WDA™ – Certified Associate Web Developer EXAM SYLLABUS

(Exam WDA-41-01)

Last revised: (September 15, 2025)

Module 1: HTML Fundamentals (10) (25%)

1.1 Document Structure and Markup Basics (3)

Objective 1.1.1 Identify and apply *DOCTYPE*, *html*, *head*, and *body* tags correctly

A. Place the HTML5 declaration `<!DOCTYPE html>` at the very top of every page.
B. Use a single root `<html lang="...">` element that wraps all content, with language set appropriately.
C. Include a `<head>` section for metadata (character encoding, viewport, title) and a `<body>` for visible content.
D. Ensure the basic skeleton is correctly ordered and closed to avoid quirks mode and parsing errors.

Objective 1.1.2 Incorporate appropriate metadata elements

A. Set character encoding and viewport: <meta charset="utf-8">, `<meta name="viewport" content="width=device-width, initial-scale=1">`.
B. Provide a concise, unique `<title>` and a descriptive `<meta name="description">` for search and sharing.
C. Add social preview metadata, favicons, and `<meta name="robots">` directives when needed.
D. Keep metadata inside `<head>` only; avoid deprecated or duplicate tags.

Objective 1.1.3 Construct well-formed markup

A. Use proper nesting and close all non-void elements; quote attribute values and avoid duplicate IDs.
B. Prefer semantic, standards-compliant elements; avoid obsolete tags and inline presentation.
C. Escape special characters (`&`, `<`, `>`) where required.

1.2 Structured and Semantic HTML Content (3)

Objective 1.2.1 Apply semantic tags like *header*, *footer*, *nav*, *section*, and *article* accurately

A. Use `<header>` for introductory or navigational content and `<nav>` for primary sets of links.
B. Structure pages with `<main>` (one per page), grouping related content with `<section>` and standalone pieces with `<article>`.
C. Complement with `<aside>` for tangential content and `<footer>` for page or section endings.

Objective 1.2.2 Construct well-organized content using tables, lists, and paragraphs in HTML

A. Reserve `<table>` for tabular data; include `<caption>`, `<thead>`, `<tbody>`, `<tfoot>`, and scope on `<th>`.
B. Use `<ul>`/`<ol>` with `<li>` for lists and `<dl>`, `<dt>`, `<dd>` for term-definition pairs.
C. Write text in `<p>` paragraphs and avoid using tables or multiple `<br>` for layout or spacing.

Objective 1.2.3 Implement headings, thematic breaks, and line breaks to structure content properly

A. Use a logical heading hierarchy (`<h1>` to `<h6>`), typically one `<h1>` per page.
B. Apply `<hr>` for thematic shifts between sections and `<br>` for meaningful single line breaks (e.g., addresses, poetry).

1.3 Media, Forms, and Navigation (4)

Objective 1.3.1 Integrate multimedia content using *img*, *video*, and *audio* tags effectively

A. Provide informative `alt` text for images; use `width`/`height` to reduce layout shift and `srcset`/`sizes` for responsiveness.
B. Wrap media in `<figure>` with `<figcaption>` when a caption is needed.

C. Enable controls for `<video>`/`<audio>`, add captions/subtitles via `<track kind="captions">`, and provide fallback text.
D. Use appropriate loading strategies (e.g., `loading="lazy"` for images) and consider performance of media assets.

Objective 1.3.2 Embed external content using iframes and ensure it is responsive and accessible

A. Include a descriptive `title` attribute for `<iframe>` content and allow fullscreen where appropriate.
B. Apply security-related attributes like `sandbox` and `referrerpolicy` as needed.
C. Make embeds responsive using CSS (e.g., `aspect-ratio` or a wrapper that preserves proportions).

Objective 1.3.3 Implement *form* elements accurately to interact with users and collect data

A. Choose appropriate native controls – `<input>` (e.g., `email`, `number`, `date`, `file`), `<select>`/`<option>`/`<optgroup>`, `<textarea>`, and `<button type="submit|reset|button">` – with meaningful `name` and `id` attributes.
B. Use `<datalist>` for suggestions and `<output>` for computed values.
C. Associate each control with a `<label>` (via `for`/`id` or by wrapping), group related fields with `<fieldset>` and a descriptive `<legend>`, and link help/error text using aria-describedby.
D. Configure the `<form>` with the appropriate `method` (`GET` for idempotent queries, `POST` for data changes), `action`, and `enctype` (e.g., `multipart/form-data` for file uploads); set `autocomplete` and use `novalidate` only when justified.
E. Leverage native HTML5 constraints (`required`, `pattern`, `min`, `max`, `step`, `minlength`/`maxlength`, `multiple`) and entry aids (`inputmode`, `placeholder`) to improve data integrity without scripting.

Objective 1.3.4 Develop navigational structures using hyperlinks, ensuring proper linking and user-friendly URL structures

A. Use clear, descriptive link text within `<a href>`.
B. Organize site navigation with lists inside `<nav>`, add breadcrumb and skip links where appropriate.
C. Use meaningful and human-readable URLs.
D. Use relative and absolute links appropriately.
E. Apply `rel="noopener noreferrer"` when using `target="_blank"` to improve security and performance.

Module 2: HTML Fundamentals (9) (22.5%)

2.1 Core CSS Concepts (3)

Objective 2.1.1 Recognize and use standard CSS syntax, selectors, properties, and values correctly

A. Write valid declarations using the pattern `selector { property: value; }` and end each rule with a semicolon.
B. Use common selectors (type, class, ID, attribute, pseudo-classes, pseudo-elements) appropriately and avoid overly broad matches.
C. Apply core properties for the Box Model, typography, colors, and sizing with consistent units (`rem`, `em`, %, `px`).

Objective 2.1.2 Manage the CSS cascade, understand and apply specificity and inheritance of styles effectively

A. Explain how source order, specificity, and `!important` determine the final computed style.
B. Calculate specificity for compound selectors and reduce it when possible to prevent conflicts.
C. Leverage inheritance for text properties and override intentionally with explicit rules.
D. Adopt naming conventions (e.g., BEM) to minimize collisions and improve predictability.

Objective 2.1.3 Apply basic CSS styling to HTML elements

A. Style text with font families, sizes, weights, line heights, and spacing for readability.
B. Control colors, backgrounds, borders, and basic spacing using margin and padding.
C. Use the Box Model intentionally, including `box-sizing: border-box;` for layout consistency.
D. Create simple utility classes for common patterns (e.g., visually hidden, text centering).

2.2 Layout, Effects, and Positioning (3)

Objective 2.2.1 Manipulate layout and appearance of elements using advanced CSS properties

A. Control display contexts with `display`, intrinsic sizing, overflow, and aspect ratio.
B. Use modern sizing and spacing techniques (`min()`, `max()`, `clamp()`, logical properties) for responsive designs.
C. Enhance presentation with shadows, rounded corners, filters, and blend modes judiciously.
D. Apply custom properties (CSS variables) to centralize design tokens and reduce repetition.

Objective 2.2.2 Implement dynamic styles and transitions using CSS3 properties

A. Create smooth state changes with `transition` on opacity, transform, and color.
B. Animate with `@keyframes` and `animation` while preserving accessibility and performance.
C. Use `transform` (`translate`, `scale`, `rotate`) and `will-change` when appropriate to improve rendering.
D. Respect user preferences such as `prefers-reduced-motion` to reduce motion effects.

Objective 2.2.3 Apply various CSS positioning schemes to control layout flow and appearance

A. Differentiate between `static`, `relative`, `absolute`, `fixed`, and `sticky` positioning.
B. Use offsets (`top`, `right`, `bottom`, `left`) intentionally and understand containing blocks.
C. Manage stacking contexts and `z-index` to resolve overlap predictably.

2.3 Frameworks, Preprocessors, and Performance (3)

Objective 2.3.1 Utilize CSS frameworks like Bootstrap for rapid, responsive design development

A. Include framework CSS correctly and use the grid system, utilities, and components as designed.
B. Override framework styles with minimal, scoped additions to preserve upgradability.
C. Customize themes via variables/tokens and build steps when available.

Objective 2.3.2 Implement CSS preprocessors like Sass or Less to write more maintainable and scalable CSS

A. Organize styles into partials and use imports to structure large codebases.
B. Use variables, nesting, mixins, and functions to encapsulate reusable patterns.
C. Leverage extends/placeholders carefully to avoid unintended selector bloat.
D. Configure build pipelines to compile, autoprefix, and generate source maps.

Objective 2.3.3 Optimize CSS for performance and maintainability

A. Minify and concatenate assets, employ HTTP caching, and defer noncritical CSS (e.g., critical CSS inlining).
B. Reduce unused CSS with audits, pruning, and tree-shaking tools where safe.
C. Limit selector complexity and depth to improve rendering performance.

Module 3: Integrating HTML & CSS (10) (25%)

3.1 Stylesheets, precedence, and project structure (4)

Objective 3.1.1 Link external stylesheets and embed internal stylesheets in HTML documents, understanding their precedence

A. Link external CSS with `<link rel="stylesheet" href="css/styles.css" media="all">` placed in `<head>`, and order files from base to components to utilities.
B. Embed internal CSS with a single `<style>` block in `<head>` for page-specific rules or prototypes.
C. Understand precedence: inline > internal/external (by source order) > user agent, with specificity and `!important` further affecting the cascade.
D. Use media queries on links (e.g., `media="print"`) and prefer external files for caching and maintainability.

Objective 3.1.2 Apply inline styles when necessary, understanding their benefits and limitations

A. Reserve inline styles for one-off overrides, email templates, or dynamic values injected by scripts.
B. Recognize that inline styles increase specificity, can hinder reuse, and complicate theming.
C. Avoid overusing `style=""` and prefer utility classes or tokens for consistency.
D. Remove temporary inline styles during refactors to restore a clean cascade.

Objective 3.1.3 Manage style conflicts and ensure style consistency across the website

A. Adopt naming conventions (e.g., BEM) and design tokens (CSS custom properties) to standardize decisions.
B. Reduce specificity by favoring class selectors over IDs and deep descendants.
C. Establish a layered stylesheet order (base → layout → components → utilities) to control overrides.
D. Avoid `!important` except for utilities or accessibility fixes, and document any use clearly.

Objective 3.1.4 Understand and implement proper directory structures

A. Organize assets into predictable folders such as */css*, */js*, */img*, */fonts*, and */assets*.
B. Use relative paths consistently (e.g., `href="../css/app.css"`) and avoid fragile deep nesting.
C. Separate source and build outputs (e.g., `/src` and `/dist`) when using tooling.
D. Document path conventions and entry points (e.g., *index.html*) for team consistency.

3.2 Forms and interactive elements (3)

Objective 3.2.1 Construct and style HTML forms using appropriate *form* elements, attributes, and input types

A. Lay out forms with CSS Grid or Flexbox, align labels and inputs, use `gap` for spacing, and let fields wrap on small screens.
B. Keep forms readable with consistent typography and spacing; avoid fixed heights and prefer low-specificity class selectors.
C. Reuse styles with CSS custom properties for colors, spacing, and font sizes.

Objective 3.2.2 Validate user inputs and interactions using HTML5 and simple JavaScript, ensuring form data integrity

A. Use native HTML5 validation first; add small scripts only to prevent submit and show clear messages when needed.
B. Show errors near fields and style `:invalid`/`:valid` states without causing layout shift.
C. Move focus to the first error and announce a brief summary via an `aria-live` or `role="alert"` region.

Objective 3.2.3 Develop and style interactive elements ensuring user-friendly experience

A. Use semantic controls (buttons for actions, links for navigation) and clear visual states (`:hover`, `:focus-visible`, `:disabled`).
B. Ensure full keyboard access with a logical tab order and a visible focus indicator; avoid hover-only interactions.
C. Provide comfortable touch targets and sufficient contrast, and keep motion subtle while respecting `@media (prefers-reduced-motion)`.

3.3 Standards compliance and debugging (3)

Objective 3.3.1 Validate HTML and CSS to ensure they are free of syntax errors and comply with standards

A. Run documents through HTML and CSS validators to catch syntax issues early.
B. Confirm correct doctype, character encoding, and closing/nesting of elements.
C. Use autoprefixing and linting tools to improve cross-browser robustness and consistency.

Objective 3.3.2 Debug and solve styling issues arising from CSS cascading and specificity conflicts

A. Inspect computed styles and specificity to identify the winning rule and unintended overrides.
B. Simplify or restructure selectors, reduce depth, and reorder files to resolve conflicts.
C. Replace `!important` with proper architecture and utilities where possible.
D. Test across breakpoints and themes to ensure fixes do not create regressions.

Objective 3.3.3 Use developer tools to inspect and debug HTML and CSS issues

A. Leverage the Elements panel to edit rules live, toggle properties, and examine the Box Model.
B. Use layout overlays, grid/flex inspectors, and responsive design mode to diagnose layout problems.
C. Profile rendering and paint events when animating or handling large DOM trees.
D. Persist fixes back to source files and commit with clear, focused messages.

Module 4: Responsive Web Design and Layout Techniques (5) (12.5%)

4.1 Responsive design fundamentals (2)

Objective 4.1.1 Create responsive web designs using media queries, fluid grids, and flexible images

A. Adopt a mobile-first approach, then layer breakpoints with media queries such as `@media (min-width: 640px)`.
B. Build fluid grids using percentages, `fr` units, and functions like `min()`, `max()`, and `clamp()`.
C. Make images flexible with `max-width: 100%;`, and serve responsive sources using `srcset` and `sizes`.
D. Use consistent spacing scales and type scales that adapt across breakpoints.

Objective 4.1.2 Develop web designs that are mobile-friendly, focusing on optimal viewports and touch-friendly navigation

A. Set the viewport meta tag for proper scaling: `<meta name="viewport" content="width=device-width, initial-scale=1">`.
B. Provide touch-friendly targets (about 44×44 px), adequate spacing, and clear focus states.
C. Avoid hover-only interactions; offer visible toggles and keyboard-accessible controls.

4.2 Modern layout systems (2)

Objective 4.2.1 Employ CSS Flexbox and Grid to create advanced, responsive layouts

A. Use Flexbox for one-dimensional alignment, gap management, and content reordering when appropriate.
B. Use CSS Grid for two-dimensional layouts with `grid-template`, `auto-fit`/`auto-fill`, and `minmax()`.
C. Leverage `gap` instead of margins for clean, consistent spacing between items.
D. Combine grid areas and utility classes to keep markup semantic and styles maintainable.

Objective 4.2.2 Optimize layouts for various devices and screen sizes, ensuring cross-browser compatibility

A. Test across common breakpoints using responsive design tools and real devices.
B. Apply progressive enhancement and feature queries with `@supports` for fallbacks.
C. Prevent overflow and layout shifts by setting intrinsic sizes and using content wrapping wisely.
D. Use autoprefixing and conservative CSS features where necessary to support older browsers.

4.3 Performance for responsive experiences (1)

Objective 4.3.1 Implement performance optimization techniques such as lazy loading, image optimization, and code minification

A. Enable native lazy loading with `loading="lazy"` and defer non-critical assets.
B. Serve optimized images (correct dimensions, compression, and modern formats like WebP/AVIF).
C. Minify and bundle CSS, and inline critical CSS for faster first paint.
D. Leverage caching, preconnect/preload hints, and a performance budget to maintain speed.

Module 5: Accessibility, Usability, and Best Practices (6) (15%)

5.1 Accessibility and usability (2)

Objective 5.1.1 Implement accessibility features, ensuring content is accessible to people with disabilities

A. Provide text alternatives for non-text content with informative `alt` text, captions, and transcripts, and use semantic landmarks like `<main>`, `<nav>`, and `<footer>`.
B. Ensure keyboard navigation and visible focus, manage focus order, and offer skip links for bypassing repetitive content.
C. Meet color-contrast guidelines (e.g., 4.5:1 for body text) and do not convey meaning with color alone.
D. Associate labels with form controls, provide clear error messages via `aria-live`, and respect motion preferences with `@media (prefers-reduced-motion)`.

Objective 5.1.2 Evaluate and enhance the usability of web content, focusing on user-centric designs and clear, intuitive navigation

A. Design clear information architecture with descriptive labels, consistent navigation, and breadcrumbs where appropriate.
B. Use readable typography, adequate spacing, and scannable headings to improve comprehension.
C. Provide predictable interactions, helpful empty states, and clear feedback for loading, success, and errors.
D. Conduct lightweight usability reviews or tests and iterate based on observed friction points.

5.2 Best practices and quality assurance (2)

Objective 5.2.1 Develop web content adhering to industry best practices, including reusability, maintainability, and separation of concerns

A. Separate structure, presentation, and behavior by keeping HTML semantic, CSS modular, and JavaScript focused on interactions.
B. Adopt reusable components, design tokens (CSS custom properties), and naming conventions (e.g., BEM) for consistency.
C. Organize files into predictable folders, document conventions, and use version control with clear commit messages.

Objective 5.2.2 Test and validate web content for cross-browser compatibility, performance, and adherence to web standards

A. Validate HTML and CSS, and audit accessibility, internationalization, and print styles.
B. Define a browser/device support matrix and use feature queries (`@supports`) and progressive enhancement for fallbacks.
C. Measure core performance metrics (e.g., LCP, CLS, INP) and maintain a performance budget.

5.3 SEO and analytics (2)

Objective 5.3.1 Implement SEO best practices to optimize website visibility in search engine results

A. Write unique, descriptive `<title>` and `<meta name="description">` tags and use a logical heading hierarchy.
B. Use semantic HTML, internal linking with descriptive anchor text, and meaningful image `alt` attributes.
C. Provide sitemaps and robots directives, use canonical URLs, and consider structured data where appropriate.
D. Improve technical SEO with fast loading, mobile-friendly design, and stable layouts.

Objective 5.3.2 Understand and utilize web analytics to monitor website performance and user engagement

A. Define KPIs and events that reflect business goals (e.g., sign-ups, purchases, or task completion).
B. Implement pageview and event tracking with clear naming, parameters, and UTM conventions for campaigns.
C. Segment reports by channel, device, geography, and cohort to identify patterns and opportunities.
D. Respect privacy and consent requirements, anonymize data where needed, and iterate based on insights.

MQC Profile

A Minimally Qualified Candidate (MQC) for the WDA-41-01 exam is an individual with foundational knowledge of semantic HTML and core CSS, plus an understanding of web standards and responsive design. The candidate can structure valid documents, apply styles using selectors and the cascade, integrate stylesheets effectively, and deliver accessible, mobile-friendly pages.

The MQC understands document structure and metadata, semantic elements and content grouping, forms and media, CSS syntax and specificity, layout with Flexbox and Grid, media queries, and basic performance practices. The candidate is familiar with debugging via browser developer tools, cross-browser testing, and the essentials of usability, SEO, and analytics; and can apply frameworks (e.g., Bootstrap) and preprocessors (e.g., Sass/Less) at a basic level.

This profile represents a blend of standards awareness, practical styling, and debugging skills needed to produce maintainable, responsive, and accessible web content.

Block 1: HTML Fundamentals

Weight: 25% of total exam

Minimum Coverage – the Candidate can:

- Assemble a valid HTML document with <!DOCTYPE html>, language on <html>, and appropriate <head> metadata (charset, viewport, title).
- Apply semantic elements such as <header>, <nav>, <main>, <section>, <article>, <aside>, and <footer> correctly.
- Organize content with paragraphs, headings, lists, and data tables that use captions, headers, and proper scope.
- Integrate images, video, and audio with alternatives and basic responsiveness, and build clear, descriptive navigation links.

Block 2: CSS Fundamentals

Weight: 22.5% of total exam

Minimum Coverage – the Candidate can:

- Write valid CSS using common selectors, properties, and units; and structure stylesheets for readability.
- Manage the cascade, specificity, and inheritance to resolve conflicts without overusing !important.
- Apply core styling for typography, color, spacing, and the box model, including box-sizing: border-box;.
- Use modern properties, basic transitions/animations, and positioning appropriately while prioritizing maintainability.

Block 3: Integrating HTML & CSS

Weight: 25% of total exam

Minimum Coverage – the Candidate can:

- Link external stylesheets, embed internal styles, and recognize precedence relative to inline styles and source order.
- Use inline styles sparingly, prefer class-based patterns, and maintain a clear directory structure for assets.
- Construct and style forms with proper labels, input types, basic HTML5 validation, and accessible states.
- Validate markup and CSS, inspect computed styles with developer tools, and resolve cascade/specificity issues.

Block 4: Responsive Web Design and Layout Techniques

Weight: 12.5% of total exam

Minimum Coverage – the Candidate can:

- Create responsive layouts with mobile-first media queries, fluid grids, and flexible images (srcset/sizes).
- Employ Flexbox for one-dimensional alignment and CSS Grid for two-dimensional layouts using modern gap controls.
- Set optimal viewports and provide touch-friendly, keyboard-accessible navigation patterns.
- Optimize for different screens and browsers using progressive enhancement and appropriate fallbacks.

Block 5: Accessibility, Usability, and Best Practices

Weight: 15% of total exam

Minimum Coverage – the Candidate can:

- Implement accessibility fundamentals: semantic landmarks, labels, focus visibility, contrast, alternatives, and motion preferences.
- Enhance usability with clear IA, consistent navigation, readable typography, and helpful feedback states.
- Apply separation of concerns, reuse patterns and tokens, audit performance, and test cross-browser compliance.
- Follow basic SEO practices (titles, descriptions, headings, alt text, internal links) and read analytics to guide improvements.

Passing Requirement

To pass the WDA exam, a candidate must achieve a **cumulative average score of at least 75%** across all exam blocks.

WDA-41-01 Exam Structure Summary

The WDA™ exam consists of 40 single-select and multiple-select items, each designed to assess key competencies in authoring semantic HTML, applying maintainable CSS, and building responsive, accessible web pages. Every item is worth a maximum of 10 points, regardless of its format. After the exam is completed, the total raw score is normalized, and the candidate's performance is reported as a percentage. The exam covers four blocks, each weighted proportionally based on the number and complexity of items.

Block Number	Block Name	Number of Items	Weight
1	HTML Fundamentals	10	25%
2	CSS Fundamentals	9	22.5%
3	Integrating HTML and CSS	10	25%
4	Responsive Web Design and Layout Techniques	5	12.5%
5	Accessibility, Usability, and Best Practices	6	15%
	Total	**40**	**100%**