**Lab Exercise 4- Implementing WCAG 2.1 Robust Features in a Web Page**

**Objective**

Ensure your web content is **robust** — meaning it can be reliably interpreted by a wide variety of user agents, including current and future assistive technologies.

**Pre-requisites**

* Basic HTML and ARIA knowledge
* Familiarity with browser developer tools
* Access to accessibility validation tools (e.g., Axe, WAVE)

**Part 1: Using Valid, Semantic HTML**

**Goal**

Use correct HTML tags and semantic elements so assistive technologies can understand and interpret content properly.

**Steps**

1. Use semantic tags like <header>, <nav>, <main>, <footer>, <article>, <section>, <button>, <form>, etc.

<header>

<h1>My Website</h1>

<nav>

<ul>

<li><a href="#home">Home</a></li>

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

</ul>

</nav>

</header>

<main>

<article>

<h2>Welcome!</h2>

<p>This is the main content.</p>

</article>

</main>

<footer>

<p>&copy; 2025 My Website</p>

</footer>

Validate your HTML with W3C Validator to ensure there are no syntax errors.

**Part 2: Use ARIA Roles and Properties Correctly**

**Goal**

Enhance semantic meaning for custom components using ARIA attributes without misusing or overusing them.

**Steps**

1. For custom interactive components that are not native HTML elements, add ARIA roles:

<div role="button" tabindex="0" aria-pressed="false" onclick="toggle(this)">

Toggle Info

</div>

<script>

function toggle(el) {

const pressed = el.getAttribute('aria-pressed') === 'true';

el.setAttribute('aria-pressed', !pressed);

}

</script>

1. Use ARIA landmarks to aid navigation:

<nav role="navigation">...</nav>

<main role="main">...</main>

<aside role="complementary">...</aside>

**Part 3: Ensure Compatible DOM and Script Behavior**

**Goal**

Ensure dynamic changes (like showing/hiding content) are communicated properly to assistive tech.

**Steps**

1. Use ARIA live regions for dynamic content updates:

<div aria-live="polite" id="status"></div>

<script>

function updateStatus(message) {

document.getElementById('status').textContent = message;

}

</script>

1. Avoid breaking semantic structure with JavaScript.

**Use Case Scenario**

**Use Case**: An online store uses custom dropdown menus and dynamically updates product availability. The store must ensure the menus are accessible and updates are announced to screen reader users.

**Implementation**:

* Custom dropdown uses role="listbox" and aria-expanded attributes.
* Product availability changes update an aria-live region.
* HTML validates cleanly without errors.

**Outcome**:

* Assistive tech reliably interacts with custom components.
* Dynamic updates are announced.
* Website works consistently across browsers and screen readers.

**Accessibility Testing Checklist (Robust)**

| **Feature** | **Test Description** | **Status** |
| --- | --- | --- |
| Valid Semantic HTML | Validate HTML with W3C Validator | ✅ |
| Proper ARIA Roles Used | Roles match element purpose, no misuse | ✅ |
| ARIA Properties Reflect State | aria-pressed, aria-expanded etc. update correctly | ✅ |
| Dynamic Content Announcements | Use of aria-live regions for updates | ✅ |
| Works in Multiple Browsers/AT | Tested with NVDA, JAWS, VoiceOver across browsers | ✅ |

**Conclusion**

This lab exercise focuses on the **Robust** principle by helping developers:

* Write clean, valid HTML markup
* Use ARIA appropriately to enhance accessibility
* Ensure assistive technologies can interpret the content consistently
* Handle dynamic content changes accessibly

Implementing these ensures future compatibility and better user experiences across diverse technologies.