Accessibility Worksheets

NEXXER

Contents

NEXER

1 Introduction

The purposes of this workshop is to improve the HTML provided.

The original HTML, CSS and JavaScript were written by the University of Washington and is licensed under a Creative Commons Attribution-Non Commercial -Sharealike license. Nexer have updated the HTML to include some additional tasks.

As you work through each step in the worksheets, don't forget that after each change you need to validate the html, if you are working with notepad you can do this using https://validator.w3.org/ and copying and pasting your html into "Validate by Direct Input".

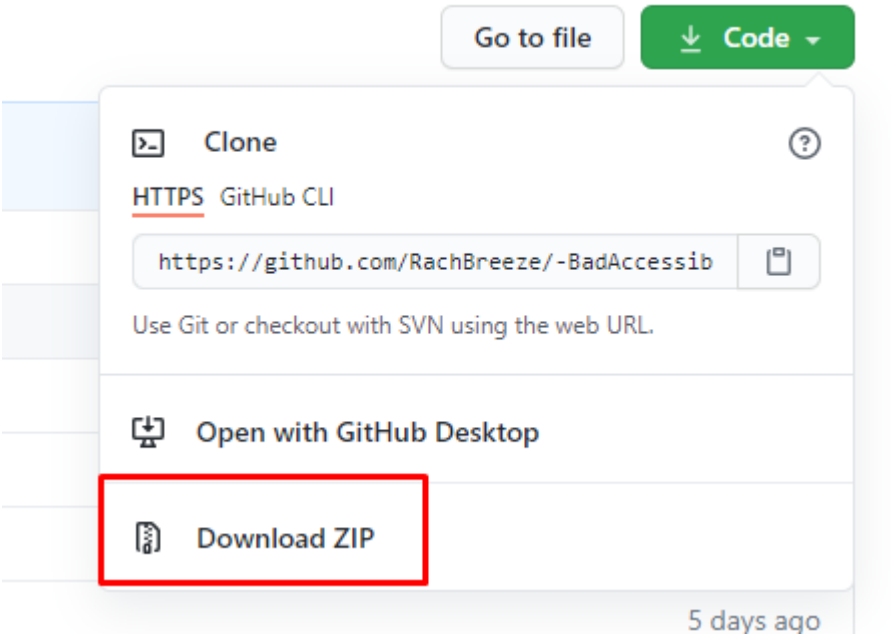
Every element mentioned in the worksheet has a start element such as `<header>` and is closed with an ending element `</header>`.

Suggested process is to keep a copy of the "original" in accessible html, and create a copy so that you can compare the improvements you make as you go along.

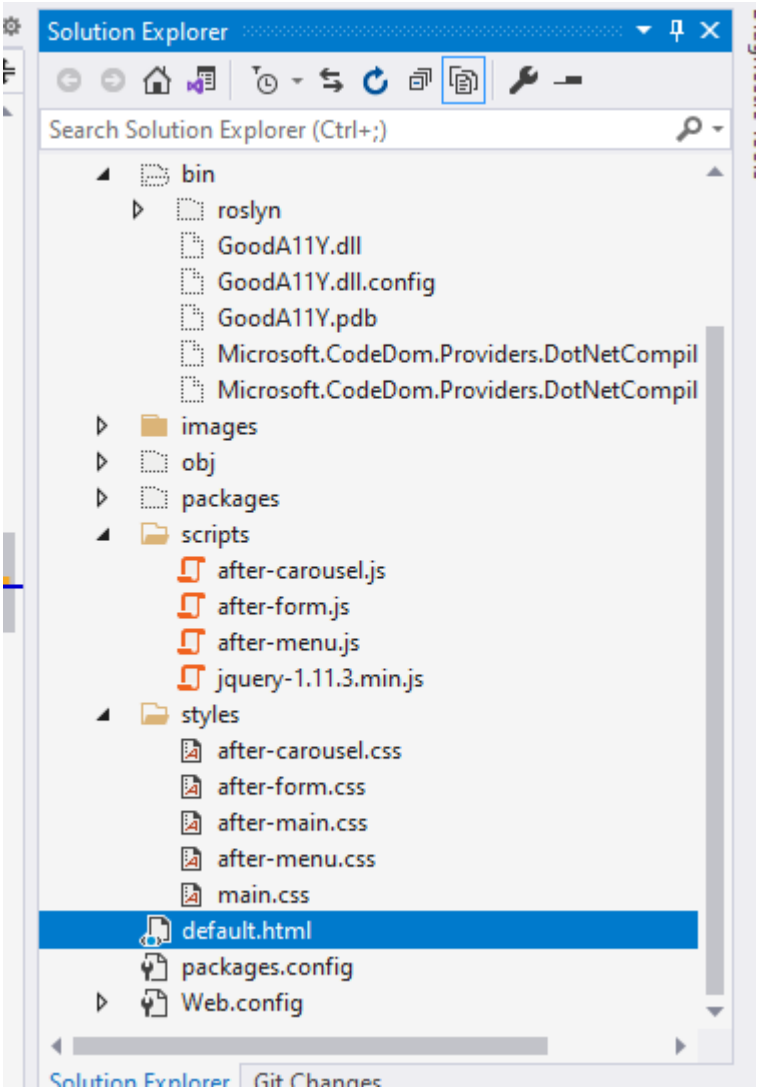
2 Get the code

2.1 Running the code in visual studio

The code is available at https://github.com/RachBreeze/-BadAccessibility pull the code down by clicking download zip from here

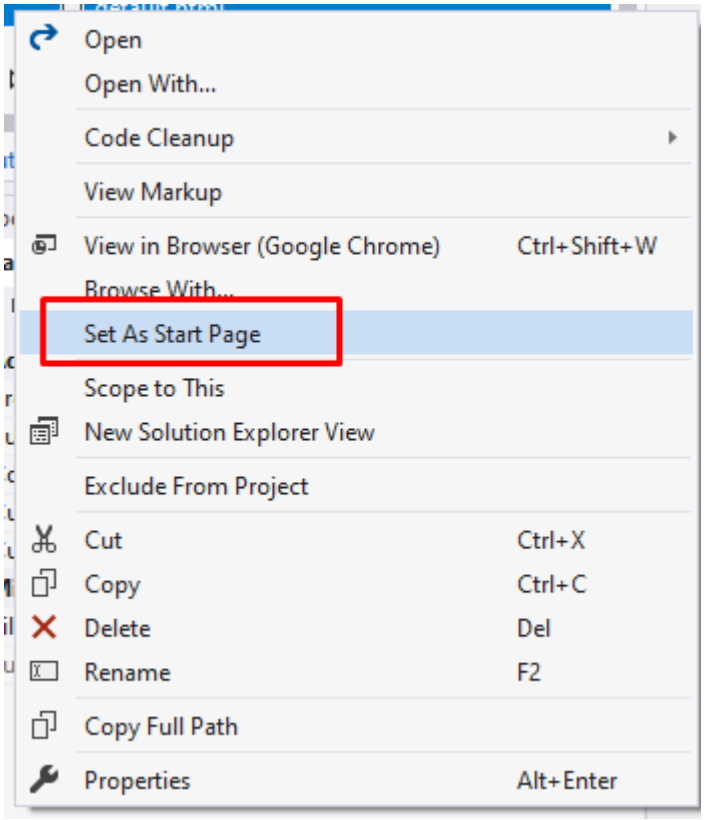


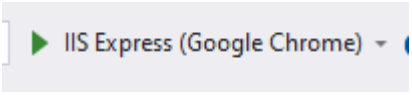
- Open the zip file and extract the files to a folder on your computer ”PinkProgrammingWorkshop”
- Double click BadA11y.sln and visual studio should load.
- Right click on default.html in the solution explorer



- Select set as start page

NEXER



- Press the green arrow in the top 

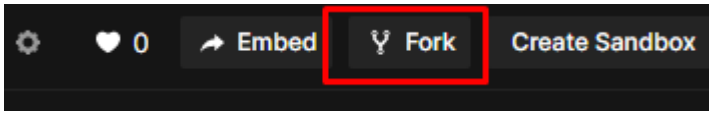
This should launch the site.

To make the changes detailed below, double click the file you want to edit in the solution explorer and it will open the files to edit. To run your changes click save and then, if your site is running refresh the browser you just loaded. If it is not running click the green go button. You can tell if the site is already running because the green go button isn't available to click.

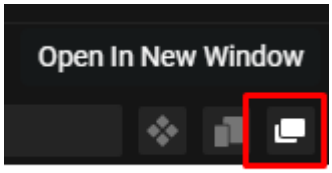
2.2 Running the code in sandbox

Go to https://codesandbox.io/ and login/ register if you haven't already logged in.

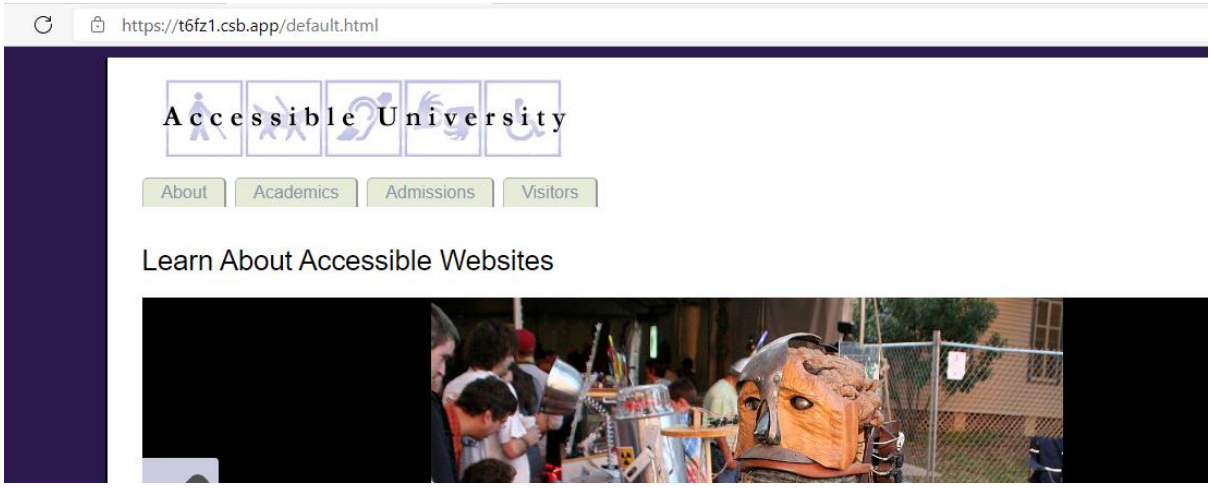
Go to https://codesandbox.io/s/t6fz1?file=/default.html and click fork. This brings the code into your sandbox and lets you edit it there:



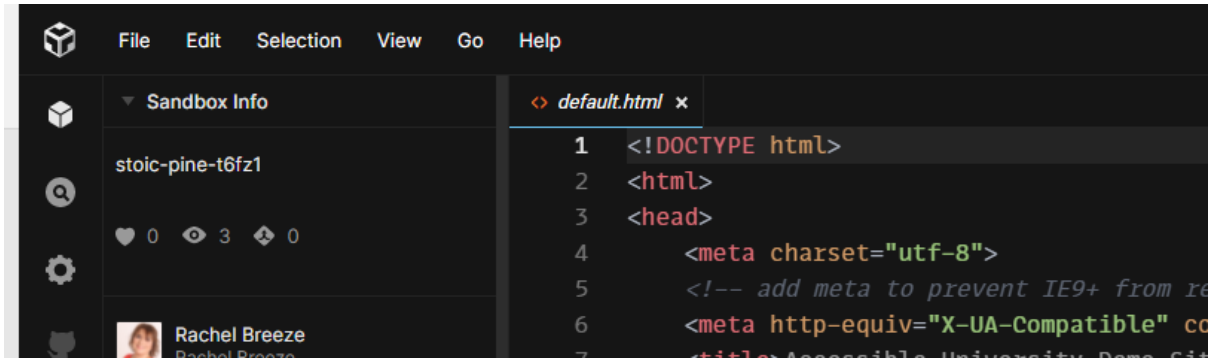
Your site is already live, to find the URL click the folders icon, "open in new window" on the right hand side of your browser:



Add default.html to the url, this will then load the site into your browser:

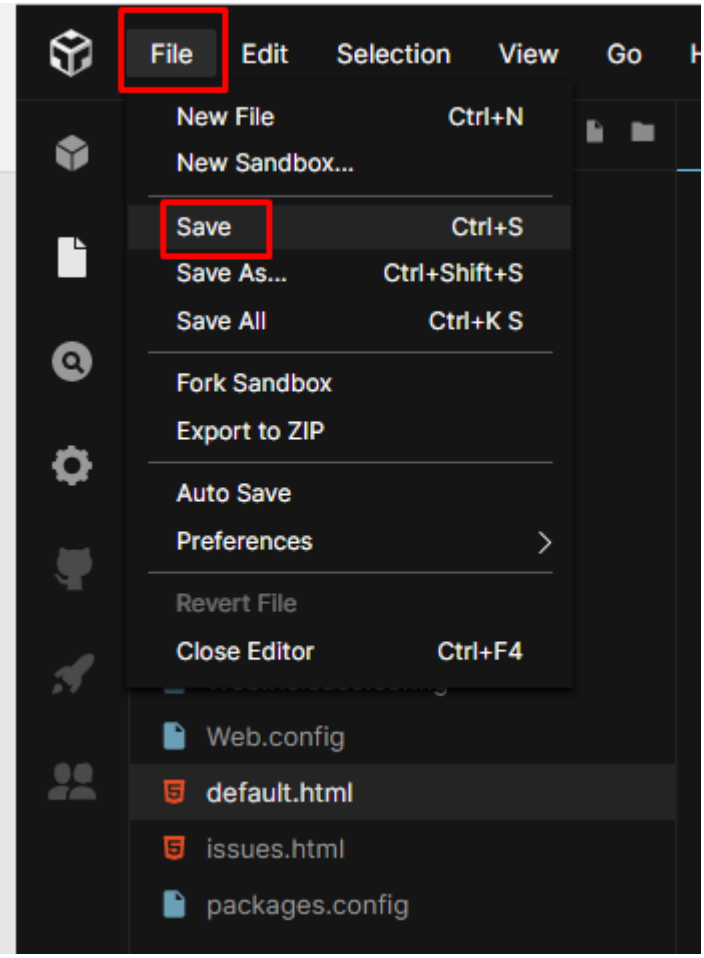


To edit your site make edits in sandbox on default.html:



Then save your edits by pressing control + s or going to file in the menu and click save:

NEXER



Only when you've saved your edits are they visible on the site we've just launched in a new url.

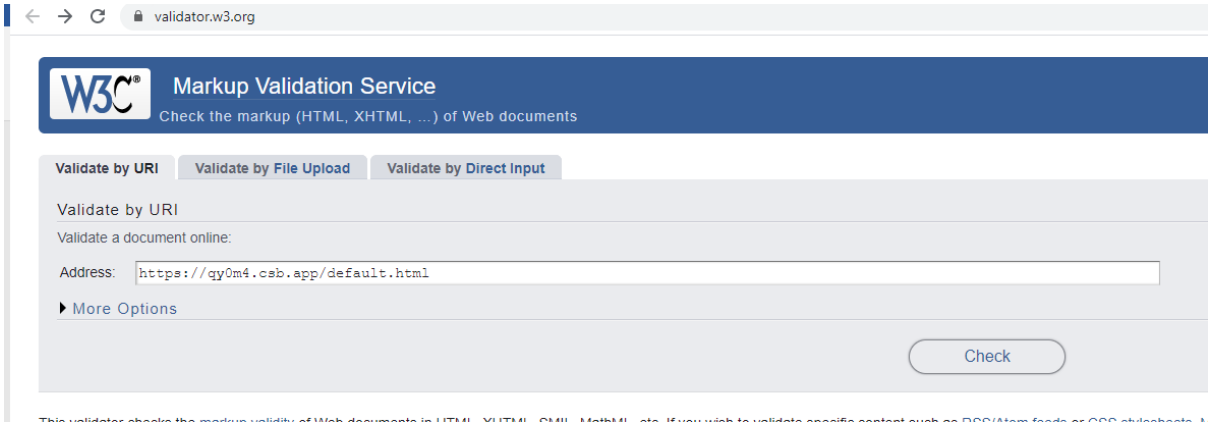
3 Validate the html

A good rule of thumb when picking up any new HTML is to validate it before you start improving it. For creating accesible websites all html must validate against the schema specified, HTML schema is set by the `<!DOCTYPE>` element and is the first element on every page.

3.1 Code sandbox

Take the url you found for loading the site in step 2.2 and copy into the validator.

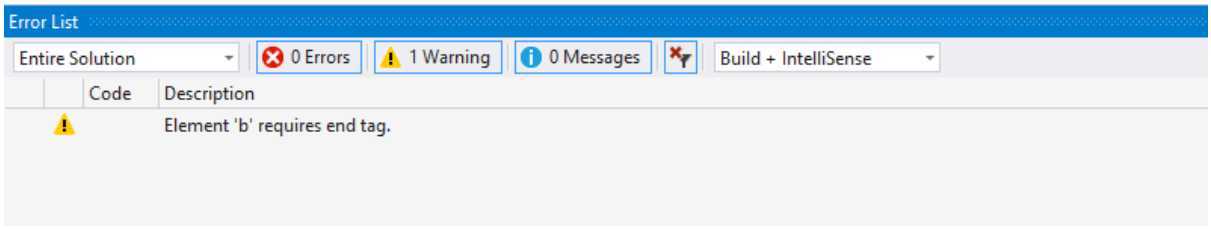
Goto https://validator.w3.org/ place the URL in the URI checker tab, make sure that default.html is after it and click check, an example is:



Then a series of errors will be returned return to the sandbox screen and edit the HTML to fix those errors. Don't forget to retest your amends to make sure there are no errors.

3.2 Visual Studio

Visual studio will provide error messages on HTML but the list will be smaller than in the W3 Validator. These errors can be found on the error list screen:



Fix this error and then open https://validator.w3.org/ and then select the validate by Direct input tab. Copy the html from default.html and past into the textbox and run the check. Then edit the HTML in your visual studio to fix those errrors. Don't forget to retest your amends to make sure there are no errors.

4 Page Structure

Using the correct semantic HTML helps screen readers and other assistive technology to navigate a web page easily. Elements such as `<header>`, `<nav>`, `<main>` and `<aside>` of the elements listed below help assistive technology understand the page structure.

When a role is added to these elements assistive technology can better understand what the purpose of that section of the page is. Each area of the page is self-contained.

An example is a header element that contains a banner:

```
<header role="banner">
```

A list of roles can be found here https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Roles/

The example HTML does not provide enough page structure and it needs to be updated to include the following.

Note it is not valid to nest `<header>`, `<nav>`, `<main>`, `<footer>` and `<aside>` inside one another.

4.1 <header> element

Banners should and logos at the start of the page should be wrapped in a `<header>` element can you identify the banner and apply the correct mark up?

Hint the existing banner is wrapped in a div element with an id of banner.

4.2 <nav> element

Menu items should use the `<nav>` element to allow users to access the menu markup directly.

Can you update the menu item so that it:

- Is wrapped in `<nav>` elements
- Includes an attribute `role` of with the type navigation. Adding this role tells assistive technology this is the navigation element.
- Has an `aria-label` attribute which lets assistive technology know this is the main menu element.

Hint the existing menu is wrapped in `<ul>` tags

4.3 `<main>` element

This allows the main body of the page to be surfaced. Can you identify where to add this and add a main element?

4.4 Final elements

Can you identify where to add and add the following elements:

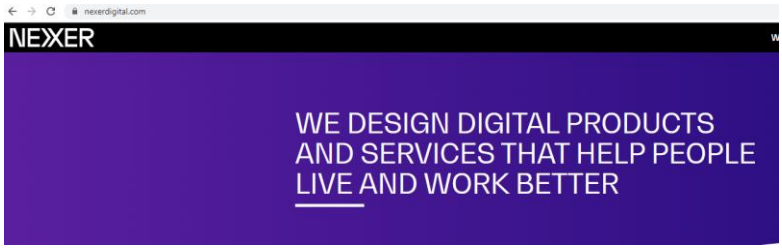
- `<aside>`
- `<footer>`

5 Skip Links

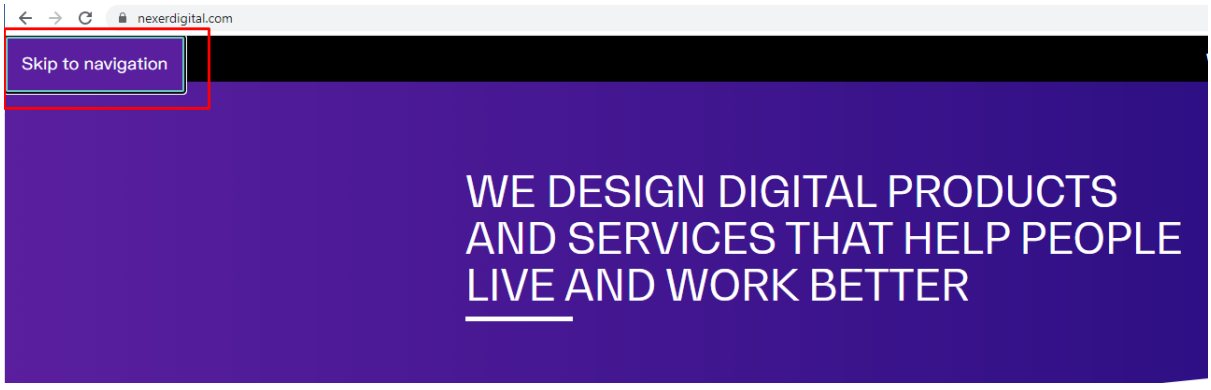
Skip links allow users to skip content on a page. Skip link information can be found here: https://www.w3.org/WAI/WCAG21/Understanding/bypass-blocks.html

They are not visible to the user until the user tabs onto them; an example is on the Nexer Digital site: https://www.nexerdigital.com/

Before tabbing to the skip link:



When a skip link is tabbed to:



Skip links make use of anchors on a page. To use a skip link the link needs to be added as follows with an anchor link:

```
<a href="#nav">Skip to navigation</a>
```

Then an id is added to the page corresponding to the id added in the anchor. In this case we would update the `<nav>` element to include an id as shown:

```
<nav id="nav" role="navigation" aria-label="Main Menu">
```

Skip links should be added straight after the `<body>` tag. Here at Nexer we add them to a `<div>` and because the div contains skip links we add an aria label saying this, and also add a role to the `<div>` called navigation. This allows assistive technology to identify the purpose of the links. Example skip links would be:

```
<div aria-label="Skip links" role="navigation">
    <a href="#nav">Skip to navigation</a>
    <a href="#main">Skip to main content</a>
    <a href="#footer">Skip to footer</a>
  </div>
```

5.1 Add Skip links

- Add skip links to your web page.
- For each skip link added add the id used on the anchor to the element you want to jump to.
- Test that when you click a link it takes you to the area of the page you want to jump to.

5.2 Hide skip links

Once you have completed task 1 you should have the following at the top of your page.

These should be hidden from the user, but still visible when you tab to them and still read out by screen readers.

To do this, we apply a class to the skip links added in task 1 that removes the item from view and bring it into focus when it is clicked on.

This task, doe assume knowledge of CSS, if you get stuck please do feel free to ask for the sheet that suggests a class to add to the CSS.

6 Headings

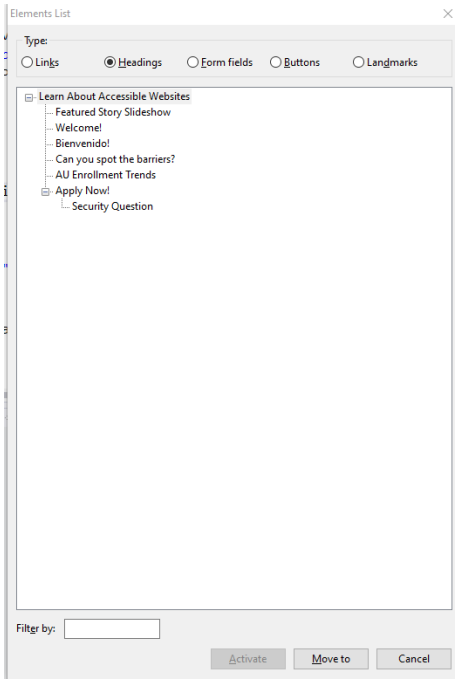
Making text larger to show headings is a great way to help users understand the page's structure. However applying a class or making the font bold does not allow people who use assistive technology to understand the page structure.

Heading elements such as`<h1>`,`<h2>`, `<h3>` … should be used instead. Heading elements have a `<h1>`, `<h2>` etc, Headings are only required for WCAG 2.1 AAA compliance but because they really help people with assistive technology they should be used as much as possible or where the page has visually added headings.

6.1 Add headings

Use a screen reader is used to view the headings in the sample HTML. There are none.

Can you add headings so that the page is more structured?



7 Alt Text

Alt text is used to describe images, not only does alt text help people using assistive technology it also helps people on slow internet connections.

Descriptions of images should be provided based on the context of the page.

Images should always have an alt element, however images which are decorative can have an empty alt element:

```
<img id="logo" src="images/au123456789.gif" width="441" height="90"
alt=””>
```

More information on writing alt text can be found here https://axesslab.com/alt-texts/

7.1 Add missing alt text

Identify the informative images missing alt text and apply alt text (you may have done this already as part of the validation step).

7.2 Fix alt text on decorative images

Use a screen reader and identify the decorative images.

Note when those images are used multiple times it makes the flow for the screen reader difficult to follow.

Either add empty alt tags, these are used for decorative images only. Or move the decorative images in the style sheet and present them as decorative images.

This task, does assume knowledge of CSS, if you get stuck please do feel free to ask for the sheet that suggests a class to add to the CSS.

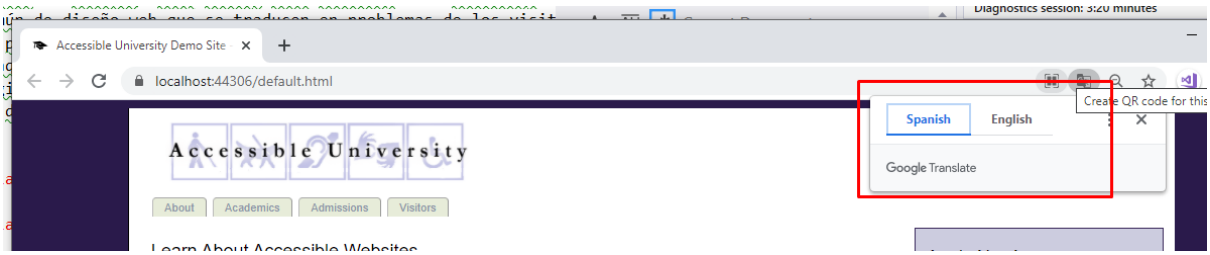
8 Languages

The lang attribute specifies the language of a page, or an element of the page https://www.w3schools.com/tags/att_lang.asp. By specifying the language of the page or an element in the page:

- Screen readers will pronounce the content correctly for a whole page demo can be found here: https://www.youtube.com/watch?v=QwOoU8T24UY
- A demonstration of why it is important to set the language on individual elements of a page can be found here: https://vimeo.com/210258413
- Adding language also:
    - Helps translation tools to identify the language in the page easily
    - Language specific fonts to be added, this can be helpful in multilingual sites: https://www.w3.org/International/questions/qa-lang-why

The lang attribute is missing from two elements on the page before setting adding the elements:

- Listen to the page with a screen reader
- Also in Google Chrome note that the translate the page option is shown:



Now add `lang="en"` and `lang="es"` to the relevant html tags and listen to the page again with a screen reader and to have translation in Chrome you now need to add the translate extension.

Note you may have added `lang="en"` as part of the validation task.

9 Forms

9.1 Add ariaRole

Form elements should be grouped into an Aria Role of "`form`". Find the form on the page and add a role of form to the `<div>` element that contains the form.

9.2 Tell the user the form title

Visually on the template it is possible to see that the form relates to "Apply Now!" that by filling in the form the user will be applying. However when you listen to the screen reader this context is missing. To allow users to know what the purpose of the form is a user of assistive technology should get the same context.

Listen to the form without a title on your screen reader. Now identify the heading field that tells the user to "Apply Now!" and give that heading an id. In the same `<div>` you gave the role of "form" to, tell the browser that the form is labelled by the id you've just given to the heading. Eg:

```
<div id="appForm" role="form" aria-labelledby="appFormHeading">
          <h2 id="appFormHeading">Apply Now!</h2>
```

Now listen to the form using a screen reader, you should hear "Apply Now, form" when you tab into the form.

9.3 Colour and required fields

Colour alone should not be used to convey meaning. Use of colour alone impacts people who are using assistive technology and also users who are colour blind. In this worksheet we have used colour to convey meaning on the form, but colour can be used anywhere on a site for example in red to mean stop and green to mean go.

The example form contains required fields that are marked in blue text. Update the form to:

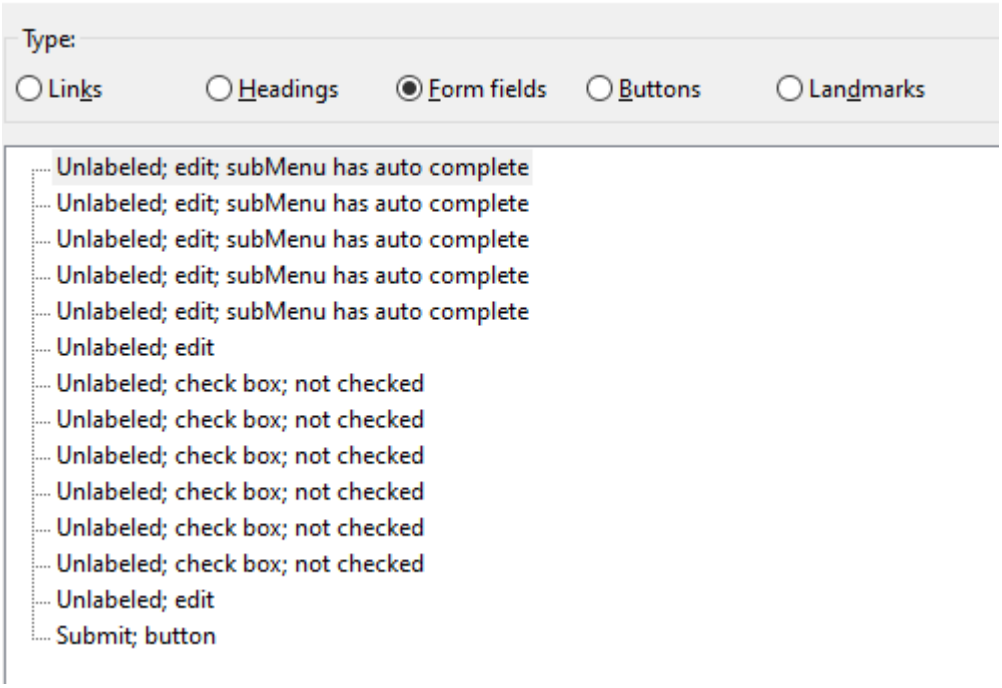
- Indicate the required fields:
    - In the labels – this is done by adding (required) or * to the required fields
    - In the html add semantic mark up to the input to indicate that the field is required e.g. `<input type="email" name="email" required />`
- Indicate to the user in the form instructions how to identify the required fields. To do this at the start of the form indicate that some form fields are required

and provide a key as to how the required fields are indicated, e.g. via (required) or an * in the label.

9.4 Indicate which labels are for which fields

Tab through the form whilst using a screen reader, you will hear “edit has auto complete” or if you have indicated the required fields correctly “edit has auto complete, required”

If you open the elements list, you will also see something like this for the form fields. The current form fields are not marked up so that assistive technology can tell the user what each of the fields are for:



To fix the labelling issue each input element needs an id assigning to it and it’s corresponding label needs a for attribute assigning e.g.:

```
    <label for="email">Email: (required)</label>
<input id="email" type="email" name="email" required />
```

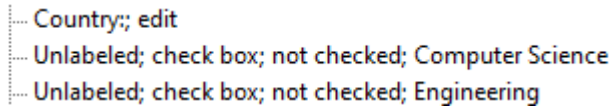
Once you have added the id attributes to the inputs and the for attributes to the labels, tab over the form and listen to the screen reader, you should now hear the screen reader say the label of the text box as you tab into it, and if the text box is required.

If you have added label fors to the checkbox inputs like this:

```
    <label for="cs">Computer Science</label>
```

```
<input type="checkbox" name="major_cs" id="cs" />
```

You will still see this in NVDA:



This can be resolved by reversing the label for concept and using aria labels:

```
<label id="cslabel">Computer Science</label>

<input type="checkbox" name="major_cs" aria-
labelledby="cslabel"/>
```

Whilst both `<label for="..">` and `<input type=".." aria-labelledby="…"/>` are both valid mark up it is always good to check out how the mark up is displayed in the screen reader.

9.5 Fieldsets

Fieldsets are used to group questions together. Can you identify a question which has multiple answers on the form?

Looking at the form we can see there are a group of checkbox answers that relate to the "Desired Major(s)" question. We should wrap these in a `<fieldset>` element this then tells assistive technology that there are a group of fields to read through.

A `<fieldset>` element also requires a `<legend>` element advising the user what is in the field set. Can you add the `<fieldset>` and `<legend>` elements to the form such that you hear "Desired major has grouping list with 6 items.

9.6 Security question/ CAPTCHA

Online forms often use images to help prevent spam and bots from filling and submitting forms automatically, these are called Captcha, which stands for Completely Automated Public Turing test. These are often not accessible, as only sighted users can use them. Visual Captcha cannot use alt text as these can be used by bots to break the captcha, and audio captcha cannot have a text file describing it for the same reason.

Google has implemented a tool called reCaptcha which will not display a Captcha if it believes a person is interacting with the site rather than a bot, however it will display one if it does not believe a human is interacting with the site, you can read more about Google's Captcha here https://developers.google.com/recaptcha/intro. WebAim has

also written about creating spam free accessible forms here http://webaim.org/blog/spam_free_accessible_forms/ .

For the purposes of this exercise the University of Washington have mimicked a Captcha and the supplied “after-form.js” can be used instead of the captcha provided allowing you to use the following mark up in the form, there is no server side validation of the captcha and this task is to illustrate the issues with Captchas:

```
<div id="captcha"></div>
```

9.7 Form help messages

When building forms it is important to provide users as much information as possible about the fields they are entering data in. In more complex forms this may include adding help text to the form. Forms should also prevent users from adding data in the wrong format. For instructions are required for a field use `aria-describedby to` help relate the description to the field for example :

```
<div class="required">
<label for="email">Email: (required)</label>
<p id="email_help">This should be the email address the university can contact you on</p>
<input id="email" type="email" name="email" required aria-describedby="email_help"/>
</div>
```

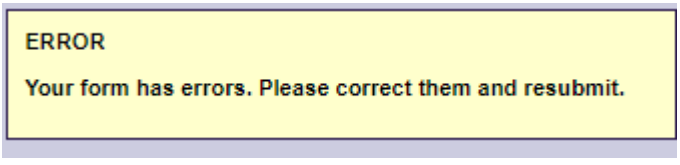
Add some help messages to your form and make sure that the screen reader can read the messages out correctly when you move into the related form control.

9.8 Task 8 Form Validation

Form validation is usually performed when the form is submitted, if the validation fails:

- The user must be told about the error and if using assistive technology the user must not have to focus on the error to determine what the error is
- The user must be provided with guidance on the error
- The user must be able to easily identify the fields with an error in, and focus should be made on the first field with the error.

If you submit the in accessible form the error is reported as:



This makes it hard to identify the errors and fix them.

9.8.1 HTML5 syntax validation

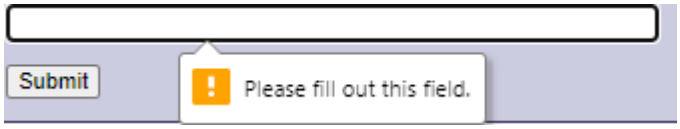
By using HTML5 syntax such as defining the type of input as shown below, the inputs on the form will validate the fields are in the correct format.:

```
<input id="email" type="email" name="email">
```

In this example, if you submit the form with an invalid email address (eg without an @ sign) you are currently not prompted to fix this. Try is, submit the form with an invalid email address and see what happens, now add `type="email"` to the email input field and see how much more improved the error message is.

9.8.2More user friendly error messages

Now required has been set on the form you will see messages like this if the field is empty:



More information than this should be given to the screen reader user this can be done by using the "`oninvalid`" attribute of inputs for example, to ask a user to enter their name on the required name input the following can be added:

```
<input id="name" type="text" name="name" required
oninvalid="this.setCustomValidity('Please enter your name')"/>
```

Add an oninvalid element to all the required fields in the form, and make sure the screen reader can hear the message played back correctly when the fields are empty (please do not worry about the Captcha here)

9.9 Tell the user when they have submitted the form successfully

Successfully submitting a form should take the user to a page that tells the user their application has been successful and what the next steps are.

However, in this task here we want to explore the "alerts". Alerts are used to tell screen readers when the state of the page has changed. The alert role must be used sparingly as it should only be used when immediate attention from the user is required.

For more information on how let the user know about a change when they have stopped interacting with a page, or are considering adding a progress bar, marquee or to your page then recommend reading about aria-live regions here https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/ARIA_Live_Regions

The sample code contains after-form.js which will update a div with an id of feedback with a message regarding successful submission of the form. Add this div to your page below the `<form>` element:

```
<div id="feedback"></div>
```

And then submit the form without error, if you listen to the interaction on a screen reader you will not hear the the form submit has been successful. Now update the div to include an alert role, as shown below and listen to the form submit again, as a screen reader user you can hear the form submit has been successful:

```
<div id="feedback" role="alert"></div>
```

10 Tables

The table mark up is inaccessible. Tables are challenging for screen reader users to understand, as such they should only be used for displaying data and not for styling.

Always consider if there is another option for presenting the data for example a series of bulleted lists.

10.1 Use a screen reader to read the table

Listen take the mouse over the table and listen to what is being said. To do this in NVDA click on the toolbar in chrome, press tab and then press t, this will take you to the table.

Once on the table hold down the ctl button and press down, can you follow what the screen reader is saying? It's not easy to understand which column and row is being detailed.

10.2 Describe the contents of the table

Screen reader users need to know what the data in the table is describing. An example in this case may be:

This table shows enrollment over a two-year period by subject and gender. The academic years 2019-20 and 2020-21 are arranged in columns with six subjects arranged as columns nested within the year columns. Total enrollment, and male and female enrollment percentages, are arranged in rows.

This can be wrapped in a `<div>` or a `<p>` tag with an id assigned to it for example:

```
<div id="table-summary">
```

This table shows enrollment over a two-year period by subject and gender. The academic years 2019-20 and 2020-21 are arranged in columns, with six subjects arranged as columns nested within the year columns. Total enrollment, and male and female enrollment percentages, are arranged in rows.

```
</div>
```

Then on the table element the aria-describedby element can be added:

```
<table id="enrollment" aria-describedby="table-summary">
```

In bootstrap css you can add an class called "`sr-only`" which will only allow screen readers to see the code, it is visually hidden from people using the browser. In the

supplied css “after-main” we have added a class “`sr-only`” which performs the same task, so if you wish to hide the content from the rendered page you can do this here.

Add a description for the table and tell the table about the description.

10.3 Add a caption for the contents of the table

All tables need a caption descripting the contents of the table. This is done by adding a `<caption>` tag after the opening element of the table e.g:

```
<table id="enrollment" aria-describedby="table-summary">
<caption class="sr-only">AU Enrollment Trends</caption>
```

This example also hides the caption from the layout so that only screen reader users know about the caption.

Add a caption to your table and use a screen reader to understand the improvement to the page.

10.4 <th> Element

The `<th>` element defines the cell as a header of a group of table cells, with the exact nature defined by the scope. The scope is either “`col`” as the cell is a header for a cell, or “`row`” as the cell is a header for a row.

In the example mark up can you identify and replace the `<td>` elements which defines the header for the column cell groupings and replace them with `<th>` and a column scope eg:

```
<th scope="col"> 2019-20</th>
```

Next can you identify the `<td>` which defines the header for the row groupings and replace with with the row scope eg:

```
<th scope="row">Total</th>
```

10.5 <td> element

The `<td>` element can be extended to use a “headers” attribute. This attribute tells screen readers where all the headings from the element can be read. To do this, unique ids should be added to all the `<th>` elements and then the header attribute can have all those ids added to it to let the screen reader know what the data represents.

For example the year row may have:

```
<tr>
<td style="width:6em">Year</td>
<th scope="col" id="y19-20" colspan="6">2019-20</th>
<th scope="col" id="y20-21" colspan="6">2020-21</th>
</tr>
```

And then the first elements in the degree row may have:

```
<tr>
<td>Degree</td>
<th scope="col" id="subj-cs1">CS</th>
              <th scope="col" id="subj-eng1">Eng</th>

Then in the total row we will have:
           <th scope="row" id="total">Total</th>
<!--2019-20 data-->
<td headers="y19-20 subj-cs1 total">84</td>
<td headers="y19-20 subj-eng1 total">126</td>
```

In the cell 84 by using headers=”y19-20 subj-cs1 total” we have told the screen reader to use, the headings from the cells with the ids

- `y19-20`
- `subj-cs1`
- `total`

Can you add headers to the table to allow screen readers to read the titles of elements?

10.6 <thead>, <tfoot> and <tbody>

There are three table elements that can help give structure to the `<table>` and can act as hooks for any html you write for the css. They are

- `<thead>` this is wrapped around all the header elements on the table.
- `<tfoot>` this is wrapped around all the footer elements on a table.
- `<tbody>` this is wrapped around all the other elements in the table.

There is only one `<thead><tfoot>` or `<tbody>` element in a table, and they cannot be nested. Can you add these elements to the table?

10.7 No empty cells on the table

Ensure that there are no empty cells on the table. This is difficult because of the way the table has been designed. Can you consider easier ways of building the table for readability?

11 Abbreviations

The tables explored in worksheet 9 contain abbreviations. Abbreviations can be difficult to read and can be misinterpreted for example “Eco” could refer to ecology or economics.

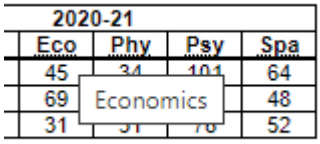
By adding abbreviation titles around the abbreviations the full text is displayed on mouse over and the full text is also readout by screen readers if their screen reader is configured to support this functionality:

```
<abbr title="Economics">Eco</abbr>
```

Inside a table header it is possible to use the `abbr` attribute but this has yet to be fully adopted by all browsers:

```
<th scope="col" id="subj- eco1" abbr=" Economics "> Eco</th>
```

Can you add `<abbr>` to the table elements so when a user hovers over the abbreviations they can see the full title of the item like that shown below:



12 Links

12.1 Uninformative link text

Open the screen read and look at the links tab, there are a number of “click here” links, this means the content is ambiguous. The WCAG 2.1 Success criteria for level A says that “the purpose of each link can be determine from the link text alone”, you can read more about the success criteria and the rules here https://www.w3.org/WAI/WCAG21/Understanding/link-purpose-in-context.html

12.2 Link Styling

12.2.1 Links are difficult to distinguish from other text

The links are only distinguishable from the other text by colour, and this is not easy for most users to spot. Even if the colour difference was stronger, some users may still be unable to distinguish the links from non-link text. This is why browsers underline links by default and it’s good accessible practice for developers and designers to not override this feature, by turning off under line links in their style sheets.

12.2.2 Insufficient visual clues

As a keyboard user navigates through the page it hard to determine where they are on the page

Task – resolve the link styling issues.

Hint: the a tag is used to reference the link elements in style sheet

13 Carousel

The carousel does not auto rotate or scroll which is good, carousels should never do this as users who are distracted by movement or who need more time to read the content need to be able to pause it

Users of assitive technology need to be able to interact with all the controls and in this case users are unable to to access the left and right controls.

13.1 Allow assitive technology to group the elements

In the supplied mark the carousel elements are wrapped in a `<div>`. There are a lot of elements within the `<div>` and it performs a specific task. It is also a significant task. It should therefore be given an aria role of `region`. Region landmark roles content should make sense if it is separate from the main content of the document.

Roles should also have an `aria-describedby` attribute which links to an id containing the details of the content in the region or an `aria-label` attribute.

Task:

- Add the `role="region"` mark up to the `<div>` that contains the carousel.
- Add an `<h2>` element inside the `<div>` that describes what the region does
- In the `<div>` element add an `aria-describedby` attribute that points to the id for the `<h2>` you have just added.

13.2 Add keyboard support for previous and next buttOns

13.2.1 Use valid semantic html

The buttons have been added in "carousel.js". There are two variables, `var prevButton` and `var nextButton` which have been added as `<div>`. When writing html either directly in the browser or via JavaScript, the semantic html should be used, screen readers do not interact with `<div>` elements, but they do interact with `<button>` elements. Change the variables to use `<button>` not `<div>`.

13.2.2 Adding prev/ next buttons with JavaScript

Because the buttons are being added by JavaScript they also need the button type attribute adding this is done as follows:

```
var prevButton = $('<button>')
  .attr('type','button')
```

Add the button type attribute to both `prevButton` and `nextButton` variables.

13.2.3 Add slide indicators

The slide buttons at the bottom of the carousel are interacted with via clicking on an `<li>` this is not valid semantic html; users don't normally click on line indents to interact with elements. The code which adds the `<li>` can be found in "carousel.js". It is this line of code:

```
var lentil = $('<li>')
```

We want to continue to use `<li>` for the display of the slide indicators but need to add a button that is appended to the `<li>` to allow users to interact with it.

To do this lets change the JavaScript so it reads as follows, note we moved all interactions to the button:

```
var lentil = $('<li>');
  var lentilButton = $('<button>').attr({
    'data-slide' : i
  }).on('click',function() {
    showSlide($(this).data('slide'));
  });
```

We then need to append the button to the `lentil` this is done by adding the line:

```
lentil.html(lentilButton);
```

Before the line:

```
lentils.append(lentil);
```

13.2.4 Add slide number indicators

We need to tell asitive technology what number slide they are on. To do this in the for loop

- We need to keep a count of the slide number
- Write out some HTML that is only visible to screen readers
- Add the HTML to the button `lentilButton`

To keep a count of the slides in the for loop `for (var i=0; i<slideCount; i++) {`

Add `var slideNumber = i + 1;`

Then write a line of code that captures the html we want to display:

```
var lentilHTML = '<span class="sr-only">Slide </span> ' +
slideNumber;
```

Then append the lentilHTML to the `lentilButton` html:

```
lentilButton.html(lentilHTML);
```

13.2.5 Test keyboard interactions with carousel

After 13.2.3 and 13.2.4 your `for (var i=0; i<slideCount; i++) {` loop in carousel.js should look like this, and you should be able to access the carousel items using the keyboard

```
for (var i=0; i<slideCount; i++) {
 var lentil = $('<li>');
 var lentilButton = $('<button>').attr({
  'data-slide' : i
 }).on('click',function() {
  showSlide($(this).data('slide'));
 });
 var slideNumber = i + 1;
 var lentilHTML = '<span class="sr-only">Slide </span> ' + slideNumber;
 lentilButton.html(lentilHTML);
 lentil.html(lentilButton);
 lentils.append(lentil);
}
```

13.2.6 Indicate to the screenreader which is the current slide the are on via the button text

The `showSlide` function is called each time a user clicks to view a slide.

This routine sets an active class using:

```
$('.lentils li').eq(index)
  .addClass('active');
```

This should be extended to display on the button that the user is on the current slide. This should only be visible to screen readers. Also because we want to make it easy to remove, when the user picks another slide we want to add a class that makes it easy to find an remove in JavaScript. This means the JavaScript becomes:

```
 $('.lentils li').eq(index)
  .addClass('active')
  .find('button').append('<span class="sr-only current-
slide">(current slide)</span>');
```

Each time a user moves between slides the (current slide) text will continue to be displayed. So, we need to remove it if it exists before it is added back into the html. To do this the following should be added before the above routine is called, this says remove the attributes with the class `current-slide` from the document:

```
$('span.current-slide').remove();
```

The show slide routine should now look like this:

```
function showSlide(index) {

 // hide the current visible slide
 $('.slide:visible').removeClass('current')
 $('span.current-slide').remove();
 // and show the new one
 $('.slide').eq(index).addClass('current');

 // also update the slide indicator
 $('.lentils li.active').removeClass('active');
 $('.lentils li').eq(index)
  .addClass('active')
  .find('button').append('<span class="sr-only current-
slide">(current slide)</span>');
}
```

13.2.7 Advanced: Add a live region to let screen reader users know the slide has changed

In default.html add an `aria-live` region "`polite`" that the JavaScript in carousel.js can call to let screen reader users know the slide has changed, also assign the id attribute to it:

```
<div id="slideStatus" aria-live="polite"></div>
```

In the `showSlide` routine create a message that alerts the user saying "`Now showing slide [slideNumber] of [number of slides].` "

Listen to the page load with the screen reader and make sure that you only hear the alert when the user clicks on the "<" or ">" buttons.

14 Main Menu

Menu items should always have a focus state, and if there are sub menu items these should be accesible via a keyboard.

14.1 Add a focus state to the menu

In the example HTML the menu has an id of ”menu” and the ”menu.css” hooks into this with `#menu`. For example:

```
#menu li a {
          padding: 0.25em 1em;
     display: block;
          color: #8A94A8;
}
```

Says on the html with the id “menu”, for each `<li>` element which has an `<a>` element apply this style. So this is the CSS you need to edit to get a color when a user tabs into it.

14.2 Identify the code displaying the sub menu

In the examle ”menu.js” can you identify the code that is displaying the sub menu and determine why the menu is not displaying.

Fixing the issue will take some time, so we will provide some sample JavaScript, which will show one way to resolve the problem.

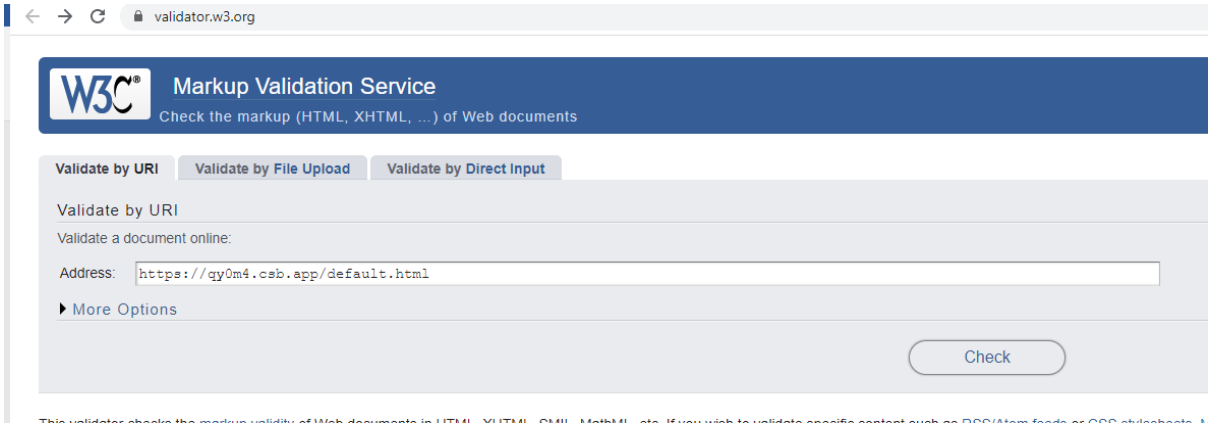
15 Final tests

This section provides some basic tests for you to perform on the work you have done and is not a guide to a full audit of the work.

15.1 Validate your html

15.1.1CodeSandbox

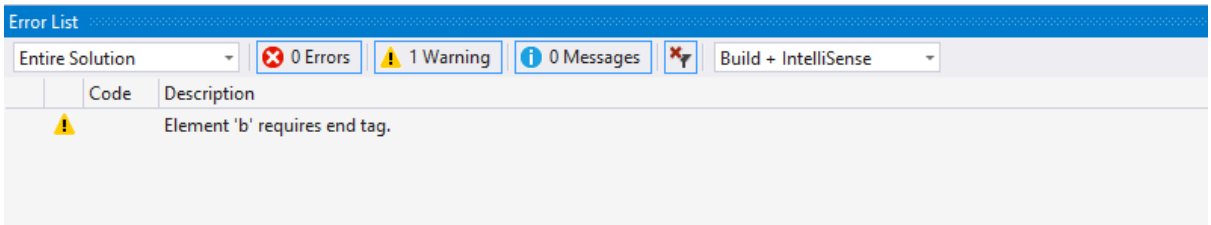
Tak the url you have been working on and goto https://validator.w3.org/ place the URL in the URI checker tab, make sure that default.html is after it and click check, an example is:



Then a series of errors will be returned return to the sandbox screen and edit the HTML to fix those errors.

15.1.2 Visual Studio

Visual studio will provide error messages on HTML but the list will be smaller than in the W3 Validator. These errors can be found on the error list screen:



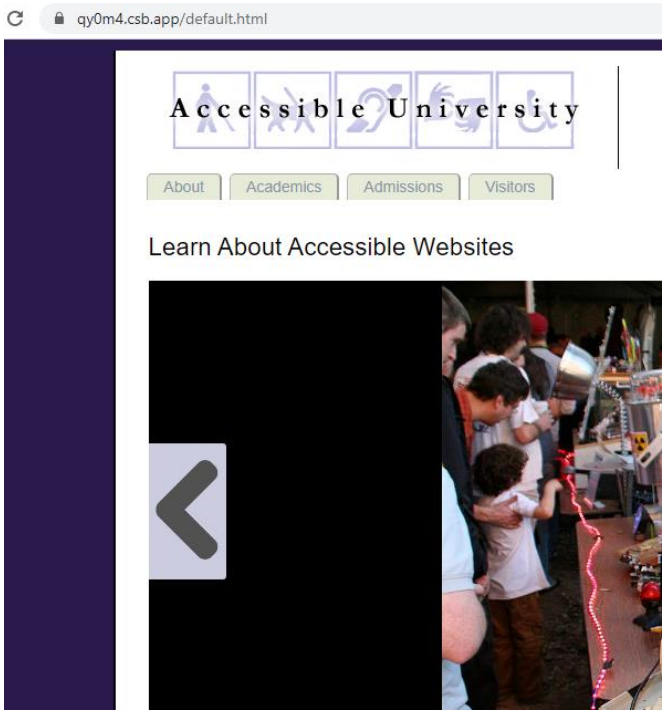
Fix this error and then open https://validator.w3.org/ and then select the validate by Direct input tab. Copy the html from default.html and past into the textbox and run the check. Then edit the HTML in your visual studio to fix those errrors.

15.2 Chrome Dev Tools

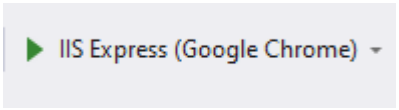
15.2.1 CodeSandbox

Tak the url you have been working on and

And open a new chrome window so it looks like this then jump to 15.2.3 ”Launch Chrome Dev Tools”:



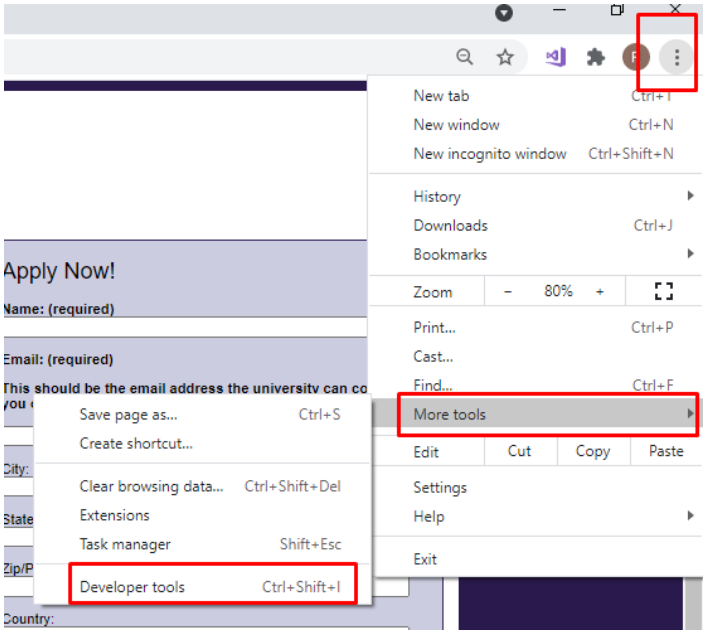
15.2.2 Visual Studio

Run your project bly clicking this button 

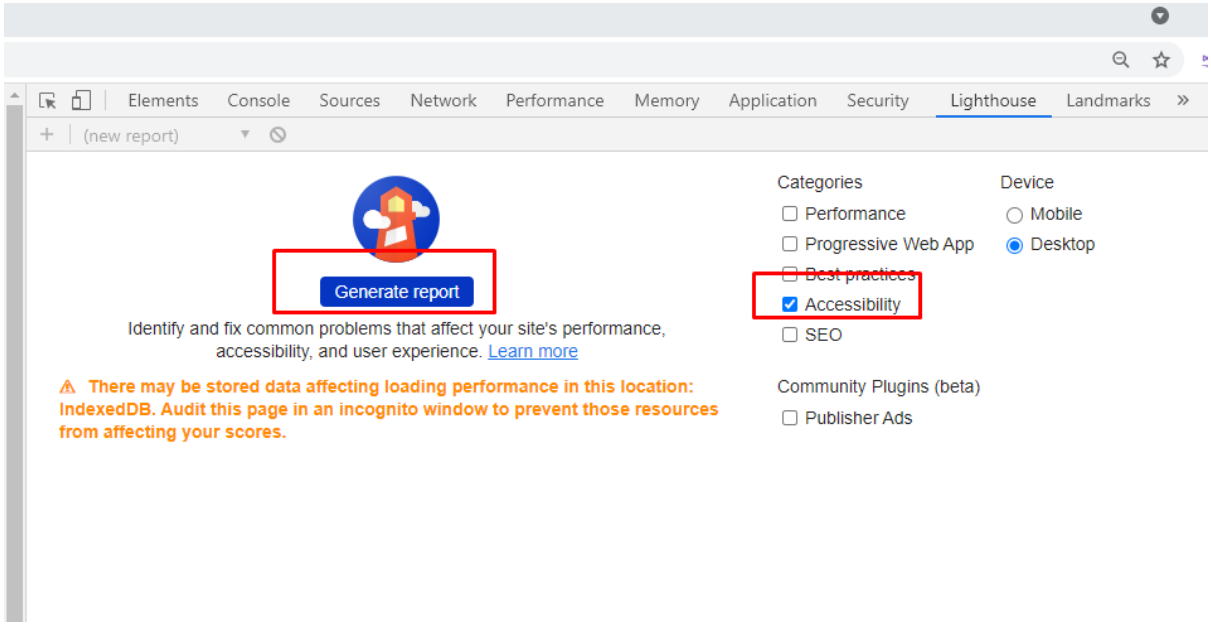
or pressing F5 then jump to 15.2.3 ”Launch Chrome Dev Tools”

15.2.3 Launch Chrome Dev Tools

Launch Chrome Dev Tools, to do this press the three dots in the chrome browser. Select more tools, then developer tools:



Once loaded select the lighthouse tab and ensure that just accesibility is ticked.then click generate report:



Light house will report an accessibilty score, detailing items that have been missed and also provide details of things for you to manually check.

15.3 NVDA/ Voice Over

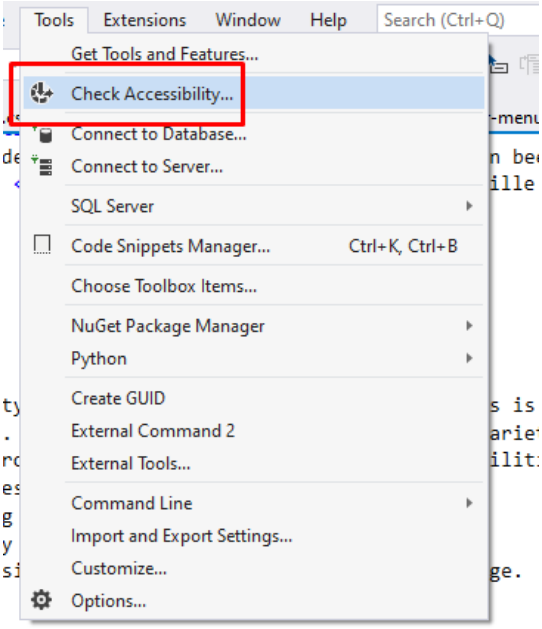
With the browsers open and serving your new page, open NVDA/ Voice Over

- Check that the screen reader is reporting what you can see.
- Is the screen reader reporting table contents correctly
- Is the screen reader reading out form entries correctly
- Are all links distinct and make sense out of context

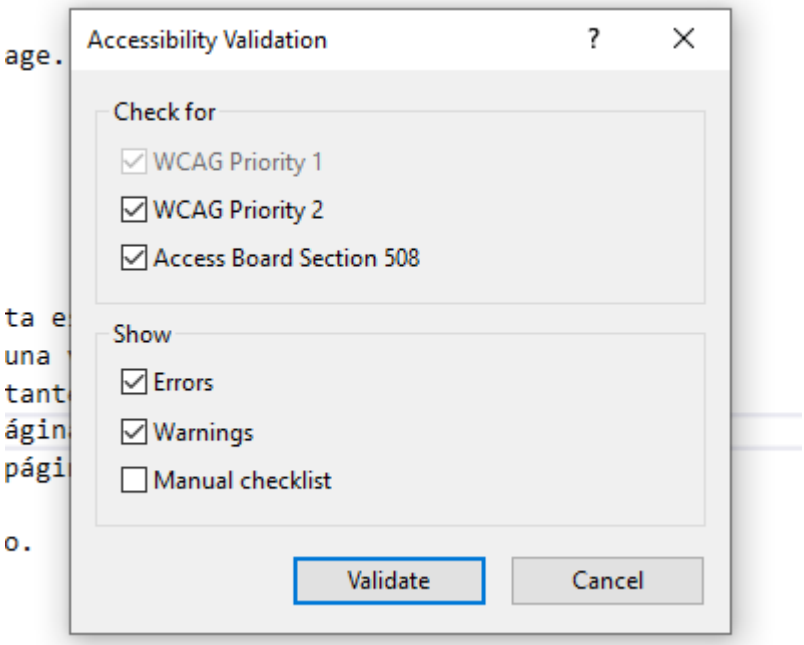
- Are heading structured and in a sensible order
- Are all form fields labelled up correclty
- Are buttons distinct and make sense out of context
- Do the landmarks represent the layout of the page

15.4 Visual Studio Accesiblity Checker

If you have visual studio accesibilty extension installed run this on your default.html. You can do this by making sure that default.html is open in your editing environment and clicking on Tools -> Check Accessibilty:



Then select the level of validation you want:



This will provide some extra checks on your code, but can sometimes report false postivies so you will need to review each of the errors reported.

16 Word and Pdf document

Word and Pdf documents are not always accesible, when publishing content on the web, the prefered method of publishing is to use html. Not word or pdf documents. As well as being more accesilble, html is easier for search engines to crawl.

17 Sample code

Sample solutions can be found at https://github.com/RachBreeze/-BetterAccessibility or https://codesandbox.io/s/divine-http-7tsp4