Website Accessibility Fundamentals

David Fisher

May 22, 2018

Contents

Chapter 1

Disability groups and issues

1.1 Issues

1.1.1 Why bother?

- It's the law
- It's a form of redundancy (failover)
- Richer user experience
- Standards compliance
- Save time and money on development and maintenance
- SEO
- It's a useful skillset to have

1.1.2 What sort of accessibility — for whom?

- **Most problems can affect anyone**
  - ◇ *Software* dependencies — browser, plugin, device, network, incl. config
  - ◇ *Intellectual* dependencies — knowledge, culture, language, etc.
- Some problems affect **specific groups** disproportionately:
  - ◇ *Visual* information — without textual and audio backup
  - ◇ *Audio* information — without textual and visual backup
  - ◇ Precise *mousing*
  - ◇ Average *intellectual demands*
- Design for the **lowest common denominator**
  - ◇ High and low capability users — edge cases
  - ◇ Standards compliance — works for all
  - ◇ Redundancy — works for all

1.1.3 Difference between disability groups

- **Visually disabled** — tend to be worst affected
    - ◇ Web (wrongly) treated as a *graphic medium*
    - ◇ Assistive technology (AT) — that doesn't implement web standards
- **Motor or mental** disability next — depends on degree
    - ◇ Both struggle with interactivity
    - ◇ Severe cases — can be worse than *minor* visual impairment
    - ◇ Both often overlooked — by tokenistic accessibility
- **The deaf** and hard of hearing
    - ◇ Traditionally least disadvantaged
    - ◇ Little site-critical information in audio streams — *changing*
- Two things to remember:
    - ◇ Practice creates problems and solutions — not technology
    - ◇ Differences *within* groups can be as significant as *between*

1.1.4 The blind

- **Common obstacles:**
    - ◇ Images/video — with no alternative text/audio description
    - ◇ Complex, hard to describe, images — maps, graphs, charts
    - ◇ Tables that make no sense when read serially — L-R, cell-by-cell
    - ◇ Forms without a logical tabbing sequence — or with poor labelling
    - ◇ Browsers and tools without *full* keyboard support
    - ◇ Proprietary formats/APIs — that screen readers struggle with
- **May use**
    - ◇ Screen readers extract text from GUI — e.g. JAWS
        - » Output to a speech synthesizer and/or braille
    - ◇ Text browsers (e.g. Lynx) — *direct* output to speech/braille
    - ◇ True audio browsers — voice input *and* output
- **Often use**
    - ◇ Rapid navigation strategies — e.g. tab through headings & links

1.1.5 Visually-impaired — or partially-sighted

- Poor acuity, tunnel vision, central field loss, clouded vision, etc.
- **Common obstacles:**
    - ◇ *Absolute font sizes* — don't scale easily

- ◇ Layouts you can't navigate when enlarged — *loss of context*
- ◇ Pages or images with *poor contrast*
  - » Especially if contrast cannot be overridden by user style sheets
- ◇ *Text as images* — preventing line wrapping when magnified
- ◇ Obstacles *specific to disability type* — and extent

- **May use:**
  - ◇ *Extra-large monitors* — & increased font/image sizes
  - ◇ Screen *magnifiers* or *screen enhancement* software
  - ◇ Specific text-colours — e.g. 24pt bright yellow on black
  - ◇ Legibility-enhanced *typefaces*

1.1.6 Colour blindness

- Meaning lack of sensitivity to certain colours
- Common forms include difficulty distinguishing between:
  - ◇ Red/green
  - ◇ Yellow/blue
  - ◇ Many other red, green, blue and yellow combinations
- May not perceive any colour at all
- **Common obstacles:**
  - ◇ Colour as a *unique marker* — e.g. for emphasis
  - ◇ Inadequate *contrast* — text vs. background
    - » Colour or pattern
  - ◇ Browsers that do not let users override designer style sheets
- **May use:**
  - ◇ Personal style sheets — to override author settings

1.1.7 The deaf

- May not read fluently — or speak clearly
  - ◇ Writing/speech may be 'second languages' for signers
- **Obstacles include:**
  - ◇ Lack of captions or transcripts — for audio
  - ◇ Lack of *content-related* images — on long/text-rich pages
  - ◇ Complex language/speech requirements — for voice input
- **May need:**
  - ◇ Captions or transcripts — for audio content
  - ◇ To concentrate harder — to read

    - ◇ Supplemental images — to highlight context

1.1.8 Hard-of-hearing

- Mild to moderate hearing impairment
- **Obstacles:**
    - ◇ Lack of toggleable captions
    - ◇ Lack of controls on streaming media
- **May need:**
    - ◇ Captions for audio content
    - ◇ Above average audio amplification

1.1.9 Motor disabilities

- Can include
    - ◇ Weakness
    - ◇ Limitations of muscular control, e.g.
        - » Involuntary movement, poor coordination, paralysis
    - ◇ Limitations of sensation
    - ◇ Joint problems
    - ◇ Missing limbs
    - ◇ Pain that impedes movement
- **Common obstacles:**
    - ◇ Lack of *keyboard alternatives* — to mousing
    - ◇ Forms — without logical tabbing sequences
    - ◇ Time-limited response requirements

1.1.10 Learning/intellectual disabilities

- Learn slowly — difficulty with *complex* and *abstract* concepts
- Wider definitions can include…
    - ◇ Substantial proportions of the population
- **Common obstacles:**
    - ◇ Unnecessarily *complex language*
    - ◇ Lack of *supportive* graphics
    - ◇ Unclear or inconsistent *organization*
- **May benefit from:**
    - ◇ More time — to complete site tasks
    - ◇ Supportive graphics — to enhance understanding

- ◇ Lower reading/hearing age requirements
  - » Minimum needed to achieve a site's purpose
- ◇ Consistent organisation — from sentence to site

1.1.11 Dyslexia or dyscalculia

- May have difficulty:
  - ◇ Processing *written* language — or images
  - ◇ Processing *spoken* language
  - ◇ Processing *numbers* — when read visually or heard
- **Common obstacles:**
  - ◇ Lack of alternative forms of information, e.g.
    - » Alternative text (convertible to audio) — to supplement visuals
    - » Lack of captions — for audio
- **May rely on:**
  - ◇ Getting information in several forms — at the same time
  - ◇ Synthesized speech
  - ◇ Captions — or captions *with* sound *as well*

1.1.12 Attention deficit disorder

- Have difficulty focusing on information
- **Common obstacles:**
  - ◇ Distracting visuals or audio — that can't be turned off *easily*
  - ◇ Lack of clear and consistent site or page organization
- **May need:**
  - ◇ To turn off animations — in order to focus on content
  - ◇ Reduced content — summaries

1.1.13 Memory impairments

- May have problems with:
  - ◇ Short-term memory
  - ◇ Missing long-term memory
  - ◇ Some loss of language
- **Obstacles:**
  - ◇ Multi-page articles
  - ◇ Inconsistent site organization
  - ◇ Dynamically changing/hidden navigation

1.1.14 Emotion-related mental health problems

- May have difficulty:
    - ◇ Focusing on information — like ADHD
    - ◇ With blurred vision or hand tremors — drug side effects
- **Obstacles include:**
    - ◇ Distracting visuals/audio — that can't be turned off easily
- **May need to:**
    - ◇ Turn off distracting visual or audio elements
    - ◇ Have distractions off *by default* — better

1.1.15 Seizure disorders

- Can triggered by audio/video signals at particular frequencies
- **Obstacles:**
    - ◇ Accidental triggering of seizures
    - ◇ By inadvertent use of key visual or audio frequencies
- **May need to turn off:**
    - ◇ Animations
    - ◇ Blinking text
    - ◇ Sound

1.1.16 Additional and multiple disabilities

- Disabilities outlined above — far from exhaustive
- Millions suffer from one or other — at mild to moderate levels
- Many suffer *multiple disabilities*
    - ◇ Significantly reducing flexibility — in the face of obstacles
    - ◇ E.g. Using strength in one area — to overcome weakness in another
- **Older people** often experience declining capabilities
    - ◇ Across a range of mental and physical activity
- Concentrating solely on the most severe forms of disability
    - ◇ May not be wise

1.1.17 Common versus specific problems

- A broad accessibility strategy not only. . .
    - ◇ Meets the needs of more disability groups
- It is often cheaper:

    - ◇ One solution — often meets the needs of many groups
    - ◇ Solutions for moderate disabilities — benefit the general population
    - ◇ Significantly reducing the number of (expensive) special cases
- Leaving resources available for specific solutions
    - ◇ To severe and particular cases

1.1.18 'Non-web' accessibility

- Web accessibility doesn't start and stop at web design
- To get onto the web in the first place
    - ◇ You need accessible software, computers and other devices
- Stuff on the web is produced and consumed off it
- Accessible websites need to interface with AT
    - ◇ Assistive technologies
    - ◇ The tools disabled people use in their daily lives
- Maybe it's non-web tools and practices that should change?
    - ◇ Is screen reading really the best way for blind people ...
    - ◇ To access computing?

1.1.19 Lynx: frequently used keys

```
g                   Goto/Open URL (inc. http://)
TAB or DOWN         Scroll forward through links
SHIFT-TAB or UP     Scroll backwards through links
ENTER or RIGHT      Follow/activate a link
u or LEFT           Back to previous page
h or ?              Help and key bindings
k                   List current key bindings
/                   Search within page
n                   Next occurrence of search term
N                   Previous occurrence of term
+ or SPACE or PgDn  Next screen down
- or b or PageUp    Previous screen up
CTRL-A or Home      Top of page
CTRL-E or End       End of page
CTRL-n              Next line
CRTL-p              Previous line
d                   Download
a                   Add bookmark
v                   View bookmarks
```

1.2 Exercises

1. The Disabled Experience

Both of the exercises below follow the same basic pattern, using different browsers. The first uses the Lynx text-based browser, and the second uses Chrome/Chromium without images or CSS, to create a similar experience.

1.1. Lynx

If you have Lynx installed, use it to:

1. (`g`)o to a typical page on your site
2. Scroll (`TAB`) through links on the page
3. Go directly to one specific link by searching (`/`) in the page
4. Activate that link (`ENTER`)
5. Loo(`k`) up current lynx key bindings
6. Return to website with `Left` key
7. Use those keys to browse around the site
8. Find and 'view' an image-rich page (the homepage?)
9. Note how images and links are labelled (or not)
10. Complete a simple form (e.g. site search, or Google?)
11. Try to complete a more complex form (e.g. shop catalogue, insurance quote, etc.). Note any difficulties.
12. Find and read a relatively complex table-based layout (e.g. Amazon)
13. Find and 'read' a relatively complex data table (e.g. football results)
14. Note any difficulties with either or both types of table

1.2. Chrome or Chromium

You can simulate a text-only browser in graphical browsers using the 'disable-HTML' extension to turn off features. Use the menu popup dots icon and go to the 'More tools' submenu, then click on 'Extensions'. Click on the 'browse the Chrome Web Store' or 'Get more extensions' link and search for "disable-HTML". The one you want is published by Daniel Lucks, and is likely to be the first result. Click on the 'Add to Chrome' button and approve its access to the necessary browser features.

To make your browser behave like a text-only one:

1. Click on the bullseye icon ◉ that should have been added to your address bar.
2. In the popup: switch images and CSS to 'OFF', and reload the page for this to take effect. The page should devolve into having stock browser styling only, and showing 'alt' text instead of images.

Try to do these exercises with just the keyboard.

1. Go to a typical page on your site. (`Ctrl+T` to open a new tab, then you should be able to type an address into the search bar.)
2. Step through links on the page (`Tab` and `Shift-Tab`).
3. Go directly to one specific link by searching (`Ctrl+F`) in the page. If the search matches in multiple places, cycle through them with `Ctrl+G` or `Enter`.
4. Deactivate the search tool (`Esc`) and activate that link (`Enter`)
5. Find and 'view' an image-rich page (the homepage?).
6. Note how images and links are labelled (or not).
7. Complete a simple form (e.g. site search, or Google?).
8. Try to complete a more complex form (e.g. shop catalogue, insurance quote, etc.). Note any difficulties.
9. Find and read a relatively complex table-based layout.
10. Find and 'read' a relatively complex data table (e.g. football results).
11. Note any difficulties with either or both types of table.

Chapter 2

Assistive technology

2.1 Assistive technology

2.1.1 Introduction

- Sites that don't support AT are inaccessible — by definition
- Problems:
  - ◇ Many ATs — per disability
  - ◇ Designs to support one product — can obstruct others
  - ◇ A lot of AT is poorly adapted — to web technologies and standards
- Keep it simple (KISS)
  - ◇ Fewer 'special' features = fewer to maintain
  - ◇ *Collate* AT requirements
    - » One technique — often answers many problems
    - » Don't code for proprietary products, devices, formats, protocols
  - ◇ Every AT *could* support web standards
    - » Designed for machine control and data processing — i.e. what ATs do

2.1.2 Screen Readers and Aural Interfaces

- **Screen reader** — the common term for voice *output*

- Designed to represent visual display — aurally
- Screen text *and* user keystrokes — are 'spoken'
  - By synthesized voices
- **Audio browser** — same output technology, but:
  - Speech-driven input
  - Reading functions are limited — to the browser
  - Some can be hooked up — to phone access
- Audio browsers may be used with
  - Speech-driven, non-GUI, operating environments
  - E.g. Emacspeak

2.1.3 Braille Displays and Interfaces

- Patterns of 6–8 raised dots represent letters and numbers
- Refreshable displays mechanically lift pins as needed
- After one line is read, the display refreshes to show the next

- Markup is comparatively easy to read and navigate
  - If semantically rich — but structurally simple
  - Complex markup (tables and forms) can be a nightmare
- Simple test:
  - Does the page make sense, read as a single column of text? (Serialised)

    Braille systems around the world vary greatly. Most modern (8-dot) systems are ASCII compatible. Although some use non-alphanumeric characters to represent common letter groupings, rather than just individual letters. Braille readers may be embedded in other computing devices, but are more often found alongside conventional computers, using a conventional keyboard for input.

2.1.4 Switch Input Devices

- Allow commands to be issued as…
  - ◇ Single yes-no type responses
- Typically used with software interfaces to the OS and apps
  - ◇ Including on-screen keyboards
- Can be very laborious to use
  - ◇ Demanding very simple interfaces
- Many specific variations on the main generic types:
  - ◇ *Mechanical* pressure switches — large buttons, caps, cup, pads
  - ◇ *Pneumatic* air switches — squeezable, sip & puff
  - ◇ *No-pressure* touch switches — plates, pads and membranes
  - ◇ *No-touch* switches — eye-brow raising, finger-flex, body tilting
  - ◇ *Sensor* switches — infra-red, sound, magnetism, etc

2.1.5 TDD and TTYs

- TDDs are 'Telecommunication Device for the Deaf'
- TTYs are text terminals — used over standard phone lines
  - ◇ Often called *minicom* in the UK (a former brand name)
- Keyboard, 1–2 line display + character encoder/decoder
- Used by deaf, hearing-/speech-impaired — for phone calls
- A TTY is normally used at both ends of the call
  - ◇ But voice users talk via Telecommunications Relay Service (TRS)

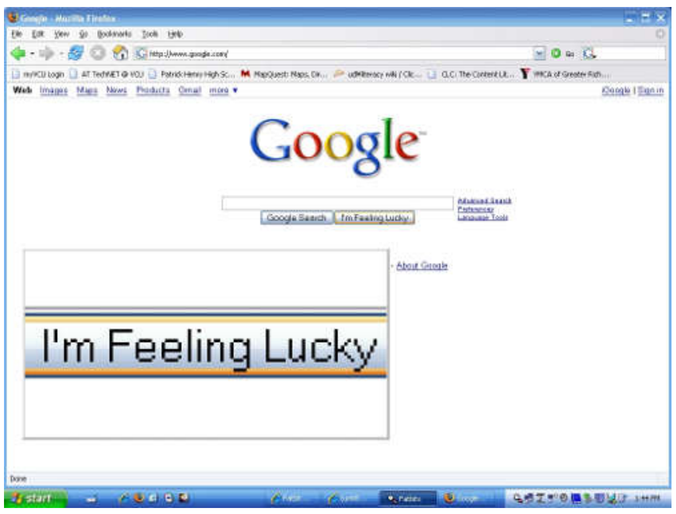
- ◇ TRS operator translates using both a voice phone *and* a TTY

  Not all TTYs use ASCII encoding, although most are dual ASCII/Baudot. And a few blind people use braille TTYs as computer terminals.

2.1.6 Modified Keyboards, Mice & Pointers

- Plug into keyboard and mouse ports — replace standard ones
- Keyboarding options include:
  - ◇ Cording keyboards — with fewer keys
  - ◇ Adaptive flat surface keyboards — with overlays and grids
  - ◇ Large keys, print, high contrast
  - ◇ Keyboard re-mapping — & alternative hardware layouts
- Mousing options include:
  - ◇ Adapters that filter out shakes
  - ◇ Haptic interfaces — tactile feedback
  - ◇ Head-mounted pointers
  - ◇ Trackballs, mats, pads and touch screens
- Key issue — is the site universally 'drivable'?
  - ◇ With standard keyboards — *and* single-button mice

2.1.7 Magnifiers



- Work like a magnifying glass
- Make part of the page/screen larger and easier to read
- Usually let users zoom in/out of a particular screen area
- Suitable for those with low or blurred vision
- Unnecessary in standards compliant pages/browsers
  - ◇ Text resizing and full page zoom is mandated
  - ◇ Not properly supported in ancient IEs
  - ◇ Fixed-dimension designs often break under text resizing

2.1.8 Optical Character Recognition (OCR)

- Converts scanned images of text into digital text
- Allowing all sorts of further processing, e.g.
  - ◇ Text to synthesised speech
  - ◇ Text to braille
  - ◇ Transforming display font typeface, colour, size
- Shouldn't really be necessary on standards compliant sites
  - ◇ Since HTML is already digitally encoded plain text
  - ◇ Plain text can be extracted from print standards like PDF
- May be used for text descriptions in bitmap images

2.1.9 Speech Recognition

- Software translates speech to digitally encoded text — for
  - ◇ Dyslexics
  - ◇ Visually impaired
  - ◇ Users with limited manual dexterity
- To dictate content — *and* issue commands
- Dictation — fairly straight-forward
  - ◇ Modern software should enable spoken markup
  - ◇ May need tweaking to produce standard markup
- Spoken commands, nav, form filling, etc. — requires support
  - ◇ In the designer's markup — *and* browser's UI

2.1.10 Touch Screens

- Enable control by pointing or touching — an area of the screen
- Typically used — by those with limited manual dexterity
- Older plastic overlays — replaced by native monitors, tablets, phones
- May be used as keyboards
- Should work with any mouse-driven software
  - ◇ Design interfaces for large *fingers*
  - ◇ *Not* small pointers — or cursors

2.1.11 Head/Eye Control

- Typically used by those with severe motor disability
- Range from laser pointing devices to simple switches
  - ◇ Mouth sticks
  - ◇ Head wands
  - ◇ Sip and puff switches
  - ◇ Eye tracking and eye mice
- May be used with touch screens and other sensors
- Typically requires:
  - ◇ Strict logical progression and navigation
  - ◇ Simple site/page structures

2.1.12 Word Prediction and Correction

- Simple spell/grammar checking — to sophisticated prediction
- Prediction ranges from simple word completion. . .
  - ◇ To longer sentence/text composition
- For dyslexia, limited manual dexterity, learning difficulties
  - ◇ Typically for form input — *and* content authoring
  - ◇ Less often — for reading and comprehension
- May need browser access to system tools — dictionaries
- Authoring & input helped — by standard usability features
  - ◇ Easy form access — and navigation
  - ◇ Sensible box sizes — line lengths, word-wrapping, scrolling
  - ◇ Replacing free form input — with simple selection
- Dictionary-based reading/writing often fails
  - ◇ On very generic — or very obscure — vocabulary

2.2 Exercises

1. Choice of platform

Use one of the two sets of exercises below, depending on what platform you're using. The first set of exercises are for Windows, and the second for Linux.

2. Using the NVDA screen reader on Windows

The purpose of this first set of exercises is to give you a feel for what the web sounds like, when you are browsing by audio. The key objective is to get you to browse guided by sound and without using a mouse. Few students find that they can actually browse by audio and keyboard alone (i.e. without looking at the screen), but it may be worth giving that a try, once you have mastered the basics.

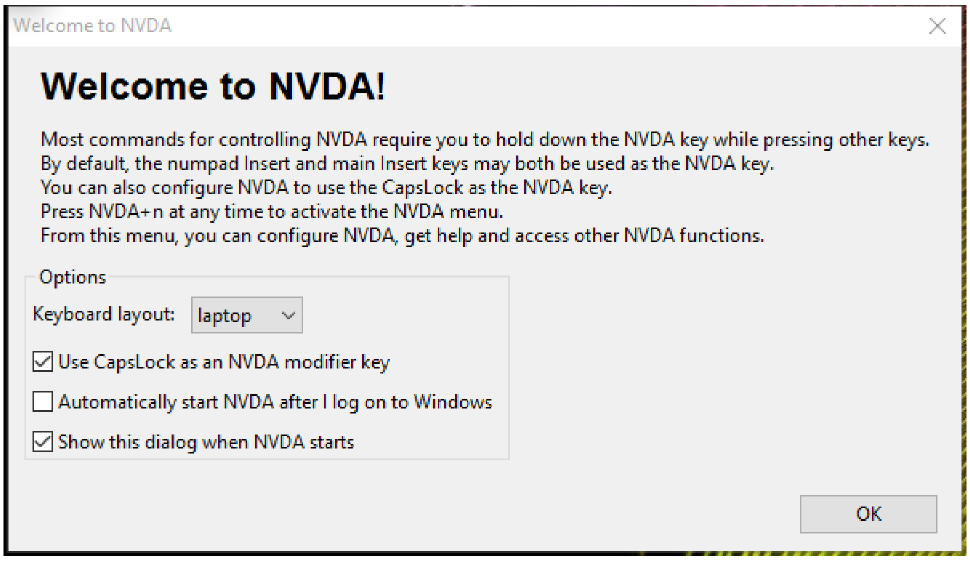
These exercises are designed to be done with the free NVDA screen reader on Windows, using Firefox as the web browser.

For these exercises, use the Pubs in Leeds example pages found in the *resources* folder, in *at/examples/*.

2.1. Starting NVDA

Start by turning on the NVDA screen reader. To do this, press '`Ctrl+Alt+N`' (that is, hold down the '`Control`' and '`Alt`' keys and press the '`N`' key). After a moment this will pop up a dialog box with some options, and start reading it to you. You can press '`Ctrl`' to stop it talking for the moment.

Make sure you have it in 'Laptop' mode, and that the option to use '`CapsLock`' as the NVDA key is turned on, as shown below. Then press OK.



Welcome to NVDA!

Most commands for controlling NVDA require you to hold down the NVDA key while pressing other keys.
By default, the numpad Insert and main Insert keys may both be used as the NVDA key.
You can also configure NVDA to use the CapsLock as the NVDA key.
Press NVDA+n at any time to activate the NVDA menu.
From this menu, you can configure NVDA, get help and access other NVDA functions.

Options
Keyboard layout: laptop
☑ Use CapsLock as an NVDA modifier key
☐ Automatically start NVDA after I log on to Windows
☑ Show this dialog when NVDA starts

You can check that NVDA is enabled by hovering the mouse over things like paragraphs in a web browser. It will start reading out the text under the cursor until you move away.

2.2. Quitting NVDA

To completely shut down NVDA, press '`CapsLock+Q`'. It will bring up a dialog box to confirm. Shut it down and use the mouse test above to check it's no longer running, then start it up again.

Another useful combination to have in mind, just in case, is '`CapsLock+N`' to bring up the NVDA settings menu.

2.3. Changing speech mode

From now on, the '`CapsLock`' key can be used to access features of the screen reader. For example, try '`CapsLock+S`' to turn the speech off, and you should hear it say "speech mode off".

The '`CapsLock+S`' combination actually switches between three modes:

1. Talk mode — when it reads out text as necessary
2. Off — when it's silent
3. Beep mode — when it just beeps when something changes

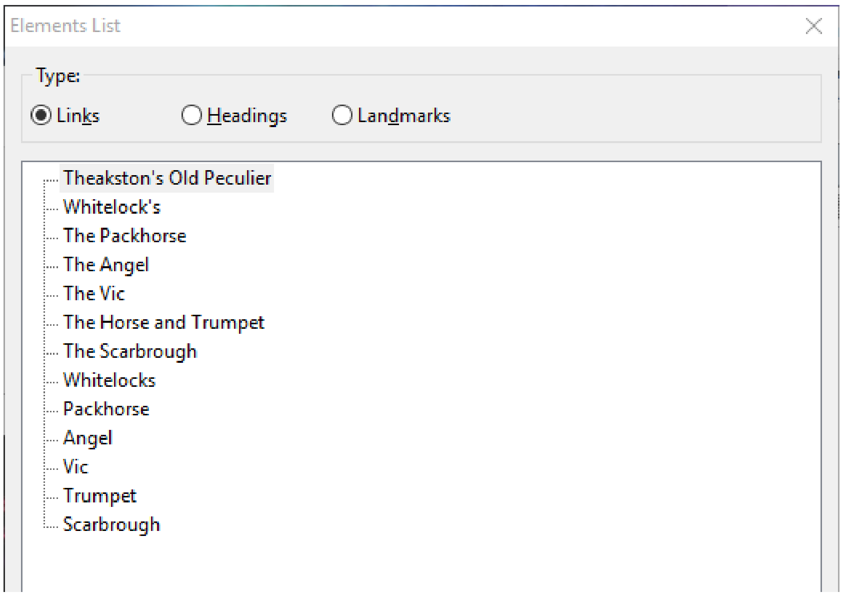
Whenever you change mode it'll tell you what one you've switched to.

Switch it back to talk mode. You'll have to switch twice to skip over the beep mode, which we don't need.

2.4. Basic navigation

Load the example page *pubs.html* (you can use '`F5`' to reload it at any time if you need to restart from the top). NVDA should start reading the page from the beginning.

1. Interrupt the screen reader with '`Ctrl`' to stop it talking, and restart it with '`CapsLock+A`'. That command means to start reading from your current position.
2. Skip ahead to the next heading, and cycle through them by pressing the '`H`' key repeatedly. You can go back to the previous heading with '`Shift+H`'.
3. Use '`H`' and '`Shift+H`' until you get to the heading "Ale trail…". Skip to the paragraph after it with '`Ctrl`' and the down arrow key. Try using the up and down arrows with '`Ctrl`' held to switch to each paragraph.
4. Try reading paragraphs in 'chunks'. Stop reading with '`Ctrl`' and this time start with '`CapsLock`' followed by the down arrow key. This will read a piece of the text up to a point, which is what NVDA calls a 'line', although they don't match the lines of text displayed visually. When it stops you can continue reading from where it left of by pressing '`CapsLock+Down`' again.
5. Use '`CapsLock+F7`' to get a list of headings. (Note that on some laptops you'll have to hold down a '`Fn`' key as well to access the function keys.)



Type: [✓ Links] Headings Landmarks

- Theakston's Old Peculier
- Whitelock's
- The Packhorse
- The Angel
- The Vic
- The Horse and Trumpet
- The Scarbrough
- Whitelocks
- Packhorse
- Angel
- Vic
- Trumpet
- Scarbrough

(a) Select the 'Headings' type to get a list only of headings. (It's easiest to do this with a mouse, but you can get there by keyboard, by using '`Shift+Tab`', using left and right arrows to change the setting, then using '`Tab`' to go back to the list.)

(b) Use the '`Up`' and '`Down`' keys to move through the list. NVDA should read each name out.

(c) Find the section about The Packhorse, then use '`Return`' to jump to it.

(d) Listen to the pubs description by continuing to read from that point with '`CapsLock+A`'.

2.5. Navigating around links

1. Try navigating around the links on the page. You can use '`K`' for links just like '`H`' for headings, including '`Shift+K`' to go to the previous link, or use '`CapsLock+F7`' to get a list of links.
2. Try '`U`' and '`V`' in the same way. They work just like '`K`', but limit themselves to only unvisited or visited links respectively.
3. Try navigating to the links at the bottom of each pub section. What difference do the `title` attributes on those links make, compared to the Theakston's Old Peculier link which doesn't have a title?

2.6. Navigating ARIA landmarks

Most screen readers nowadays support the main ARIA landmarks, whether specified with the ARIA attributes or the plain HTML5 elements that become landmarks by default. Bring up the list dialog again with '`CapsLock+F7`' and go to the 'Landmarks' view. See if you can understand the relationship between the parts of the page, the HTML5 structural elements used to create them, and the ARIA landmarks they implicitly create.

Note that NVDA reads out the type of landmark when you click on one in the list, such as "complementary landmark", so even to use a screen reader it helps to know what these areas mean.

Try navigating between landmarks directly in the page using '`D`' and '`Shift-D`'.

2.7. Images

Go to the pubs example page. Navigate to one of the pubs headers and have the screen reader read from there on, paying attention to how the image is presented. The first three pub images have `alt` attributes containing a description. The last three have an empty `alt=""` attribute. Find out what difference the attributes make to the screen reader.

2.8. Navigating tables

Load the example file *table.html*.

1. When it's loaded, jump straight to the table of beers '`T`'. It works like the other navigation keys. Notice what information is read out when you reach the table.
2. Use '`CapsLock+A`' to start reading the contents of the table in linear fashion (this will also take you from the caption into the table itself).
3. Stop the speech with '`Ctrl`', then hold down '`Ctrl+Alt`' while using the arrow keys to move from cell to cell. Move to the 'Description' column. Move down the column listening to the descriptions, until you get to the description of Old Peculier. Notice that the screen reader reads out the headings from the left-hand column, to help you keep track of where you are.
4. Move to the cells to the right in the Old Peculier row to find the price of a pint. Notice how this time, since you're changing column, the column headers are read out.

2.9. Doing a Google search

Go to the pubs example page. Find your way to the search input control:

Use '`Tab`' or '`Shift+Tab`' to switch through the focusable parts of the page until you hear the screen reader announce that you're in the "Search the web" entry. You can also use '`F`' or '`Shift+F`' to jump to form fields.

Note that NVDA can operate in 'browse mode', which is what you've been using to move around a page, and 'form mode' where typing letters like '`H`' will enter them into a form control instead of taking you to a header. When you go into a form control NVDA will switch to form mode and make a beep noise and a key-click sound effect to let you know. You can then enter a value into the form. To get back to browse mode, press '`CapsLock+Space`', which toggles between the modes.

Type in a search term you're interested in and press '`Return`'. The form should submit your query to Google and give you a Google results page.

In the Google page, try navigating forward until you find the search results landmark and have it read out with '`CapsLock+A`'. See if you can recognize what the search results are without following visually.

2.10. Navigating and filling in forms

Load the example file *form.html*.

1. Use '`Tab`' and '`Shift+Tab`' to go through the form controls (which will include the search box at the top of the page). Notice how the names of the fields are read out, because they use `<label>` elements.

2. Switch between the fields in the 'About you' and 'Choices' sections. Notice how the screen reader uses the fieldsets to give you context about where you are in the form.
3. Go back to the 'Title' field and select an appropriate one, using only the keyboard and screen reader.
4. Enter a name, but leave the email address field blank.
5. Listen to how the checkboxes are presented when you tab to them, so that you know whether they have been checked or not. Try changing one by pressing '`Space`'.
6. Try submitting the form. Without an email address (an `<input>` field which is marked `required`) it should give you an error. The screen reader should tell you there's an error, and take your cursor to the field that needs filling in. Enter an email address (a fake one is fine) and try submitting the form again.

3. Using the Orca screen reader on Linux

The purpose of this first set of exercises is to give you a feel for what the web sounds like, when you are browsing by audio. The key objective is to get you to browse guided by sound and without using a mouse. Few students find that they can actually browse by audio and keyboard alone (i.e. without looking at the screen), but it may be worth giving that a try, once you have mastered the basics.

These exercises are designed to be done with the Orca screen reader on Ubuntu Linux, using Firefox as the web browser. Chromium doesn't yet have the necessary accessibility support on Linux to work with a screen reader (although the Windows and Mac versions are accessible).

For these exercises, use the Pubs in Leeds example pages found in the *resources* folder, in *at/examples/*.

Before you do anything else, make a note of these essential key bindings for Orca:

Key	**What it does**
'`CapsLock+S`'	Stop or restart speech.
'`CapsLock+;`'	Start reading continuously, from the current point on.
'`CapsLock+H`'	Go to 'learn mode'. Press '`F1`' to open help dialog. Press '`Esc`' to exit mode.
'`Tab`' or '`Shift+Tab`'	Move focus to next or previous focusable item.

3.1. Activate caret browsing

It's easier to use a screen reader and see where you are visually on the page if you activate 'caret browsing' in Firefox, which displays a text cursor where the screen reader is pointing (it doesn't update while reading though, only when you stop). Make sure you're focussed on Firefox, and then activate caret browsing by pressing '`F7`' (with the '`fn`' key if necessary, eg on HP Stream laptops) and answering 'Yes' in the dialog box if it comes up. The cursor should then appear.

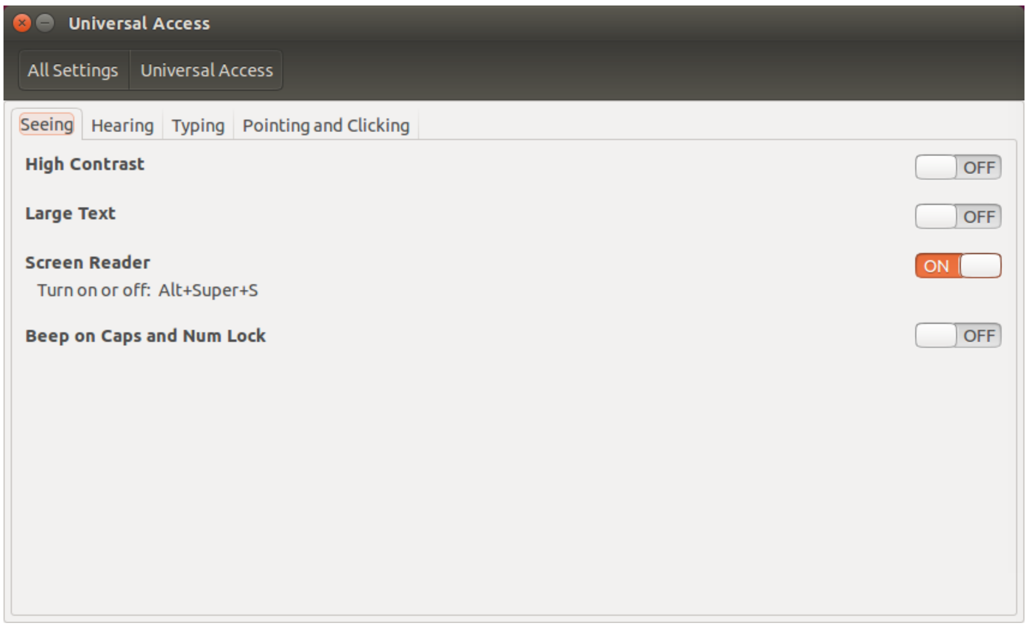
If you need to deactivate caret browsing, go to 'Edit' -> 'Preferences' in the menu, then the 'Advanced' page, the 'General' tab, and uncheck "Always use the cursor keys to navigate within pages".

3.2. Basic navigation

Start by turning on the Orca screen reader. To do this, open the settings window by clicking on the cogs icon on the left side of the screen:

Then click on the blue 'Universal Access' icon in the bottom row:

You should now see an option for the screen reader, with a switch next to it to turn it on. Click the switch.



Universal Access

All Settings | Universal Access

Seeing | Hearing | Typing | Pointing and Clicking

High Contrast	OFF
Large Text	OFF
Screen Reader Turn on or off: Alt+Super+S	ON
Beep on Caps and Num Lock	OFF

From now on, the '`CapsLock`' key can be used to access features of the screen reader. For example, try '`CapsLock+S`' to turn the speech off, and then use it again to turn the speech back on.

Load the example page *pubs.html* (you can use '`Ctrl+R`' to reload it at any time if you need to restart from the top). Orca should start reading the page from the beginning.

1. Interrupt the screen reader with '`Esc`' or '`Return`', and restart it with '`CapsLock+;`' (that's a semicolon). The semicolon command means to start reading from your current position.
2. Skip ahead to the next heading, and cycle through them by pressing the '`H`' key repeatedly. You can go back to the previous heading with '`Shift+H`'. What happens when you've gone past the first or last heading?
3. Use '`H`' until you get to the heading "Ale trail. . . ", and then skip to the paragraph after it with '`P`'. Try using '`P`' repeatedly switch to the next paragraph.
4. Use '`Alt+Shift+H`' to get a list of headings. Use the '`Up`' and '`Down`' keys to move through the list. Find the section about The Packhorse, then use '`Return`' to jump to it. Listen to the pubs description by continuing to read from that point with '`CapsLock+;`'.

Note: navigating like this, based on elements such as headings and paragraphs, is called *structural navigation* by Orca.

3.3. Navigating around links

1. Try navigating around the links on the page. You can use '`K`' for links just like '`H`' for headings, including '`Shift+K`' to go to the previous link, and '`Alt+Shift+K`' to get a list of links.
2. Try '`U`' and '`V`' in the same way. They work just like '`K`', but limit themselves to only unvisited or visited links respectively.
3. Try navigating to the links at the bottom of each pub section. What difference do the `title` attributes on those links make?

3.4. Using Firefox in-page search

1. Use Firefox's in-page search feature (activated with '`Ctrl+F`') to find the word "Smiths", which is mentioned twice in the description of The Angel. Orca will read out the line of text where it finds a match to help you tell whether you've found what you want.
2. Try pressing '`Return`' to cycle through the matches. When you're on the first of the two matches, press escape to switch to browsing at that position.
3. Read from that point on using the 'read all' option ('`CapsLock+;`'). Orca should start reading from the start of the current sentence.

3.5. Images

Navigate to one of the pubs headers and have the screen reader read from there on, paying attention to how the image is presented. The first three pub images have `alt` attributes containing a description. The last three have an empty `alt=""` attribute. Find out what difference the attributes make to the screen reader.

3.6. Doing a Google search

Find your way to the search input control.

Use '`Tab`' or '`Shift+Tab`' to switch through the focusable parts of the page until you hear the screen reader announce that you're in the "Search the web" entry. It should say that you're now in 'focus mode', which means your keystrokes will be entered into the box, instead of being used to navigate the page. For example, if you type 'H' it should appear in the text box, instead of jumping to the next heading.

Enter a search term you're interested in. The form should submit your query to Google and give you a Google results page.

3.7. Navigating tables

Load the example file *table.html*.

1. When it's loaded, jump straight to the table of beers using structural navigation (if necessary make sure it's turned on with '`CapsLock+Z`'). Either use '`T`' to jump to the next table, or use '`Alt+Shift+T`' to pop up a list of tables and jump to the right one. Notice what information is provided when you enter the table.

2. Use ‘`Alt+Shift+Right`’ to move to the ‘Description’ column. The other cursor keys with ‘`Alt+Shift`’ also work. Move down the column listening to the descriptions, until you get to the description of Old Peculier. Notice that the screen reader reads out the headings from the left-hand column, to help you keep track of where you are.
3. Move to the cells to the right in the Old Peculier row to find the price of a pint. Notice how this time, since you’re changing column, the column headers are read out.

3.8. Navigating and filling in forms

Load the example file *form.html*.

1. Get a list of form controls with ‘`Alt+Shift+F`’. Jump to the email field.
2. Use ‘`Tab`’ and ‘`Shift+Tab`’ to go forward to the first checkbox, and then back to the email field. Notice how the screen reader uses the fieldsets to give you context about where you are in the form.
3. Go back to the ‘Title’ field and select an appropriate one, using only the keyboard and screen reader.
4. Enter a name, but leave the email address field blank.
5. Listen to how the checkboxes are presented when you tab to them, so that you know whether they have been checked or not. Try changing one by pressing ‘`Space`’.
6. Try submitting the form. Without an email address (an `<input>` field which is marked `required`) it should give you an error. The screen reader should tell you where the cursor has been placed to fix the error (although at time of writing the combination of Firefox and Orca don’t read out the actual error message, it’s just displayed visually).

Chapter 3

The Equality Act (2010)

3.1 The Equality Act

3.1.1 The Equality Act 2010

- Subsumes the Disability Discrimination Act (DDA) 1999
  - ◇ Mostly re-bundles the DDA — with other anti-discrimination law
- Ensures access to goods, facilities and services — for disabled
  - ◇ Courts can require companies to make ‘reasonable adjustments’
    - » To ensure access
  - ◇ No specific mention of websites in the Act
  - ◇ But it does apply to them
- Policed by the Equality and Human Rights Commission
- Through ‘Codes of Practice’ — practical interpretations of the Act
  - ◇ With the force of law — since 2011
- Most website-relevant codes:
  - ◇ Code of Practice on Services, Public Functions and Associations
  - ◇ Guidance for service users

3.1.2 The EA covers websites — EHRC *service code*

From section 11.18

> "Websites provide access to services and goods, and may in themselves constitute a service, for example, where they are delivering information or entertainment to the public"

Example:

> “... The [organization] is responsible for ensuring that reasonable adjustments have been made where needed, for example by changing the size of the font, to ensure

> that disabled users are able to get the information, without being placed at a substantial disadvantage **(even if the [organization] employs an external organisation to build and maintain its website)**."

From section 11.9

> "... If an ISSP is established in Great Britain, then Part 3 [of the Act] applies to the provision of information society services where these are accessed in Great Britain or any EEA member state (other than the UK)."

An **ISSP** is an 'Information Society Services Provider' — as defined by Article 2 (a) of the EU's *E-Commerce Directive*

Chapter 13 of the *Service Code* describes exceptions that apply to ISSPs if they are mere conduits, or provide 'caching' or 'hosting' for other ISSPs.

3.1.3 EHRC guidance on websites and internet services

- equalityhumanrights.com/en/multipage-guide/websites-and-internet-services
- An ISSP must make sure:

  "That it does not allow discriminatory advertisements and information to appear on its website (whatever the advertisement is for)."

  "That it does not accept requests for the placing of information that unlawfully discriminates against people because of a protected characteristic in using a service."

  "That it makes reasonable adjustments to make sure that its website is accessible to disabled people."

3.1.4 "Reasonable Adjustments"

- If you discriminate — you *must* make "reasonable adjustments" to any:

  "Practice, policy or procedure which makes it impossible or unreasonably difficult for a disabled person to use a service"
- For example:

  * People with visual impairments — who use text-to-speech software

  * People with manual dexterity impairments — who cannot use a mouse

  * People with dyslexia and learning difficulties

  In making reasonable adjustments, **a service provider is not allowed to wait until a disabled person wants to use their services**. They must think in advance about what people with a range of impairments might reasonably need. If they have not done this and a disabled person wants to use a service, then the service provider must make the reasonable adjustments as quickly as possible.

3.1.5 "Reasonable Adjustments" ... continued

- Reasonable steps must be taken to provide:
  - ◇ "Auxiliary aids and services" — e.g. accessible website
  - ◇ Where these would enable or facilitate the use of a service

- "Reasonable" is not defined in the Act
- But the Code of Practice indicates that it will depend upon:
  - ◇ The type of service provided
  - ◇ The type of organisation you are and resources available
  - ◇ The impact on the disabled person

3.1.6 Powers of the EHRC under the EA

- Powers not as strong as on race and gender inequality
- But still substantial:
  - ◇ To conduct formal investigations
  - ◇ To serve non-discrimination notices
  - ◇ To enter into compliance agreements
  - ◇ To act over persistent discrimination — County Court
  - ◇ To provide assistance to plaintiffs
  - ◇ To issue Codes of Practice
    - » Failure to implement does not make companies liable
    - » But courts *must* consider their provisions
  - ◇ To conciliate — in disputes
- Requires disabled person to sue
  - ◇ Take non-compliant company to the County Court
  - ◇ But the EHRC can help them prosecute or enforce

3.1.7 Could you be sued?

- No UK case law at present — so hard to tell
- But severe or persistent offenders are at risk
  - ◇ Given the technical ease and low costs of conversion
  - ◇ Given common law precedents like the Sydney Olympic site
- RNIB have threatened action
  - ◇ Typically results in needed changes being made
- DRC investigation of 1,000 companies in 2004:
  - ◇ 80% considered practically impossible to use — by disabled people
- So low-profile sites & minor infractions — will *not* be priorities
  - ◇ Especially if they offer comparable services — via other media
  - ◇ E.g. 24 hour phone services
  - ◇ Even the EHRC doesn't publish its reports in HTML!

3.1.8 DRC Code of Practice and the WCAG

- The DRC's 2004 investigation used the W3C's Web Content Accessibility Guidelines (WCAG) as a benchmark
  - ◇ Expect UK courts to do the same
- Level of WCAG compliance required varies — according to:
  - ◇ Site services
  - ◇ Site user profile
- See BS 8878 Web accessibility — Code of Practice
  - ◇ Supersedes PAS 78 — but costs £50–£100
  - ◇ Overview — https://www.access8878.co.uk/bs8878-overview.aspx
- RNIB-suggested minimum:
  - ◇ Complete compliance with WCAG Level 1 checkpoints (A) — all sites
    - » But aim for Level 2 (AA)
  - ◇ Complete AA compliance — gvmt & universal service providers
  - ◇ AAA is encouraged — but impossible to verify

3.1.9 Top WCAG checkpoint violations

- According to the DRC's 2004 investigation:
  1. Provide *text equivalents* for non-text elements
  2. Sufficient foreground/background *contrast*
  3. Usable pages with *scripts* turned off
  4. Avoid *moving content* without agent support
  5. No *popups* or changes to current window
     - ◇ *Without* informing the user and providing agent support
  6. Divide large info blocks into *manageable chunks*
  7. Clearly identify *link targets*
  8. Clearest and *simplest language* appropriate
- A more up to date report:
  - ◇ A Quantitative Analysis Of WCAG2.0 Compliance For Some Indian Webportals
- W3C's big list of technical WCAG2.0 failures:
  - ◇ Failures for WCAG 2.0

3.1.10 The Cost of Non-Compliance

- High court costs are only likely for persistent offenders
  - ◇ The Sydney *Para-Olympic* site's fine was 20,000 AUSD
- Cost to brand image could be greater

  - ◇ Whether or not the prosecution is successful
  - ◇ Odeon and National Rail fiascos
- The cost of lost business will be substantial
  - ◇ Inaccessible sites typically deter unimpaired users
  - ◇ Competitors will spot this weakness
- N.B. On the web brand loyalty is often weak and short-lived

3.2 Exercises

1. THE DDA and the EHRC

1. To get a feel for just how illegal a site can be, try using Lynx to browse this page on the old Odeon Cinema website:
   - www.odeon.co.uk
2. See this article, for summary of the Odeon cinema site's story:
   - Odeon rolls credits on copycat website
3. Try using Lynx or a screen reader to find *and browse* the EHRC's Code of Practice on their website.
4. Try using Lynx or a screen reader to find *and browse* the British Standards Institution's *PAS 78: a guide to good practice in commissioning accessible websites*. ,end

Chapter 4

Web Content Accessibility Guidelines (WCAG) 2.0

4.1 Web Content Accessibility Guidelines 2.0

4.1.1 WCAG 2.0

- The internationally recognised standard for web accessibility
  - ◇ Created & managed by the W3C's Web Accessibility Initiative (WAI)
  - ◇ Often a benchmark for interpreting anti-discrimination legislation
- Released 11 Dec 2008
  - ◇ Updating WCAG 1.0 to account for evolution of web tech and practices
  - ◇ Was initially controversial — rejected by significant accessibility advocates
  - ◇ Still imperfect — but no longer debated much
  - ◇ N.B. some legislation may still be based on version 1.0

4.1.2 Four layers

- WCAG 2.0 is organised through 4 layers:
  1. Principles
  2. Guidelines
  3. Success criteria
  4. Techniques to meet the criteria
- See WCAG 2.0 Layers of Guidance

4.1.3 Four principles

- An accessible website must be:
  1. Perceivable

    - ◇ It must *not* be invisible to all of their senses
  2. Operable
     - ◇ The interface cannot require interaction that a user cannot perform
  3. Understandable
     - ◇ The content or operation cannot be beyond users' understanding
  4. Robust
     - ◇ As technologies evolve — content should remain accessible
- See Understanding the Four Principles of Accessibility

4.1.4 Twelve guidelines — mapped to principles

- Perceivable
  - ◇ Guideline 1.1 Text Alternatives
  - ◇ Guideline 1.2 Time-based Media
  - ◇ Guideline 1.3 Adaptable
  - ◇ Guideline 1.4 Distinguishable
- Operable
  - ◇ Guideline 2.1 Keyboard Accessible
  - ◇ Guideline 2.2 Enough Time
  - ◇ Guideline 2.3 Seizures
  - ◇ Guideline 2.4 Navigable
- Understandable
  - ◇ Guideline 3.1 Readable
  - ◇ Guideline 3.2 Predictable
  - ◇ Guideline 3.3 Input Assistance
- Robust
  - ◇ Guideline 4.1 Compatible

4.1.5 Success criteria

- Each guideline has one or more **testable** success criteria
- Each success criterion matches 1 of 3 levels of conformance:
  - ◇ A (lowest)
  - ◇ AA (medium)
  - ◇ Triple-A (highest)
- W3C assigned levels based on the answers to these questions:
  1. Is this criterion *essential*?
     - ◇ If not met, user agents (incl. AT) cannot reproduce the web page

2. Is this criterion applicable for *all* sites
    - ◇ Regardless of type, content, technology used, etc.
3. Can creators *learn* how to meet the criterion in a week, or less?
4. How will this criterion *limit* functions or "look & feel" of the page?
    - ◇ Incl. presentation, freedom of expression, design or aesthetic
    - ◇ Everyone, including not techies, should be able to contribute
5. Is there an *alternative*, indirect, way of meeting the criterion?

4.1.6 Techniques

- Each guideline and success criterion has its own techniques
    - ◇ Advice about how to achieve the success criterion and satisfy the guideline
    - ◇ See: Techniques for WCAG 2.0
- 2 categories of technique:
    1. **Sufficient** — meet the criteria, but are not mandatory
    2. **Advisory** — suggestions to improve accessibility, beyond sufficiency
- Neither are mandatory
    - ◇ You could fail a test for a "sufficient" technique
    - ◇ But satisfy the success criterion in some other way
- Advisory techniques are divided into categories:
    - ◇ General, HTML, CSS, client-side scripting, server-side scripting, SMIL, plain text, ARIA, Flash, Silverlight, PDF
- WAI also documents common errors — to be avoided
    - ◇ See: Failures for WCAG 2.0
    - ◇ 97 at the time of writing — early 2016

4.2 Conformance

4.2.1 Conformance

- Relatively easy to assess, because:
    1. WCAG 2.0 provides a suggested way of *testing* each criterion
    2. And *every* success criterion is assigned a level
- For testing, alternative, content is considered *part of* the page
    - ◇ So if a failing page has an alternate version that fulfills the criterion ...
    - ◇ You can say that a page has achieved a level — subject to certain conditions
    - ◇ See next few slides
- W3C warnings about conformance claims:

  - ◇ You *can* say "We are level A, but have also passed these AA criteria..."
    - » But you cannot claim that a site is AA — if a single page fails an AA criterion
  - ◇ AAA is hard to get and maintain, so be careful about AAA claims
    - » Especially if content is user-contributed, multimedia, etc.

4.2.2 Conformance and the 'whole page'

- Not just HTML pages or CSS styles — but any included content:
  - ◇ Videos, flash movies, games, sound, etc.
  - ◇ Linked alternatives for un-accessible content — e.g. a "longdesc" document
- If a web page is part of a process — e.g. a shop checkout
  - ◇ The complete process must be audited
- So, if there is any inaccessible info or function — e.g. AJAX bits ...
  - ◇ You must provide an accessible alternative — e.g. a non-JavaScript version
- User agents must be able to understand — and use — alternatives
  - ◇ I.e. inaccessible content or tech — must *not* block other controls
  - ◇ E.g. Flash usually traps the 'focus' — preventing keyboard use elsewhere
- Technology may be — "Turned on", "Turned off", "Not supported"
- So, give particular attention these success criteria:
  - ◇ 1.4.2 Audio Controls
  - ◇ 2.1.2 Keyboard Traps
  - ◇ 2.2.2 Pause, Stop, Hide
  - ◇ 2.3.1 Three Flashes or Below Threshold

4.2.3 The 3 parts of a conformance claim

1. Required:
   - Date — of the claim
   - Guidelines followed — title, version and URI
     - ◇ E.g. "Web Content Accessibility Guidelines 2.0 at http://www.w3.org/TR/2008/REC-WCAG20-20081211/"
   - The conformance level — A, AA or AAA
   - A brief description of the pages — with the URIs or equiv. expression
   - Content technologies used — PDF, flash, etc.
     - ◇ Incl. whether the page still conforms — if they are turned off or unsupported
       - » Links to 'known' software that can render this content are recommended
2. Optional
   - Success criteria passed — beyond the conformance claimed

   - Non-conformant tech — for which an accessible alternative is provided
   - User agents (incl. AT) you have tested with
   - A metadata version of specific technologies relied upon
   - A metadata version of the conformance claim
3. 3rd party content (required) — monitor+repair v partial conformance

4.2.4 Writing a conformance claim

- 3 parts:
  1. Which pages conform
     - ◇ E.g. All, a site section, regex-defined set of pages, individual pages
  2. What technologies used are "accessibility supported"
     - ◇ Listed — on a page (URL), page fragment (URL#), the claim itself
  3. Optional components
     - ◇ Additional criteria met, tech used (relied & not relied upon), agents tested with
- See Examples of conformance claims
  - ◇ `oneguidelineaday.com`

4.3 Testing

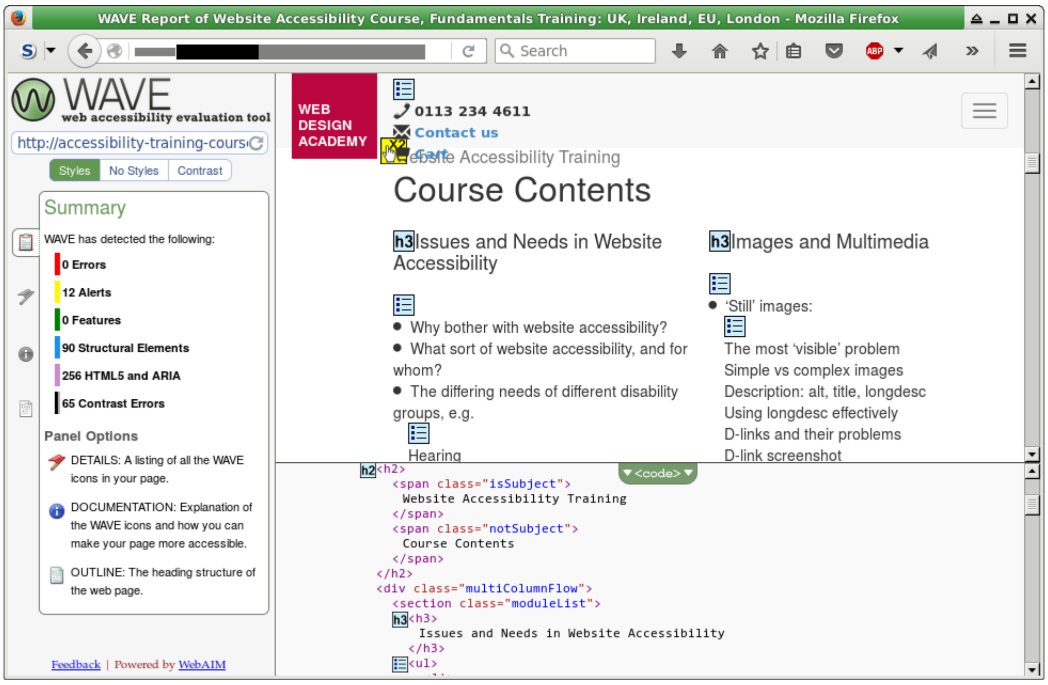
4.3.1 Automatic testing

- Automatic accessibility testing can help
  - ◇ Some problems easy to detect
  - ◇ Invalid HTML, broken links, missing `alt`, ...
- Others impossible to detect:
  - ◇ Does hierarchy of headings make sense?
  - ◇ Are descriptions like table captions accurate?
- Testing tools can highlight issues to look at
  - ◇ But won't catch everything
  - ◇ Worth using — might find problems you wouldn't have noticed
  - ◇ But don't need to blindly follow their advice
- Warnings might not mean a problem:
  - ◇ Eg, Missing `alt` value
  - ◇ Maybe not be important if it's just a decorative image
- Use tools to look for problems
  - ◇ But beware of their limitations
  - ◇ No substitute for human judgement

4.3.2 WebAim WAVE

- WAVE — Web Accessibility Evaluation tool
  - ◇ wave.webaim.org
- Uses JavaScript to evaluate page
  - ◇ Shows icons where there might be issues
  - ◇ Also highlights ARIA etc metadata
- Plug your domain into web page
  - ◇ Or install browser plugin for Chrome
  - ◇ (Firefox toolbar plugin has now been discontinued, none for IE)
- Shows three pane view:
  - ◇ Web page, with icons added where info available
  - ◇ Source code can be popped up underneath
  - ◇ Menu on left to filter results
- Not just errors — also highlights AIRA & HTML5 structure
- Click help icon for explanation of issues

4.3.3 WebAim WAVE — screenshot


WAVE Report of Website Accessibility Course, Fundamentals Training: UK, Ireland, EU, London - Mozilla Firefox

[redacted]

WAVE web accessibility evaluation tool

http://accessibility-training-cours

Styles | No Styles | Contrast

Summary

WAVE has detected the following:

- 0 Errors
- 12 Alerts
- 0 Features
- 90 Structural Elements
- 256 HTML5 and ARIA
- 65 Contrast Errors

Panel Options

DETAILS: A listing of all the WAVE icons in your page.

DOCUMENTATION: Explanation of the WAVE icons and how you can make your page more accessible.

OUTLINE: The heading structure of the web page.

Feedback | Powered by WebAIM

WEB DESIGN ACADEMY

0113 234 4611

Contact us

Cart

Website Accessibility Training

Course Contents

h3 Issues and Needs in Website Accessibility

- Why bother with website accessibility?
- What sort of website accessibility, and for whom?
- The differing needs of different disability groups, e.g.

Hearing

h3 Images and Multimedia

- 'Still' images:

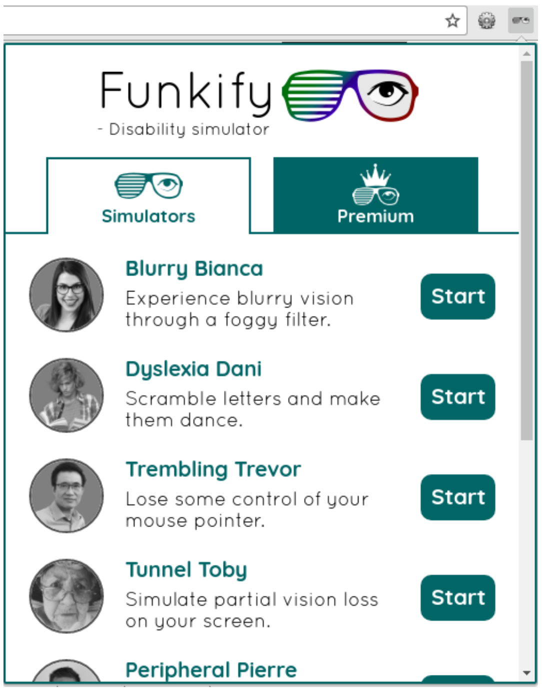
The most 'visible' problem
Simple vs complex images
Description: alt, title, longdesc
Using longdesc effectively
D-links and their problems
D-link screenshot

```
h2<h2>
    <span class="isSubject">
     Website Accessibility Training
    </span>
    <span class="notSubject">
     Course Contents
    </span>
  </h2>
  <div class="multiColumnFlow">
    <section class="moduleList">
    h3<h3>
       Issues and Needs in Website Accessibility
      </h3>
    <ul>
```

4.4 Exercises

1. Test sites with Funkify

Try the ‘Funkify’ Chrome/Chromium extension on your own website and/or a competitor’s site (some other suggested sites you can check for specific issues are listed below), by clicking the icon in the toolbar to get the popup to appear.



Blurry Bianca
Experience blurry vision through a foggy filter.

Dyslexia Dani
Scramble letters and make them dance.

Trembling Trevor
Lose some control of your mouse pointer.

Tunnel Toby
Simulate partial vision loss on your screen.

Peripheral Pierre

Test with each of the modes and evaluate whether the site is particularly badly effected by the problems being simulated.

Some specific sites you can check:

1. The homepage of https://www.theatlantic.com/
   - Evaluate with Tunnel Toby.
   - Try finding your way around the page. E.g. can you find the list of most popular current articles?
2. The footer at the bottom of http://www.art.co.uk/
   - Evaluate with Sunshine Sue — what’s the problem?
3. The text on the homepage of http://www.printmag.com/
   - Evaluate with Blurry Bianca or Dyslexia Dani.
4. The websites http://shouldiuseacarousel.com/ — which itself provides some useful accessibility advice.
   - Evaluate with Trembling Trevor and try to go to different slides of the carousel.

Chapter 5

ARIA

5.1 Accessible Rich Internet Applications

5.1.1 Introduction

- ARIA — Accessible Rich Internet Applications
  - ◇ Published by W3C WAI — Web Accessibility Initiative
- Problem:
  - ◇ Modern web sites are also applications
  - ◇ Extend HTML's user interaction with JavaScript
  - ◇ Assistive technologies don't understand this new behaviour
  - ◇ Sometimes web apps rely on interaction not supported in restricted environments
- Examples:
  - ◇ Drag-and-drop for user limited to keyboard
  - ◇ Navigate calendar popup control with speech-based browsing
  - ◇ Be alerted to dynamic updates to page
- Solution: extra markup extending HTML

  It should be noted that not all — or even most — web pages will need to use ARIA's markup extensions. There are particular cases where content can be made more accessible by annotating it with ARIA attributes, but the most common use will be for JavaScript-powered web applications, in which dynamic elements in the page provide a richer interface than HTML alone would support. ARIA can be used to expose those enhancements to the assistive technology support of the browser and operating system.

5.1.2 ARIA roles

- ARIA defines an HTML attribute `role`:

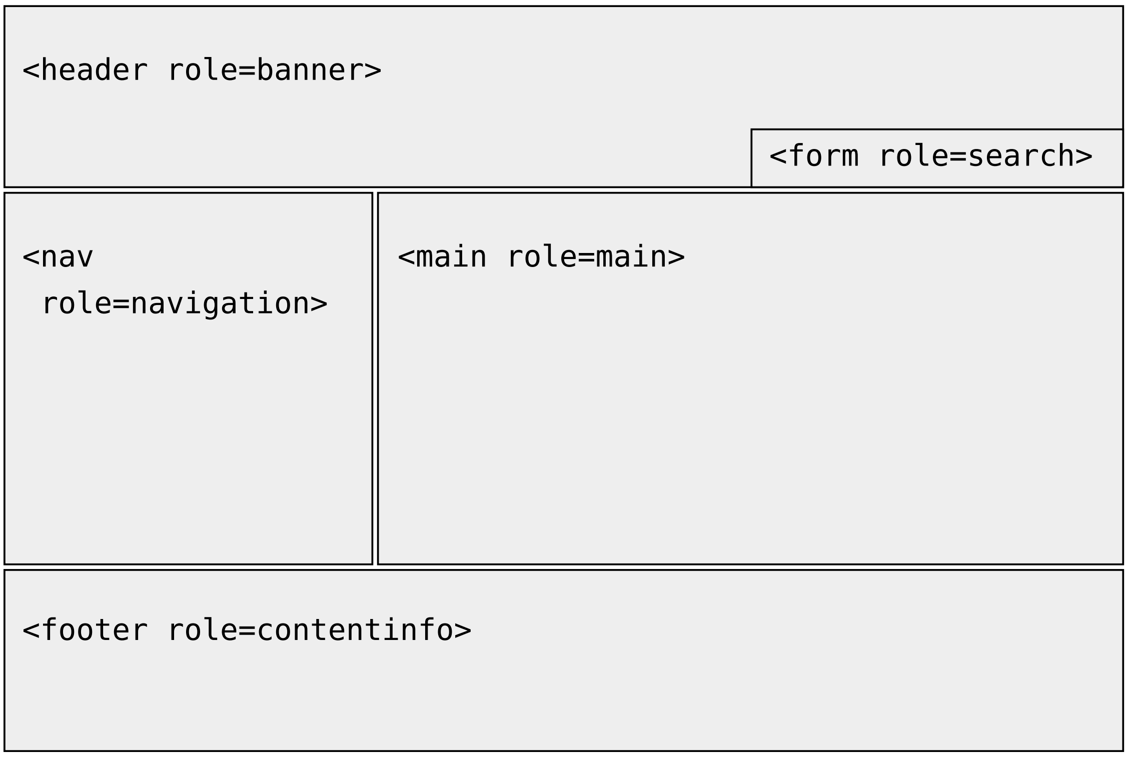
  ```
  <p role=button>Title</p>
  ```
- Overrides native HTML semantics
  - ◇ E.g., if `<button>` element can't be used for some reason

- Doesn't affect behaviour or appearance in visual browsing
  - ◇ Looks and behaves like a paragraph
  - ◇ But assistive technology treats it as a button
- Combine with CSS to make appearance consistent
- Use JavaScript to give expected behaviour
  - ◇ E.g., must respond to clicks and keyboard
- But use ordinary HTML instead when possible
  - ◇ Use ARIA to enrich, not replace, HTML semantics

5.1.3 Roles for landmarks

- Landmarks divide page into meaningful groupings
  - ◇ Context for speech browsing, and easier navigation by keyboard
  - ◇ Sophisticated version of older 'skip links' technique
- Some HTML5 elements have default roles
  - ◇ Spec says they shouldn't be specified with `role` attributes
  - ◇ But though redundant, can be a good idea to help accessibility
  - ◇ Until implementations all know HTML5's default roles
- Between them, the landmarks must include all the content
  - ◇ Or it will be hard to navigate to
  - ◇ Blind users might not realise outside content exists

5.1.4 Landmarks example



Note that in this example, the roles for the `<nav>` and `<main>` elements are actually their defaults, as defined in the HTML specification. For the time being, it's probably still worth specifying them redundantly, so that their roles are recognized by browsers which support ARIA, but aren't up to date with the latest changes in HTML.

5.1.5 Roles example: pull-down menus

- A web-based application with traditional pull-down menus
  - ◇ Start with good markup: nested lists of links, etc.
  - ◇ Add CSS and JavaScript to make them work as expected
  - ◇ Add ARIA roles to tell screen-readers etc. what they are
- For example:

```
<nav role=menubar>
 <ul role=menu>
  <li><a href="#" role=menuitem>Normal item</a>
  <li><input type=checkbox role=menuitemcheckbox>
  <li><input type=radio role=menuitemradio>
  ...
```

- JavaScript would override what link does when clicked
  - ◇ Make checkbox and radio button menu items work as expected
  - ◇ Good to add the `role` attribute with JS code
    - » Unless JavaScript not used, or always required

5.2 Other ARIA attributes

5.2.1 The `aria-describedby` attribute

- Points to content describing this element
  - ◇ E.g., full textual description of an image
  - ◇ Value is ID of element or elements containing description
- Example, a tooltip where a simple `title` isn't sufficient:

  ```
  <div aria-describedby=tooltip tabindex=0>
   Something which should have a tooltip
  </div>
  <div id=tooltip role=tooltip>
   Complex tooltip content
  </div>
  ```
- Here, CSS or JavaScript would be added to make tooltip work
- ARIA makes tooltip available to assistive technology
  - ◇ Otherwise, user might never know it's there
- `tabindex` attribute discussed later

5.2.2 aria-label + aria-labelledby

- Provides context
  - ◇ E.g., what's this content I'm looking at now?
- `aria-label` puts the label in the attribute itself:

  ```
  <ul role=menu aria-label="Main menu">
  ```
- If label is part of content — use `aria-labelledby` instead
- Identify labeling element by its ID:

  ```
  <h1 id=edit-hdr>Document Editing Area</h1>
  <textarea aria-labelledby="edit-hdr">
   ...
  </textarea>
  ```
  - ◇ Can have multiple labels — separate IDs with spaces
- Don't use `aria-label` if you can use `aria-labelledby`
  - ◇ Better to have labels visible in the content

5.2.3 Example of `aria-labelledby`

- From HTML spec:

  ```
  <figure role="img" aria-labelledby="fish-caption">
   <pre>
   o           .'`/
       '      /  (
     O    .-'` ` `'-._      .')
  ```

```
        _/ (o)        '.  .' /
        )       )))     &gt;&lt;  &lt;
        `  |_      _.'  '. \
          '-._  _ .-'       '.)
      jgs     `\__\
  </pre>
  <figcaption id="fish-caption">
   Joan G. Stark, "<cite>fish</cite>".
   October 1997. ASCII on electrons. 28×8.
  </figcaption>
 </figure>
```

- Label tells screen-readers to look at caption
    - ◇ Role `img` says ASCII art is image, not text to be read out

5.3 Dynamic pages

5.3.1 Dynamically changing pages

- JavaScript updating pages can cause problems
    - ◇ Accessibility software needs to know what's happening
    - ◇ And if user should be notified
- Mark parts of HTML as 'live regions'
    - ◇ Element whose content is updated by JavaScript
    - ◇ ARIA roles describe its function

5.3.2 Live regions

- Parts of a web page which get updated
    - ◇ E.g. chat logs, live sports scores
- ARIA can tell screen readers which elements are 'live'
- Add `aria-live` attribute:

  `<div id=chat-log aria-live=polite></div>`
- Default value is `off` — meaning:
    - ◇ No changes will happen
    - ◇ Or, they are not important — like the ticks of a clock
- Alternative values:
    - ◇ `polite` — alert user when they're otherwise idle
    - ◇ `assertive` — alert user immediately
- Use with caution
    - ◇ Don't annoy users with alerts they don't need

5.3.3 Alerts

- When the user should be alerted of something important
  - ◇ E.g. error submitting a form, password incorrect, etc.
  - ◇ Should only be used for a pressing issue
- Use `role=alert`, either:
  - ◇ Statically, in HTML sent to browser, or
  - ◇ Dynamically, by JS code adding new element
- Element with `role=alert` should be static text
  - ◇ Not interactive
  - ◇ Doesn't need to be focusable
- Alerts are always announced
  - ◇ Implies `aria-live=assertive`

5.3.4 Alert dialogs

- For interactive alerts
  - ◇ E.g. login form required before continuing
- Use `role=alertdialog`
  - ◇ Must have `aria-labelledby` or `aria-label` attribute
  - ◇ Must be modal — means a break out of normal control flow
  - ◇ So only for alerts which require immediate attention
  - ◇ E.g. login about to time out — do you want to stay logged in?
- Use `alertdialog` when focus change is required
  - ◇ E.g. when user must press a button before continuing
  - ◇ Not just new information — action required
  - ◇ If just informing user, use `role=alert`

5.3.5 Other dynamic roles

- `role=status`
  - ◇ Element which is updated
  - ◇ But updates shouldn't be announced to user
- `role=timer`
  - ◇ Element showing count-down, stop-watch, etc
- `role=marquee`
  - ◇ Element showing scrolling text
  - ◇ Implies `aria-live=polite`

    - So changes announced, but not urgently
- `role=log`
    - Where content is added dynamically
    - Content should always be added at the end
    - E.g. a chat log

5.3.6 The `application` role

- For elements containing an interactive web application
    - Where all keyboard/mouse use is handled by JavaScript
    - Tells screen-reader to leave keyboard handling to JS code
- Should be very rarely needed
    - If used inappropriately, will block accessibility
    - For same reason, use only on smallest element necessary
        - Allow normal keyboard use on rest of page
- Example where it would make sense:

```
<canvas id=space-invaders
        role=application>
  ...
```

    - Normal interaction with page wouldn't make sense for the game
    - So leave interaction with game to customized keyboard controls

5.4 ARIA with JavaScript

5.4.1 Accessible custom widgets

- 'Widgets' are interactive controls not provided by HTML
    - Implemented with JavaScript
- Example: hierarchical 'tree' control
    - Start with hierarchical bullet lists with `<ul>`
    - CSS and JS to hide unexpanded parts
    - Clicks/keyboard can open or close nested lists
- ARIA attributes added/updated by JS enable accessibility
- For a tree control:
    - Without ARIA, assistive technology treats it as a normal list
    - Doesn't know it is a tree widget
    - Doesn't know how to handle changes in visibility
- Ready-made implementations available for free, e.g.:
    - https://github.com/filamentgroup/jQuery-Tree-Control

5.4.2 ARIA attributes for tree widgets

- Label highest-level list as a tree:

  `<ul role=tree>`

- Nested lists that can be expanded are groups:

  `<ul role=group>`

- Each item listed in tree is a 'tree item'
  - And is either expanded or not:

    ```
    <li role=treeitem aria-expanded=true>
     <a href="...">Item in tree</a>
    </li>
    ```

  - Set `aria-expanded` to `false` if item is collapsed
  - CSS needed to actually hide/show the nested lists
- These ARIA attributes should be added by JS
  - Along with event handlers to handle keyboard/mouse interaction
  - Without JS, default is just normal nested bullet lists

5.5 More on ARIA

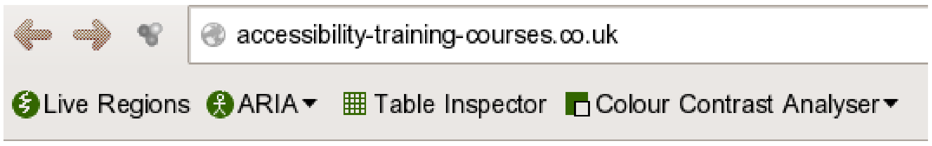
5.5.1 Using **`tabindex`** with ARIA

- Use the `tabindex` attribute to make elements focusable
  - E.g., tabbing through page should visit visible tree items
  - And skip over ones currently collapsed
  - Previous example of tooltip required `tabindex`:
    - Allow keyboard access to tooltip content
- Three different kinds of value:
  - Zero — makea element focusable
    - It's tabbing order will depend on it's position in page
  - Number bigger than zero — sets explicit position in tab order
  - -1 makes it focusable *only by JavaScript*
    - Can't get to it by hitting 'Tab' key
    - Effectively turns off focusability
    - But JavaScript could provide alternative keyboard navigation to it

5.5.2 Resources

- Main W3C page, linking to specification and guides:
  - http://www.w3.org/WAI/intro/aria.php

- HTML5 specification recommends use of some parts of ARIA:
  - ◇ http://www.whatwg.org/specs/web-apps/current-work/multipage/elements.html#wai-aria
  - ◇ Also adds requirements on how it should be used
- Firefox plugin Juicy Studio can check use of roles:
  - ◇ https://addons.mozilla.org/en-US/firefox/addon/juicy-studio-accessibility-too/

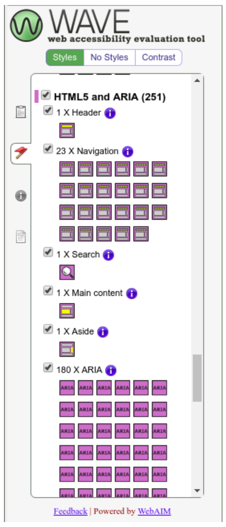


5.6 Exercises

1. Testing for ARIA features

Use the 'WAVE' extension in Chrome/Chromium to test websites. Visit the page and click the circled 'W' icon Ⓦ to view the results. You can also use the website http://wave.webaim.org/ to access the service in other browsers, but using the extension is more effective.

1. Use the tool on the Google homepage. The page should be annotated with information about the markup as coloured text and flags. Use them to find out what information is provided by ARIA markup about the search text box.
2. Click the 'Code' tab ▲<code>▲ at the bottom of the page to view the HTML source, which is also annotated with flags. Click on the magnifying glass icon in the web page to jump to the code that describes it. Find out what ARIA markup is being used.
3. Click on the flag icon on the far left of the WAVE toolbar to bring up a list of all the errors and features it found. Check to see what errors are detected. Try clicking on the buttons in the sidebar to highlight the corresponding position in the actual page, and use the information icons ⓘ to get explanations of the errors. What information is the page *not* providing and why might it be a problem for some users?
4. Go to https://whatwg.org/ and use the WAVE tool to see if it has any ARIA landmarks. Use the list of errors/features in the sidebar and scroll down to 'HTML5 and ARIA'.



2. Creating ARIA markup

1. Create a sample page divided into four parts: a header, a navigation column, a main content area, and a footer.
2. Assign appropriate landmark roles to the four areas. Make sure that between them they contain all the page's content.
3. Test the roles with the Web Developer extension in Chrome/Chromium:
   (a) Open the popup menu for the tool with the cog icon ⚙ on the toolbar.
   (b) Go to its 'Information' tab.
   (c) Use the 'Display ARIA Roles' feature.
4. Test the page with and without the roles using a screen-reader to see whether the roles make navigation around the page easier.

Chapter 6

Text and colour

6.1 Making text accessible

6.1.1 Text-only parallel sites

- Widely perceived as
  - ◇ A solution to *designer* problems rather than disabled people's
  - ◇ A form of ghettoization
  - ◇ Cheap, poor quality, software translations
  - ◇ Obstacles to the improvement of main sites
- May actually be discriminatory — and therefore illegal
  - ◇ If they lack functionality available on the graphic site
  - ◇ If they are harder to use — it happens
- Perpetuates myth — that text-only is *intrinsically* accessible
  - ◇ Web textually is often the primary obstacle for learning disabled
- More expensive than 1 site that gracefully degrades
  - ◇ Fixing bugs in one often has no benefit for the other

6.1.2 Headers: order, level and tabbing

- Screen and braille readers run through pages *serially*
- No equivalent of the random access that sighted people use
  - ◇ To block out adverts and leap to bits that interest them
- Skips, `accesskeys` and `tabindex` can help
  - ◇ But not the same as taking in an entire page — at a glance
  - ◇ And even read the good bits — without moving
- Properly marked up and nested `h1`–`h6` is essential
  - ◇ Lets text/audio browsers signify structure/importance in other ways

  - ◇ Never use style to denote headings
  - ◇ Override unwanted default styles in CSS
- Consider making headings into target anchors or even links
  - ◇ Since tabbing from item to item is the norm
  - ◇ And headings are significant summaries

6.1.3 Phrasal markup

- HTML provides many *phrase elements* which enable
  - ◇ Users and software to pick out *significant* words
  - ◇ From surrounding text
  - ◇ Aids skimming and scanning — by all sorts of users
- Hardly ever used — because of the geeky selection?
- But that deficiency is no excuse for:
  - ◇ Failing to use the more generically useful ones
  - ◇ Misusing them — e.g. using `em` and `strong` for graphic effect
  - ◇ Using purely presentational phrase markup instead (`i`, `b`, etc)
  - ◇ Misusing arbitrary tags instead (`span`)
- Get to know at least
  - ◇ `em`, `strong`, `abbr`, `dfn`, `cite`, `blockquote`, `q`
- Make up for the missing tags with `<em>` plus:
  - ◇ A meaningful `class` name
  - ◇ A more descriptive `title` attribute

6.1.4 `<abbr>`

- Has obvious accessibility benefits
  - ◇ At least for the mildly learning disabled
  - ◇ Highlighting the word(s) and providing a definition/explanation
- Some examples:

```
<abbr title="World Wide Web">WWW</abbr>
<abbr lang="fr" title="Société Nationale
                   des Chemins de Fer">SNCF</abbr>
<abbr title="Radio Detection and Ranging
  (used to locate and track moving objects)">radar
</abbr>
<abbr title="societate per azioni" lang="it">SpA
</abbr>
```

6.1.5 Quotation

- Quotations with sources help readability and provide context
- Long ones should be in `<blockquote>`
    - ◇ With the source URL in `cite` attribute

      ```
      <blockquote
       cite="http://theregister.co.uk/pdf_tags_blind/">
        ...
      </blockquote>
      ```
- Short ones should be in `<q>` with a `cite=""` attribute

  ```
  <q cite="http://www.gutenberg.org/etext/1122">
    To be or not to be</q>
  ```
- Workaround poor `<q>` and `cite=""` support in old IEs:

  ```
  &ldquo;To be or not to be&rdquo;
  <cite>
    <a href="http://www.gutenberg.org/etext/1122">
      Hamlet, by William Shakespeare</a>
  </cite>
  ```

6.1.6 Language and encoding

- HTML *require* character set declarations, e.g.

  ```
  <meta charset="iso-8859-1">
  ```
- This HTML5 method is much simpler than earlier ones
    - ◇ And is widely supported
    - ◇ See manual for outdated HTML4 and XHTML methods
- Many uses, e.g. helps search engines select readable texts
- Vital: signal any text written in a non-default language, e.g.

  ```
  <q>Don't take a <em lang="fr">laissez faire</em>
    attitude</q>
  ```
    - ◇ Enabling screen readers to switch to a French voice
- But *avoid foreign languages* — with learning disabled users
    - ◇ If unavoidable — flagging it up may reduce confusion or distress
- HTML4 and XHTML *require* character set declaration:

  ```
  <meta http-equiv="Content-type"
    content="text/html; charset=iso-8859-1" />
  ```
- XHTML also requires the document language to be specified

  ```
  <html xmlns="http://www.w3.org/1999/xhtml"
    xml:lang="en" lang="en">
  ```

6.1.7 Font size and scaling

- Designers can't actually stop users from resizing text
  - ◇ Attempts to do so often hurt the visually impaired
  - ◇ And ruin pixel-imperfect page layouts
- Ideally all text elements would be proportional to one another
  - ◇ Leaving the user's system default to set the baseline
  - ◇ Can be approximated (IE compatibly) by setting CSS:

    `body {font-size: 100%;}`
  - ◇ Then every other component to a proportion of that
- Screen magnifiers can reduce the harm of fixed font sizes
  - ◇ But their usage is hindered by fixed page/column widths
  - ◇ Fluid pages with scalable fonts are, arguably, better
  - ◇ Magnifiers may be unnecessary with scalable dimensions and fonts
- The *halation* effect of dark backgrounds is debatable
  - ◇ Do softer edges make perceived characters bigger or smaller?

6.1.8 Text quantity and complexity

- While well-marked up text helps most disabled groups
  - ◇ But text-only pages can cause problems for the *learning disabled*
  - ◇ Text-to-speech synthesis is unlikely to help
- Reading extended text is a common problem for this group
- Mixed text and graphics will not necessarily be accessible
  - ◇ But may be *less inaccessible*
- Plain English may help — but only if it fits site purposes
- Icons rarely encapsulate complex ideas simply/accessibly
  - ◇ Effective icons rely on implicit knowledge and/or interpretive skill
- Realistic pictograms, clipart, photographs — often better
- The web may never be fully accessible to learning disabled
  - ◇ It's hard to imagine the web without significant textual content
  - ◇ And stripping out text would disadvantage other groups

6.1.9 Text colour and colour deficiencies

- Colour blindness — inability to distinguish wavelengths
  - ◇ Mostly red-green and green-blue pairs
  - ◇ Red may perceived as darker — in some red-green confusion

  - ◇ No problem with black, white, grey — millions of colours/shades
- Far less trouble for web designers — than widely believed
  - ◇ Doesn't even limit you to using non-confusable colours
  - ◇ Just avoid confusable colours...
    - » When *adjacent* colour distinction is 'meaningful'
- E.g. text, links, navigation, and any icon/pictogram
  - ◇ Used for **critical action** (e.g. 'buy', 'submit' 'skip intro')
  - ◇ Contrasts with black, grey or white cover almost every case
  - ◇ Most useful for link/menu text and borders
- Set these 5 on every page
  - ◇ Foreground and background text + unvisited, visited, active links
  - ◇ Hovered links and focused links are optional

6.1.10 Setting basic colours

- In contemporary CSS:

```
body { color: black; background-color: white }
a:link { color: #990000; background-color: white;
         text-decoration: underline }
a:visited { color: purple; background-color: white;
            text-decoration: none }
a:focus { color: white; background-color: red;
          text-decoration: underline }
a:hover { color: white; background-color: red;
          text-decoration: underline }
a:active { color: white; background-color: purple;
           text-decoration: underline }
```

  - ◇ CSS mandates the LV(F)HA order — won't work without it
  - ◇ Never require hovering to make links/actions visible
  - ◇ `:focus` caters for keyboard-/touch-driven browsers
    - » They can't 'hover'

6.1.11 Ultra-safe colour combinations

- If you want to do a belt-and-braces job, the following specify:
  - ◇ Colour pairs that can be used together
  - ◇ Colour steps (gradations) that can be used to show progressions, ranges, or to differentiate multiple adjacent objects
- Red/blue, with these steps:
  - ◇ Dark, medium and light red — light, medium, and dark blue
- Orange/blue, with these steps:

  - ◇ Dark, medium and light orange — light, medium, and dark blue
- Orange/purple, with these steps:
  - ◇ Dark, medium and light orange — light, medium, and dark purple
- Yellow/purple, with these steps:
  - ◇ Yellow — light, medium, and dark purple (note the restricted list)

6.1.12 System colours for links

- Colours that users see are invariably only *estimates*
  - ◇ You don't control the user's hardware, software or environment
  - ◇ Designers need to get used to this — it will *never* change
- CSS2 introduced the idea of keywords for *system colours*
  - ◇ Partly, to assist in accessibility
  - ◇ **For users** to override designs — with colours from their system
  - ◇ Using `!important`
  - ◇ Enhanced by SVG named colours in HTML5
  - ◇ I.e. 256 names that all modern browsers support
- System colours can be useful to designers...
  - ◇ Who want to *maximise* usability for the color blind
  - ◇ By picking from ultra-safe palettes

6.2 Exercises

1. Text and colour

1. If you have Firefox installed:
   (a) Use the Web Developer Toolbar to '*Disable*' page colours, note any changes this makes to the usability of your selected pages.
   (b) Use the Web Developer Toolbar to '*Disable*' minimum font size, note any changes this makes to usability.
   (c) Use the Web Developer Toolbar CSS button to '*Disable*' disable all styles, note any changes this makes to usability.
   (d) Use the browser's Options/Preferences to reset the default font size to something very small (e.g. 6pt) and prevent pages from choosing their own fonts. Note any changes this makes to usability.
   (e) Repeat the previous exercise, but this time increase the default font size to something quite large (e.g. 24–30pt) and set this to be the *minimum*.
2. Take screenshots of some of your selected pages (using Gimp or a tool you are familiar with) and convert the screenshot images to greyscale (monochrome), note any changes to usability.
3. If you have IE installed:

(a) Try repeating the Lynx exercises from Module 1 in IE, *without* using the mouse, i.e. by keyboard only.

4. If you know CSS reasonably well, you might want to try using Firefox's Web Developer Toolbar 'Edit CSS' feature, to see how your pages perform when you deliberately reduce the contrast between foreground and background colours.

Chapter 7

Images

7.1 Accessible images

7.1.1 The most 'visible' problem

- Information conveyed in images is obviously less accessible
  - ◇ To the blind and poorly sighted
  - ◇ But also to software and hardware processors
- Excessive embedded *images undermine* the accessibility and comprehensibility of *textual content*
  - ◇ E.g. screen readers have to wade through mountains of pointless descriptions of spacers, borders and other decoration
- Provide text alternatives for key information and functionality
- May be impossible for some (Google Maps?)
  - ◇ Comparatively easy for most sites using modern HTML and CSS
- Virtually all decorative images can be handled by the CSS
  - ◇ Significantly improving performance for all users

7.1.2 Simple vs. complex images

- Most image accessibility problems are easy to solve
- Just think explicitly about the use and function of images, e.g.
  - ◇ Remove *decorative images* from markup — CSS backgrounds
  - ◇ Provide text description of key *supportive illustrations*
  - ◇ Don't describe trivial *illustrative pictures* at length
- The least tractable problems are:
  - ◇ Graphics which simplify complex data — graphs and charts
  - ◇ Graphics which encapsulate masses of data — maps
- Video imagery is less problematic than you might think

  - ◇ Blind people can be avid consumers of TV

7.1.3 Description: `alt`, `title`, `longdesc`

- The `alt` attribute is mandatory on `img`
  - ◇ An *alternative* for non-viewable images — not a supplement
  - ◇ Should describe image *function*, very briefly
  - ◇ Remember that it occupies screen space and listener time
  - ◇ See special cases for purely decorative images (below)
- `title=""` for longer, but still brief, description of *content*
  - ◇ Supplemental information — browser-native moused pop-up
  - ◇ Available on most elements — not just `img`
- N.B. Mixing `alt` and `title` safely — relies on:
  - ◇ First describing function — second describing content
- `longdesc=""` url of a *full* text description of *content*
  - ◇ Description is contained in a complete HTML page at that address
  - ◇ Designed for use alongside `alt` and `title`
  - ◇ Can also be used on `frame` and `iframe` in HTML transitional
  - ◇ See below for usage policy

7.1.4 Using `longdesc` effectively

- Poorly supported by browsers
  - ◇ Deprecated in HTML5 — but may come back
  - ◇ Originally only only iCab, Opera and Gecko/Firefox
- Screen reader support was very poor — but now OK
  - ◇ Jaws (4.01), Windows-Eyes (4.5), IBM Home Page Reader have it
  - ◇ Inconsistent: JAWS opens new window, HPR auto-fetches the page
- You should still consider it
  - ◇ Should you deprive those who do have the support/need?
  - ◇ It has significant advantages over kludges like D-links (see below)
  - ◇ Don't give browser providers an excuse to continue ignoring it
- Make sure the description page links back to the original page location — and closes superfluous windows
- *Don't over-use it* — following links, then reading unnecessary descriptive detail is time-consuming and irritating

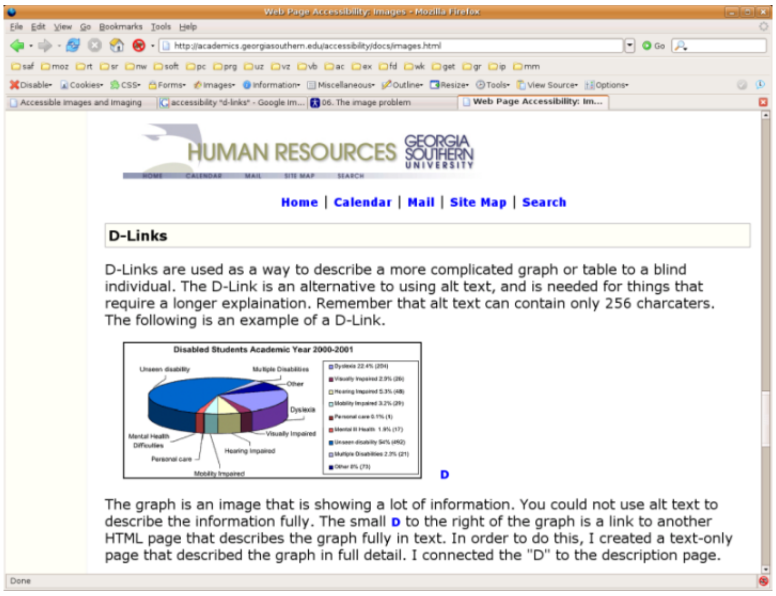
- Make sure the description page is valid HTML

  Longdesc was originally deprecated from HTML5, but has now been revived as a distinct HTML5 module http://www.w3.org/TR/html-longdesc/. For a succinct discussion of its problems, see http://tink.co.uk/2013/03/solving-the-longdesc-problem/

7.1.5 D-Links and their problems

- WAI supported attempt to remediate poor `longdesc` support
- Put a visible link to a full text description of an image right next to it
  - ◇ Containing something like 'D' or '[D]' in its anchor text
- Criticised by many:
  - ◇ Difficult to ensure D-link is always associated with the right image
  - ◇ No standard, widely recognised, format for the link text
  - ◇ Time-consuming to code, unless automated
  - ◇ Undermines efforts to get `longdesc` compliance from browsers
  - ◇ Usually looks hideous
- Don't use them to describe the overall appearance of a page
  - ◇ Doesn't really work and not much use if it did
  - ◇ Falls into the category of well-intentioned, but counter-productive

7.1.6 D-Link screenshot


Web Page Accessibility: Images - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://academics.georgiasouthern.edu/accessibility/docs/images.html
Accessible Images and Imaging
accessibility "d-links" - Google Im...
06. The image problem
Web Page Accessibility: Im...
HUMAN RESOURCES
GEORGIA SOUTHERN UNIVERSITY
Home | Calendar | Mail | Site Map | Search
Disabled Students Academic Year 2000-2001
Unseen disability
Multiple Disabilities
Other
Dyslexia
Visually Impaired
Hearing Impaired
Mobility Impaired
Personal care
Mental Health Difficulties
Done

D-Links

D-Links are used as a way to describe a more complicated graph or table to a blind individual. The D-Link is an alternative to using alt text, and is needed for things that require a longer explaination. Remember that alt text can contain only 256 charcaters. The following is an example of a D-Link.

D

The graph is an image that is showing a lot of information. You could not use alt text to describe the information fully. The small D to the right of the graph is a link to another HTML page that describes the graph fully in text. In order to do this, I created a text-only page that described the graph in full detail. I connected the "D" to the description page.

7.1.7 Text in images

- Text shouldn't really be ever be presented as bitmaps
  - ◇ Rarely rescalable — pixel units

  - ◇ Poor readability when re-scaled
  - ◇ Page layouts break when real text is re-scaled around them
- Was popular for graphic headings — no longer necessary
  - ◇ You can use background images or embed free fonts
- If someone insists on it:
  - ◇ Retain normal markup, but use CSS to hide the plain text from modern graphical browsers (`z-index` is one of several options)
  - ◇ Will never work perfectly — the web is not a fixed size medium
- Informational text may be embedded in an image — photos
  - ◇ Use `title` and/or `longdesc`, depending on quantity
  - ◇ Make sure the description distinguishes between the overall image and text contained within it

7.2 Special purpose images

7.2.1 Symbols and bullets

- Symbols like arrow images should rarely occur in markup
  - ◇ Font icons are OK in CSS backgrounds
    - » Unknown to much AT — and depend on correct charset
    - » But essentially harmless — if supported by plain text markup
  - ◇ In markup, named characters are probably better, e.g. arrows:
    - » `&larr;` or `&#8592;` for left arrow
    - » `&rarr;` or `&#8593;` for right arrow
- No bitmap images for bullets in markup, either
  - ◇ Set CSS `list-style-type` to `disc`, `circle`, or `square`
  - ◇ For bitmap prettiness use CSS `list-style` with a non-bitmap backup:

```
ul { list-style: url(/images/filename.png) disc }
```

7.2.2 SKIP ASCII art

- A real pain for screen reader users
  - ◇ Screen readers read out every character
  - ◇ No chance of listener being able envisage the image
- Use a [Skip] link to the text below
  - ◇ A source anchor:

```
<a href="#target">[Skip]</a>
```

  - ◇ to a target anchor:

```
<a name="target">
```

- Could use a `title` attribute on a `div` container to describe the image
  - May just be "too much information"

7.2.3 Backgrounds, borders, spacers, etc.

- Purely decorative images — should not be in markup at all
  - Use CSS for all spacing
  - `background` for backgrounds — images and gradients
  - `border-radius` for rounded borders
  - `text-shadow` and `box-shadow` for shadows
  - Etc.
- If you must embed decorative images in legacy markup…
  - Give them empty `alt` attributes e.g.

    ```
    <img src="foo.gif" alt="">
    ```

7.2.4 Graphs, charts and maps

- These are intrinsically graphical forms
- Difficult, often impossible, to describe textually
  - Because their purpose is to provide, easier to understand…
    - **Visual** representations of *complex* data
  - Transforming graphs and charts back into text and numbers…
    - Makes the data *harder* to understand
  - Maps (especially 3D) — just *too much* data to describe
- Some options:
  - Don't even try to encapsulate some complexity in text
    - It only confuses
  - Simple summary of key data/features
    - Using `alt`, `title` and `longdesc`
  - Very small data sets may be presented in tables alongside
    - With a `caption` — and possibly a summary para

7.2.5 Photo galleries

- Blind people's interest in photo galleries is debatable
  - But you should not deny users choice
- Use both *supportive* and *alternative* text description
- Code in HTML5 whenever possible

```
<figure>
  <img src="coppi.jpg" alt="Fauto Coppi">
  <img src="bartali.jpg" alt="Gino Bartali">
  <img src="magni.jpg" alt="Fiorenzo Magni">
  <figcaption>
    Italian cycling's post-war campionissimi.
    From left to right, Fauto Coppi, Gino Bartali,
    Fiorenzo Magni.
  </figcaption>
</figure>
```

7.2.6 Scalable Vector Graphics (SVG)

- The great hope for accessible images and scalable pages
- Vector graphics are intrinsically scalable and essentially text
  - ◇ Text describes object co-ordinates and paths between them
- SVG is a W3C standard specifically designed for web graphics
  - ◇ Images are defined by XML compliant markup
  - ◇ Intrinsically self-describing, plus great support for metadata
- Native support in all modern browsers
  - ◇ Firefox, Safari, Chrome, Opera, IE9+
  - ◇ None in IE6–8
  - ◇ Several plugin viewers, e.g. Adobe SVG Viewer
  - ◇ You can always fall back to figcaption, alt, title metadata
- Historically poorly supported in AT, but
  - ◇ Some screen readers can drive plugin viewers
  - ◇ Native support technically simple — should come soon
  - ◇ Consider transforming SVG XML to text-based alternative via XSLT
  - ◇ Successful research experiments in rendering SVG on tactile maps

7.3 Exercises

1. Images and accessibility

The resources you'll need for these exercises can be found in *resources/images/*.

1. Find the resource file called *undescribed-images.html*. It contains a mini photo gallery.
   (a) Save a copy of the file, calling it *described-images.html*
   (b) Add `alt` and `title` attributes to *all* the images in the copy.
   (c) Add `longdesc` to at least one of the images. You'll need to either create a new page containing the description so that you can provide its URL, or put the description somewhere on the existing page and use a fragment to reference it.

(d) You can check that your `longdesc` is linking to the right place in Firefox by right-clicking on the image and selecting 'View Description'. If possible, test the effectiveness of these attributes in a text browser and a screen reader.

(Answer on page 113)

1.1. D-Links instead of `longdesc`

If there is time, try replacing the `longdesc` attributes with 'D-Links'. Test them in several modern graphical browsers.

1.2. Objects and iframes

If there is time, try replacing the `<img>` tags with `<object>` and/or `<iframe>` elements. Test them in several modern graphical browsers.

Chapter 8

Navigation

8.1 Accessible navigation

8.1.1 Understanding focus and cursors

- Page components have *focus* if *selected*, *active*, or the *next action* applies to them, e.g.
  - ◇ Selecting text gives it focus — e.g. for copying
  - ◇ Clicking in browser window gives it focus — follow links, scroll, etc.
  - ◇ Click in address bar and it gets focus — to type in a URL or search
- Practically impossible to drive a browser if:
  1. You don't know **where** the focus is
  2. You can't **shift** focus to where you need it to be to perform actions
- General users only notice it when a browser bug crops up
  - ◇ Usually working around it, by clicking where they want focus to be
- Difficult for the blind — so avoid:
  - ◇ Frames, pop-ups, etc. that shift/detach focus and nav context
- Software uses multiple cursors to track focus, e.g. one in the browser window, one in a plugin and one in a screen reader
  - ◇ Easily manageable by clever software, less so by humans

    The location of focus is not always as obvious as it is in the examples given in this slide. For example, if you click on an IE favourite, the URL in the address bar is selected, but the browser window gets the focus. Some browsers allow users to make their window cursor visible and let the user control it directly from the keyboard, which is useful. In Firefox, pressing the F7 key will turn on this 'Caret Browsing' feature.

8.1.2 Consistent and unambiguous

- Internal site consistency is important
  - ◇ Don't change block positions, metaphors, or key decoration

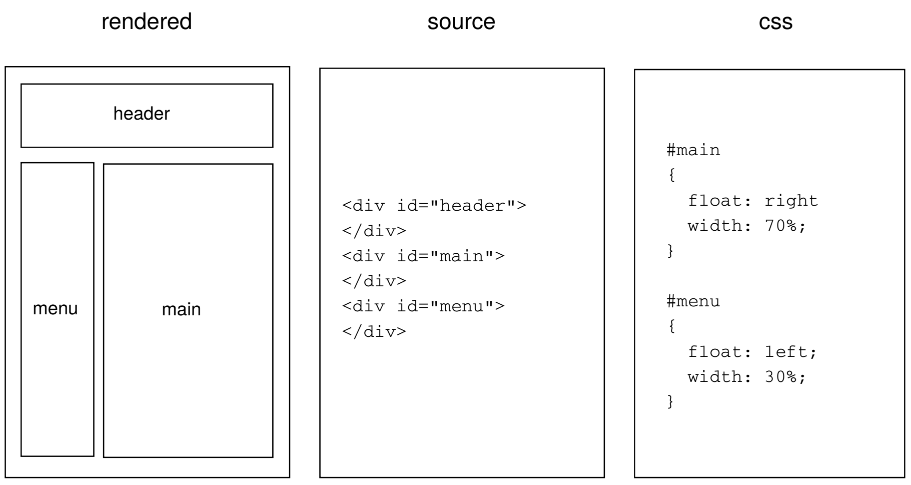
- Consistency with the rest of the web maybe more
    - ◇ People don't learn to navigate your site by visiting it
- On every page:
    - ◇ Clickable logo to homepage — top left
    - ◇ Search box — top right
    - ◇ Text link to home page (top of menu) — plus breadcrumbs?
- Label and distinguish nav from similar *nearby* structures
    - ◇ Other lists, links, feeds in the same column
- Avoid 'virtual' site structures which shift contexts, disappear, are separated from browsing, overly complex
    - ◇ If site size/complexity requires these, reduce the site/section
    - ◇ Reveal as much real structure as users can handle
- N.B., ordinary links are navigation — don't mess with them

8.1.3 How and why to skip navigation

- Classic problem: irreducibly long left-hand menus
    - ◇ No one wants to tab or read through them to get to page content
- Two solutions:
    1. Out of order CSS positioning:
        - ◇ Put main content first in the markup and menus second
        - ◇ Float content to the right leaving space on the left for menus
    2. Simple text link to a target anchor at the top of content
- Another option: skip intermediate menu items
    - ◇ Use metadata links in the `<head>` to browse without having to tab through uninteresting menu items

```
<link rel=home href="/" title="Training Home Page">
<link rel=contents href="index.html" title="Section">
<link rel=prev href="visibility.html" title="Visibility">
<link rel=next href="usability.html" title="Usability">
```

8.1.4 Float out of order diagram



```
<div id="header">
</div>
<div id="main">
</div>
<div id="menu">
</div>
```

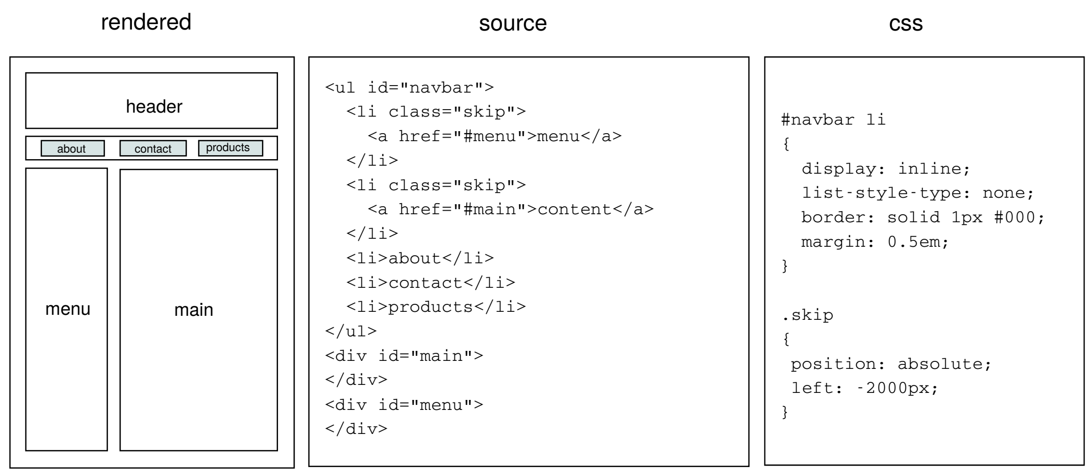
```
#main
{
  float: right
  width: 70%;
}

#menu
{
  float: left;
  width: 30%;
}
```

8.1.5 Skipping 2-Part Navigation Structures

- Navbars and left menus have different functions
- Don't assume that because people want to skip one. . .
  - ◇ That they necessarily want to skip both
- Simple navbars may be tabbed through without grief
  - ◇ And provide site orientation on first visit
- For more complex ones, put skip links in the navbar itself
  - ◇ To the top of left menus — first
  - ◇ To the top of the main content
  - ◇ Gives the option of section orientation *before* reading

8.1.6 Skip navigation diagram



```
<ul id="navbar">
  <li class="skip">
    <a href="#menu">menu</a>
  </li>
  <li class="skip">
    <a href="#main">content</a>
  </li>
  <li>about</li>
  <li>contact</li>
  <li>products</li>
</ul>
<div id="main">
</div>
<div id="menu">
</div>
```

```
#navbar li
{
  display: inline;
  list-style-type: none;
  border: solid 1px #000;
  margin: 0.5em;
}

.skip
{
 position: absolute;
 left: -2000px;
}
```

8.1.7 Navigation skipping and search

- May need to skip **to** search — depends on markup order
- *Never skip the search itself* — it’s a critical nav structure
- Top positioned search usually doesn’t need skipping to
  - ◇ But top-right in table layout may do
- If you provide a skip, integrate it with skips to the menus
  - ◇ Navbar first, then menus, then search
- If search is on a separate page, fix that
  - ◇ Obstructs usability for everyone — and irritates
- If you really must skip to a separate search page
  - ◇ Warn about it using `title=""` in the link
  - ◇ Try to provide a way back to the current *context*

8.1.8 Other page landmarks

- Identify essential and non-essential landmarks on every page
- Other nav landmarks that may be too important to skip:
  - ◇ A login section, with userID and password fields
  - ◇ Links to printable pages or similar (PDF?), “Email this article”, etc.
  - ◇ Links to sidebars in an article and supplementary articles
  - ◇ Next, Back, and Top of Page buttons
  - ◇ A contact or `mailto:` link on a personal homepage

8.1.9 Page extremities

- Don't put an input box (e.g. search) first in the source
    - ◇ Text browser cursors can get stuck in it — notably Lynx
    - ◇ Lynx automatically selects the first actionable region on a page
    - ◇ Not a big problem — usually site logo links come first
        - » Lynx users are a small minority
- Linking to page top from the bottom might seem redundant
    - ◇ Most keyboards have `Home` and `PageUp` keys
- But some don't:
    - ◇ Macs, some Unix/Linux, some handhelds, etc
- Also, scrolling to the top doesn't put the cursor there
    - ◇ Requiring further keystrokes or actions
- Linking to page bottom may be redundant
    - ◇ Depends on content and structure

8.2 Links

8.2.1 Identifying link destinations

- Ensure that users know what to expect before they click
    - ◇ Browsing is 'costly' for many disabled groups
    - ◇ Especially recovery from mistaken navigation choices
- Link anchors should say where and what they lead to:
    - ◇ Never "Click Here"
    - ◇ Always fullest textual description possible
    - ◇ Differentiate one link from another
- Put `title` attributes on links
    - ◇ Mouse hovering reveals a fuller description — *before* clicking
    - ◇ Useful, but unreliable
- Suggested `title` attribute content:
    - ◇ *Name* of destination site/section
    - ◇ *Relationship* of target to the current page or element
    - ◇ *Warnings* about possible problems at the destination
        - » E.g. Flash, frames, subscription, PDF, video ...

8.2.2 Links to page fragments

- Skipping to page fragments is technically easy, but
  - ◇ Too many skips — reproduce the clutter you are trying to skip
  - ◇ Context needs to be clearly labelled
- Skipping within the *current* page?
- Or to a fragment of a *different* page?

8.2.3 Separating consecutive links

- Two adjacent links can easily blur into one another, e.g.
  - ◇ If only space, colour, or element boundary separate them
- Ensure clear visible/audible separation for *all* users
  - ◇ Provided *automatically* by list markup for menu blocks
  - ◇ But not for ordinary, inline, links — by default
  - ◇ Should be done with list markup and CSS
    - » I.e. make list items stack horizontally *in lines*
    - » Using `display:inline;`
- Old methods of separation — with `|` and `<hr>`
  - ◇ Should rarely be used — but can sometimes be handy

8.3 Access keys

8.3.1 The `accesskey` attribute

- Adds keyboard shortcuts to web pages
- Allowing users to
  - ◇ *Skip* directly to a specific location
  - ◇ Pass the *focus* to that location
  - ◇ And *activate* that location
- E.g. typing a one- or two-letter shortcut
  - ◇ Jumps to a link, form input box, paragraph, etc.
- Put an `accesskey` attribute on these elements:
  - ◇ `<a>`, `<area>`, `<button>`, `<input>`, `<label>`, `<legend>`, `<textarea>`
  - ◇ Giving each `accesskey` a *unique* single character value
- Remember that the `<a>` element is for *target anchors* as well as ordinary links
- *Don't over-use it!*

8.3.2 Problems with `accesskey`

- Typically helps the motor-impaired and hinders the blind
  - ◇ The blind are heavy keyboard customisers — of necessity
- Potential character-encoding, keyboard mapping, slip-ups
  - ◇ Stick to ASCII `A–Z` plus `0–9`
- Invoking access keys is browser dependent
  - ◇ Several graphical browsers follow the IE model — `ALT+KEY`
  - ◇ A single keystroke (e.g. Lynx) makes much more sense
- Some browsers don't support `accesskey` at all

8.3.3 Making access keys visible

- List site-wide access keys high on your 'Accessibility' page
- Several page-specific 'solutions' have serious problems
  - ◇ Underlining the `accesskey` letter in link text
  - ◇ Using the CSS `:first-letter` pseudo-class
  - ◇ Signaling the `accesskey` letter in an `alt` attribute
- Ideally, put `accesskey` *in plain text*, straight after the link:

```
<a href="/home.html" title="Homepage" accesskey="H">
 Homepage</a> [<kbd class="accesskey">H</kbd>]
```

  - ◇ Sometimes called an *accesskey legend*
  - ◇ Use a `kbd.accesskey` CSS selector to style it distinctively
- Put `title` on legend to explain the system you're using to:
  - ◇ Non-disabled users, mobility-impaired users, screen readers

```
<a href="/home.html" title="Homepage" accesskey="H">
  Homepage</a> [<kbd class="accesskey"
  title="To activate press: S, Alt-S, or Ctrl-S">
  H</kbd>]
```

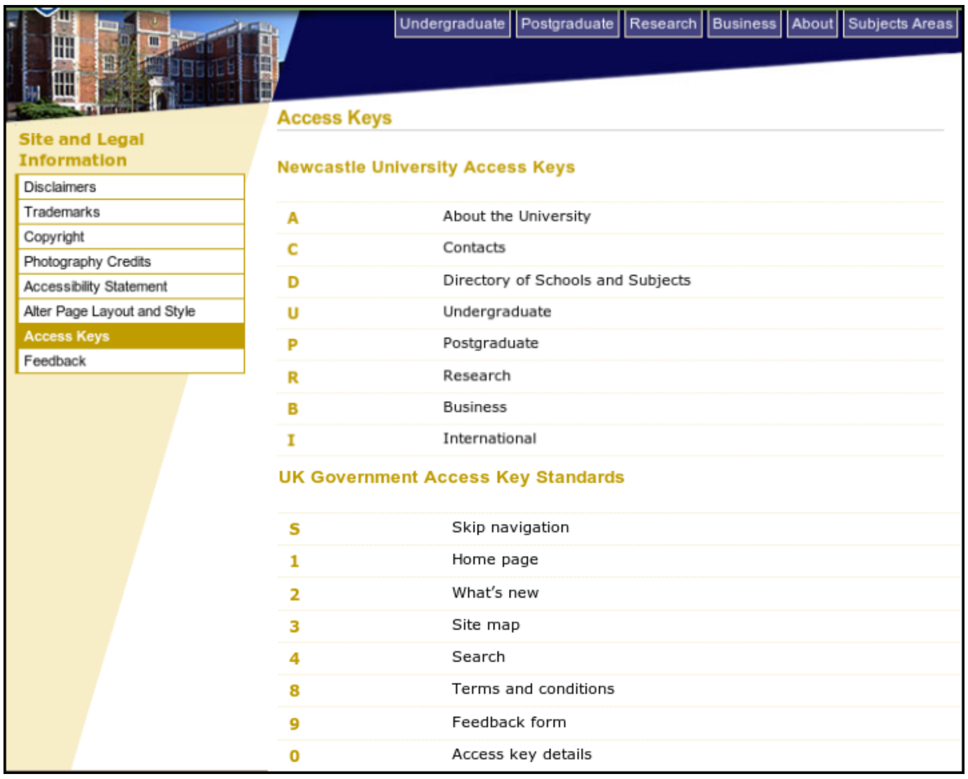
8.3.4 Using JavaScript to display access keys

- HTML5 adds the `accessKeyLabel` method
  - ◇ Returns text describing the *actual* key-binding
  - ◇ E.g., if `accesskey=x` in Firefox:
  - ◇ `element.accessKeyLabel` yields `Alt+Shift+x`
- JavaScript can add explanatory text to links, buttons, etc.
- For example, an access key on a link:

```
<nav>
  <ul>
    <li><a href="/sitemap.html" accesskey=m
           id=sitemap>Sitemap</a></li>
  </ul>
</nav>
<script>
var link = document.getElementById("sitemap");
if (link.accessKeyLabel && link.accessKeyLabel != "")
    link.innerHTML += " (" + link.accessKeyLabel + ")";
</script>
```

- If you use underlined letters you come up against these problems:
  1. The letter you want may not normally occur in your chosen location
  2. The `<u>` element is deprecated
- # And, *most importantly*, links are normally underlined!
- The `:first-letter` pseudo-class is not supported in old IEs.
  - ◇ Moreover it has to be *first* letter
  - ◇ And the technique wraps *every* access link in a paragraph.
- Putting `accesskey` in an `alt` tag is palpable nonsense
  - ◇ `accesskey` is for mobility impaired — not blind people.
- People often use smallcaps to style access key legends distinctively.

8.3.5 List accesskeys on-site



Access Keys

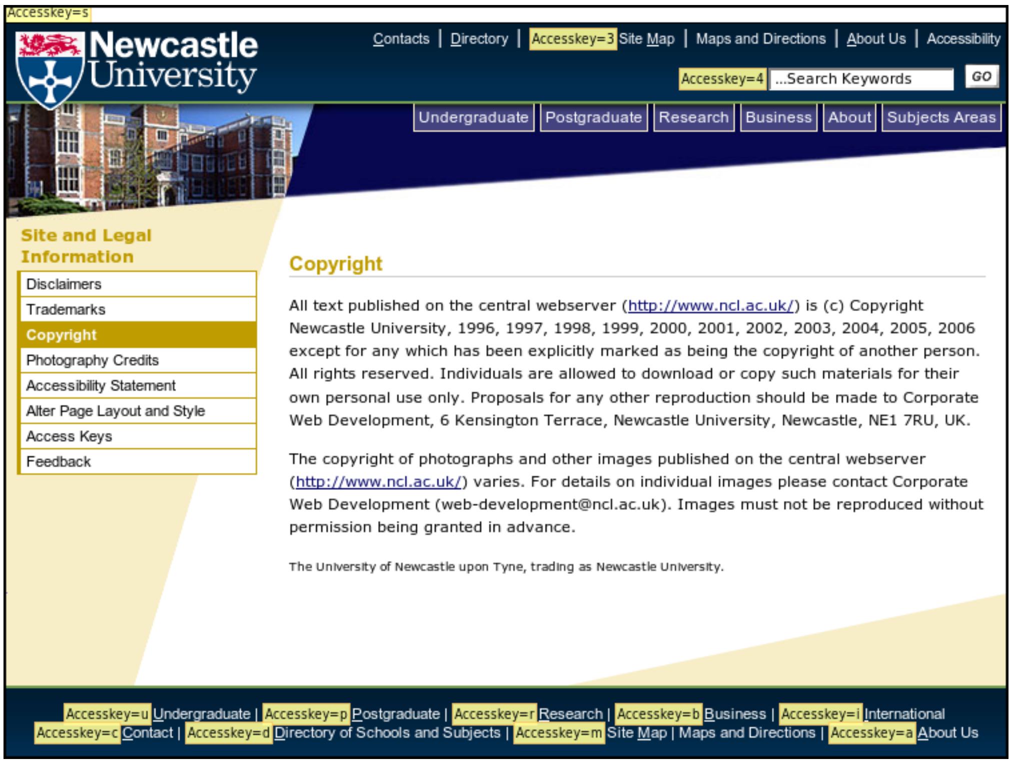
Newcastle University Access Keys

A	About the University
C	Contacts
D	Directory of Schools and Subjects
U	Undergraduate
P	Postgraduate
R	Research
B	Business
I	International

UK Government Access Key Standards

S	Skip navigation
1	Home page
2	What's new
3	Site map
4	Search
8	Terms and conditions
9	Feedback form
0	Access key details

8.3.6 Accesskeys in Web Developer Toolbar


Accesskey=s

Newcastle University

Contacts | Directory | Accesskey=3 Site Map | Maps and Directions | About Us | Accessibility

Accesskey=4 ...Search Keywords GO

Undergraduate | Postgraduate | Research | Business | About | Subjects Areas

Site and Legal Information

- Disclaimers
- Trademarks
- Copyright
- Photography Credits
- Accessibility Statement
- Alter Page Layout and Style
- Access Keys
- Feedback

Copyright

All text published on the central webserver (http://www.ncl.ac.uk/) is (c) Copyright Newcastle University, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006 except for any which has been explicitly marked as being the copyright of another person. All rights reserved. Individuals are allowed to download or copy such materials for their own personal use only. Proposals for any other reproduction should be made to Corporate Web Development, 6 Kensington Terrace, Newcastle University, Newcastle, NE1 7RU, UK.

The copyright of photographs and other images published on the central webserver (http://www.ncl.ac.uk/) varies. For details on individual images please contact Corporate Web Development (web-development@ncl.ac.uk). Images must not be reproduced without permission being granted in advance.

The University of Newcastle upon Tyne, trading as Newcastle University.

Accesskey=u Undergraduate | Accesskey=p Postgraduate | Accesskey=r Research | Accesskey=b Business | Accesskey=i International
Accesskey=c Contact | Accesskey=d Directory of Schools and Subjects | Accesskey=m Site Map | Maps and Directions | Accesskey=a About Us

8.3.7 `accesskey` naming conflicts

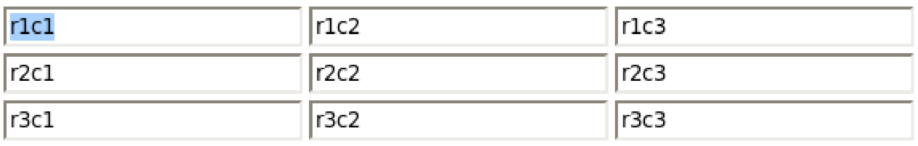
- Fewer access keys mean fewer complexities and conflicts
  - ◇ With other targets, users, browsers, page-specific keys, etc
- Site-wide consistency — accommodating variation
  - ◇ Template pages — same keys for core features (nav, search)
  - ◇ Record universal and variable keys in a conflict resolution table
- Consider consistency with UK government keys
- Naming scheme suggestions:
  - ◇ Letters for navbar, numbers for left menus
  - ◇ 1st letter of *first word* in *menu labels*
    - » Then 1st letters in second word — to resolve conflict
  - ◇ Last letter of *ordinary links*
  - ◇ Avoid zero — looks like letter 'O'
  - ◇ For numbers > 9 consider `!@#$%^&*` — risky but maybe necessary
  - ◇ `!` to put the cursor in a form field
  - ◇ `/` for search — common in many OSes and apps
    - » Browsers that use it for in-page search are usually remappable

8.4 Tab indexes

8.4.1 Tab key navigation and `tabindex`

- Tabbing was the original browsing method
  - ◇ Press TAB to hop from one page component to another (in order)
  - ◇ Press ENTER to invoke link components
  - ◇ Type to enter data in form components — then tab to button/link
- Vital for text, audio and screen reading browsers
  - ◇ Retained in the vast majority of graphical browsers (pace Macs)
- Adding the `tabindex` attribute to page components
  - ◇ Allows you to override strict markup order
  - ◇ Each `tabindex` attribute is given a number (0–32767)
  - ◇ Each TAB moves focus and cursor to the next number
  - ◇ Designed to move you *past* components
- Applicable to the following elements:
  - ◇ `<a>` `<area>` `<button>` `<input>` `<object>` `<select>` and `<textarea>`

8.4.2 Tab through a tabular form vertically



r1c1	r1c2	r1c3
r2c1	r2c2	r2c3
r3c1	r3c2	r3c3

```
<form action="handler.php">
 <table>
  <tr>
   <td><input type=text name=1 value=r1c1 tabindex=110></td>
   <td><input type=text name=2 value=r1c2 tabindex=140></td>
   <td><input type=text name=3 value=r1c3 tabindex=170></td>
  </tr>
  <tr>
   <td><input type=text name=4 value=r2c1 tabindex=120></td>
   <td><input type=text name=5 value=r2c2 tabindex=150></td>
   <td><input type=text name=6 value=r2c3 tabindex=180></td>
  </tr>
  <tr>
   <td><input type=text name=7 value=r3c1 tabindex=130></td>
   <td><input type=text name=8 value=r3c2 tabindex=160></td>
   <td><input type=text name=9 value=r3c3 tabindex=180></td>
  </tr>
 </table>
</form>
```

8.4.3 Problems with `tabindex`

- You are taking choice and control away from users
    - ◇ Forcing them to skip stuff than might interest them
    - ◇ Forcing them to proceed in your preferred order
- May disorientate:
    - ◇ Blind people, used to tabbing ploddingly from one item to another, suddenly find their cursor goes somewhere else
- Allows users to tab from one block to another, but:
    - ◇ May make items within a block completely inaccessible
    - ◇ Especially problematic for menus and form components
- CMSs with dynamic page/URL generation:
    - ◇ Can be difficult to identify target URLs — shouldn't be
- `tabindex` conflict tracking is vastly more difficult
    - ◇ Than `accesskey` conflict tracking

8.4.4 `tabindex` tips and tricks

- Use sparingly — with careful planning *and testing*
    - ◇ Don't disable normal tabbing for little gain
- Don't user it to target specific blocks — especially menus
    - ◇ Use `accesskey` for that
- Use `tabindex` most:
    - ◇ To ensure all form components are completable in correct order
    - ◇ Where markup order is not particularly meaningful or helpful
        - » E.g. image maps
    - ◇ Where there is an obviously sensible reading/skimming sequence
        - » E.g. section headings
- Don't use consecutive numbers for `tabindex` values
    - ◇ Steps of 10 allow you infill later, if necessary

8.4.5 Some final dos and don'ts

- *Don't* bother with **site-maps** — for large and complex sites
    - ◇ They rarely reduce complexity — except for the non-disabled
    - ◇ They rarely reduce the sheer quantity of navigation
- *Do* provide site-maps for simple sites
    - ◇ With **legacy inaccessibility**
    - ◇ Which can't be immediately remedied

- *Don't* bother **describing image maps** themselves
  - ◇ Just the actions or pages they link to
- *Do* make sure that you can **tab through image maps**
- *Don't* use **frames**
  - ◇ They detach the cursor from navigation context
- *Don't* use JavaScript, VBScript, Java, etc **to action links**
  - ◇ Text browsers don't have built-in interpreters

8.5 Exercises

1. Links

These exercises use resources in the folder *resources/nav/*.

1. Make a copy of the resource file called *starters/links.html*.
2. Change the navigation menu to use a bullet list, so that the links aren't run up against each other with nothing but whitespace in between. You'll need to remove the styling for the `nav a` selector to allow the list to be formatted naturally.
3. Adjust the 'more' and 'click here' links so that their cover text is more informative about where they lead.
4. Add `title` attributes to the links for which the cover text still isn't informative enough, so that the appearance of the page remains the same, but more information is available about what the links take you to.
5. Add a skip link to the header to skip over the navigation directly to the article. After testing it, make it hidden in visual browsers.

(Answer on page 113)

2. Tabindex

Make a copy of the resource file called *starters/tabindex.html*.

1. Check that the tab order is as expected — it should match the source order of the elements.
2. Add `tabindex=0` to some of the `<input>` elements and confirm that it leaves the tabbing order as before.
3. Set `tabindex=-1` on one of the `<input>` elements and confirm that it takes that element out of the tab order, making it unreachable without a mouse.
4. Set a different tab index value on each of the `<input>` elements, such that the cursor will start by going through the elements on the bottom row, left to right, then the middle row, and finally the top row.
5. Find out what happens to that tab order if a few of the elements have their `tabindex` attributes removed or set to zero.

(Answer on page 113)

Chapter 9

Tables

9.1 Accessible tables

9.1.1 Tables and accessibility

- Tables are often the most convenient way of presenting data
  - ◇ But some disabled groups will find them difficult to read
- The blind are most obviously affected
  - ◇ Reading software has to trawl serially through every cell (L–R, T–B)
  - ◇ Can get 'lost' in nested table structures
  - ◇ Table metadata (column/row labels) is detached from cell data
  - ◇ Or tediously repeated in every cell context
- Complex table structures and presentation hurt
  - ◇ Even the mildly learning disabled — disproportionately
- On the other hand, tables can be made a lot more accessible
  - ◇ Using the support that HTML provides for audio devices
  - ◇ Reducing structural complexity to the necessary minimum
  - ◇ Reducing presentational complexity
  - ◇ Often by simply omitting overly detailed/distracting content

9.2 Tables for Layout

9.2.1 Layout tables

- Tables have often been used to arrange page components
  - ◇ Not allowed by latest HTML specifications
  - ◇ Tables are for 2-dimensional related bits of information
  - ◇ Use CSS for layout instead

- Traditionally used when stretchy layouts required
  - ◇ Now CSS table display properties can be used instead
- Even so, not *necessarily* an accessibility disaster
  - ◇ Screen-readers have been forced to handle them
- If impractical to refit an old layout table
  - ◇ Tell screen-readers what it is:

    `<table role=presentation>`
- W3C suggests `border=""` or `border="1"` for data tables
  - ◇ To tell screen-reader's it's *not* a layout table
  - ◇ But WHATWG spec says `border` is obsolete and non-conforming
  - ◇ Only use to fix screen-readers misinterpretation if it happens

9.3 Table metadata

9.3.1 Table captions

- Add an explanatory caption with `<caption>` element
  - ◇ Puts table in context
  - ◇ Gives a brief overview
  - ◇ Can be used to identify table, e.g., *Table 1.3*
- Caption should be first thing inside `<table>`, e.g.:

```
<table>
 <caption>Table 1:
    Sales Figures for 2013</caption>
 <tr>
  ...
</table>
```

- `<caption>` can contain 'flow content'
  - ◇ That is, inline content, paragraphs, etc.
  - ◇ So long as the caption doesn't contain a nested table!

9.3.2 Table captions in figures

- Tables can be placed inside `<figure>`
  - ◇ Indicates it's a separate piece of content
  - ◇ Probably referenced from the main text
- Figures can have their own caption in `<figcaption>`
  - ◇ Don't use both captions — would be confusing
- Rule is, use `<figcaption>` if `<figure>` contains:
  - ◇ The `<table>` element

    - ◇ The `<figcaption>` element — before or after table
    - ◇ No other elements or text (except whitespace)
- Otherwise, use `<caption>` as normal
- Either way, the caption is associated with the table

```
<figure>
 <figcaption>Table 2.
    Dates of mass extinctions</figcaption>
 <table> ... </table>
</figure>
```

9.3.3 Table summaries

- Complex or unusual tables may need a summary
    - ◇ Explanation of their structure
    - ◇ If a `<caption>` isn't enough on its own
- Best practice: use normal content before or after table
    - ◇ Table can be connected to summary with ARIA:

    ```
    <p id=summary> ... </p>
    <table aria-describedby=summary>
    ```

    - ◇ Or use `<caption>` containing new `<details>` element:

    ```
    <caption>Simple caption for everyone
     <details>
      <summary>Click for details</summary>
      <p>More details given here, initially hidden...
    ```

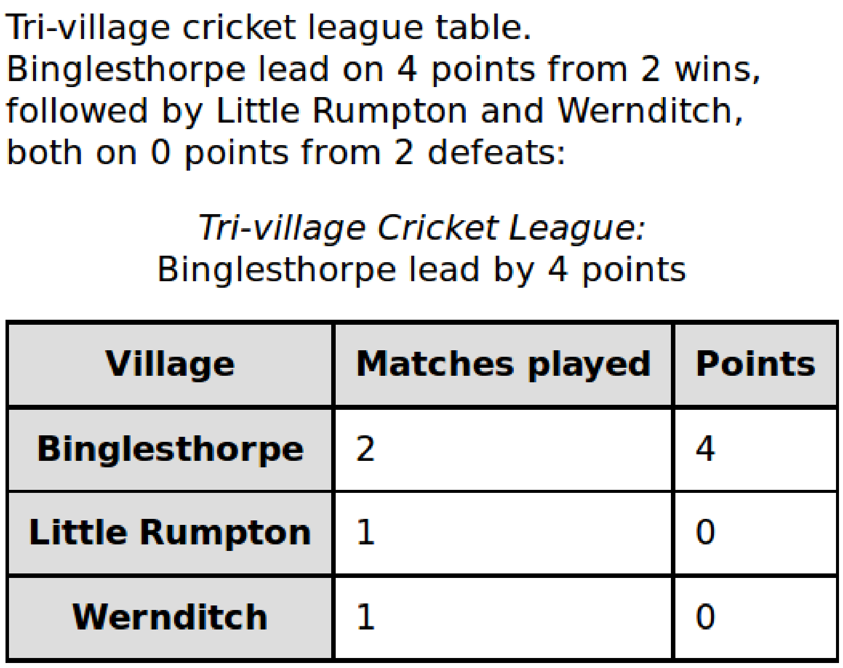
        - » Caveat: so far `<details>` only supported in Chrome
        - » Although can be made to work with JavaScript polyfill
- Don't use the deprecated `summary` attribute
    - ◇ Never used consistently enough to be useful

9.3.4 Example: table caption and summary

```
<p id=summary>Tri-village cricket league table. Binglesthorpe lead
 on 4 points from 2 wins, followed by Little Rumpton and Werndитch,
 both on 0 points from 2 defeats:</p>
<table aria-describedby=summary>
  <caption><em>Tri-village Cricket League:</em><br>
    Binglesthorpe lead by 4 points
  </caption>
  <tr>
    <th>Village</th>        <th>Matches played</th> <th>Points</th>
  </tr>
  <tr>
    <th>Binglesthorpe</th>  <td>2</td>              <td>4</td>
  </tr>
  <tr>
```

```
    <th>Little Rumpton</th> <td>1</td>              <td>0</td>
  </tr>
  <tr>
    <th>Wernditch</th>      <td>1</td>              <td>0</td>
  </tr>
</table>
```

9.3.5 Example: table metadata rendering



Tri-village cricket league table.
Binglesthorpe lead on 4 points from 2 wins,
followed by Little Rumpton and Wernditch,
both on 0 points from 2 defeats:

Tri-village Cricket League:
Binglesthorpe lead by 4 points

Village	Matches played	Points
Binglesthorpe	2	4
Little Rumpton	1	0
Wernditch	1	0

9.3.6 Keep table structure simple

- Multi-level headings are graphic and, partly, print-oriented
  - ◇ Cramming many data and relations into one page/block
  - ◇ Consider splitting the data across several tables
- Similar criticism could be made of column grouping
  - ◇ Especially since columns are a fairly shaky concept in HTML
  - ◇ Ask yourself whether it is strictly necessary
- Never let cell border width or visibility be the only clue
  - ◇ About grouping and spanning
- Spanning individual cells across multiple rows or columns
  - ◇ Should also be done structurally rather than graphically
  - ◇ Using the `colspan=""` and `rowspan=""` attributes on the `td`

9.3.7 Defining groups of columns

- Multiple columns can be collected in a logical 'group'
  - ◇ E.g., prices including and excluding VAT
  - ◇ Each in separate column, with its own header
  - ◇ Header above that for 'Price' with `rowspan=2`
- Define how columns are grouped — `<colgroup>` and `<col>`

  ```
  <colgroup><col>
  <colgroup><col><col>
  ```

  - ◇ First group contains first column
  - ◇ Second group contains second and third columns
- `<colgroup>` must go at start of table
  - ◇ Before everything except `<caption>`, if any
  - ◇ No need to close the elements, happens automatically
- Both elements can be used for styling
  - ◇ E.g., add `id` or `class` attribute and select with CSS

9.3.8 Defining groups of rows

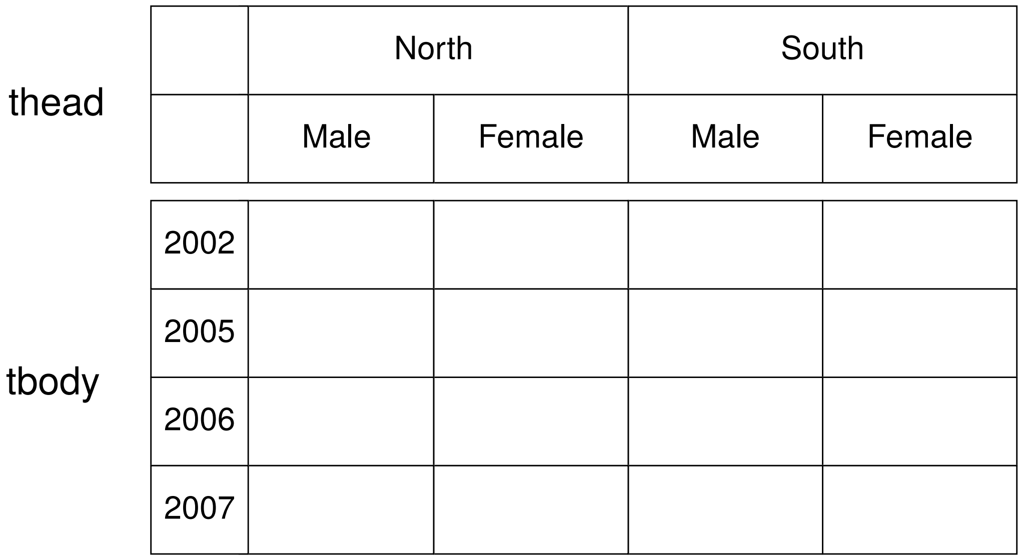
- Rows can be groups just like columns
  - ◇ But there's no `<rowgroup>` or `<row>` element
- Instead, wrap rows in these grouping elements, in this order:
  - ◇ `<thead>` for header rows
  - ◇ `<tbody>` for data rows
  - ◇ `<thfoot>` for footer rows
- Only one `<thead>` and one `<tfoot>` allowed
  - ◇ But can have multiple `<tbody>`s for groups of data
  - ◇ E.g., one group for rows in each month's records

9.3.9 Table headers and footers

- Tables are often more conceptually complex than we notice
- Arguably, the HTML spec makes them even more so
- Always use `<th>` for column and row headings, not `<td>`
  - ◇ Provides a non-graphical signal that these are headings
  - ◇ Allows software to remind blind users about which row/column they are currently reading
  - ◇ Can be used in the middle of a table, if they apply to rows below
- `<thead>` is not strictly necessary

  - ◇ But helps to distinguish headers from the data (`<tbody>`)
  - ◇ Most assistive technology recognises `<thead>`
  - ◇ Gives a better chance of deciphering multi-level headings
- `<tfoot>` is for labelling the bottom of long columns
  - ◇ Gives serial readers extra (pointless) stuff to read

9.3.10 Illustration: `<thead>` and `<tbody>`


thead
tbody

	North		South	
	Male	Female	Male	Female
2002				
2005				
2006				
2007				

9.3.11 Scoping row and column headers

- A simple 2-step procedure lets talking book software read out table data sensibly:
  1. Read *current* row header, followed by the col headers in order
  2. Then read the cells from that row, in the same order
- The following allows screen readers to do the same thing
  - ◇ Use `<th scope="col">` for column headers
    - » They apply to the `<td>` cells below that header
  - ◇ Use `<th scope="row">` for row headers
    - » Apply to the data cells to the right of the header
- For column and row groups
  - ◇ Use `scope="colgroup"` and `scope="rowgroup"`

9.3.12 Multi-level headings

- The `scope` attribute only handles one header level at a time
- HTML has a mechanism for handling multi-level headings:
  - ◇ Allowing heading cells at right or bottom of table
  - ◇ Add `id=""` and a unique name to each header cell
  - ◇ For each data cell — add `headers=""`
  - ◇ List every header `id` that applies to the cell — separate by spaces
- Also works recursively:
  - ◇ Use `headers` on a subheading `<th>`
  - ◇ Data cells which reference that get main `<th>` automatically
- Incredibly tedious and time consuming to code
- No adaptive technology support
- No authoring software support
- But it could make a difference
  - ◇ Use if you can code support into your authoring tools

9.3.13 Tables without headers

- Not every table needs to have headers
  - ◇ E.g., a table displaying a Sudoku puzzle's solution
  - ◇ Each number in a separate `<td>`
  - ◇ Or a mathematical matrix
- For example:

```
<p>This is an <dfn>identity matrix</dfn>:</p>
<table>
 <tr> <td>1 <td>0 <td>0 </tr>
 <tr> <td>0 <td>1 <td>0 </tr>
 <tr> <td>0 <td>0 <td>1 </tr>
</table>
```

- Which might be displayed as:

  This is an *identity matrix*:

1	0	0
0	1	0
0	0	1

9.3.14 Concluding advice on tables

- Tables are never likely to be fully accessible to the blind
  - ◇ In the sense of being able to randomly locate and access data items, rather browse through an entire dataset

- But it is possible to make table browsing a great deal easier
- Mostly down to table authors, rather than web designers
  - ◇ By using simpler, clearer, table structures
- For basic accessibility, designers:
  - ◇ Don't use tables for layout if at all possible
  - ◇ Should use basic header information for data tables (`<th>`)
  - ◇ Should use `<caption>` elements
  - ◇ Summarise complex tables to aid understanding
- For intermediate accessibility:
  - ◇ Use the `scope` attribute for relatively complex data table headers
- For advanced accessibility:
  - ◇ Use `id=""` and the full range of data table `headers=""` attributes

9.4 Exercises

1. Table Accessibility

In the *resources/tables/* folder you will find a sample file called *starters/datatable.html* to complete these exercises. Make a copy of it, and then:

1. Add a suitable caption. The table is supposed to represent sales figures for a group of clients over several months.
2. Instead of repeating the month name, make it spread out over all the client's orders in the same month.
3. Define a row group for the header, and a separate row group for each month's figures.
4. Define column groups so that the 'Sales' and 'Profit' pairs of columns are each collected together in a two-column group, while the first two columns are column groups all by themselves.
5. Add appropriate `scope` attributes to column and row header cells.
6. Move the 'Month' column to the right-hand side of the table. Unfortunately, this means that it's `<th>` cells for each month will no longer automatically apply to the data cells to its left. For one client's data in one month, add `headers` attributes to define which headers apply to each cell, and the necessary `id` attributes on header cells.

(Answer on page 113)

Chapter 10

Forms

10.1 Accessible forms

10.1.1 Generic form accessibility issues

- Most are no different from those faced by any user, e.g.
    - ◇ Excessive form length + complexity
    - ◇ Excessively small focusable areas
    - ◇ Poor labelling — position, readability, comprehensibility
    - ◇ No redundancy for fragile optimisations, e.g. CSS, JS
    - ◇ Obscure, winding, path to completion
- Some much more extreme
    - ◇ AT dependency on **keyboard** navigation and input
    - ◇ Fine-grained **mouse control** — motor disabled
    - ◇ **Labelling**, instructions, help — learning disabled, incl. AD/HD

10.2 Movement and navigation

10.2.1 Assistive Technology & the Keyboard

- Blind and motor disabled depend on keyboard
    - ◇ For form navigation and activation
- Screen reader problems
    - ◇ Can get 'lost' in conventional forms
    - ◇ Incorrectly identifying the next form stage/component
    - ◇ Placing the cursor in the wrong place
    - ◇ Failing to associate a form control with its instructions
    - ◇ Connecting controls with components (e.g. button with text box)

- Main causes:
  - ◇ Table-based forms — progress order doesn't match cell order
  - ◇ Buggy screen readers — don't use or understand standard markup
- Solutions:
  - ◇ Group and label controls properly
  - ◇ Serialise their order
  - ◇ Code to standards — not for screen reader bugs

10.2.2 Moving to and within forms

- *Always* highlight focussed area — with high-contrast
- If form filling is the primary purpose of the page, consider:
  - ◇ Skipping to first form control/label
  - ◇ Via `autofocus`, `tabindex`, `accesskey`
  - ◇ Give first control `autofocus` — and/or low `tabindex` (1?)
  - ◇ Remember to skip over earlier forms, like search boxes
- Ensure that every *landmark* form *section* has a tabindex
  - ◇ But do *not* put `tabindex` on every individual radio/checkbox
  - ◇ Only 1st in each series — cursor keys navigate/select between
  - ◇ See next slide
- Consider adding `accesskey`s, if you can make them visible
- Browser default styling for focus and placeholder text is typically low-contrast

10.2.3 Tabindex only first in a series of radio buttons

```
<form>
  <label for=fullName>Full name
    <input type=text tabindex=1 name=fullName>
  </label>
  <fieldset>
    <legend>Country</legend>
    <label for=england>
      <input type=radio tabindex=2 name=country id=england value=England>
      England
    </label>
    <label for=scotland>
      <input type=radio name=country id=scotland value=Scotland>
      Scotland
    </label>
    <label for=wales>
      <input type=radio name=country id=wales value=Wales>
      Wales
    </label>
  </fieldset>
```

```
</form>
```

10.2.4 Selecting form landmarks

- Simple forms — e.g. search boxes:
  - May not need an `accesskey` or `tabindex`
  - Good coders put them first and allow `RETURN` to trigger them
- Long forms for full completion:
  - `accesskey` and `tabindex` on the first action
  - And on any control affecting the whole form (e.g. `submit`)
- Long forms which are partially skipped:
  - `accesskey` on the essential *groupings* that you skip *to*
  - And on any button performing a link function — e.g. 'Go Back'
  - **Not** `tabindex` — enforced skipping may be harmful

10.2.5 Time-outs and progress

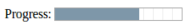
- Input time limits should be avoided
  - Unless absolutely necessary
- WAI-ARIA timer role counts down/up to/from a set time
- Timers don't announce changes to the user — by default
  - Can be overridden by the developer
- Consider allowing users to save progress, or auto-save state
  - As they move through longer forms
- Chunk long forms — clearly indicating progress
- HTML5's `progress` element provides a simple way to do this
  - Accessible JQuery fallback with WAI-ARIA-enabled widgets
  - http://access.aol.com/aegis/#goto_progressbar

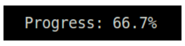
10.2.6 The `progress` element

- Not strictly a form control — more generic
  - So cannot be associated with a `<label>`
- Example:

```
<form name="f" action="..." method=get>
  <p>Progress:
    <progress value="66.7" max=100>66.7%</progress>
  </p>
</form>
```

- No standard browser rendering — not necessarily a 'bar'

    - ◇ Falls back to plain text — in non-supporting browsers
    - ◇ jQuery UI progress bar plugin an alternative option
- Firefox: 

- Lynx text browser 
Progress: 66.7%

10.2.7 Subordinate reset buttons

- Reset buttons are usually a bad idea
- Make them impossible to activate accidentally by keypress
- *Do not* give the Reset button an `accesskey`
    - ◇ Under *any* circumstances
    - ◇ Too easy to press the access key accidentally
    - ◇ Especially because browsers/readers hide `accesskey`s
- Don't give a Reset button a `tabindex` either
    - ◇ You don't want to make it easy to select the button
- If you must assign `tabindex` to a Reset button:
    - ◇ Give it the very last numeric value on the page (32767)

10.3 Select boxes and datalists

10.3.1 Completing Forms by Selection

- Widely used to prevent errors, e.g. invalid data entry
- Can cause the disabled enormous grief, if:
    - ◇ Individual items are inadequately separated
    - ◇ Overly long
        - » Tedious and time-consuming — e.g. country selections
        - » Its easy to loose track of position — e.g. phone interruptions
        - » Simply confuse — especially the learning disabled
        - » Hard work — for the motor disabled
- But better than long series of individual radios/checks
- **Enhancements** — detailed below
    - ◇ Use **datalists** instead — like combo boxes
    - ◇ Group options — `optgroup`
    - ◇ Group and label form sections/controls — `fieldset` + `legend`
- To test select box usability, try mousing with your non-dominant hand

10.3.2 'Combo' boxes and free-text entry

- Select from pre-set options — or enter free text
- Often easier for disabled groups to handle — than pure selection
  - ◇ Particularly motor disabled
  - ◇ Better than forcing users to do repeated manual correction
  - ◇ Can eliminate long selections and complex (menu-driven) forms
- Work best when auto-completion is plausible e.g.
  - ◇ When possible completions or spellings are *relatively* limited
  - ◇ When proffered completions are chunked
  - ◇ First 2–3 chars of optional completions are distinctive
    - » N.B. product lists with 100s of virtually identical names
- JS dependency may be problematic
  - ◇ But needn't be an absolute obstacle
  - ◇ You often have data integrity problems with free text entry
  - ◇ Failed completion or error checking shouldn't make it any worse
- Another useful form of redundancy

10.3.3 `<datalist>`

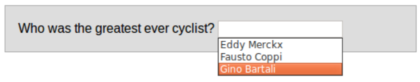
- Allows you to associate a list of options with a text field
- Similar to autocompletion
  - ◇ It applies to text fields — provides 'suggested' options
- But *different* — user can make a custom entry
  - ◇ Unlike exclusive autocompletion — or select boxes
- When input element is focused — a list of options drops down
  - ◇ If its `list` attribute links to the datalist

```
<form>
  <label>Who was the greatest ever cyclist?
    <input list="cyclists" name="greatest">
    <datalist id="cyclists">
      <option value="Eddy Merckx">
      <option value="Fausto Coppi">
      <option value="Gino Bartali">
    </datalist>
  </label>
</form>
```

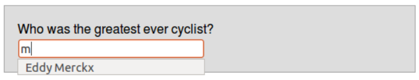
- Fewer accessibility issues than `<select>` drop-downs
  - ◇ But `<select>` may still be needed
  - ◇ For non-supporting browsers

10.3.4 Browser support for list and datalist

- Partial in most current browsers (early 2016) — not Safari
  - ◇ See: caniuse.com
- See jQueryUI autocomplete plugin — for a similar fallback
- Default fallback to standard text field — without that
- Opera:



- Firefox:



10.3.5 Grouping Options With `optgroup`

- `<optgroup>` lets you group `<option>` elements
  - ◇ *Within* a `<select>` box
  - ◇ Just wrap the tags around sub-groups of the option list
  - ◇ Always add `label=""` attribute to `<optgroup>`
  - ◇ A few browsers provide cleverly controlled display
  - ◇ Most modern browsers group display groups distinctly
    - » And helpfully
  - ◇ Not all screen-readers use it
    - » But it won't do any harm
  - ◇ N.B. You cannot nest `<optgroup>` tags
- `<select>` can contain both grouped *and* ungrouped options
  - ◇ Probably not a good idea — could be confusing

10.3.6 Chunking with `fieldset` & `legend`

- `<fieldset>` lets you chunk form fields into sets, e.g.
  - ◇ Names — first name, middle initial, surname
  - ◇ Addresses — street address, city, county, post code, country
  - ◇ Telephone numbers — work, home, mobile, modality
  - ◇ Online addresses — website, email, IM
- Almost mandatory for radio/checkbox sets
  - ◇ Proper labelling for both the group *and* individual controls
- Use CSS to remove the ugly bounding box, e.g.

`fieldset {border: none;}`

- Use `<legend>` to add a caption to each `fieldset`

10.4 Labels and instructions

10.4.1 Associate `labels` with controls

- `<label>` describes controls that don't have implicit labels
  - ◇ Should be as short, clear, and simple as possible
    - » Especially important for learning disabled + AD/HD
  - ◇ Instructions not in `<label>` often unread by screen-readers
    - » And might be skipped over with *Tab*
- 2 ways of associating the form control with label text:
  1. **Link** them from a label like `<label for="idvalue">Label Text</label>` by adding an `id` attribute to the control
     - ◇ Most AT will link the two — fairly reliably now
     - ◇ Logically associated, *regardless* of graphic or source code order
  2. **Enclose** the control inside `<label></label>`
     - ◇ Put *Label text* before the control's opening tag
     - ◇ RNIB: label text after the control on radio buttons and checkboxes
     - ◇ Logical, graphical, *and* source code order association

     The RNIB suggest that label text should appear *after* the control on checkboxes and radio buttons. The likely reason for this is that users have become used to it, since it has been a convention in graphical user interfaces for decades. Even if an alternative were measurably 'better', the cost of making users work out how to use an unfamiliar interface is too high.

     Note that although a user reading an English-language page will expect a label to appear to the left of an input box, and to the right of a checkbox, the exact opposite will be true when the page is predominantly written in a language whose writing goes right-to-left across the page. Pages written in languages such as Arabic and Hebrew will in general have their layout reversed horizontally, including the positioning of labels for form controls.

10.4.2 Using label wrapping

- Label wrapping is recommended in the HTML5 spec
  - ◇ Lets you mouse-focus a large area around the input control
    - » A major advantage for the motor disabled
  - ◇ `<label>` can contain any *inline* element, including images
  - ◇ `accesskey` on the `<label>` passes focus to the control
- Use both methods together for maximum reliability

- And extra metadata for maximum redundancy:

  ```
  <label for="emailaddr" accesskey="e" title="Type
    your email address in this box">Email address:
     <input type="text" name="emailaddr"
       id="emailaddr" value="j.bloggs@elsewhere.com"
       size="20" maxlength="20">
  </label>
  ```

10.4.3 Using `title` attributes

- Using `title` is *optional*
  - ◇ Expression should match visible labelling — but can expand on it
  - ◇ Redundant textual explanation is a core accessibility technique
  - ◇ But the repetition will irritate blind users — understandably
- Particularly useful on `<input>` and `<textarea>`, e.g.
  - ◇ 'Type your surname (family name), then any other names'
  - ◇ 'Explain the problem you are having (in under 100 words)'
  - ◇ 'If you don't know your post code, leave the field blank'
- And on individual radio buttons, e.g. to ensure correct association between answer text and the right button:

  ```
  <input type="radio" name="seatgroup" value="window"
     title="Select a seat next to the window">
  <input type="radio" name="seatgroup" value="aisle"
     title="Select a seat next to the aisle" checked>
  <input type="radio" name="seatgroup" value="centre"
     title="Select the seat in the middle of a row">
  ```

10.4.4 Graphical buttons

- Some people want their submit buttons to look more 'fancy'
- Ideal way — use normal `<input type=submit>` and CSS
  - ◇ No accessibility issues — unless styling itself obscures meaning
  - ◇ Try http://cssbuttongenerator.com/ for some examples
- Old fashioned way — use an image button
  - ◇ Works like `<img>` — must have a `src` attribute
  - ◇ And a *non-empty* `alt` attribute e.g.

    ```
    <input type=image src="submit.png" alt="Submit Form">
    ```
  - ◇ `alt` value should match text in the image
  - ◇ Accessibility tools should treat it same as `<input type=submit>`
  - ◇ Again, don't let styling obscure its meaning
    - » Don't make it *too* fancy — should still look like a button

  - ◇ Side-effect — they submit coordinates of where user clicked them

    Note that the `<input type=image>` control was originally not designed to act as a fancy submit button. It was actually designed to allow users to select a position on an image, and submit the coordinates of the position they clicked. This could be used, for example, for the user to select a position on a map of the country.

    This use of image buttons is quite rare now — they're mostly just used for fancy submit buttons — but it does mean that when the form is submitted you'll get extra form information with those coordinates. Even if you activate the button with the keyboard, the browser will submit coordinates of $(0,0)$. This shouldn't be a problem so long as they don't get in the way of the rest of your form.

    Just so that you know: if you don't put a `name` attribute on your image button, then it will submit fields named~`x` and~`y`. If you have `name=foo` then you'll get fields `foo.x` and `foo.y` instead.

10.4.5 Placeholder text

- Prompts, or hints, in text boxes — deleted on focus
  - ◇ Built-in to HTML5 browsers
  - ◇ Previously JavaScript dependent — although unproblematic
- Example:

```
<form>
  <input name="q" type=search
    placeholder="Search this section">
  <input type="submit" value="Search">
</form>
```

- Replace low-contrast default colour — via CSS3 selectors

```
-webkit-input-placeholder { color: #333 }
-moz-input-placeholder { color: #333 }
```

- Pre-HTML5 browsers just ignore the placeholder attribute and its value
- Using JavaScript to place and auto-delete placeholder text inside controls for older browsers is not a problem for accessibility:
  - ◇ The auto-deletion is non-essential — can be done manually, if labouriously, in text browsers
  - ◇ Modern screen readers understand JS

10.4.6 Re-Thinking Form Layout

- HTML is intrinsically linear
  - ◇ Stacking page components serially
  - ◇ Ideal for AT access to most forms — where progress is serial
- CSS overcomes design limitations
  - ◇ E.g. toggle horizontal/vertical stacking, multicolumn flow, etc.
  - ◇ Vary display order with positioning, float order, flex flow order

- But KISS — aesthetics should *never* compromise functionality
- Layout best practice:
  - ◇ Single-column
  - ◇ Labels positioned above most controls — radio/checkbox excepted
  - ◇ Labels text left aligned
  - ◇ Straight path to completion — vertical scan line at left
  - ◇ Submit button on the PtC scan line
  - ◇ Error messages — inline, adjacent, but offset right
- If must you must stack form controls horizontally or use multiple columns:
  - ◇ Ensure access and sequence order with `tabindex`

10.4.7 New HTML5 form controls

- Most new HTML5 input types allow free text input
  - ◇ Enabling keyboard and mouse input via AT and older browsers
  - ◇ Even when modern browsers provide special UI widgets
  - ◇ E.g. date pickers, colour pickers, spinboxes, etc
- And almost all degrade to `type=text`
- So most accessibility issues concern:
  - ◇ Buggy older user agents that don't degrade properly
  - ◇ Built-in processing, e.g. validation, required fields, autofocus, etc.

10.4.8 Numbers and Spinboxes

- Currently easier than ranges
  - ◇ For motor disabled and keyboard
  - ◇ Because they allow **free-text entry**
  - ◇ Control spinners with cursor keys — if focused
- The `type` attribute `number` requires numerical input

```
<form>
  <input name="n" type="number" min="0" max="10"
         step="0.5" value="6">
  <input type="submit" value="Go">
</form>
```

- Fully supporting browsers will validate
  - ◇ Opera 10
- Has a basic fallback to `<input type="text">`, but:
  - ◇ Associated attributes like `min`, `max` and `step` will be ignored
  - ◇ Firefox 22

10.4.9 Ranges

- The `range` type attribute works much like `number`
  - But fully supporting browsers will render with **sliders**

    ```
    <form>
      <input name="r" type="range" min="1" max="11" value="9">
      <input type="submit" value="Go">
    </form>
    ```

- **Keyboard accessible** in HTML5 browsers — if focusable
  - **Un-usable** in some pre-HTML5 text browsers
- Example renderings:
  - Opera 10
  - Firefox 22
  - Firefox 25
- Fall back to accessible and unobtrusive JS range-type controls
  - http://www.frequency-decoder.com/demo/fd-slider
  - Often don't use the native range, but combine. . .
  - CSS style, WAI-ARIA semantics, and JS to capture interactions

10.4.10 The `autofocus` attribute

- All web form controls can have an `autofocus` attribute
- Moves the input focus to the given input — on page load
  - Browsers should allow users to disable it — important
- Example:

  ```
  <form>
    <input type=search name=q autofocus>
    <input type=submit value=Search>
  </form>
  ```

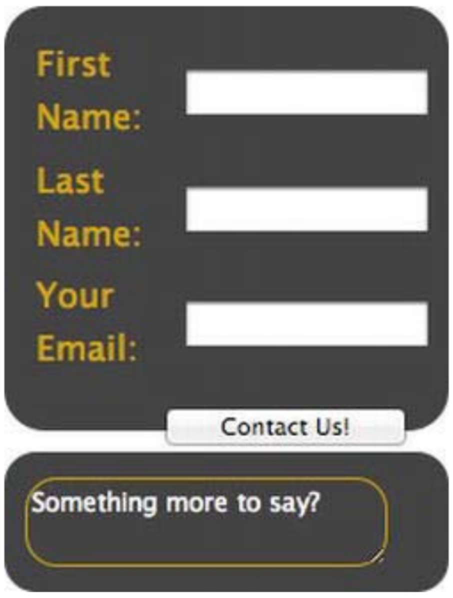
- Not acceptable to just position the focus on a field
- **If used:**
  - Inline error messages should included in the label
  - Or make `aria-described` point to messages elsewhere

10.4.11 The `form` attribute

- Available on `input`, `output`, `select`, `textarea`, `button`, `label`, `object`, and `fieldset`
- Lets you associate them with a form
- So these elements can be placed anywhere on a page
  - Not just as descendants of the form element

```
<div id=container1>
  <form id=contact action=post>
    <label for=forename> First Name: </label>
    <input type=text id=forename name=fname>
    <label for=surname> Last Name:</label>
    <input type=text id=surname name=sname>
    <label for=mail>Your Email:</label>
    <input type=email id=mail name=mail>
    <input type=submit id=submit value="Contact Us!">
  </form>
</div>
<div id="container2">
  <textarea id=extra title="Enter extra Info here and
                          tab back to submit form"
  form=contact placeholder="Something more to say?">
  </textarea>
</div>
```

10.4.12 `form` attribute screenshot



10.5 Required fields and validation

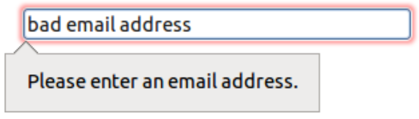
10.5.1 The `required` attribute

- Well supported in modern screen readers
- Completely ignored by older screen readers
- Q+D workaround:
    - `aria-required="true"` alongside HTML5 `required`
- Drawbacks:

  - ◇ Repetition
  - ◇ Both unknown to older AT
- More robust backward compatibility:
  - ◇ Include the word "Required" within the label
  - ◇ Repetitive — no way to detect what kind of AT a person is using
  - ◇ Better than an unexpected error message

10.5.2 HTML5 validation

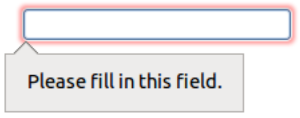
- Supporting browsers test for
  - ◇ Correct data format for input type



bad email address

Please enter an email address.

  - ◇ Presence/absence of required data



Please fill in this field.

- Modern browsers give good inline error messages — not IE9
  - ◇ May be picked up by modern screen readers — FF is
  - ◇ May pass focus to the first problem field — FF does
- Load webforms2.js for validation in older browsers

```
<script type="text/javascript" src="webforms2.js">
```

- Modern screen readers should pick this up
  - ◇ But there should always be a server-side fallback

10.5.3 Dynamic or conditional forms

- Dynamic forms can really help
  - ◇ Eliminating unnecessary components, stages, data entry, etc
  - ◇ Update the display and auto-enter data in response to questions
- Can be seriously counter-productive — if implemented badly
- Client-side scripting is less robust
  - ◇ And more likely to create transaction problems
  - ◇ For users and site owners
  - ◇ May disable form completion and advance — entirely
  - ◇ Failure can be catastrophic, e.g.
    - » Trapping screen readers — in a state that they can't escape

    - ◇ Speed advantages matter less to many disabled groups
    - ◇ But greater interface usability may help
- Two options:
    - ◇ Stick entirely to server-side scripting
    - ◇ Use standard JS and *always* provide:
        - » Redundant HTML-based advance
        - » WAI-ARIA metadata for structures, controls and events

10.5.4 Interacting with the Real World

- Web forms often initiate offline activity, e.g.
    - ◇ Sending out print documents, telephone enquiries, etc
- Unthinking form design can deny this activity to some groups
- Remember, deaf people use telephones via TTYs
    - ◇ Make sure forms prompt for phone *mode* beside phone entry fields
    - ◇ Ideally, enable calls via the UK relay service (Typetalk)
        - » 0800 500 888 (TTY)
        - » 0800 7311 888 (Voice)
- Blind and other disabled groups read aurally — or in Braille
    - ◇ Ensure forms provide the option of requesting alternate formats
- You don't have to go into alternate format publishing yourself
    - ◇ But you break the law if your forms prevent people using options that are available, e.g. transcription/translation services

10.6 Exercises

1. Grouping items

The resources for these exercises are in the folder *resources/forms/*.

Start with a copy of the resource file *starters/countries-list.html*. Make it easier to find a country by grouping them into logical categories, such as 'North America', 'South America', and 'Europe'.

(Answer on page 113)

2. Arranging form fields

Start with a copy of the resource file *starters/login-form.html*.

Users generally expect form fields to be laid out vertically, as *login-form.html* would be in a 320px window. But resizing the window to a typical tablet/desktop width causes the three inputs to be laid out horizontally.

Fix this, by using HTML only (no CSS) to stack the form fields vertically whatever the browser width. You'll need block level wrappers, which stack vertically by default in all browsers.

(Answer on page 114)

2.1. Form field labels

Currently, there's no easy way to tell what the text fields are supposed to be for. Add text to label them, showing that the first text box is for a username, and the second is for a password. Use the `<label>` element to associate the textual labels with the corresponding controls. Try the two different ways of associating labels with form controls, one on each.

(Answer on page 114)

2.2. Submit button value

This is a form for logging in to a web service, so use the `value` attribute to give a more meaningful name to the submit button.

(Answer on page 114)

2.3. Labels above controls

By default, labels and inputs are laid out inline, i.e. horizontally, so the labels sit alongside the inputs. Usability testing demonstrates that most classes of user, but especially learning disabled users, complete forms more quickly when labels are top-aligned, i.e. above the control to which they refer. Use CSS to style labels and controls so that they all begin on new lines.

(Answer on page 114)

Chapter 11

Multimedia

11.1 Video, audio, and animation

11.1.1 Making multimedia less inaccessible

- Video *on the web* doesn't work perfectly — for anyone
  - ◇ No standard format (file or stream) & insufficient bandwidth
  - ◇ Limited AT support for HTML multimedia tags (see below)
- TV is vastly more accessible — high technical standardisation
  - ◇ Web video still mostly handled by Flash players
- Audio better — lower bandwidth & a standard format (MP3)
  - ◇ But streaming audio has similar standards problems
- Multimedia web accessibility is, therefore, mainly about:
  - ◇ Maximising redundancy for multimedia site 'content'
  - ◇ Captioning — for the deaf and hard of hearing
  - ◇ Audio description — for the blind and visually impaired
  - ◇ Subtitling and dubbing — for others
- Subtitling and dubbing are hardly covered in this module
  - ◇ Not strictly web issues — tied to production hardware and software
  - ◇ Web designers/developers can't do much about them

11.1.2 HTML5 — provide multiple video/audio sources

- And provide **non-video fallbacks**
- Provide `<source>` elements instead of `src` attribute

```
<video controls>
  <source src="video.webm" type="video/webm">
  <source src="video.mp4" type="video/mp4">
  Fallback content goes here.
```

```
</video>
```

- Browser uses first source it knows how to play
- Same for `<audio>` element:

```
<audio controls>
  <source src="song.mp3" type="video/mpeg">
  <source src="song.ogg" type="video/ogg">
  Fallback content goes here.
</audio>
```

11.1.3 Browser problems with HTML4/XHTML

- HTML4/XHTML media should be nested in `<object>` tags
  - ◇ `<object>` is a generic tag for including any non-HTML object inside a web page (from computer programs to sound files)
  - ◇ Nesting allows infinite redundancy (graceful degradation)
  - ◇ If the user agent can't support one format, pass it a series of alternatives
    - » Ending in plain text if all else fails
  - ◇ See an example on the next slide
- This method suffers from:
  - ◇ Patchy support for the `object` tag in older IEs
  - ◇ Continued use of non-standard or deprecated tags (`embed`, `applet`)

11.1.4 Example: `<object>`

```
<object data="film.mpg" type="video/mpeg"
  width="320" height="240" title="Conference Video">
  <param name="autoplay" value="true">
  <param name="controller" value="true">
  <object data="film.wmv" type="video/x-ms-wmv"
  width="320" height="240" title="Conference Video">
    <param name="autoplay" value="true">
    <param name="controller" value="true">
    <object data="image.jpg" type="image/jpeg"
      width="320" height="240" title="Conference Foto">
      <p>Sorry, you don't seem to be able to see
      pictures, let alone Windows videos.  Here is a
      transcript of the press conference: ...</p>
    </object>
  </object>
</object>
```

- Note the use of redundant `title=""` attributes

11.1.5 Alternative datastreams and captions

- You need alternative datastreams within video formats to support alternative audio tracks, audio description and/or captioning

- Old media have loads of them:
  - ◇ UK TV: several streams + 100s of channels for full-screen text
  - ◇ DVD: up to 32 subtitle tracks and 8 audio tracks
- The picture among popular online video formats is patchier:
  - ◇ QuickTime — no predefined limit to no. of text and audio tracks
  - ◇ Windows Media — Multiple captions, but no audio descriptions
  - ◇ Flash — nothing whatsoever built into the data structure
    - » But you can add text in a way that functions as captions
- HTML5's `track` element + the Timed Text Track API
  - ◇ Enables you to overcome most of these limitations
  - ◇ The API provides the tools to add accessibility, translated transcripts, or extended content to your videos
  - ◇ Beyond the scope of this course

11.1.6 SMIL

- Synchronised Multimedia Integration Language is W3C markup language to create accessible media files
  - ◇ Includes captioning and audio description
  - ◇ Platform-neutral, industry-standard
  - ◇ Lets you cue text, audio, and video together in any combination
  - ◇ Describes what should appear when, in any "time-based medium"
- Player support is pretty good
  - ◇ QuickTime 4.1 and later
  - ◇ RealPlayer 8 and later
  - ◇ Windows Media Player
    - » Actually supports a Microsoft-only subset of SMIL 2.0 (HTML+Time)
    - » Standard SMIL not guaranteed to work, but usually does
  - ◇ Open source: Totem (complete), Mplayer (incomplete)
- Lots of authoring tools:
  - ◇ http://www.w3.org/AudioVideo/#Authoring
- If you want accessible web video, use it!

"Time-based media" include: cinema, TV, radio, music, slideshows (e.g. PowerPoint), GIF animations, etc. Microsoft also have Synchronised Access Media Interchange (SAMI) —a SMIL-like markup language for time-based media. It's not particularly difficult to learn, but don't bother, because:

- It's non-standard
- There are no authoring tools
- It only works in Windows Media Player

- It has been rendered obsolete by SMIL and HTML+Time

11.1.7 Media player interface problems

- Visual player controls affect disabled groups differently
  - ◇ No problems for the deaf
  - ◇ Difficulties for motor and learning disabled
  - ◇ Practically insurmountable problems for the blind
- Actually, the problem for blind people is the screen reader
  - ◇ Designed to represent GUIs
  - ◇ Sits on top of the OS and a stack of applications (inc. browsers)
  - ◇ How does the reader move its cursor to a player's controls?
  - ◇ How does the reader access the browser-embedded controls?
  - ◇ Given access, how does it control the media player non-graphically?
- Designers have little influence over this — so don't bother
- Pressurise the software industry — or join a FLOSS project
  - ◇ E.g. port a text-driven Linux player to your platform

11.1.8 Caption, transcription and description

- Video is harder to understand without sound — than pictures
- So deaf people suffer disproportionately
  - ◇ Really need captioning for full accessibility
- *How to* caption, transcribe or compose audio descriptions
  - ◇ Is *well* beyond the scope of this course:
    - » All three are *massive* topics in their own right
    - » Merely putting a toe in the captioning water depends on access to large amounts of video material and equipment
    - » The techniques, conventions and skills involved take years to acquire
    - » The skills are not remotely related to designer/developer's job
    - » Poor quality accessibility — often worse than none at all
  - ◇ In short, leave them to specialist professionals

11.1.9 DIY captioning and fansubbing

- If you still fancy 'having a go' — a few things to consider:
  - ◇ Watch TV and DVD with the subtitling switched on and sound off
  - ◇ Work out which info the professionals include and exclude
  - ◇ Work out how non-speech info is represented — in captions

- ◇ Learn the correct markup languages and conventions
- ◇ Learn how to typeset appropriately
  - » Video formats impose enormous technical constraints

- *Do not* go into web serving captioned video
  - ◇ Until the content has been thoroughly tested
- Fortunately, testing is cheap, if time-consuming:
  - ◇ Turn the sound off, ensuring subjects don't already know the movie

11.1.10 Open vs. closed captioning

- If you can author captioned video and serve it from your site:
  - ◇ Try to ensure that you offer open captioning
- Open captions are switched on by default
- Closed captions are the kind you normally see on TV
  - ◇ Switched off by default, requiring user action to turn them on
  - ◇ The exception — subtitled foreign films (usually open)
- Closed, because everyone receives the same feed
  - ◇ And *non*-disabled people find the subtitles irritating
- Web services don't have this limitation
  - ◇ The extra cost of providing parallel downloads is negligible
  - ◇ Both feeds contain the same data — the only difference is config

11.1.11 Scripting multimedia

- JavaScript can enhance accessibility in some areas
  - ◇ But implementation details are beyond the scope of this course
- Priority: avoid JavaScript usage that hinders accessibility
  - ◇ Navigation — using keyboard or assistive technology
  - ◇ Hidden content — ATs can't trigger reveal events
  - ◇ User control — often lacking when content changes automatically
  - ◇ Confusion/Disorientation
    - » Altering/disabling normal user agent functionality
    - » Triggering events that the user may not be aware of
- Ensure pages are fully functional with JavaScript switched off
  - ◇ Easy to test — just view pages without it in many browsers

11.1.12 Keyboard alternatives to mousing

- Ideally, JS image links should use progressive enhancement:
  - ◇ All *basic* functionality works in HTML alone
  - ◇ As much enhancement as possible — via CSS-only methods
  - ◇ JS code tops up with the JS-only markup — when it's enabled
- Better than text links in `<noscript>` element
  - ◇ Generally disapproved of — relegates disabled to 2nd class
- Simple links and buttons can use `onclick`
  - ◇ Keyboard alternatives trigger a fake click

    `onclick="openWin(foo.htm)"`
- When using JS event handlers, return false
  - ◇ So `href` link is not followed by JavaScript-enabled browsers
- Custom controls may need to respond to keyboard
  - ◇ Use JS to make them react like standard controls
- Ensure selections using `OnChange` work with the keyboard
  - ◇ By including a submit button: http://jimthatcher.com/webcoursea.htm#Webcourse10.1.5

Chapter 12

Ebooks

12.1 Ebook formats

12.1.1 Using HTML for ebooks

- Ordinary web pages can be 'ebooks'
  - ◇ Can use all HTML's accessibility features
  - ◇ And provide flexibility of how to read
  - ◇ Avoid styling that would be bad for narrow screens
- Most ebook formats are just packaged HTML
  - ◇ But they place restrictions on markup use
  - ◇ To avoid full complexity of markup found on the web
- Makes good 'source' format for ebooks
  - ◇ And for publishing on web
  - ◇ But for reader devices, convert to an ebook format
  - ◇ Suggested: MOBI for Kindles, EPUB for everything else

12.1.2 EPUB format

- EPUB, sometimes called ePub, file extension *.epub*
  - ◇ Supported by almost all e-readers
  - ◇ And phone apps, tablet apps, and PC applications
- *Not* supported on Amazon's Kindles
  - ◇ Although converters are available
  - ◇ Converts well to MOBI format for Kindle
- Actually just a Zip archive file
  - ◇ Eg. you can change extension to *.zip* and open archive
  - ◇ Contains XHTML files for content

  - ◇ Plus CSS styling and images
  - ◇ And some metadata files
- Supports all HTML accessibility features

12.1.3 Other ebook formats

- MOBI — based on Palm OS archive files
  - ◇ Package containing HTML
  - ◇ Originally for Mobipocket reader software
  - ◇ Now supported much more widely
- AZW — Amazon's original format
  - ◇ Based on MOBI — compatible, when DRM isn't used
- Kindle now using new version AZW3 aka KF8
  - ◇ Supports better typography
  - ◇ Including proper hyphenation

12.2 Digital Rights Management

12.2.1 DRM and accessibility

- DRM — Digital Rights Management
  - ◇ Usage restrictions chosen by publisher
  - ◇ Enforced by hardware and/or software, with encryption
- Mostly to prevent copying
  - ◇ Eg. can't give away copies of your DRM ebooks
  - ◇ But also can't copy to more accessible device
  - ◇ So can't upload Kindle ebook to PC or tablet
    - » To use its screen reader software
- Amazon allow publishers to prevent text-to-speech on a book
  - ◇ Originally to prevent competition with audio books
  - ◇ But applied even to books with no audio alternative
  - ◇ Kindle Fire has built-in text-to-speech facility
  - ◇ But DRM prevents its use with many books
- For broadest accessibility, avoid DRM
  - ◇ Give users flexibility on how to access
  - ◇ Different platforms and tools are suited to different users

12.3 Accessibility of PDF

12.3.1 PDF documents

- Can be accessible, if created properly
    - ◇ PDF originally a visual medium — designed for printing
    - ◇ But can have *tags* added
    - ◇ Like HTML markup describing text
    - ◇ Headings, paragraphs, etc — overlaid on visual markup
- Untagged PDF:
    - ◇ Can be scaled up for larger text
    - ◇ Can't be converted to speech or braille
    - ◇ Can't be reflowed onto a narrow screen
        - » Either shrink to tiny text, or scroll each line
- Tagged PDF has *potential* to be accessible
    - ◇ Like HTML, only if the tags are actually meaningful

12.3.2 Accessible PDF from MS Word

- Original Word doc must be accessible
    - ◇ Use proper styles — headings, emphasis, etc
        - » Not just font size settings, italic, colours
    - ◇ Use proper lists, tables for data, columns for layout
- Add alternative text to images
    - ◇ Right click image, select "Format Picture..."
    - ◇ Except in Word 2007, where it's in "Size..." dialog
    - ◇ See WebAIM tutorial for gory details: webaim.org/techniques/word/
- Word 2010 has Accessibility Checker tool
    - ◇ File -> Info -> Check for Issues -> Check Accessibility
- Note: all this also helps with conversion to HTML
- Save as a PDF file
    - ◇ Go to "Options" in save dialog
    - ◇ Check "Document structure tags for accessibility"
- Word 2007 requires either Adobe Acrobat, or MS PDF add-on
    - ◇ Earlier versions require Adobe Acrobat
    - ◇ Word 2010 onwards don't — PDF tagging built-in

12.3.3 Accessible PDF from LibreOffice

- LibreOffice aka OpenOffice
  - ◇ Free (open source) alternative to MS Office
- Like Word, use proper styles for accessibility
- Add alt text for images:
  - ◇ Right click image, select “Picture. . . ”
  - ◇ Go to dialog’s “Options” tab
  - ◇ Edit “Alternative (Text only)”
- Use “Export as PDF. . . ” option in File menu
  - ◇ In “General” tab, check “Tagged PDF”

Appendix A

Exercise answers

7. Images solutions

1. Images and accessibility (↑ from page 64)

   The file called *solutions/described-images-complete.html* provides one example of a possible ‘solution’.

8. Navigation solutions

1. Links (↑ from page 78)

   The file called *solutions/links-complete.html*, provides one example of a possible ‘solution’.
2. Tabindex (↑ from page 78)

   A sample solution is in the file *solutions/tabindex-complete.html*

9. Tables solutions

1. Table Accessibility (↑ from page 86)

   Complete solutions: *solutions/tables-basic-complete.html*

1. Table Accessibility (↑ from page 86)

   Complete solutions: *solutions/tables-scope-complete.html*

1. Table Accessibility (↑ from page 86)

   Complete solutions: *solutions/tables-headers-complete.html*

10. Forms solutions

1. Grouping items (↑ from page 100)

   Something like this should show the countries grouped by region:

```
<select name=country>
  <optgroup label="Europe">
    <option>Belgium</option>
    <option>France</option>
    <option>Germany</option>
    <option>Italy</option>
    <option>Spain</option>
    <option selected>UK</option>
  </optgroup>
  <optgroup label="North America">
    <option>Canada</option>
    <option>Mexico</option>
    <option>USA</option>
  </optgroup>
  <optgroup label="South America">
    <option>Brazil</option>
    <option>Ecuador</option>
    <option>Venezuela</option>
  </optgroup>
</select>
```

Complete solution: *solutions/countries-list-complete.html*

2. Arranging form fields (↑ from page 100)

The simplest way is just to wrap a block element like `<div>` round each of the controls, like this:

```
<div><input type=text name=username></div>
<div><input type=text name=password></div>
<div><input type=submit></div>
```

2.1. Form field labels (↑ from page 101)

For the username we'll use the simple way, and just wrap the label and control inside a `<label>` element:

```
<div>
  <label>Username:
    <input type=text name=username></label>
</div>
```

For the password we'll keep them separate and connect them with the `for` attribute:

```
<div>
  <label for=password>Password:</label>
  <input type=text name=password id=password>
</div>
```

The submit button doesn't need a label, since it has its own built in.

2.2. Submit button value (↑ from page 101)

```
<div><input type=submit value="Log in"></div>
```

2.3. Labels above controls (↑ from page 101)

One simple way is just to make the `<label>` and `<input>` elements display as blocks. We'll also space them out with a bit of margin below each control, so that the labels are visually associated with the controls directly under them.

```
<style>
  label, input {
    display: block;
  }
  input {
    margin-bottom: 1em;
  }
</style>
```

For this to work we just need to line up the two kinds of elements, so that the block display will stack them all vertically:

```
<form method=get action="">
  <label for=username>Username:</label>
  <input type=text name=username id=username>

  <label for=password>Password:</label>
  <input type=text name=password id=password>

  <input type=submit value="Log in">
</form>
```

Complete solution for all these exercises: *solutions/login-form-complete.html*