VA | U.S. Department of Veterans Affairs

Getting Started with Keyboard Navigation & Screen Readers

Accessibility Beyond Compliance (ABC)

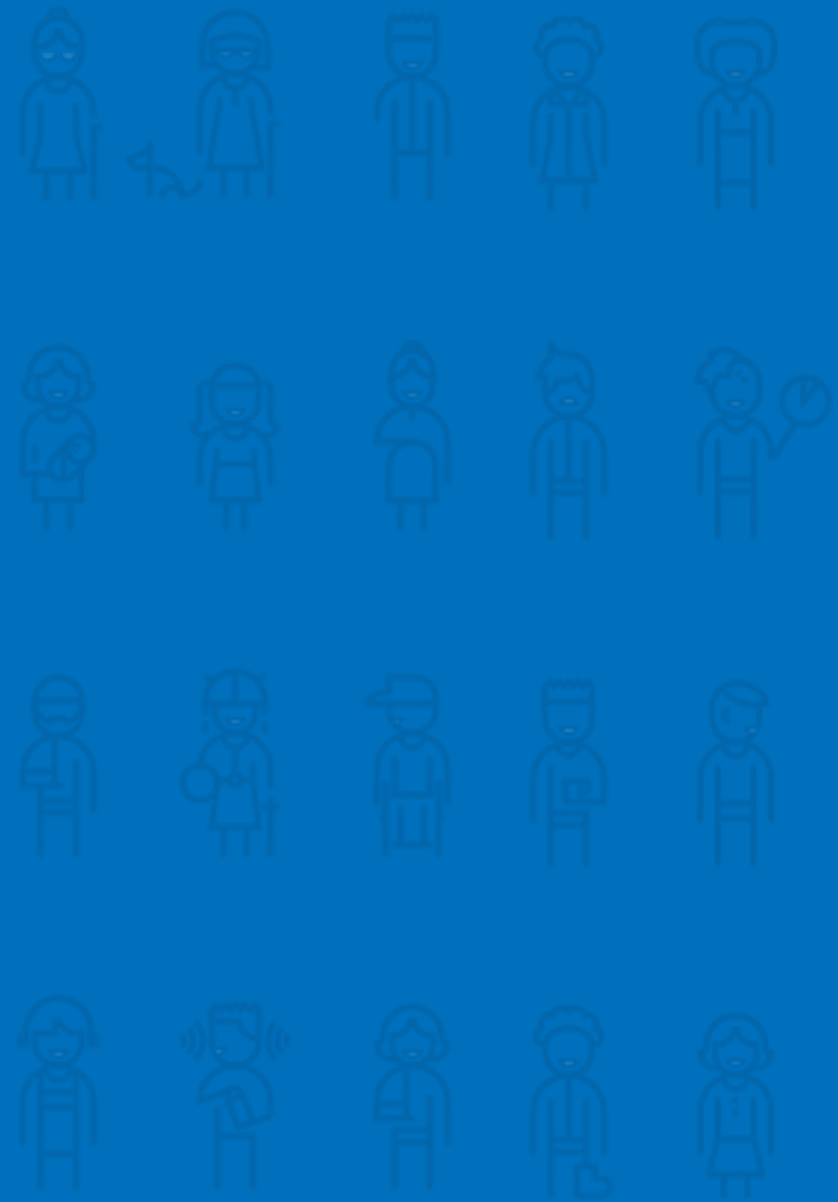
Trevor Pierce
Designer + Accessibility Specialist, VSP
trevor@adhocteam.us

Jennifer Strickland
Designer + Accessibility Specialist, VSA
jennifer.strickland@adhocteam.us

December 20, 2019

Schedule

1. 5 minutes — Chat, wait for stragglers
2. 5 minutes — Introduction
3. 25 minutes — Content
4. 15 minutes — Q&A
5. 10 minutes — Wiggle room, or free for a bio break before next meeting

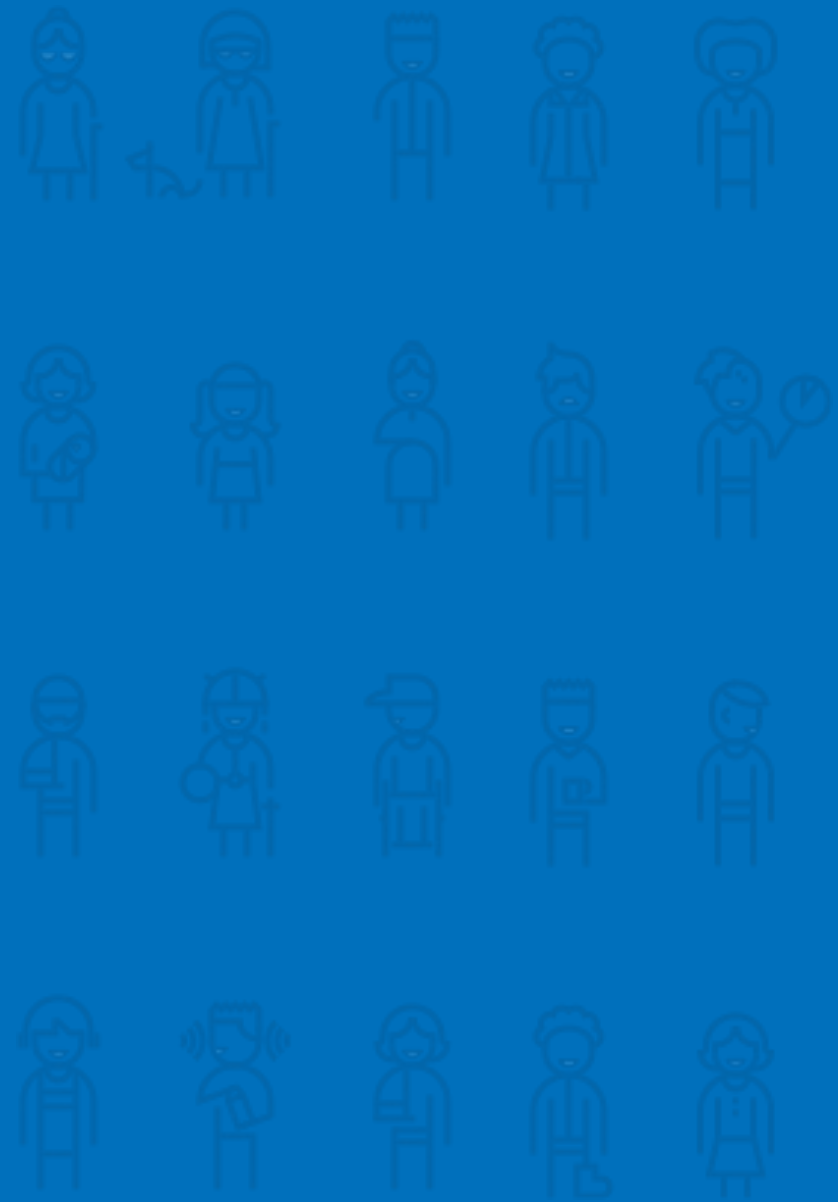


Upcoming Learning Sessions

Jan 24 Cognitive Considerations

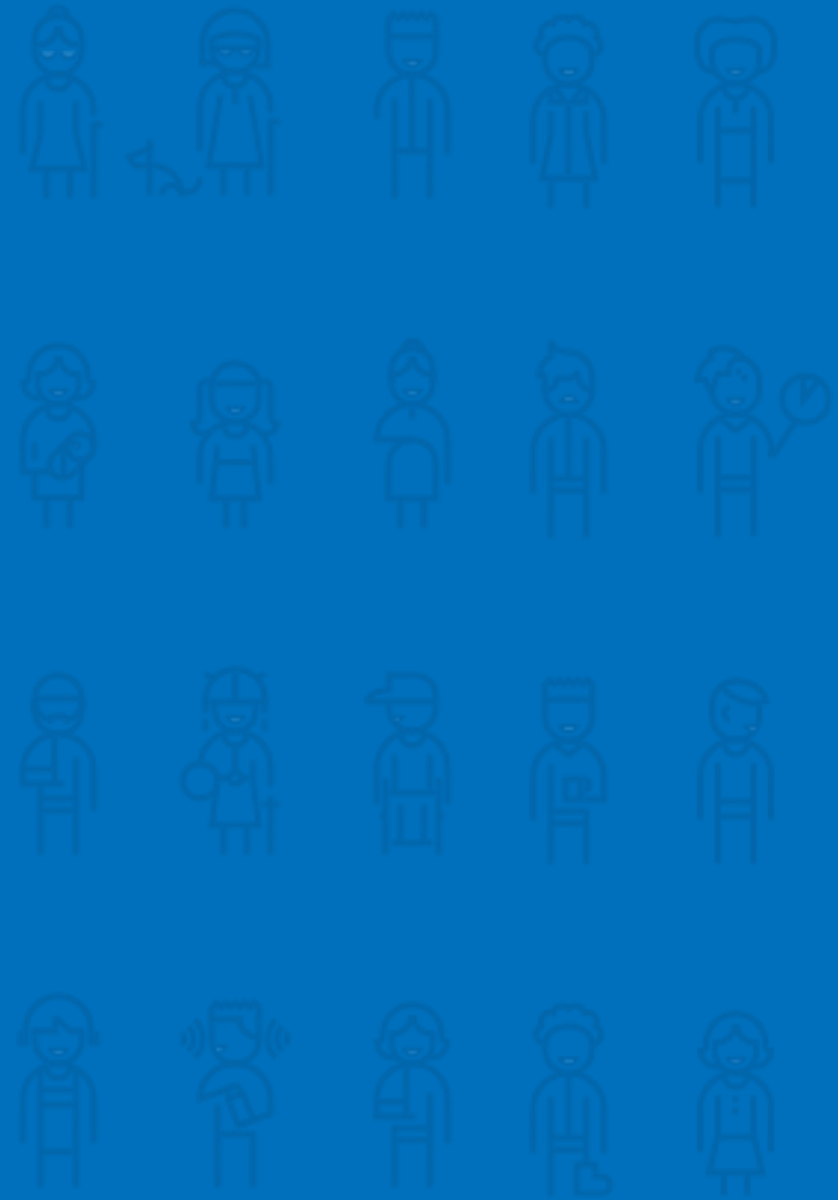
Feb 21 Web Performance Impact on UX & Accessibility

Then, we'll evaluate how these learning sessions are going, to ensure this is a useful format.



Agenda

- Keyboard navigation overview
- Screen reader navigation overview
- Accessibility API high points
- Semantic HTML
- Live demo of keyboard and screen readers



Resources

- WebAIM guide on keyboard navigation
- Ally.js - What does "focusable" mean?
- Paciello Group - the tabindex attribute
- MDN: What is the accessibility API
- Web Fundamentals - the accessibility tree
- How accessibility trees inform assistive tech
- A List Apart - semantics to screen readers
- Leonie Watson - Understanding screen reader interaction modes
- NVDA download
- NVDA focus highlight plugin
- WebAIM guide to using NVDA
- MacOS VoiceOver guide

**If you can’t navigate the keyboard,
you won’t navigate the screen reader either.**

Keyboard navigation should be the first thing we test manually. It can be abstracted (somewhat) with automated tests, but ultimately we are responsible for finding the edges and the corners.

“If I have to fill out your form with a mouse, you’ve already failed.”

— Me, trying to fill out the orthopedist’s intake form

ABC » Getting Started with Keyboard Navigation & Screen Readers

Keyboard Navigation

Basic keyboard commands

Navigation is primarily handled with these keys or chords (keys pressed together)

- **TAB** — to move forward by one focusable element
- **ENTER** — to register a click event on buttons or links, submit forms
- **SPACE** — to open select menus, or register click events on buttons
- **DOWN** and **LEFT ARROW** — to cycle forward through a radio group with focus
- **UP** and **RIGHT ARROW** — to cycle backward through a radio group
- **UP** and **DOWN ARROW** — to scroll the page up and down by small increments or interact with open select menus
- **PAGE UP**, **PAGE DOWN** — to scroll the page by larger increments in the viewport
- **SHIFT** + **TAB** — to move backward by one focusable element

Basic keyboard commands

Navigation is primarily handled with these keys or chords (keys pressed together)

TAB	to move forward by one focusable element
ENTER	to register a click event on buttons or links, submit forms
SPACE	to open select menus, or register click events on buttons
DOWN and **RIGHT ARROW**	to cycle forward through a radio group with focus
UP and **LEFT ARROW**	to cycle backward through a radio group
UP and **DOWN ARROW**	to scroll the page up and down by small increments or interact with open select menus
PAGE UP, **PAGE DOWN**	to scroll the page by larger increments in the viewport
SHIFT + **TAB**	to move backward by one focusable element

What is focusable? What is tabbable?

Ally.js defines five states an HTML element can be (https://allyjs.io/what-is-focusable.html)

1. **Inert:** Element is not interactive, cannot take keyboard focus
2. **Focusable:** Element can be focused by JavaScript and possibly the mouse.
   Often focused with the JS `element.focus()` method.
3. **Tabbable:** Element is keyboard focusable, part of the sequential tabbing order
4. **Only tabbable:** Element is only keyboard focusable, cannot be focused with scripting
5. **Forwards focus:** Element sets focus on another element instead of taking focus itself

Why does keyboard focus matter?

Keyboard focus is an important piece of the user experience.

It *should*:

- Follow the natural reading order of the page
- Reset when users add, remove, or change things on the page
- Have a clear, easy to find focus state (halo)

It *should not*:

- Skip around or scroll far on the page, then jump abruptly
- Leave users to reorient themselves when the page changes

ABC » Getting Started with Keyboard Navigation & Screen Readers

Screen Reader Navigation

How do users navigate with screen readers?

Generally speaking, there are two ways to navigate with a screen reader:

- Users can navigate focusable elements with TAB and SHIFT + TAB
- They also navigate by a second paradigm, called the virtual cursor. Leonie Watson describes the virtual cursor:

> "...by creating a virtual copy of the document, screen readers make it possible for blind people to interact with content in ways that would otherwise be impossible on the Windows platform. This happens because the screen reader intercepts most keypresses before they reach the browser, triggering an interaction with the virtual document instead."

Virtual cursor for the win

Screen readers generally have two interaction models for their virtual cursor:

- Virtual/browse mode
- Forms/focus mode

Virtual/browse mode:

- Pressing DOWN ARROW or Ctrl + Opt + RIGHT ARROW moves to the next text element
- Pressing UP ARROW or Ctrl + Opt + LEFT ARROW moves to the previous text element
- When users interact with text nodes, they are in virtual/browse mode and the screen reader generally just reads the text
    - If the text is a heading, it will read “Your text here, heading level <N>”
    - There are also shortcuts to move by headings, tables, links, forms

Virtual cursor, continued

Forms/focus mode:

- When users interact with a form element like a text input or checkboxes, Windows screen readers switch to forms/focus mode
- JAWS announces Forms mode with an audible “click”
- NVDA announces the input label, and “Focus mode”
- At this point, users can type “Horse” or “How are you” and the screen reader will register those keypresses in the input
- VoiceOver on MacOS and iOS doesn’t have this dual interaction model. It explains what element has focus and how users can interact with that element.

Interpreting the page

Screen readers are able to read the page and tell me what I’m interacting with. How is this possible?

Through the magic of **accessibility APIs**!

Accessibility is built on understanding

If I have an HTML snippet:

```
<h1 id="heading-one" class="va-heading va-heading-one">
    Facility Locator
</h1>
```

- Screen readers can tell us a lot about it:
  - This is a heading level one
  - It has an ID of “heading-one”
  - It has two classes, “va-heading” and “va-heading-one”.
    CSS can change elements’ availability in the accessibility API.
  - It has one child text node “Facility Locator”

ABC » Getting Started with Keyboard Navigation & Screen Readers

Semantic HTML

What is it?

Wikipedia defines semantics thusly:
semantics refers to the meaning of language constructs, as opposed to their form (syntax).

- Good HTML should look like a term paper outline
- My favorite example is a heading. It can be marked up two ways:
  - <h1>This is the purpose of the page</h1> ***versus…***
  - This is the purpose of the page

  These can **look** identical, but their meaning is **very** different
  - Sighted users see a "heading". It's the top of the visual hierarchy.
  - Screen reader users hear:
    - This is the purpose of the page, heading level one ***versus…***
    - This is the purpose of the page
- The H1 has semantic meaning and good descriptive text
- The SPAN asks users to assign importance to the text

Live demo time!

Live demo URLs

Generic demo:

- Any Corp Scranton - generic

Semantic demo:

- Any Corp Scranton - semantic

Automated test limits:

- Building the most inaccessible site possible with a perfect Lighthouse score