Keyboard navigation

Users must be able to *navigate* to all interactive interface components using a keyboard.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement records elements in the target page that receive the input focus.

1. Use the keyboard to navigate through all the interactive interface components in the target page.
    a. Use **Tab** and **Shift+Tab** to navigate between widgets both forwards and backwards.
    b. Use the arrow keys to navigate between the focusable elements within a composite widget.
2. If a “keyboard trap" prevents the focus from leaving a widget:
    a. Use your mouse to move the focus to the next widget.
    b. Resume testing.
3. If you encounter any trigger component that reveals hidden content:
    a. Activate the trigger.
    b. Navigate through the revealed content.
    c. Close the revealed content.
    d. Resume navigating the page.
4. Verify that you can navigate to all interactive components using the keyboard.
5. Record your results:
    a. If you find any failures, select **Fail**, then add them as failure instances.
    b. Otherwise, select **Pass**.

No keyboard traps

Users must be able to *navigate away* from all components using a keyboard.

How to test

1. Use standard keyboard commands (unmodified Tab and arrow keys) to navigate through all the interactive interface components in the target page.
2. If you can't navigate away from a component using standard keyboard commands:
    a. Examine the component's accessible name and accessible description to determine whether they describe an alternative keyboard command.
    b. If an alternative keyboard command is documented, test whether it works.
3. Verify that you can navigate away from all components using *either*
    a. Standard keyboard commands, or
    b. An alternative keyboard command that's described to users.
4. Record your results:
    a. If you find any failures, select **Fail**, then add them as failure instances.
    b. Otherwise, select **Pass**.

On focus

Navigating to a component must not trigger any unexpected change of context.

How to test

1. Use standard keyboard commands to navigate through all the interactive interface components in the target page.
2. Verify that moving focus to a component does not trigger any *unexpected* change of context, such as:
    a. Submitting a form automatically
    b. Launching a new window
    c. Shifting focus automatically to another component
3. Record your results:
    a. If you find any failures, select **Fail**, then add them as failure instances.
    b. Otherwise, select **Pass**.

On input

Changing a component's settings must not trigger any unexpected change of context.

How to test

1. Use standard keyboard commands to navigate through all the interactive interface components in the target page.
    a. Enter data in each text field and then **Tab** away from the field.
    b. Change selections for selectable components such as toggle buttons, radio buttons, check boxes, and list boxes.
2. Verify that changing the component's settings does not trigger any unexpected change in context, such as:
    a. Submitting a form automatically
    b. Launching a new window
3. Record your results:
    a. If you find any failures, select **Fail**, then add them as failure instances.
    b. Otherwise, select **Pass**.

No keystroke timings

Components must not require specific timings for individual keystrokes.

How to test

1. Use the keyboard to perform all functions that are available using a mouse.
2. Verify that individual keystrokes do not require specific timings, such as requiring users to:
    a. Repeatedly press a key within a short period of time
    b. Enter a series of keystrokes within a short period of time
    c. Press and hold a key for an extended period of time
3. Record your results:
    a. If you find any failures, select **Fail**, then add them as failure instances.
    b. Otherwise, select **Pass**.

Character key shortcuts New for WCAG 2.1

Users must be able to turn off or remap character key shortcuts.

How to test

1. Examine the target page to identify any keyboard shortcuts that can be activated using only printable keys, such as letters, numbers, symbols, or punctuation. (Alt, Shift, and Ctrl are non-printable keys.)
2. Verify that at least one of the following is true:
    a. A mechanism is available to turn off the shortcut.
    b. A mechanism is available to remap the shortcut to use one or more non-printable keyboard characters.
3. Exception: This requirement does not apply to a keyboard shortcut for a user interface component if the shortcut is active only when that component has focus.
4. Record your results:
    a. If you find any failures, select **Fail**, then add them as failure instances.
    b. Otherwise, select **Pass**.

Visible focus

Components must provide a visible indication when they have the input focus.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement records elements in the target page that receive the input focus.

1. Use the keyboard to navigate through all the interactive interface components in the target page.
    a. Use **Tab** and **Shift+Tab** to navigate between widgets both forwards and backwards.
    b. Use the arrow keys to navigate between the focusable elements within a composite widget.
2. As you move focus to each component, verify that it provides a visible indication that it has received the focus. (In addition to the circle drawn by Accessibility Insights for Web.)
3. Record your results:
    a. If you find any failures, select **Fail**, then add them as failure instances.
    b. Otherwise, select **Pass**.

Revealing content

Activating a component that reveals hidden content must move input focus into the revealed content.

How to test

1. Examine the target page to identify any "trigger" components (typically buttons or links) that reveal hidden menus or dialogs.
2. Use the keyboard to activate each trigger component.
3. Verify that focus is moved into the revealed content. (It is acceptable to **Tab** once or use an arrow key to move focus into the revealed content.)
4. Record your results:
    a. If you find any failures, select **Fail**, then add them as failure instances.
    b. Otherwise, select **Pass**.

Modal dialogs

Users must not be able to Tab away from a modal dialog without explicitly dismissing it.

How to test

1. Examine the target page to identify any "trigger" components that open modal dialogs.
2. Use the keyboard to activate each trigger component.
3. Use the **Tab** and arrow keys as needed to move focus all the way through the content of the dialog.
4. Verify that you cannot move focus out of any modal dialog using just the **Tab** or arrow keys.
5. Record your results:
    a. If you find any failures, select **Fail**, then add them as failure instances.
    b. Otherwise, select **Pass**.

Closing content

Closing revealed content must return input focus to the component that revealed it.

How to test

1. Use the keyboard to activate any trigger component that reveals hidden content, such as menus, dialogs, expandable tree views, etc.
2. If needed, use the **Tab** or arrow key to move focus into the revealed content.
3. Use the keyboard to close or hide the revealed content.
4. Verify that focus returns to the original trigger component. (It is acceptable to use **Shift+Tab** once or use an arrow key to move focus to the trigger.)
5. Record your results:
    a. If you find any failures, select **Fail**, then add them as failure instances.
    b. Otherwise, select **Pass**.

Focus order

Components must receive focus in an order that preserves meaning and operability.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement records elements in the target page that receive the input focus and their focus order.

1. Use the keyboard to navigate through all the interactive interface components in the target page.
    a. Use **Tab** and **Shift+Tab** to navigate between widgets both forwards and backwards.
    b. Use the arrow keys to navigate between the focusable elements within a composite widget.
2. If you encounter any trigger component that reveals hidden content:
    a. Activate the trigger.
    b. Navigate through the revealed content.
    c. Close the revealed content.
    d. Resume navigating the page.
3. Verify that all interactive interface components receive focus in an order that preserves the meaning and operability of the web page.
4. Record your results:
    a. If you find any failures, select **Fail**, then add them as failure instances.
    b. Otherwise, select **Pass**.

Focus not obscured New for WCAG 2.2

For elements receiving keyboard focus, its focus indicator must be at least partially visible and not obscured by author-created content which overlays it, unless the focused element can be revealed without requiring the user to advance focus in the UI.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement records elements in the target page that receive the input focus.

Note: the AAA criterion Focus Not Obscured (Enhanced) calls for focusable elements to be entirely unobscured when receiving keyboard focus.

Note: this rule covers standard keyboard focus. It does not pertain to “focus indicators” that screen readers can provide to illustrate where the assistive technology is presently reading.

1. Use the keyboard to navigate through all the interactive interface components in the target page.
    a. Use **Tab** and **Shift+Tab** to navigate between widgets both forwards and backwards.
    b. Use the arrow keys to navigate between the focusable elements within a composite widget.
2. As you move focus to each component, verify that the focused element is not completely obscured by other content. (In addition to the circle drawn by Accessibility Insights for Web.)
   Note: Focus can be obscured by user rendered content and still pass this requirement if that content can be dismissed via a keyboard command (e.g., pressing the Escape key).
3. Record your results:
    a. If you find any failures, select **Fail**, then add them as failure instances.
    b. Otherwise, select **Pass**.

Landmark roles

A landmark region must have the role that best describes its content.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement highlights all landmarks in the target page.

Note: If no matching/failing instances are found, this requirement will automatically be marked as pass.

1. In the target page, examine each landmark to verify that it has the **`role`** that best describes its content:
    a. **Banner** - Identifies site-oriented content at the beginning of each page within a website. Site-oriented content typically includes things such as the logo or identity of the site sponsor, and a site-specific search tool. A banner usually appears at the top of the page and typically spans the full width.
    b. **Complementary** - Identifies a supporting section of the document, designed to be complementary to the main content at a similar level in the DOM hierarchy, but which remains meaningful when separated from the main content.
    c. **Contentinfo** - Identifies common information at the bottom of each page within a website, typically called the "footer" of the page, including information such as copyrights and links to privacy and accessibility statements.
    d. **Form** - Identifies a set of items and objects that combine to create a form when no other named landmark is appropriate (e.g. main or search). To function as a landmark, a form must have a label.
    e. **Main** - Identifies the primary content of the page.
    f. **Navigation** - Identifies a set of links that are intended to be used for website or page content navigation.
    g. **Region** - Identifies content that is sufficiently important for users to be able to navigate to it AND no other named landmark is appropriate. To function as a landmark, a region must have a label.
    h. **Search** - Identifies a set of items and objects that combine to create search functionality.
2. Record your results:
    a. Select **Fail** for any instances that do not meet the requirement.
    b. Otherwise, select **Pass**. Or, after you have marked all failures, select **Pass unmarked instances.**

Primary content

The **`main`** landmark must contain all of the page's primary content.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement highlights the page's **`main`** landmark.

Note: If no landmarks are found, this requirement will automatically be marked as pass.

1. Examine the target page to verify that all of the following are true:
    a. The page has exactly one **`main`** landmark, and
    b. The **`main`** landmark contains all of the page's primary content.
2. Exception: If a page has nested **`document`** or **`application`** roles (typically applied to **`<iframe>`** or **`<frame>`** elements), each nested document or application may *also* have one **`main`** landmark.
3. Record your results:
    a. If you find any failures, select **Fail**, then add them as failure instances.
    b. Otherwise, select **Pass**.

No repeating content

The **`main`** landmark must not contain any blocks of content that repeat across pages.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement highlights the page's **`main`** landmark.

Note: If no landmarks are found, this requirement will automatically be marked as pass.

1. Examine the target page to verify that all of the following are true:
    a. The page has exactly one **`main`** landmark, and
    b. The **`main`** landmark does not contain any blocks of content that repeat across pages (such as site-wide navigation links).
2. Exception: If a page has nested **`document`** or **`application`** roles (typically applied to **`<iframe>`** or **`<frame>`** elements), each nested document or application may *also* have one **`main`** landmark.
3. Record your results:
    a. If you find any failures, select **Fail**, then add them as failure instances.
    b. Otherwise, select **Pass**.

Heading function

An element *coded* as a heading must *function* as a heading.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights coded headings in the target page. Coded headings include HTML tags **`<h1>`** through **`<h6>`** and elements with **role="heading"**.

Note: If no matching/failing instances are found, this requirement will automatically be marked as pass.

1. In the target page, examine each highlighted element to verify that it *functions* as a heading:
    a. An element functions as a heading if it serves as a descriptive label for the section of content that follows it.
    b. An element does not function as a heading if it serves any other purpose.
2. Record your results:
    a. Select **Fail** for any instances that do not meet the requirement.
    b. Otherwise, select **Pass**. Or, after you have marked all failures, select **Pass unmarked instances.**

No missing headings

Text that *looks like* a heading must be *coded* as a heading.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement highlights coded headings in the target page.

1. Examine the target page to verify that each element that *looks like a* heading is *coded* as a heading (highlighted).
2. Record your results:
    a. If you find any failures, select **Fail**, then add them as failure instances.
    b. Otherwise, select **Pass**.

Heading level

A heading's *programmatic* level must match the level that's presented *visually*.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights coded headings in the target page.

Note: If no matching/failing instances are found, this requirement will automatically be marked as pass.

1. In the target page, examine each heading to verify that its *programmatic* level matches the level that's presented *visually* (through font style).
    a. Lower-level headings should be more prominent than higher-level headings. (Level 1 should be the most prominent, level 6 the least.)
    b. Headings of the same level should have the same font style.
2. Record your results:
    a. Select **Fail** for any instances that do not meet the requirement.
    b. Otherwise, select **Pass**. Or, after you have marked all failures, select **Pass unmarked instances.**

Heading level

A heading's *programmatic* level must match the level that's presented *visually*.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights coded headings in the target page.

Note: If no matching/failing instances are found, this requirement will automatically be marked as pass.

1. **In the target page, examine each heading to verify that its *programmatic* level matches the level that's presented *visually* (through font style).**
    a. **Lower-level headings should be more prominent than higher-level headings. (Level 1 should be the most prominent, level 6 the least.)**
    b. **Headings of the same level should have the same font style.**
2. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Bypass blocks

A page must provide a keyboard-accessible method to bypass repetitive content.

How to test

1. **Examine the target page to identify:**
    a. **The starting point of the page's primary content.**
    b. **Any blocks of content that (1) *precede* the primary content and (2) appear on multiple pages, such as banners, navigation links, and advertising frames.**
2. **Refresh the page to ensure that it's in its default state.**
3. **Use the Tab key to navigate toward the primary content. As you navigate, look for a bypass mechanism (typically a skip link). The mechanism might not become visible until it receives focus.**
4. **If a bypass mechanism *does not* exist, select Fail, then add the failure instance.**
5. **If a bypass mechanism *does* exist, activate it.**
6. **Verify that focus shifts past any repetitive content to the page's primary content.**
7. **Record your results:**
    a. **If you find a failure, select Fail, then add the failure instance.**
    b. **Otherwise, select Pass.**

Consistent navigation

Navigational mechanisms that appear on multiple pages must be presented in the same relative order.

How to test

1. **Examine the target page to identify any navigational mechanisms (such as site navigation bars, search fields, and skip links) that appear on multiple pages.**
2. **Verify that the links or buttons in each navigational mechanism are presented in the same relative order each time they appear. (Items should be in the same relative order even if other items are inserted or removed between them.)**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Consistent identification

Functional components that appear on multiple pages must be identified consistently.

How to test

1. **Examine the target page to identify any functional components (such as links, widgets, icons, images, headings, etc.) that appear on multiple pages.**
2. **Use the Accessibility pane in the browser Developer Tools to verify that the component has the same accessible name each time it appears.**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Consistent help New for WCAG 2.2

Ensure help – or mechanism(s) to request help – are consistently located in the same relative location across a set of web pages/screens.

How to test

Note: this criterion does not require help to be provided.

1. **Examine the target page to identify "help" mechanisms (for example links to help, etc.) on the page. Determine if this is a set of web pages with blocks of content that are repeated on multiple pages.**
2. **Verify that all helpful information and mechanisms provided are consistent with other pages in terms of location, behavior and relative to the other content of the page & UI for all components where help resides.**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Exception: The location of a help mechanism can change based on user input, for example resizing of the window that changes the location of the help link – this would still pass this rule so long as the help link was consistently presented in the same location across the same set of web pages at the adjusted window size.

Link function

If an anchor element functions as a custom widget, it must have the appropriate ARIA widget role.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights anchor elements that are possible custom widgets. These elements don't have an ARIA widget role, but they do have some custom widget markup, such as tabindex="-1", an ARIA attribute, a non-widget role, or no href.

Note: If no matching/failing instances are found, this requirement will automatically be marked as pass.

1. **In the target page, examine each highlighted anchor element to verify that it functions as a link (i.e., it navigates to new content in the current page or in a new page).**
2. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Link purpose

The purpose of a link must be described by its link text alone, or by the link text together with preceding page context.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights links in the target page.

Note: If no matching/failing instances are found, this requirement will automatically be marked as pass.

1. **In the Instances list below, examine each link to verify that its accessible name describes its purpose.**
    a. **If a link navigates to a document or web page, the name of the document or page is sufficient.**
    b. **Links with different destinations should have different link text.**
    c. **Links with the same destination should have the same link text.**
2. **If a link's purpose is clear from its accessible name, mark it as Pass.**
3. **If a link's purpose is *not* clear from its accessible name, examine the link in the context of the target page to verify that its purpose is described by the link together with its preceding page context, which includes:**
    a. **Text in the same sentence, paragraph, list item, or table cell as the link**
    b. **Text in a parent list item**
    c. **Text in the table header cell that's associated with cell that contains the link**
4. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Label in name New for WCAG 2.1

A link's accessible name must contain its visible text label.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement highlights links that have visible text on the target page.

Note: If no matching/failing instances are found, this requirement will automatically be marked as pass.

1. **In the target page, examine each highlighted link to identify its visible text label.**
2. **Compare each link's text visible label to its accessible name (displayed in the Instances list below).**
3. **Verify that:**
    a. **The accessible name is an exact match of the visible text label, or**
    b. **The accessible name *contains* an exact match of the visible text label, or**
    c. **The link does not have a visible text label.**
4. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Widget function

If a native widget *functions* as a custom widget, it must have the appropriate ARIA widget role.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights native widgets that are possible custom widgets. These elements don't have an ARIA widget role, but they do have some custom widget markup, such as `tabindex="-1"`, an ARIA attribute, or a non-widget role.

Note: If no matching/failing instances are found, this requirement will automatically be marked as pass.

1. **In the target page, examine each highlighted widget to verify that it *functions* as a simple native widget.**
2. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Instructions New for WCAG 2.1

If a native widget has a visible label or instructions, they must be programmatically determinable.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement highlights native widgets. Native widgets include `<button>`, `<input>`, `<select>`, and `<textarea>` elements.

Note: Both WCAG 2.0 and 2.1 require a widget's visible label and instructions (if present) to be programmatically determinable. WCAG 2.1 also requires a widget's visible label and instructions (if present) to be included in its accessible name and description.

1. **In the target page, examine each highlighted element to determine whether it has a visible label or instructions.**
2. **If a widget does have a visible label or instructions, verify that they are also displayed in the Instances list:**
    a. **The accessible name must be (or include) an exact match of any visible text label.**
    b. **The accessible description must include any additional visible instructions. If any non-text instructions are provided (for example, icons or color changes), the accessible description must include a text equivalent.**
3. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Expected input

A native widget must have a label and/or instructions that identify the expected input.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights native widgets.

Notes: (1) If no matching/failing instances are found, this requirement will automatically be marked as pass. (2) If a native widget has no programmatically-related label, it will fail an automated check.

1. **Examine each widget in the Instances list below to verify that its accessible name and/or instructions identify the expected input, including any unusual or specific formatting requirements.**
2. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Cues

If a native widget adopts certain interactive states, it must provide appropriate cues.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights native widgets. Native widgets include `<button>`, `<input>`, `<select>`, and `<textarea>` elements.

Note: If no matching/failing instances are found, this requirement will automatically be marked as pass.

1. **In the target page, interact with each native widget to determine whether it adopts any of these states:**
    a. **Disabled**
    b. **Read-only**
    c. **Required**
2. **If a widget *does* adopt any of these states, inspect its HTML using the browser Developer Tools to verify that the states are appropriately coded.**
    a. **HTML properties (e.g., `readonly`) should be used on elements that support them.**
    b. **ARIA properties (e.g., `aria-readonly`) should be used on elements that don't support HTML properties.**
3. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Autocomplete New for WCAG 2.1

Text fields that serve certain purposes must have the correct HTML5 `autocomplete` attribute.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement highlights text fields.

1. **In the target page, examine each highlighted text field to determine whether it serves an identified input purpose.**
2. **If a text field *does* serve an identified input purpose, verify that it has an autocomplete attribute with the correct value.**
3. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Widget function

If a native widget *functions* as a custom widget, it must have the appropriate ARIA widget role.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights native widgets that are possible custom widgets. These elements don't have an ARIA widget role, but they do have some custom widget markup, such as `tabindex="-1"`, an ARIA attribute, or a non-widget role.

Note: If no matching/failing instances are found, this requirement will automatically be marked as pass.

1. **In the target page, examine each highlighted widget to verify that it *functions* as a simple native widget.**
2. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Instructions New for WCAG 2.1

If a native widget has a visible label or instructions, they must be programmatically determinable.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement highlights native widgets. Native widgets include `<button>`, `<input>`, `<select>`, and `<textarea>` elements.

Note: Both WCAG 2.0 and 2.1 require a widget's visible label and instructions (if present) to be programmatically determinable. WCAG 2.1 also requires a widget's visible label and instructions (if present) to be included in its accessible name and description.

1. **In the target page, examine each highlighted element to determine whether it has a visible label or instructions.**
2. **If a widget does have a visible label or instructions, verify that they are also displayed in the Instances list:**
    a. **The accessible name must be (or include) an exact match of any visible text label.**
    b. **The accessible description must include any additional visible instructions. If any non-text instructions are provided (for example, icons or color changes), the accessible description must include a text equivalent.**
3. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Expected input

A native widget must have a label and/or instructions that identify the expected input.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights native widgets.

Notes: (1) If no matching/failing instances are found, this requirement will automatically be marked as pass. (2) If a native widget has no programmatically-related label, it will fail an automated check.

1. **Examine each widget in the Instances list below to verify that its accessible name and/or instructions identify the expected input, including any unusual or specific formatting requirements.**
2. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Cues

If a native widget adopts certain interactive states, it must provide appropriate cues.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights native widgets. Native widgets include `<button>`, `<input>`, `<select>`, and `<textarea>` elements.

Note: If no matching/failing instances are found, this requirement will automatically be marked as pass.

1. **In the target page, interact with each native widget to determine whether it adopts any of these states:**
    a. **Disabled**
    b. **Read-only**
    c. **Required**
2. **If a widget *does* adopt any of these states, inspect its HTML using the browser Developer Tools to verify that the states are appropriately coded.**
    a. **HTML properties (e.g., `readonly`) should be used on elements that support them.**
    b. **ARIA properties (e.g., `aria-readonly`) should be used on elements that don't support HTML properties.**
3. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Autocomplete New for WCAG 2.1

Text fields that serve certain purposes must have the correct HTML5 `autocomplete` attribute.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement highlights text fields.

1. **In the target page, examine each highlighted text field to determine whether it serves an identified input purpose.**
2. **If a text field *does* serve an identified input purpose, verify that it has an autocomplete attribute with the correct value.**
3. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Design pattern

A custom widget must have the appropriate ARIA widget role for its design pattern.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights custom widgets. (A custom widget is an element with a valid ARIA widget role.)

Note: If no matching/failing instances are found, this requirement will automatically be marked as pass.

1. **Familiarize yourself with the ARIA design patterns for custom widgets.**
2. **In the target page, examine each custom widget to determine which design pattern best describes its function.**
3. **In the Instances list below, verify that the custom widget has the right role for its design pattern.**
4. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Instructions New for WCAG 2.1

If a custom widget has a visible label or instructions, they must be programmatically determinable.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement highlights custom widgets. (A custom widget is an element with a valid ARIA widget role.)

Note: Both WCAG 2.0 and 2.1 require a widget's visible label and instructions (if present) to be programmatically determinable. WCAG 2.1 also requires a widget's visible label and instructions (if present) to be included in its accessible name and description.

1. **In the target page, examine each highlighted element to determine whether it has a visible label or instructions.**
2. **If a widget does have a visible label or instructions, verify that they are also displayed in the Instances list:**
    a. **The accessible name must be (or include) an exact match of the visible text label.**
    b. **The accessible description must include any additional visible instructions. If any non-text instructions are provided (for example, icons or color changes), the accessible description must include a text equivalent.**
3. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Expected input

A custom widget must have a label and/or instructions that identify the expected input.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights custom widgets.

Note: (1) If no matching/failing instances are found, this requirement will automatically be marked as pass. (2) If a custom widget has no programmatically-related label, it will fail an automated check.

1. **Examine each widget in the Instances list below to verify that its accessible name and/or instructions identify the expected input, including any unusual or specific formatting requirements.**
2. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Role, state, property

A custom widget must support the ARIA roles, states, and properties specified by its design pattern.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights custom widgets.

Note: If no matching/failing instances are found, this requirement will automatically be marked as pass.

1. **In the Instances list below, use the link for the design pattern that best describes the widget's function.**
2. **Familiarize yourself with the "WAI-ARIA Roles, States, and Properties" section of the design pattern spec.**
3. **Inspect the widget's HTML using the Accessibility pane in the browser Developer Tools to verify that it supports all of the roles, states, and properties specified by its design pattern:**
    a. **For a composite widget, use the Accessibility Tree to verify the role hierarchy. (For example, verify that a menuitem exists for each option in a menubar.)**
    b. **View the widget's ARIA Attributes while you interact with it to verify that required properties update according to spec. (For example, when a tree node in a tree view is expanded, aria-expanded is "true" and when it isn't expanded, it is "false".)**
4. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Cues

If a custom widget adopts certain interactive states, it must communicate those states programmatically.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights custom widgets.

Note: If no matching/failing instances are found, this requirement will automatically be marked as pass.

1. **In the target page, interact with each custom widget to determine whether it adopts any of these states:**
    a. **Disabled**
    b. **Read-only**
    c. **Required**
2. **If a widget *does* adopt any of these states, inspect its HTML using the browser Developer Tools to verify that the states are appropriately coded.**
    a. **HTML properties (e.g., `readonly`) should be used on elements that support them.**
    b. **ARIA properties (e.g., `aria-readonly`) should be used on elements that don't support HTML properties.**
3. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Keyboard interaction

A custom widget must support the keyboard interaction specified by its design pattern.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights custom widgets.

Note: If no matching/failing instances are found, this requirement will automatically be marked as pass.

1. **For each custom widget, open the spec for the design pattern that best describes the widget's function. (If the widget has the correct role, the design pattern link in the Instances list below will open the correct spec.)**
2. **Familiarize yourself with the "Keyboard Interaction" section of the spec.**
3. **Interact with the widget to verify that it supports the keyboard interactions specified by its design pattern.**
4. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Time limits

If a time limit is set by the content, the user must be able to turn off, adjust, or extend the time limit.

How to test

1. **Examine the target page to determine whether it has any content time limits (time-outs).**
    a. **Ignore any time limit that is:**
        i. **Part of a real-time event (such as an auction), and no alternative to the time limit is possible; or**
        ii. **Essential to the activity (such as an online test), and allowing users to extend it would invalidate the activity; or**
        iii. **Longer than 20 hours.**
2. **If the page *does* have a time limit, verify that:**
    a. **You can turn off the time limit, or**
    b. **You can adjust the time limit to at least 10 times the default limit, or**
    c. **You are:**
        i. **Warned about the time limit, and**
        ii. **Given at least 20 seconds to extend the time limit with a simple action (e.g., "Press the space bar"), and**
        iii. **Allowed to extend the time limit at least 10 times.**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Moving content

If content moves, blinks, or scrolls automatically for more than five seconds, users must be able to pause, stop, or hide it.

How to test

1. **Examine the target page to identify any information that:**
    a. **Moves, blinks, or scrolls, and**
    b. **Starts automatically, and**
    c. **Lasts more than 5 seconds, and**
    d. **Is presented in parallel with other content, and**
    e. **Is not part of an activity where it is essential.**
2. **If you find such content, verify that you can pause, stop, or hide it.**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Auto-updating content

If content updates automatically, users must be able to pause, stop, hide, or control frequency of the updates.

How to test

1. **Examine the target page to identify any content that:**
    a. **Updates, and**
    b. **Starts automatically, and**
    c. **Is presented in parallel with other content, and**
    d. **Is not part of an activity where it is essential.**
2. **If you find such content, verify that you can pause, stop, or hide it, or control the update frequency.**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Audio control

If audio content plays automatically for longer than three seconds, users must be able to pause or mute it.

How to test

1. **Examine the target page to determine whether it has any audio that:**
    a. **Plays automatically, and**
    b. **Lasts more than three seconds.**
2. **If you find such audio, verify that a mechanism is available, either at the beginning of the page/screen content or in platform accessibility features, that allows you to:**
    a. **Pause or stop the audio, or**
    b. **Control audio volume independently from the overall system volume level.**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Error identification

If an input error is automatically detected, the item in error must be identified, and the error described, in text.

How to test

1. **Examine the target page to identify any input fields with automatic error detection, such as:**
    a. **Required fields**
    b. **Fields with required formats (e.g., date)**
    c. **Passwords**
    d. **Zip code fields**
2. **If you find such an input field, enter an incorrect value that triggers automatic error detection.**
3. **Verify that:**
    a. **The field with the error is identified in text, and**
    b. **The error is described in text.**
4. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Error suggestion

If an input error is automatically detected, guidance for correcting the error must be provided.

How to test

1. **Examine the target page to identify any input fields with automatic error detection, such as:**
    a. **Required fields**
    b. **Fields with required formats (e.g., date)**
    c. **Passwords**
    d. **Zip code fields**
2. **If you find such an input field, enter an incorrect value that triggers automatic error detection.**
3. **Examine the error notification to verify that guidance for correcting the error is provided to the user (unless it would jeopardize the security or purpose of the content).**
4. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Error prevention

If submitting data might have serious consequences, users must be able to correct the data input before finalizing a submission.

How to test

1. **Examine the target page to determine whether it allows users to:**
    a. **Make any legal commitments or financial transactions, or**
    b. **Modify or delete data in a data storage system, or**
    c. **Submit test responses.**
2. **If the page *does* allow such actions, verify that at least one of the following is true:**
    a. **Submissions are reversible.**
    b. **Data entered by the user is checked for input errors, and the user is given an opportunity to correct them.**
    c. **The user can review, confirm, and correct information before finalizing the submission.**
3. **Record your results:**
    a. **If you find a failure, select Fail, then add the failure instance.**
    b. **Otherwise, select Pass.**

Status messages New for WCAG 2.1

Status messages must be programmatically determinable without receiving focus.

How to test

1. **Examine the target page to determine whether it generates any status messages. A status message provides information to the user on any of the following:**
    a. **The success or results of an action**
    b. **The waiting state of an application**
    c. **The progress of a process**
    d. **The existence of errors**
2. **Refresh the page.**
3. **Inspect the page's HTML to identify an empty container with one of the following attributes:**
    a. `role="alert"`
    b. `role="log"`
    c. `role="progressbar"`
    d. `aria-live="assertive"`
4. **Trigger the action that generates the status message.**
5. **Verify that the status message is injected into the container.**
6. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Page title

A web page must have a title that describes its topic or purpose.

How to test

1. **Consider the title of the target page, displayed in the Instances list below.**
2. **Verify that the page's title describes its topic or purpose:**
    a. **For pages within a website, the page title must be unique.**
    b. **For documents or single-page web apps, the document name or app name is sufficient.**
3. **Record your results:**
    a. **Select Fail if the page title does not meet the requirement.**
    b. **Otherwise, select Pass.**

Frame title

A frame or iframe must have a title that describes its content.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights all `<frame>` and`<iframe>` elements with visible content.

Notes: (1) If no matching/failing instances are found, this requirement will automatically be marked as pass. (2) If a frame or iframe doesn't have a title, it will fail an automated check and will not be displayed in the list of instances for this requirement.

1. **Examine each `<frame>` or `<iframe>`in the Instances list below to verify that that its title describes its content.**
2. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Multiple ways

Users must have multiple ways to navigate to a page.

How to test

1. **Examine the target page to determine whether:**
    a. **The page is part of a multi-page website or web app.**
        i. **If there are no other pages within the site or app, select Pass.**
    b. **The page is the result of, or a step in, a process.**
        i. **If the page is part of a process, select Pass.**
2. **Verify that the page provides two or more ways to locate pages within the site or app, such as:**
    a. **Site maps**
    b. **Site search**
    c. **Tables of contents**
    d. **Navigation menus or dropdowns**
    e. **Navigation trees**
    f. **Links between pages**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Image function

Every image must be coded as either meaningful or decorative.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights images that are coded as meaningful or decorative.

Notes: (1) If an image has no code to identify it as meaningful or decorative, it will fail an automated check. (2) Assistive technologies will ignore any image coded as decorative, even if it has an accessible name. (3) If no matching/failing instances are found, this requirement will automatically be marked as Pass.

1. **Examine each image to verify that its coded function is correct:**
    a. **An image should be coded as *meaningful* if it conveys information that isn't available through other page content.**
    b. **An image should be coded as *decorative* if it could be removed from the page with *no* impact on meaning or function.**
2. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Text alternative

A meaningful image must have a text alternative that serves the equivalent purpose.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights images that are coded as meaningful.

Notes: (1) This procedure may require use of the Web Developer browser extension. (2) If no matching/failing instances are found, this requirement will automatically be marked as Pass.

1. **Examine each image in the Instances list to verify that its text alternative serves the equivalent purpose.**
    a. **A *simple* image should have an accessible name that serves the equivalent purpose. Special cases:**
        i. **An image of text should have an accessible name that exactly matches the text within the image.**
        ii. **A CAPTCHA image should have an accessible name that communicates the purpose of the image, but not its content. (A CAPTCHA is a test to differentiate a human from a computer.)**
    b. **A *complex* image (such as a graph) should have both**
        i. **An accessible name that communicates the purpose of the image, and**
        ii. **An accessible description that communicates the content of the image.**
2. **If a CSS background image is coded as meaningful:**
    a. **Use the Web Developer browser extension (CSS > Disable All Styles) to turn off CSS.**
    b. **Verify that the information conveyed by the image is visible when CSS is turned off.**
3. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Images of text

Images of text are allowed only where a specific appearance is required (e.g., logotypes).

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights images that are coded as meaningful.

Note: If no matching/failing instances are found, this requirement will automatically be marked as pass.

1. **In the target page, examine each image to identify any images of text.**
2. **If you find an image of text, verify that it is used only where a specific appearance required, such as text in a logo.**
3. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

CAPTCHAs

If a CAPTCHA is used, alternative methods must be provided for both users without vision and users without hearing.

How to test

1. **Examine the target page to determine whether it has a CAPTCHA. (A CAPTCHA is a test to differentiate a human from a computer.)**
2. **If the page *does* have a CAPTCHA, verify that alternative methods are provided (at a minimum) for**
    a. **Users without vision**
    b. **Users without hearing**
3. **Record your results:**
    a. **If you find a failure, select Fail, then add the failure instance.**
    b. **Otherwise, select Pass.**

Language of page

A page must have the correct default language.

How to test

Note: If the `<html>` element's lang attribute is missing or invalid, it will fail an automated check.

1. **Examine the target page to determine its primary language.**
2. **Inspect the page's `<html>` tag to verify that it has the correct language attribute.**
3. **Record your results:**
    a. **If you find a failure, select Fail, then add the failure instance.**
    b. **Otherwise, select Pass.**

Language of parts

If the language of a passage differs from the default language of the page, the passage must have its own language attribute.

How to test

Note: if an element has an invalid lang attribute, it will fail an automated check

1. **Examine the target page to identify any passages in a language different from the default language of the page.**
2. **If you find such a passage, examine the containing element's HTML to verify that it has the correct language attribute.**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Text direction

If a page or a passage uses a script that is read right-to-left, it must have the correct text direction.

How to test

1. **Examine the target page to determine whether the page uses a right-to-left script - such as Arabic, Hebrew, Persian or Urdu.**
2. **If the page or a passage *does* use a right-to-left script, examine the containing element's HTML to verify that it has the correct direction attribute (dir="rtl").**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Color as meaning

Color must not be used as the only visual means for conveying meaning.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement displays the target page in grayscale.

1. **Examine the target page to identify any instances where color is used to communicate meaning, such as:**
    a. **Communicating the status of a task or process**
    b. **Indicating the state of a UI component (such as selected or focused)**
    c. **Prompting a response**
    d. **Identifying an error**
2. **For each instance, verify that at least one of these visual alternatives is also provided:**
    a. **On-screen text that identifies the color itself and/or describes the meaning conveyed by the color**
    b. **Visual differentiation (e.g., shape, position, size, underline) and a clear indication of its meaning**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Instructions

Instructions must not rely solely on color or other sensory characteristics.

How to test

1. **Examine the target page to identify any instances where instructions refer to an element's sensory characteristics, such as:**
    a. **Color**
    b. **Shape**
    c. **Size**
    d. **Visual location**
    e. **Orientation**
    f. **Sound**
2. **For each instance, verify that the instructions also include additional information sufficient to locate and identify the element without knowing its sensory characteristics. (For example, "Press the green button").**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Auditory cues

Auditory cues must be accompanied by visual cues.

How to test

1. **Interact with the target page to identify any instances where the system provides auditory cues, such as:**
    a. **A tone that indicates successful completion of a process**
    b. **A tone that indicates arrival of a message**
2. **For each instance, verify that the system *also* provides a visible cue.**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Flashing

Web pages must not have content that flashes more than three times per second.

How to test

1. **Examine the target page to identify any content that flashes.**
2. **If you find such content, determine whether it flashes faster than three times per second.**
    a. **Use this link to see an example of content that flashes three times per second. (The example will open in a new browser window.)**
3. **For any content that flashes faster than three times per second, verify that at least one of the following is true:**
    a. **The total flashing area is smaller than 21,824 pixels (in any shape).**
    b. **The relative luminance between the brightest and darkest portions of the flash is less than 10% and the flash does not include any saturated red.**
4. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

High contrast mode

Websites and web apps must honor high contrast appearance settings and functions.

How to test

Google Chrome and Microsoft Edge both support Windows high contrast themes. If you are using Microsoft Edge, make sure that the target page is not in Internet Explorer mode. Neither browser supports high contrast themes on macOS.

1. **Open the target page.**
2. **Apply a high contrast theme for your operating system using these instructions.**
3. **Verify that the target page adopts the colors specified for the theme.**
4. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Resize text

Users must be able to resize text, without using assistive technology, up to 200% with no loss of content or functionality.

How to test

Note: An automated check will fail if text scaling and zooming is disabled because the `user-scalable=no` ***parameter is used in a*** `<meta name="viewport">` ***element.***

1. **Set your browser window width to 640 logical pixels (the equivalent of 1280 logical pixels at 200% zoom):**
    a. **Set the maximum browser window size using these instructions.**
    b. **Put the browser into full-screen mode.**
    c. **Examine the target page to verify that:**
        i. **All text resizes fully, including text in form fields.**
        ii. **Text isn't clipped, truncated, or obscured.**
        iii. **All content remains available.**
        iv. **All functionality remains available.**
    d. **Exception: Images of text and captions for videos are exempt from this requirement.**
2. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Contrast

Text elements must have sufficient contrast.

Visual helper with Accessibility Insights

How to test

For this requirement, Accessibility Insights for Web highlights instances of text where the contrast ratio can't be determined, typically because the background color is not uniform. You must manually verify the contrast for these instances.

Note: If no matching/failing instances are found, this requirement will automatically be marked as pass.

1. **Examine each instance in the target page to determine whether it is text. (Because icons are assessed in Contrast > Graphics , they can be marked as Pass in this test.)**
2. **Examine each text instance to identify an area where the text and background are most likely to have a low contrast ratio (e.g., white text on a light gray background).**
3. **Use Accessibility Insights for Windows to test the contrast at that area. (If you are testing on a Mac, you can use the Colour Contrast Analyser.)**
4. **Verify that each instance meets these contrast thresholds:**
    a. **Regular text must have a ratio ≥ 4.5**
    b. **Large text (18pt or 14pt+bold) must have a ratio ≥ 3.0.**
5. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Orientation New for WCAG 2.1

Web content must not be locked to a particular screen orientation.

How to test

1. **Open the target page on a device that automatically reorients web content when the device orientation changes (e.g., a mobile device).**
2. **Examine the target page with the device oriented vertically, then horizontally.**
3. **Verify that the page content reorients when the device's orientation changes. Exception: Orientation locking is allowed if a specific orientation is essential to the underlying functionality (such as a banking application that requires horizontal orientation when photographing a check for deposit).**
4. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

ReflowNew for WCAG 2.1

Content must be visible without having to scroll in two dimensions.

How to test

The following steps assume the page uses a script read horizontally (left-to-right or right-to-left) rather than vertically (top-to-bottom).

1. **Use your system's display settings to set the display resolution to 1280 x 1024.**
2. **Use your browser's settings to set the target page's zoom to 400% and to enable full-screen mode.**
    a. ***Note: 320 x 256 is equivalent to a display resolution of 1280 x 1024 at a 400% zoom with the browser set to full-screen mode.***
3. **Examine the target page to verify that all text content is available without horizontal scrolling. Content can be displayed directly in the page, revealed via accessible controls, or accessed via direct links.**
   Exception: Horizontal scrolling is allowed for the following content:
    a. **Data tables**
    b. **Photos**
    c. **Maps**
    d. **Charts**
    e. **Games**
    f. **UI with toolbars**
4. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

ReflowNew for WCAG 2.1

Content must be visible without having to scroll in two dimensions.

How to test

The following steps assume the page uses a script read horizontally (left-to-right or right-to-left) rather than vertically (top-to-bottom).

1. **Use your system's display settings to set the display resolution to 1280 x 1024.**
2. **Use your browser's settings to set the target page's zoom to 400% and to enable full-screen mode.**
    a. ***Note: 320 x 256 is equivalent to a display resolution of 1280 x 1024 at a 400% zoom with the browser set to full-screen mode.***
3. **Examine the target page to verify that all text content is available without horizontal scrolling. Content can be displayed directly in the page, revealed via accessible controls, or accessed via direct links.**
   Exception: Horizontal scrolling is allowed for the following content:
    a. **Data tables**
    b. **Photos**
    c. **Maps**
    d. **Charts**
    e. **Games**
    f. **UI with toolbars**
4. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Hover / focus contentNew for WCAG 2.1

Content that appears on focus or hover must be dismissible, hoverable, and persistent.

How to test

1. **Examine the target page to identify any components that reveal additional content when they receive focus or pointer hover, such as a button that shows a tooltip on hover.**
2. **Verify that all of the following are true:**
    a. **Dismissible. A mechanism is available to dismiss the additional content without moving pointer hover or keyboard focus, unless the additional content communicates an input error or does not obscure or replace other content.**
    b. **Hoverable. If pointer hover can trigger the additional content, then the pointer can be moved over the additional content without the additional content disappearing.**
    c. **Persistent. The additional content remains visible until the hover or focus trigger is removed, the user dismisses it, or its information is no longer valid.**
3. **Exception: This requirement does not apply if the visual presentation of the additional content is controlled solely by the browser.**
4. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Audio-only equivalent

Pre-recorded audio-only content must be accompanied by an equivalent text alternative.

How to test

1. **Examine the target page to identify any pre-recorded audio-only content:**
    a. **Audio-only content *does not* include:**
        i. **Audio synchronized with video, slides, animations, or other time-based visuals.**
        ii. **Short sounds such as confirmation beeps or error notifications.**
    b. **Audio-only content *does* include audio accompanied by simple static visuals, such as the title of a speech and the speaker's name.**
2. **Determine whether the audio-only content is accompanied by a text transcript.**
3. **If you find any audio-only content that doesn't have a transcript, select Fail, then add the failure instance.**
4. **Compare the audio-only content to the transcript.**
5. **Verify that the transcript provides an accurate and complete description of the audio content.**
6. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Video-only equivalent

Pre-recorded video-only content must be accompanied by an equivalent text or audio alternative.

How to test

1. **Examine the target page to identify any pre-recorded video-only content:**
    a. **Video-only content *does not* include short animation effects, such as a button being highlighted when activated, or a file icon shrinking when the file is closed.**
    b. **Video-only content *does* include video content accompanied by sound (such as background music) that does not contribute meaning.**
2. **Determine whether the video-only content is accompanied by at least one of these alternatives:**
    a. **A text transcript**
    b. **An audio track**
3. **If you find any video-only content that doesn't have a transcript or an audio track, select Fail, then add the failure instance.**
4. **Compare the video-only content to the transcript or audio track.**
5. **Verify that the transcript or audio track provides an accurate and complete description of the video content, including information about actions, characters, scene changes, and on-screen text.**
6. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Captions

Pre-recorded multimedia content must have captions.

How to test

1. **Examine the target page to identify any pre-recorded multimedia content (synchronized video and audio).**
    a. **Ignore any multimedia content that is provided as an audio or video alternative to text and is clearly labeled as such.**
2. **Enable captions, then play the multimedia content.**
3. **Verify that the multimedia content has captions.**
4. **If you find any multimedia content that doesn't have captions, select Fail, then add the failure instance.**
5. **Verify that the captions provide an accurate and complete description of the audio content, including speaker identity, speech, and meaningful sounds.**
6. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

No obstruction

Captions must not obscure or obstruct relevant information in the video.

How to test

1. **Examine the target page to identify any pre-recorded multimedia content with captions.**
2. **Determine whether the multimedia content has captions.**
3. **Enable captions, then play the multimedia content.**
4. **Verify that the captions do not obscure or obstruct relevant information in the video.**
5. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Audio description

Pre-recorded video with audio must have an audio description.

How to test

1. **Examine the target page to identify any pre-recorded multimedia content (video with synchronized audio).**
2. **Enable audio descriptions, then play the multimedia content.**
3. **Verify that the multimedia content has an audio description.**
4. **If you find any pre-recorded multimedia content that doesn't have an audio description, select Fail, then add the failure instance.**
5. **Verify that the audio description adequately describes important visual content in the media, including information about actions, characters, scene changes, and on-screen text.**
6. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Synchronization

An audio description must be synchronized with the video content.

How to test

1. **Examine the target page to identify any pre-recorded multimedia content with an audio description.**
2. **Enable audio descriptions, then play the multimedia content.**
3. **Verify that the audio description is synchronized with the video content.**
4. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

No conflict

An audio description must not conflict with audible information in the sound track.

How to test

1. **Examine the target page to identify any pre-recorded multimedia content with an audio description.**
2. **Enable audio descriptions, then play the multimedia content.**
3. **Verify that the audio description does not conflict with relevant information in the soundtrack. (Descriptive narration should be added during existing pauses in dialog.)**
4. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Captions

Captions must be provided for live (streaming) video with audio.

How to test

1. **Examine the target page to identify any live (streaming) multimedia content (video with synchronized audio).**
2. **Enable captions, then play the multimedia content.**
3. **Verify that the multimedia content has captions.**
4. **If you find any live multimedia content that doesn't have captions, select Fail, then add the failure instance.**
5. **Verify that the captions provide an accurate and complete description of the audio content, including speaker identity, speech, and meaningful sounds.**
6. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

CSS positioning

Meaningful content positioned on the page using CSS must retain its meaning when CSS is disabled.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement highlights content positioned on the screen using CSS position:absolute or float:right.

Note: If no matching/failing instances are found, this requirement will automatically be marked as pass.

This procedure also uses the Web Developer browser extension.

1. **Examine the target page to determine whether it has any positioned (highlighted) content that's meaningful:**
    a. **Content is *meaningful* if it conveys information that isn't available through other page content.**
    b. **Content is *decorative* if it could be removed from the page with no impact on meaning or function.**
2. **If the page does have meaningful positioned content, use the Web Developer browser extension (Miscellaneous > Linearize page) to show the page in DOM order.**
3. **Verify that the positioned content retains its meaning when the page is linearized.**
4. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Layout tables

The content in an HTML layout table must make sense when the table is linearized.

How to test

This procedure uses the Web Developer browser extension.

1. **Use the Web Developer extension (Outline > Outline Tables) to identify any `<table>` elements in the target page.**
2. **If you find a table, determine whether it is a *layout* table:**
    a. **A *data* table uses rows and columns to show relationships within a set of data.**
    b. **A *layout* table uses rows and columns to visually position content without implying any relationships.**
3. **If you find a layout table, use the Web Developer browser extension (Miscellaneous > Linearize page) to show the page in DOM order.**
4. **Verify that content in layout tables still has the correct reading order when the page is linearized.**
5. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Columns

Content presented in multi-column format must support a correct reading sequence.

How to test

This procedure uses the Web Developer browser extension.

1. **Use the Web Developer browser extension (Outline > Outline table cells) to highlight table cells (`<td>` and `<th>` elements).**
2. **Examine the page to identify any side-by-side columns of text or data that are *not* contained in a table cell.**
3. **Using your mouse or keyboard, verify that you can select *all* of the text in one column without selecting *any* text from an adjacent column.**
4. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

CSS content

Meaningful content must not be implemented using only CSS `:before` or `:after`.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement highlights content inserted in the page using CSS `:before` or `:after`.

Note: If no matching/failing instances are found, this requirement will automatically be marked as pass.

This procedure uses the Web Developer browser extension.

1. **In the target page, examine each highlighted item to determine whether it is meaningful or decorative.**
    a. **An element is *meaningful* if it conveys information that isn't available through other page content.**
    b. **An element is *decorative* if it could be removed from the page with *no* impact on meaning or function.**
2. **If inserted content is meaningful:**
    a. **Determine whether the information in inserted content is available to assistive technologies:**
        - **Open the Accessibility pane in the browser Developer Tools.**
        - **In the accessibility tree, examine the element with inserted content and its ancestors.**
        - **Verify that any information conveyed by the inserted content is shown in the accessibility tree.**
    b. **Determine whether the information in inserted content is visible when CSS is turned off:**
        - **Use the Web Developer browser extension (CSS > Disable All Styles) to turn off CSS.**
        - **Verify that any information conveyed by the inserted content is visible in the target page.**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Table semantics

A `<table>` element must be coded correctly as a *data* table or a *layout* table.

How to test

This procedure uses the Web Developer browser extension.

1. **Use the Web Developer browser extension (Outline > Outline tables) to outline `<table>` elements on the page.**
2. **Examine each outlined table to determine its type:**
    a. **A *data* table uses rows and columns to show relationships within a set of data.**
    b. **A *layout* table uses rows and columns to visually position content without implying any relationships.**
3. **Use the Web Developer browser extension to reveal elements with ARIA roles (Information > Display ARIA Roles).**
4. **Verify that each table is coded correctly for its type:**
    a. **A *data* table *must not have* `role="presentation"` or `role="none"` on any of its semantic elements:**
        - **`<table>`**
        - **`<tr>`**
        - **`<th>`**
        - **`<td>`**
        - **`<caption>`**
        - **`<col>`**
        - **`<colgroup>`**
        - **`<thead>`**
        - **`<tfoot>`**
        - **`<tbody>`**
    b. **The `<table>` element of a *layout* table *must have* `role="presentation"` or `role="none"`.**
5. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Table headers

Coded headers must be used correctly.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement highlights elements that are coded as table headers. Coded headers include `<th>` elements and any element with a `role` attribute set to "`columnheader`" or "`rowheader`".

1. **Examine the target page to identify any *data* tables:**
    a. **A *data* table uses rows and columns to show relationships within a set of data.**
    b. **A *layout* table uses rows and columns to visually position content without implying any relationships.**
2. **Examine each data table to identify cells that function as headers:**
    a. **A cell functions as a header if it provides a label for one or more rows or columns of data.**
    b. **A cell does not function as a header if it serves any other purpose.**
3. **Verify that coded headers are used correctly:**
    a. **Cells that function as headers *must be* coded as headers, and**
    b. **Cells that do not function as headers *must not be* coded as headers.**
4. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Headers attribute

The `headers` attribute of a `<td>` element must reference the correct `<th>` element(s).

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement highlights `<td>` and `<th>` elements with a `headers` attribute set. Any element whose id is referenced by a `headers` attribute is also highlighted.

1. **Use the visual helper to reveal any `headers` attributes on the page.**
2. **If a table has `headers` attributes, inspect the page's HTML to verify that the header and data cells are coded correctly:**
    a. **Each header cell (`<th>` element) must have an `id` attribute.**
    b. **Each data cell (`<td>` element) must have a `headers` attribute.**
    c. **Each data cell's `headers` attribute must reference all cells that function as headers for that data cell.**
3. ***Note: If a `headers` attribute references an element that is missing or invalid, it will fail an automated check.***
4. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Lists

Lists must be contained within semantically correct containers.

How to test

This procedure uses the browser Developer Tools (F12) to inspect the page's HTML.

Note: This requirement applies to list containers. An automated check will fail if list items are incorrectly coded.

1. **Examine the target page to identify any content that appears to be a list. A list is a set of related items displayed consecutively. List items are usually, but not always, stacked vertically.**
2. **Examine the HTML for each list to verify that it is contained within the semantically correct element:**
    a. **An *unordered* list (plain or bulleted) must be contained within with a `<ul>` element.**
    b. **An *ordered* list (numbered) must be contained within an `<ol>` element.**
    c. **A *description* list (a set of terms and definitions) must be contained within a `<dl>` element.**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Emphasis

Words and phrases that are visually emphasized must be contained within semantically correct containers.

How to test

This procedure uses the browser Developer Tools (F12) to inspect the page's HTML.

1. **Examine the target page to identify any visually emphasized words or phrases.**
2. **Inspect the HTML for each emphasized word or phrase to verify that it's contained in an `<em>` or `<strong>` element.**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Quotes

The `<blockquote>` element must not be used to style non-quote text.

How to test

This procedure uses the browser Developer Tools (F12) to inspect the page's HTML.

1. **Search the page's HTML to determine whether the page includes any `<blockquote>` elements.**
2. **Examine each `<blockquote>` element to verify it contains a quote.**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Letter spacing

Spacing characters must not be used to increase the space between letters in a word.

How to test

This procedure uses the browser Developer Tools (F12) to inspect the page's HTML.

1. **Examine the target page to identify any words that appear to have increased spacing between letters.**
2. **Inspect the HTML for each word with increased spacing to verify that it does not include any spacing characters. Spacing characters include spaces, tabs, line breaks, and carriage returns.**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Pointer gestures New for WCAG 2.1

Functions must be operable without requiring multipoint or path-based gestures.

How to test

1. **Examine the target page to identify any functions that can be operated using either of the following:**
    a. **Multipoint gestures, such as a two-fingered pinch zoom or a three-fingered tap.**
    b. **Path-based gestures, such as dragging or swiping.**
2. **Verify that the function is *also* operable using a single-point, non-path-based gesture, such as a double-click or a long press.**
   Exception: Multi-point and path-based gestures are allowed where they are essential to the underlying function, such as freehand drawing.
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Pointer cancellation New for WCAG 2.1

Users must be able to cancel functions that can be operated using a single pointer.

How to test

1. **Examine the target page to identify any functions that can be operated using a single pointer.**
2. **Verify that at least one of the following is true:**
    a. **No down-event. The down-event of the pointer is not used to execute any part of the function.**
    b. **Abort or Undo. Completion of the function is on the up-event, and a mechanism is available to abort the function before completion or to undo the function after completion.**
    c. **Up reversal. The up-event reverses any outcome of the preceding down-event.**
3. **Exception: This requirement does not apply if completing the function on the down-event is essential to the underlying function. For example, for a keyboard emulator, entering a key press on the down-event is considered essential.**
4. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Motion operation New for WCAG 2.1

If a function can be operated through motion, it must also be operable through user interface components.

How to test

1. **Examine the target page to identify any functions that can be operated by device motion (such as shaking) or user motion (such as walking).**
2. **Verify that both of the following are true:**
    a. **The function is also operable through user interface components (such as a toggle button).**
    b. **Motion operation can be disabled by the user.**
3. **Exception: This requirement does not apply if motion activation is essential to the underlying function, such as tracking a user's steps.**
4. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Dragging movements New for WCAG 2.2

The action of dragging cannot be the only means available to perform an action, with exceptions on where dragging is essential to the functionality, or the dragging mechanism is not built by the web author (e.g., native browser functionality unmodified by the author).

How to test

1. **Examine the target page to identify elements that support dragging (such as press and hold, repositioning of pointer, releasing the pointer at end point).**
2. **Verify that there is an single pointer activation alternative that does not require dragging to operate the same function**
   Exception: This criterion does not apply to scrolling enabled by the user-agent. Scrolling a page is not in scope, nor is using a technique such as CSS overflow to make a section of content scrollable. This criterion also applies to web content that interprets pointer actions (i.e. this does not apply to actions that are required to operate the user agent or assistive technology).
3. **Record your results:**
   a. **If you find any failures, select Fail, then add them as failure instances.**
   b. **Otherwise, select Pass.**

Target sizeNew for WCAG 2.2

Touch targets must have sufficient size and spacing to be easily activated without accidentally activating an adjacent target.

How to test

1. **Examine the target page to identify interactive elements which have been created by authors (non-native browser controls).**
2. **Verify these elements are a minimum size of 24x24 css pixels. The following exceptions apply:**
    a. ***Spacing*: These elements may be smaller than 24x24 css pixels so long as it is within a 24x24 css pixel target spacing circle that doesn't overlap with other targets or their 24x24 target spacing circle.**
    b. ***Equivalent*: If an alternative control is provided on the same page that successfully meets the target criteria.**
    c. ***Inline*: The target is in a sentence, or its size is otherwise constrained by the line-height of non-target text.**
    d. ***User agent control*: The size of the target is determined by the user agent and is not modified by the author.**
    e. ***Essential*: A particular presentation of the target is essential or is legally required for the information to be conveyed.**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

UI components New for WCAG 2.1

Visual information used to identify active user interface components and their states must have sufficient contrast.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement highlights links, native widgets, and custom widgets in the target page.

1. **In the target page, examine each highlighted element in its normal state (not disabled or selected, no mouseover or input focus).**
2. **Use Accessibility Insights for Windows (or the Colour Contrast Analyser if you are testing on a Mac) to verify that the following visual information (if present) has a contrast ratio of at least 3:1 against the adjacent background:**
    a. **Any visual information that's needed to identify the component**
        i. **Visual information is almost always needed to identify text inputs, checkboxes, and radio buttons.**
        ii. **Visual information might not be needed to identify other components if they are identified by their position, text style, or context.**
    b. **Any visual information that indicates the component is in its normal state**
3. **Exception: No minimum contrast ratio is required if either of the following is true:**
    a. **The component is inactive/disabled.**
    b. **The component's appearance is determined solely by the browser.**
4. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

State changes New for WCAG 2.1

Any visual information that indicates a component's state must have sufficient contrast.

Visual helper with Accessibility Insights

How to test

Note: Contrast requirements for text within a UI component or graphic are covered in Automated checks and Adaptable content.

The visual helper for this requirement highlights links, native widgets, and custom widgets in the target page.

1. **In the target page, examine each highlighted element to determine whether it supports any of the following states:**
    a. **Focused**
    b. **Hover (mouseover)**
    c. **Selected**
2. **In each supported state (including combinations), use Accessibility Insights for Windows (or the Colour Contrast Analyser if you are testing on a Mac) to verify that the following visual information, if present, has a contrast ratio of at least 3:1 against the adjacent background:**
    a. **Any visual information that's needed to identify the component**
        i. **Visual information is almost always needed to identify text inputs, checkboxes, and radio buttons.**
        ii. **Visual information might not be needed to identify other components if they are identified by their position, text style, or context.**
    b. **Any visual information that indicates the component's current state**
3. **Exceptions:**
    a. **No minimum contrast ratio is required if either of the following is true:**
        i. **The component is inactive/disabled.**
        ii. **The component's appearance is determined solely by the browser.**
    b. **If a component has redundant state indicators (such as unique background color and unique text style), only one indicator is required to have sufficient contrast.**
    c. **Hover indicators do not need to have sufficient contrast against the background, but their presence must not cause other state indicators to lose sufficient contrast.**
4. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Graphics New for WCAG 2.1

Graphics must have sufficient contrast.

Visual helper with Accessibility Insights

How to test

The visual helper for this requirement highlights images coded as meaningful.

1. **In the target page, examine each highlighted image to identify any of the following graphics:**
    a. **Stand-alone icons (no text)**
    b. **Charts**
    c. **Diagrams**
    d. **Illustrations**
2. **For each graphic, identify the parts that are necessary for the graphic to be understood. (This is subjective.)**
3. **Use Accessibility Insights for Windows to verify that each necessary part has a contrast ratio of at least 3:1 against the adjacent background. (If you are testing on a Mac, you can use the Colour Contrast Analyser.)**
   Exception: A lower contrast ratio is allowed for logos, photos, and other cases where a specific visual presentation is essential.
4. **Record your results:**
    a. **Select Fail for any instances that do not meet the requirement.**
    b. **Otherwise, select Pass. Or, after you have marked all failures, select Pass unmarked instances.**

Redundant entry New for WCAG 2.2

Do not require people to re-enter information they have already provided via other means – e.g., as part of a previous step in the same form.

How to test

1. **Examine the target page to identify user input mechanisms that request information to be entered (for example form fields, passwords, etc.)**
2. **Verify if the information has already been requested on a previous step of the process and that the information entered previously is prepopulated in the fields or displayed on the page. If either of these conditions fail, the test is considered a failure.**
3. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**

Authentication New for WCAG 2.2

People with cognitive issues relating to memory, reading (for example, dyslexia), numbers (for example, dyscalculia), or perception-processing limitations will be able to authenticate irrespective of the level of their cognitive abilities.

How to test

Note: Text-based personal content does not qualify for an exception as it relies on recall (rather than recognition), and transcription (rather than selecting an item).

1. **Examine the target page to identify the input fields and verify whether they prevent the user from pasting or auto-filling the entire password or code in the format in which it was originally created.**
2. **Confirm whether any other acceptable authentication methods are present that satisfy the criteria such as an authentication method that does not rely on a cognitive function test.**
3. **If either of these above conditions fail, the test is considered a failure.**
4. **Record your results:**
    a. **If you find any failures, select Fail, then add them as failure instances.**
    b. **Otherwise, select Pass.**