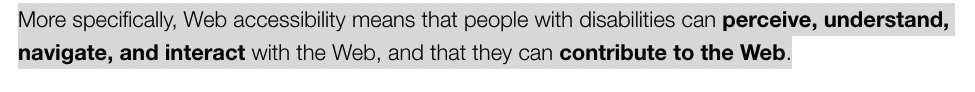
Accessibility Course

# Introduction

* **What is it?**
  + Allowing people with disabilities to use the web
    - 
* **Similar Fields (BUT NOT ACCESSIBILITY)**
  + Web Performance
    - Making website small and fast so people around the world can use it
      * Mainly on slower connection and slower devices
  + Internationalization
    - Translating app so it can be used in people’s native languages
  + UI Design
    - Making app use to use and understand
* **Types of disabilities**
  + Mobility and physical
  + Cognitive and neurological
  + Visual
  + Hearing
* **How people use the web (with disabilities)**
  + Keyboards
  + Head wand
  + Mouth Stick
  + Single Switch (one button)
  + Screen Reader
* **Curb Cut Effect**
  + Some disabled professors from Berkley poured cement from his wheelchair down a curb to create a ramp in protest of a curb being inaccessible to his wheel chair.
    - Caused human rights movement in creating curb cuts
      * Helped lots of people
        + Shopping carts
        + Strollers
        + Wheelchairs
        + Bikes
        + Skateboards
  + Lesson?
    - By making stuff more accessible for people with disabilities, it also helps other people as well.
    - Accessibility has hidden adverse effect that tend to benefit everyone.
* **WCAG**
  + 3 levels
    - A (lowest)
    - AA (recommended)
    - AAA (highest)
  + Can be hard to understand
* **WebAIM**
  + User friendly version of WCAG
    - Not the same as WCAG but a lot easier to understand
  + Checklist
    - <https://webaim.org/standards/wcag/checklist>

# Screen Readers

* **How screen readers work**
  + Turn text to speech
* **What screen readers do**
  + 1) Read all content
  + 2) List All Links on page
  + 3) List all headings in the page
* **Popular Screen Readers**
  + Chart, pie chart

    Description automatically generated
  + There can be issues with working on different screen readers
    - But will work 90-95% of the time just fine
* **Alternative Text**
  + Screen readers read the alt text when they find an image
    - If none, then they read file name
  + Background images should have empty alt text
    - Div with background image will auto ignore
  + User Uploaded Images
    - Tricky bc you can always add alt text
    - Here you can
      * 1) add generic alt
        + Alt=“User uploaded picture”
      * 2) Empty alt
        + Alt=”” (Will screen reader will skip these)
  + You can also have images hidden from S.R and images hidden from non-S.R
* **Captions of Audio**
  + Providing captions for videos so screen reader can use them

# Accessible HTML

* **Semantic Elements**
  + Some elements have no special functionality
    - <aside>
    - <footer>
    - <header>
  + Others provide a lot of functionality
    - <button>
    - <input>
    - <textarea>
  + Tips
    - Using h1-h3 properly
      * They are used for screen reader for reading headers
        + Text

          Description automatically generated
    - When you need a button, use a button
      * Don’t use div as a button
* **Labels**
  + Form fields can be confusing
    - Without using labels, it doesn’t say what to add into the input fields
  + Always add labels to forms.
    - Text

      Description automatically generated(Explicit labelling)
    - Text

      Description automatically generated(Implicit labelling)
  + Limitations for labels
    - Only the following elements can be labeled
      * Button, input, keygen, meter, output, progress, select, textarea
    - Everything else sound use Aria-label
      * 
* **Screen Reader only content**
  + Add a visually hidden class in CSS
    - Text

      Description automatically generated
  + This section on the website goes over how to turn a div into a button
    - “the life of a button” section
      * <https://learn-a11y.netlify.app/accessible-html/index.html>
    - Basically, showing all the things that a button html element does naturally and why a div that you use as a button is so much different
  + Anchor tag vs Button?
    - Use <a> when going from page to page
    - Use <Button> when providing interactivity on current page (i.e: submitting form)
  + EXERCISE 2: excises/2.html
    - Converting a page from non-accessible to accessible
      * Adding labels
      * Changing div to button
      * Adding alt text to image
      * Adding aria-label to image

# Aria Labels

* **Labels**
  + Label only works on certain html elements
  + Aria provides 3 tools for labeling non-labelable html elements
    - 1)aria-label
      * Aria-label=”github page for repo”
    - 2) aria-labelledby
      * <div id=”foo” class=”visuallyhidden”>Github page for repo</div>
      * Aria-labelleby=”foo”
    - 3) aria-describedby
  + **aria-labelledby vs aria-describedby**
    - labelledby
      * Provides essential information
        + Labelling a button
    - Describedby
      * Provides extends info
        + Mentioning that the button causes a redirect
* **Roles, States, and properties**
  + adding roles allows us to use divs instead of the native element (such as a checkbox) and bring some accessible functionality to that element
    - works for (and more)
      * button
      * checkbox
      * tree
      * banner
      * aria-autocomplete
      * aria-haspopup
    - how to use
      * <div role=” checkbox”></div>
  + aria also has states
    - learn more on mdn
* **Live Regions**
  + ****
    - When something in these changes, then the content will be read to the user
      * Even if they are past that section
    - Example: when click on “book uber” button, it changed to “finding a ride”
  + Values for aria-live (politeness settings)
    - **Text

      Description automatically generated**
    - we want to place aria-live only on direct element because it scans entire element and whenever there is a change it will read the whole element
      * have span if u need to reduce scope
  + Exercise 3
    - **Working will live regions**

# Focus Management

* Keyboard only users
  + When tabbing around, focus rings provide a clue as to the current active item
  + Table

    Description automatically generated
* Keyboard shortcuts
  + Sites like Twitter and Facebook provide keyboard shortcuts to do almost every action
    - Text

      Description automatically generated
  + Basic Keyboard Shortcut library
    - <https://github.com/jkup/shortcut>
* Skip Links
  + Skipping over the link (navbar)
    - To skip to content
  + Basic idea of creating a skip link:
    - Make a “skip to content” anchor
    - Make it first item in body of website
    - Make it visually hidden
    - When focused on, make it visible
  + Example (GitHub and NYTimes)
    - Graphical user interface, application

      Description automatically generated
* Tab Navigation
  + Default tab-able elements
    - <a>
    - <button>
    - <input>
    - <select>
    - <textarea>
    - <iframe>
  + Making element tab-able
    - Adding “tabindex” attribute
      * Types of values:
        + Text

          Description automatically generated
  + If you have a parent with a sublist of children
    - Parent should be tab-able (top level elements)
    - Then children should be moved to using arrow keys.
      * Not tab-able
    - Example:
      * Using a drop down menu
* Active Element
  + Document.activeElement returns the currently focused element.
  + When to using this?
    - In twitter when you tab into a tweet and enter a modal. When you exit that modal, we don’t want to have the tabs reset.
    - So tab caches the previously focused twitter and you will resume from there
* Tab Trapping
  + Used when within a modal
    - When user tabs through modal, at the end they will end up going to the hidden stuff under the modal
    - We don’t want that
  + Solution
    - Using JS
      * When on last tab element and there is keyDown event (TAB), change focus to first tab element in modal.
      * When on first tab element and there is a keyDown for SHIFT + TAB) then change focus to last tab element in modal
    - This makes it so the tab is always saying within the context of the modal
  + Recommended
    - Use library below to get list of focusable elements on page
    - <https://github.com/jkup/focusable>
  + PRACTICE:
    - I turned index.html for focusable into having this feature on the modal at the bottom
  + Exercise 4
    - Creating a Skip link