from the <span>Demo</span>.
3. Write XPath using the **following-sibling axis** to select the <p class="desc"> that comes after <p class="intro">.
4. Write XPath using the **preceding-sibling axis** to select the <p class="intro"> before <p class="desc">.
5. Write XPath using the **descendant axis** to select all <li> elements inside <ul id="menu">.
6. Write XPath using the **ancestor-or-self axis** to select the <p> with class intro starting from <span>Demo</span>.
7. Write XPath using the **self axis** to select the <button> with text = “Login”.

**📌 Part 4: Pro-Level XPath & CSS Selectors**

**🔹 1. Chained Functions (XPath)**

|  |  |  |
| --- | --- | --- |
| **Use Case** | **Example** | **Meaning** |
| Combine contains() + text() | //a[contains(text(),'Click') and contains(@href,'#')] | Find link with text containing "Click" and href with #. |
| With starts-with() | //input[starts-with(@name,'emp') and contains(@id,'txt')] | Input whose name starts with "emp" and id contains "txt". |
| With normalize-space() | //button[normalize-space(text())='Submit'] | Handles extra spaces before/after text. |
| Count validation | count(//ul/li) = 3 | Ensures there are exactly 3 <li> in a list. |

**🔹 2. Advanced CSS Selectors**

|  |  |  |
| --- | --- | --- |
| **Selector** | **Example** | **Meaning** |
| **Multiple classes** | button.btn.primary | Button with both btn and primary classes. |
| **Substring match** | [id\*='user'] | Attribute contains “user”. |
|  | [id^='emp'] | Attribute starts with “emp”. |
|  | [id$='txt'] | Attribute ends with “txt”. |
| **Nth child** | ul li:nth-child(2) | 2nd <li> in <ul>. |
| **Nth last child** | ul li:nth-last-child(1) | Last <li> in <ul>. |
| **Only child** | div:only-child | Div that is only child of parent. |
| **First/last of type** | p:first-of-type / p:last-of-type | First/last <p> in parent. |
| **Negation** | input:not([type='hidden']) | All inputs except hidden. |
| **Direct child** | div > p | <p> directly inside <div>. |
| **General sibling** | h2 ~ p | All <p> siblings after <h2>. |
| **Adjacent sibling** | h2 + p | First <p> immediately after <h2>. |

**🔹 3. XPath ↔ CSS Mapping (Advanced Level)**

|  |  |  |
| --- | --- | --- |
| **Requirement** | **XPath** | **CSS** |
| First element | (//div)[1] | div:first-of-type |
| Last element | (//div)[last()] | div:last-of-type |
| nth element | (//li)[3] | li:nth-of-type(3) |
| All except one | //input[not(@type='hidden')] | input:not([type='hidden']) |
| Ends-with | //input[substring(@id, string-length(@id)-2)='txt'] | input[id$='txt'] |
| Starts-with | //input[starts-with(@id,'emp')] | input[id^='emp'] |
| Contains | //div[contains(@class,'menu')] | div[class\*='menu'] |
| Parent-child | //div/p | div p |
| Direct child | //div/child::p | div > p |
| Following sibling | //h2/following-sibling::p | h2 + p (first only) / h2 ~ p (all) |

✅ With this, you now have:

* **Basics** → Absolute/Relative + Simple CSS
* **Intermediate** → Common functions (contains, normalize-space, etc.)
* **Advanced** → Axes (ancestor, sibling, etc.)
* **Pro-Level** → Chained functions + advanced CSS + mapping

**📌 Exercises – Part 4 (CSS Specific)**

👉 Use this sample DOM for practice (same one we’ve been using):

<div id="main" class="container">

<h1>Welcome to Demo Page</h1>

<ul id="menu">

<li class="item active"><a href="/home">Home</a></li>

<li class="item"><a href="/about">About</a></li>

<li class="item"><a href="/services">Services</a></li>

<li class="item"><a href="/contact">Contact</a></li>

</ul>

<div class="content">

<p class="intro">This is <span>Demo</span> content.</p>

<p class="desc">XPath vs CSS Selectors</p>

<form id="loginForm">

<input type="text" name="username" placeholder="Enter Username"/>

<input type="password" name="password" placeholder="Enter Password"/>

<button type="submit" class="btn primary">Login</button>

<button type="reset" class="btn secondary">Reset</button>

</form>

</div>

</div>

**Exercise Questions**

1. Write a CSS selector to select the <div> with **id = "main"**.
2. Write a CSS selector to select the <ul> element with **id = "menu"**.
3. Write a CSS selector to select all <li> elements with **class = "item"**.
4. Write a CSS selector to select the **active <li> element** (class = item active).
5. Write a CSS selector to select the **3rd <li> element** inside #menu.
6. Write a CSS selector using :nth-of-type() to select the **2nd <p> element** inside .content.
7. Write a CSS selector using :not() to select all <button> elements except the one with **class = "secondary"**.
8. Write a CSS selector to select the <input> element with **type = password**.
9. Write a CSS selector to select the **last <li> element** inside #menu.

**📌 Part 5: Quick Revision Cheat Sheet (All-in-One)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Level** | **Feature** | **XPath Example** | **CSS Example** | **Notes** |
| **Basic** | Absolute path | /html/body/div[1]/h1 | ❌ | Rigid, breaks with changes. |
| Relative path | //div[@id='container'] | div#container | Preferred. |
| By ID | //input[@id='user'] | input#user | Fastest. |
| By Class | //div[@class='menu'] | div.menu | CSS cleaner. |
| **Intermediate** | Text match | //button[text()='Login'] | ❌ | Case-sensitive. |
| Contains | //input[contains(@id,'user')] | [id\*='user'] | Handles dynamic. |
| Starts-with | //input[starts-with(@name,'emp')] | [name^='emp'] | Prefix match. |
| Ends-with | //input[substring(@id,string-length(@id)-2)='txt'] | [id$='txt'] | CSS simpler. |
| Normalize-space | //a[normalize-space(text())='Home'] | ❌ | Cleans spaces. |
| And/Or | //input[@type='text' and @name='email'] | input[type='text'][name='email'] | Combine filters. |
| Position | (//li)[2] | li:nth-of-type(2) | 2nd element. |
| Last | (//li)[last()] | li:last-of-type | Last element. |
|  | Count | count(//tr) | ❌ | Validation in XPath. |
| **Advanced (Axes)** | Parent | //h2/parent::div | ❌ | Go upward. |
|  | Child | //div[@class='section']/child::h2 | div.section > h2 | Direct child. |
|  | Descendant | //div[@id='container']/descendant::li | div#container li | All nested. |
|  | Ancestor | //li[@id='item2']/ancestor::div | ❌ | Go to all parents. |
|  | Following-sibling | //li[@id='item1']/following-sibling::li | li#item1 ~ li | All next siblings. |
|  | Preceding-sibling | //li[@id='item3']/preceding-sibling::li | ❌ | Only XPath supports. |
|  | Following | //h1/following::a | ❌ | All nodes after. |
|  | Preceding | //footer/preceding::h2 | ❌ | All before. |
| **Pro-Level** | Chained funcs | //a[contains(text(),'Click') and contains(@href,'#')] | ❌ | Combine conditions. |
|  | Negation | //input[not(@type='hidden')] | input:not([type='hidden']) | Exclude elements. |
|  | Only child | ❌ | div:only-child | CSS unique. |
|  | First/Last of type | (//p)[1] / (//p)[last()] | p:first-of-type / p:last-of-type | Indexing. |
|  | Adjacent sibling | ❌ | h2 + p | First sibling after. |
|  | General sibling | ❌ | h2 ~ p | All siblings after. |

✅ This table = **your interview-ready revision card** → covers **all levels in one glance**.

**📌 Exercises – Part 5 (XPath vs CSS)**

👉 Using the **same sample DOM** from Part 1.

**Exercise Questions**

1. Write **XPath** and **CSS** to select the <h1> element with text “Welcome to Demo Page”.
2. Write **XPath** and **CSS** to select the <li> element with text “Services”.
3. Write **XPath** to select the <a> element by **text = "Contact"**. Can you do the same with CSS? Why/why not?
4. Write **XPath** and **CSS** to select the **span element inside <p class="intro">**.
5. Write **XPath** to select the **parent <p> of <span>Demo</span>**. Is this possible in CSS?
6. Write **XPath** and **CSS** to select the **2nd <input> element** inside the form.
7. Write **XPath** and **CSS** to select all <li> elements where the class contains “item”.

Perfect 👍 Now we’re at **Part 5 → XPath vs CSS Selector: One Big HTML Example**.  
I’ll use a **unified HTML snippet** and then show how to locate elements both in **XPath** and **CSS**, with notes on when to use each.

**📄 Unified Sample HTML**

<html>

<body>

<div id="main" class="container">

<h1>Welcome to Demo Page</h1>

<ul id="menu">

<li class="item active"><a href="/home">Home</a></li>

<li class="item"><a href="/about">About</a></li>

<li class="item"><a href="/services">Services</a></li>

<li class="item"><a href="/contact">Contact</a></li>

</ul>

<div class="content">

<p class="intro">This is <span>Demo</span> content.</p>

<p class="desc">XPath vs CSS Selectors</p>

<form id="loginForm">

<input type="text" name="username" placeholder="Enter Username"/>

<input type="password" name="password" placeholder="Enter Password"/>

<button type="submit" class="btn primary">Login</button>

<button type="reset" class="btn secondary">Reset</button>

</form>

<table id="userTable">

<thead>

<tr><th>ID</th><th>Name</th><th>Role</th></tr>

</thead>

<tbody>

<tr><td>1</td><td>Alice</td><td>Admin</td></tr>

<tr><td>2</td><td>Bob</td><td>User</td></tr>

<tr><td>3</td><td>Charlie</td><td>Guest</td></tr>

</tbody>

</table>

</div>

</div>

</body>

</html>

**📌 XPath vs CSS Examples**

**1. Select element by ID**

* **XPath** → //div[@id='main']
* **CSS** → div#main or simply #main

✅ CSS is shorter and faster here.

**2. Select element by Class**

* **XPath** → //li[@class='item active']
* **CSS** → li.item.active

✅ CSS is clean when classes are unique.

**3. Select element by Text**

* **XPath** → //a[text()='About']
* **CSS** → ❌ Not possible directly (needs JavaScript).

✅ XPath wins when selecting by **exact text**.

**4. Select element by Partial Text**

* **XPath** → //p[contains(text(),'XPath')]
* **CSS** → ❌ Not supported.

✅ XPath better for **dynamic text matching**.

**5. Select element by Attribute**

* **XPath** → //input[@name='username']
* **CSS** → input[name='username']

✅ Both work fine; CSS is shorter.

**6. Select Nth Child**

* **XPath** → //ul[@id='menu']/li[3]
* **CSS** → #menu li:nth-child(3)

✅ CSS is simpler for ordered elements.

**7. Navigate Parent → Child**

* **XPath** → //div[@class='content']/p[@class='intro']/span
* **CSS** → .content > p.intro > span

✅ Both work well; CSS feels cleaner.

**8. Navigate Child → Parent**

* **XPath** → //span[text()='Demo']/parent::p
* **CSS** → ❌ Not supported (CSS can’t go upward).

✅ XPath is more powerful for **reverse traversal**.

**9. Select All Links**

* **XPath** → //a
* **CSS** → a

✅ Both work equally.

**10. Table Specific Example**

* **XPath** → //table[@id='userTable']//tr[td[text()='Bob']]
* **CSS** → ❌ Very hard (CSS doesn’t handle conditional table text well).

✅ XPath shines in **complex table structures**.

**⚖️ Quick Comparison**

| **Feature** | **XPath ✅** | **CSS ✅** |
| --- | --- | --- |
| By ID / Class / Attribute | ✔️ | ✔️ |
| By Text | ✔️ | ❌ |
| By Partial Text | ✔️ | ❌ |
| Move Upwards (Child → Parent) | ✔️ | ❌ |
| Move Downwards (Parent → Child) | ✔️ | ✔️ |
| Position (nth element) | ✔️ | ✔️ |
| Complex Conditions (AND/OR) | ✔️ | ❌ |
| Readability / Performance | ❌ Longer | ✔️ Shorter |

👉 Rule of Thumb:

* **Use CSS** for speed, simplicity, and stable attributes.
* **Use XPath** for text, parent traversal, or complex conditions.

Would you like me to now prepare a **final condensed 1-page version (all 5 parts merged)** as a **ready-to-print cheat sheet** 📄?

**interview-style questions**

Awesome 🚀 — let’s take **one big sample HTML page** and then create **20 interview-style questions** on XPath & CSS selectors based on it. This way, you get *realistic practice* and can apply the entire cheat sheet.

**🏗 Sample Big HTML (Unified for Questions)**

<html>

<body>

<div id="container">

<h1>Main Title</h1>

<div class="nav">

<ul>

<li><a href="/home" class="active">Home</a></li>

<li><a href="/about">About</a></li>

<li><a href="/contact">Contact</a></li>

</ul>

</div>

<div class="content">

<h2>Section 1</h2>

<p class="desc"> Welcome to Testing </p>

<ul class="items">

<li id="item1">Item 1</li>

<li id="item2" class="highlight">Item 2</li>

<li id="item3">Item 3</li>

</ul>

</div>

<div class="content">

<h2>Section 2</h2>

<p class="desc">Description here</p>

<table id="userTable">

<thead>

<tr>

<th>ID</th><th>Name</th><th>Role</th>

</tr>

</thead>

<tbody>

<tr><td>1</td><td>Alice</td><td>Admin</td></tr>

<tr><td>2</td><td>Bob</td><td>User</td></tr>

<tr><td>3</td><td>Charlie</td><td>Manager</td></tr>

</tbody>

</table>

<a href="#" id="applyLink">Apply Now</a>

<a href="#" id="cancelLink">Cancel</a>

</div>

<footer>

<p>Footer Text</p>

</footer>

</div>

</body>

</html>

**🎯 20 Interview-Style Practice Questions**

**Basics**

1. Write XPath & CSS to locate the **Main Title** (<h1>).
2. Locate the **Home link** using text.
3. Find the **nav <div> by class**.
4. Locate **3rd item in the first <ul> inside .items**.
5. Select the **footer <p> element**.

**Intermediate**

1. Write XPath to find the <p> with text "Welcome to Testing" using normalize-space().
2. Locate the <a> tag whose class contains active.
3. Select the 2nd <li> inside .items using position/nth-child.
4. Count how many rows are in the table body.
5. Locate the last <tr> inside the table using XPath & CSS.

**Advanced (Axes)**

1. From <td>Bob</td>, get its **ancestor <table>**.
2. From <td>Bob</td>, get its **following-sibling <td>** (role).
3. From <td>Charlie</td>, get its **preceding-sibling <td>** (name).
4. From <a id="applyLink">, get the **parent <div class="content">**.
5. From <li id="item2">, get all **preceding siblings**.

**Pro-Level**

1. Locate the **Apply Now link** using contains(text(),'Apply').
2. Locate the **Cancel link** using XPath starts-with(@id,'cancel').
3. Write XPath to find all <a> except the one with id="cancelLink".
4. Locate the **first row’s name cell (Alice)** using XPath and CSS.
5. Locate the **last <a> tag inside .content div**.

👉 This set mimics *exactly how interviewers ask*:

* Start with **simple locators**
* Move to **dynamic handling**
* Then test **axes**
* Finally check **chained/advanced expressions**

**🎯 Key Observations**

* CSS is **cleaner** for class/id/nth-child cases.
* XPath is **stronger** for:
  + Text search (text(), contains(), normalize-space())
  + Axes (ancestor, preceding, following-sibling, parent)
  + Validation (count(), last())

👉 This gives you **questions + answers in one go** → perfect for interview drills.

Would you like me to now make a **separate “Top 10 Trick Questions” set** (like *dynamic table handling, find element without unique attr, 2nd last element, nth occurrence of text, etc.*) — the kind that usually stumps candidates?

Perfect 👌 Here we go — **20 Interview Questions + Answers** (based on our unified big HTML).  
I’ll give **XPath** first, then **CSS equivalent** (if possible).

**✅ Answers to 20 Practice Questions**

**Basics**

1. **Main Title (<h1>)**
   * XPath → //h1[text()='Main Title']
   * CSS → h1
2. **Home link**
   * XPath → //a[text()='Home']
   * CSS → a.active or a[href='/home']
3. **nav <div> by class**
   * XPath → //div[@class='nav']
   * CSS → div.nav
4. **3rd <li> inside .items**
   * XPath → //ul[@class='items']/li[3]
   * CSS → ul.items li:nth-child(3)
5. **footer <p>**
   * XPath → //footer/p
   * CSS → footer p

**Intermediate**

1. **Normalize-space text (Welcome to Testing)**
   * XPath → //p[normalize-space(text())='Welcome to Testing']
   * CSS → ❌ (not supported in CSS)
2. **<a> tag with class contains active**
   * XPath → //a[contains(@class,'active')]
   * CSS → a[class\*='active']
3. **2nd <li> inside .items**
   * XPath → //ul[@class='items']/li[2]
   * CSS → ul.items li:nth-child(2)
4. **Count rows in table body**
   * XPath (validation) → count(//table[@id='userTable']/tbody/tr) = 3
   * CSS → ❌ (CSS can’t count directly)
5. **Last <tr> inside table**

* XPath → //table[@id='userTable']/tbody/tr[last()]
* CSS → #userTable tbody tr:last-of-type

**Advanced (Axes)**

1. **From <td>Bob</td> → ancestor <table>**

* XPath → //td[text()='Bob']/ancestor::table
* CSS → ❌ (CSS doesn’t support ancestor axis)

1. **From <td>Bob</td> → following-sibling (role)**

* XPath → //td[text()='Bob']/following-sibling::td[1] → "User"
* CSS → ❌

1. **From <td>Charlie</td> → preceding-sibling (name)**

* XPath → //td[text()='Charlie']/preceding-sibling::td[1] → "3"
* CSS → ❌

1. **From <a id="applyLink"> → parent <div class="content">**

* XPath → //a[@id='applyLink']/parent::div
* CSS → ❌ (CSS has only parent-child, not reverse)

1. **From <li id="item2"> → all preceding siblings**

* XPath → //li[@id='item2']/preceding-sibling::li
* CSS → ❌

**Pro-Level**

1. **Apply Now link (contains text)**

* XPath → //a[contains(text(),'Apply')]
* CSS → ❌ (text-based search not in CSS)

1. **Cancel link (starts-with id)**

* XPath → //a[starts-with(@id,'cancel')]
* CSS → a[id^='cancel']

1. **All <a> except cancelLink**

* XPath → //a[not(@id='cancelLink')]
* CSS → a:not(#cancelLink)

1. **First row’s name cell (Alice)**

* XPath → //table[@id='userTable']/tbody/tr[1]/td[2]
* CSS → #userTable tbody tr:nth-child(1) td:nth-child(2)

1. **Last <a> tag inside .content div**

* XPath → (//div[@class='content']//a)[last()]
* CSS → div.content a:last-of-type

**🎯 Key Observations**

* CSS is **cleaner** for class/id/nth-child cases.
* XPath is **stronger** for:
  + Text search (text(), contains(), normalize-space())
  + Axes (ancestor, preceding, following-sibling, parent)
  + Validation (count(), last())

👉 This gives you **questions + answers in one go** → perfect for interview drills.

Would you like me to now make a **separate “Top 10 Trick Questions” set** (like *dynamic table handling, find element without unique attr, 2nd last element, nth occurrence of text, etc.*) — the kind that usually stumps candidates?

**📌 XPath & CSS Interview Questions**

**🔹 Basics**

1. What is the difference between **absolute** and **relative** XPath? Which one should you use and why?
2. Can you write an XPath for the **5th <div> element** in the DOM?
3. What is the difference between // and / in XPath?
4. What are CSS selectors? How do they differ from XPath?
5. How do you locate an element by **ID, class, and attribute** using XPath and CSS?
6. Which one is faster in Selenium — CSS or XPath? Why?
7. Can you explain why XPath is not recommended in IE browsers (older versions)?

**🔹 Functions & Conditions**

1. How does contains() work in XPath? Can you write an example for locating a button with **partial text**?
2. What is the difference between contains() and starts-with() in XPath?
3. How does normalize-space() help in locating elements?
4. How do you locate an element using **multiple attributes** in XPath? Example: @id and @class.
5. Write an XPath to select a button with text = “Login” but **ignore case sensitivity**.
6. How do you handle **dynamic elements** in XPath (where attributes change on refresh)?
7. Write an XPath to find the **last element** of a list.
8. How do you use position() and last() together in XPath? Give an example.
9. Explain the usage of count() in XPath.

**🔹 Axes & Traversals**

1. What are XPath **axes**? List at least 5 with examples.
2. How do you find an element’s **parent** using XPath?
3. Write an XPath to locate the **next sibling** of an element.
4. How do you select all **preceding siblings** of an element?
5. What is the difference between ancestor, ancestor-or-self, and parent?
6. Write an XPath that selects all **descendant <span> elements** of a <div> with class content.
7. How would you select the **2nd preceding sibling** of an element in XPath?

**🔹 CSS Specific**

1. How do you select an element by ID in CSS? By class? By attribute?
2. What is the difference between element.class and element > .class in CSS?
3. How do you select the **nth-child** of a list in CSS?
4. What is the difference between :nth-child() and :nth-of-type() in CSS?
5. Can CSS select an element based on its **text content**? Why or why not?
6. How do you select all elements **except** a specific one using CSS (:not pseudo-class)?
7. Write a CSS selector for an <input> element of type password.
8. How do you select the **last element** of a list in CSS?
9. Write a CSS selector to select an <a> tag inside a <li> which itself is inside a <ul id="menu">.

**🔹 XPath vs CSS**

1. What are the advantages of XPath over CSS selectors?
2. What are the advantages of CSS selectors over XPath?
3. Which is better for performance in Selenium: XPath or CSS? Why?
4. Can CSS move from **child → parent**? Why is it a limitation compared to XPath?
5. In which scenarios would you prefer CSS selectors over XPath, and vice versa?
6. Is it possible to locate elements inside an **iframe** using XPath and CSS? How?
7. Which one would you prefer for **cross-browser automation**? Why?

**🔹 Practical / Problem-Solving**

1. Write an XPath to select the **second row, third column** from a table.
2. Write a CSS selector to select the **3rd <li> element** inside a <ul> with ID menu.
3. Write an XPath to locate a <button> element if it contains either text = “Submit” OR “Apply”.
4. How do you locate an element when multiple elements have the same attributes?
5. Write a CSS selector to find all <input> fields where the attribute name starts with “user”.
6. Write an XPath that selects all elements where attribute class contains the word “active”.
7. Write a CSS selector to find all <div> elements that contain a class starting with “col-”.
8. Write an XPath that selects the 2nd <li> after the one that has text = “Services”.
9. Write an XPath to locate a checkbox **next to a label** with text “I agree”.
10. Write a CSS selector for a <form> element with both id="loginForm" and class active.
11. Write an XPath that selects all <tr> elements in a table except the header row.

👉 This covers **50 interview-ready questions** (theory + hands-on coding).  
Would you like me to also **prepare answers with code snippets** for each, or keep it only as a **question bank** for practice?