Accessability and graphic image editing

Course overview

1. Graphical resources for the web

1. Graphical resources for web applications

2. Editing images for the web

1. Image formats
2. Editing images for the web
3. Making images web friendly

3. Optimize images for SEO

1. Optimizing images for SEO

4. Accessibility

1. Accessibility
2. Designing for accessibility
3. Coding for Accessibility
4. ARIA Roles and Attributes
5. Accessibility Guidelines and Standards
6. Accessibility Testing, Tools and Resources

Graphical resources for the web

Sections in this chapter:

1. Graphical resources for web applications

1-1. Graphical resources for web applications

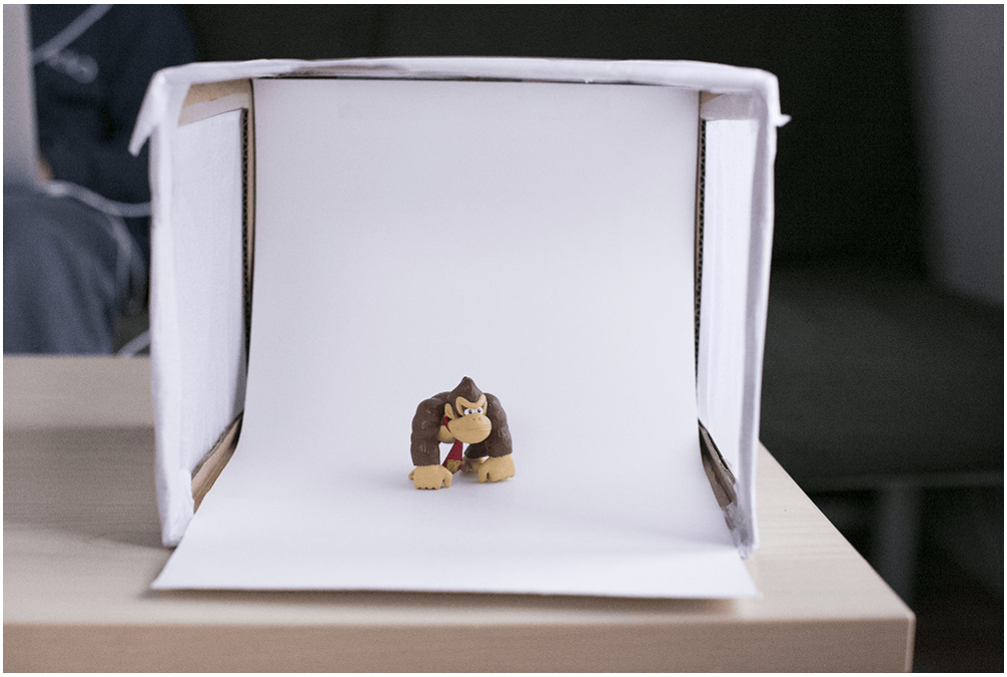
There are many **high-quality stock photo sites** where you can find great photographs to use on your page. Many of them are free and lets you download their photos for commercial use. Below is a list of a few of them: 1-1-1

- Pixabay
- Unsplash
- PicJumbo 1-1-2

However, stock photos cannot be used for **everything**. You might want to use photos of yourself, family members or your product. 1-1-3

If you need to take photographs of your own products, you can make them look **well-lit and professional**, without expensive camera equipment. 1-1-4

This can be done by creating your own **lightbox**. Link to guide: 1-1-5
http://www.jimdo.com/blog/product-photography-with-diy-light-box/



There are also lots of free and online tools for building your own infographics or photos with text overlays, like Canva and PiktoChart. 99designs can be used if you are looking for a logo. 1-1-6

An example of an infographic: 1-1-7

To help reinforce the message of the page, you should have at least **one** image on every page of your website. This can help **catch the reader's eye** and **break up your text.**

Editing images for the web

Sections in this chapter:

1. Image formats
2. Editing images for the web
3. Making images web friendly

2-1. Image formats

There are two main types of image files 2-1-1

- Raster
- Vector

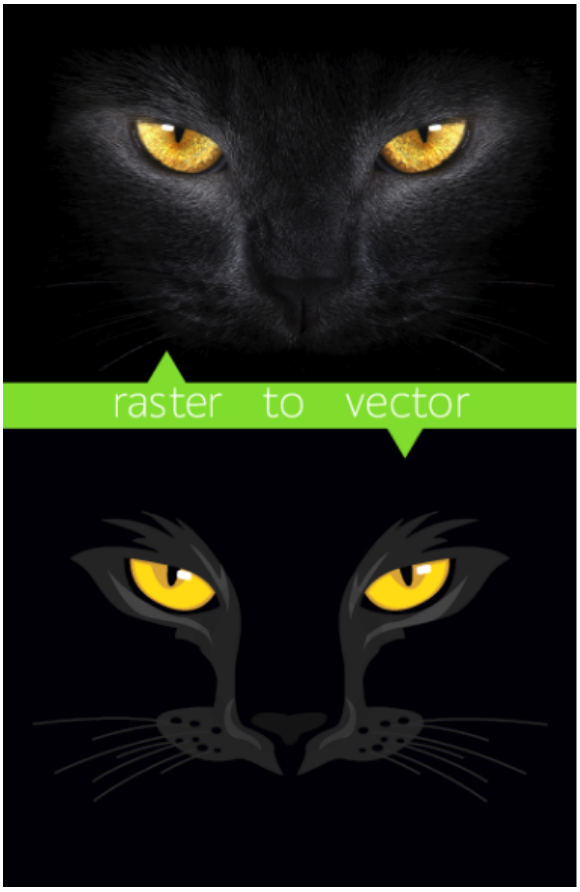
Raster images are created with **pixel-based** programs or by using a digital camera. When using a *raster* program, you *paint* an image. Colors can be blended to soften transition from one color to another. 2-1-2

Raster images are made of **pixels**. A pixel is a **single point** of the smallest single element in a display device. When zooming in on a raster image, you will eventually see lots of tiny squares. 2-1-3

Vector graphics are created with vector software. When using a *vector* program you *draw the outline of shapes*. 2-1-4

Vector images are mathematical calculations from **one point to another**, that form lines and shapes. If you zoom in on a vector graphic, it will always look the same(no pixels). 2-1-5

Photographs have soft color blends, while vector graphics have a clear distinction between each color. 2-1-6



Raster image formats

JPEG is short for **Joint Photographic Experts Group**. This format is optimized for **photographs** and can contain up to **16.8 million colors**. This means that it can display a high amount of **nuances** for each color.

Although JPEG has a very effective **compression**, meaning the file size can be **compressed** to a **high degree** without the quality of the photograph being affected, there is a limit.

The reason for this is that the method of compression used by JPEG removes information from the photograph. This is referred to as **destructive compression**.

Destructive compression means that less pixels will be saved each time the image is edited and saved, and as a result the image will appear to loose quality over time.

JPEG are very good at handling **photographs**, but not **text or drawings** (like straight lines) and such. Text stored as JPEG tend to get **blurry** and it is generally not recommended to use JPEG for text. You should instead use PNG or GIF for texts.

JPEG Pros:

a. Good at handling photographs
b. Can handle many colors
c. High degree of compression possible

JPEG Cons:

a. Uses destructive compression
b. Does not handle images with text on well
c. Bad format for drawings, schematics and illustrations

GIF is short for **Graphics Interchange Format**. Each image can handle up to 256 colors, making this format a bad fit for photographs. It is on the other hand very good at handling text and illustrations.

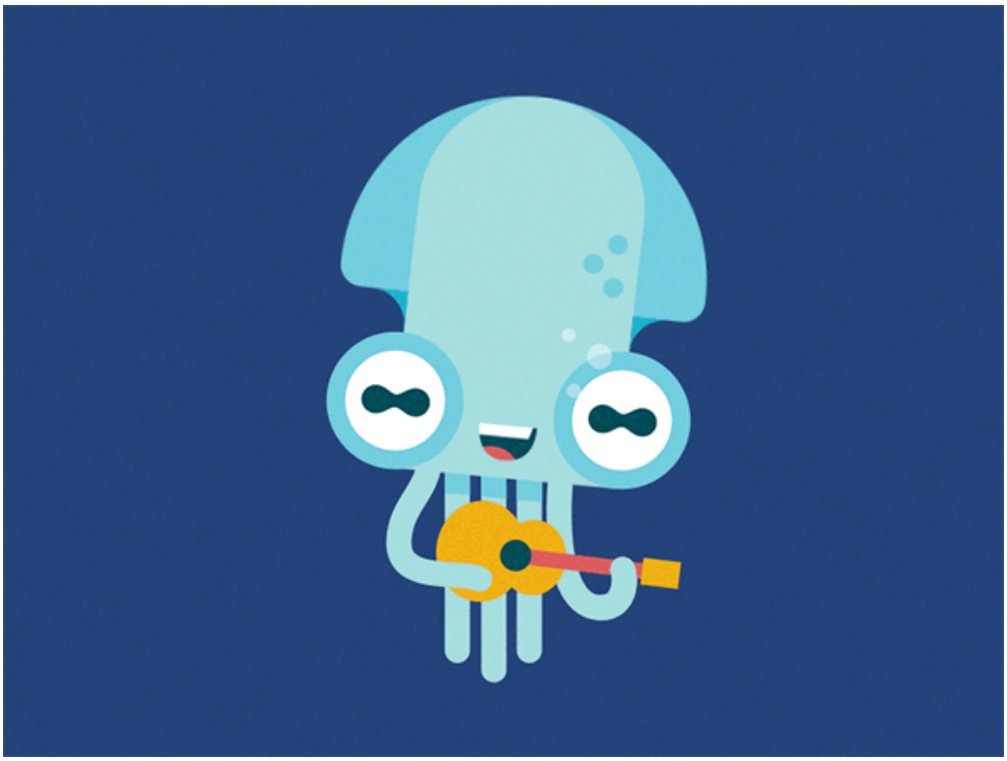
The cause for this is that less number of colors results in a high contrast (sharper contours) and since GIF uses a different form of compression that does **not** remove information from the image.

Even if the image is compressed, it will retain all information. If your image contains text, you should use **GIF** or **PNG**, since both these handle text well. 2-1-19

Another advantage of using **GIF** is the ability of making parts of the image **transparent**. This is not possible with **JPEG**. 2-1-20

An image that is partly transparent can be placed on top of other images, making parts of the underlying image visible. This can be useful for placing text or graphical objects on top of photographs or other images. 2-1-21

Another advantage of GIF is that it can be used to create animations. 2-1-22



GIF Pros 2-1-23

a. Good at handling text
b. Can handle transparent areas
c. Can be used for animations

GIF Cons 2-1-24

a Cannot handle photographs well
b Can only handle 256 color
c Compression is not effective

PNG is short for **Portable Network Graphics**. This format is younger than JPEG and GIF, and was created with the intent of replacing GIF by offering more features. 2-1-25

There are three kinds of PNG-images: 2-1-26

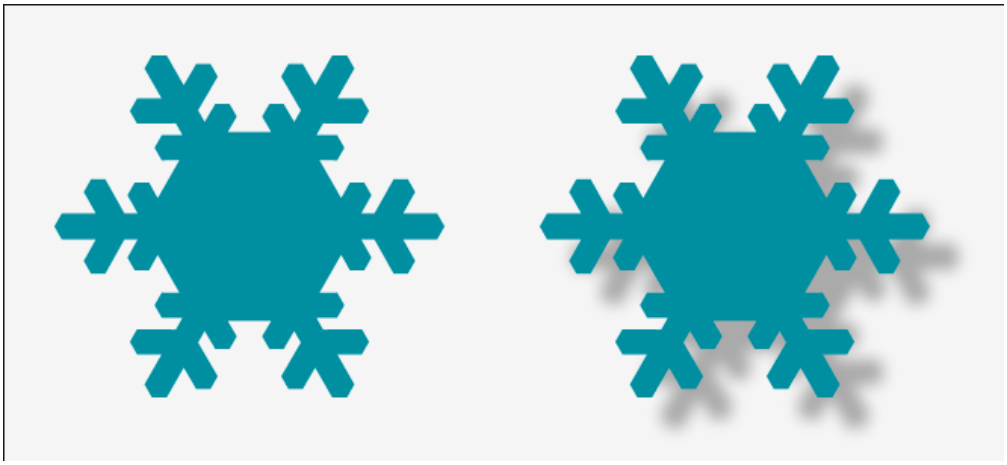
- **PNG-8**: handles up to 256 color
- **PNG-24**: handles 16.8 million color
- **PNG-32**: handles 16.8 million colors and has advanced functionality for transparency

PNG supports **transparency** just like GIF, so selected areas in an image can be made transparent. The difference between GIF and PNG is that GIF will allow you to use **percentages** for deciding how transparent an area should be. 2-1-27

This means that PNGs can **melt into the background** in a better way than GIFs, which is best observed when using **circular or asymmetric figures** that are to be placed on top of other images. 2-1-28

 2-1-29

This can be used when creating **shadow effects**, since the foreground and background image mesh together in a way that is not possible with GIFs. 2-1-30



PNG is good at handling **photographs**. Much better than GIF and usually just as good as JPEG. PNG uses a different form of **compression** than JPEG, which means that there is **no** loss of information when a PNG-image is compressed.

In other words, PNG does not use **destructive compression**. The downside is that the file size of the image cannot be shrunk in the same degree as a JPEG, resulting in PNG file sizes being quite large.

When in need of smaller file sizes, JPEG should be used instead. Older browsers may not support the PNG format, although all the new ones do.

PNG Pros

a. Very good format for text and logotypes
b. Advanced support for transparent areas
c. Can handle many colors

PNG Cons

a. Does not support animations
b. Large file sizes for detailed images
c. May lack support in older browsers

Vector graphic formats

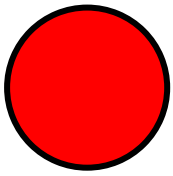
SVG is short for **Scalable Vector Grahics**. It is an **XML-based** markup than can contain **two-dimensional vectors**. Since **text** is used to describe the graphic, an SVG file can be scaled to different sizes without losing quality. 2-1-38

2-1-39

```
<!DOCTYPE html>
<html>
<body>

<svg height="100" width="100">
  <circle cx="50" cy="50" r="40" stroke="black"
          stroke-width="3" fill="red" />
  Sorry, your browser does not support inline SVG.
</svg>

</body>
</html>
```



SVG Pros 2-1-40

- Relatively small file sizes
- Can be embedded directly into HTML
- Since it is essentially text, it can be created using a simple editor

SVG Cons 2-1-41

- File size may grow fast, if the SVG consists of a large number of small elements
- Impossible to read a part of the graphic object
- Variable browser support

2-2. Editing images for the web

With images, you are most likely to encounter a JPEg (or JPG) or a PNG. As discussed in the previous sections, there are pros and cons of each, but for most cases you can remember the following: 2-2-1

- **Photographs should be saved and uploaded as JPEGs.** This file format deals with all the colors in a photograph in a very manageable, efficient way, so you won't end up with the monster file size you would if you saved a photograph as a PNG. 2-2-2

- **Graphics, especially those using large, flat areas of color, should be saved as PNG.** This includes most design files, infographics, images with a lot of text in them, and logotypes. PNGs are higher quality than JPEGs and they deal with areas of color and text with nice crips lines, so you can zoom in and not loose quality. 2-2-3

They also support transparent backgrounds (which you'll want if you are using a logo). It is recommended that the PNGs are saved as 24 bit rather than 8 bit because of the better quality and richer array of supported colors. 2-2-4

Q What if you are using a photograph with text over it? Should you use JPEG or PNG? 2-2-5

A If the majority of the image is a photograph, you can stick with JPEG. 2-2-6

You can select the type of file you want with any simple photo program by going to the **File** menu and selecting **Save as**, **Export** or **Save for web** and then choosing the file type you prefer. There are also free online tools that will convert files for you, like Zamzar. 2-2-7

You can convert from PNG to JPEG, but you don't gain any quality by converting a JPEG to a PNG. So for example, if you only have your logo as a JPEG, you will need to go directly to your designer and ask for a PNG from their original design files, rather than trying to reverse-engineer one from the JPEG. 2-2-8

2-3. Making images web friendly

With web images, you want to find the right balance between **file size** and **resolution**. The higher your resolution, the better your image will look, but the larger the file size will be. 2-3-1

Huge image files on your site can slow down the loading of your page **(page speed)**, which hurts your user experience and, eventually, your search engine ranking. 2-3-2

So, how do you strike the right balance? First, it's important to understand when it comes to images, "size" is a relative term. What you need for print is usually much larger than what you need for a website. 2-3-3

What makes up size? 2-3-4

- **File size:** The number of bytes the file takes up on your computer. File size is the factor that can slow your website way down. A 15MB(megabyte) photo is a huge file. A 125KB(kilobyte) file size is much more reasonable. If your file size is very large, it is an indicator that either your image size is too large or **resolution** is too high.

- **Image size:** The measurement of the height and width of the image, in pixels. Images on the web are measured in pixels. 2-3-5

- **Resolution:** The quality or density of an image, measured in dots per inch(dpi). Most computer monitors won't display more than 72dpi, so anything bigger tham that is overkill and just making your file unnecessarily large. When a design program has the option to "save for web", it means saving the file at a low, web-friendly resolution. 2-3-6

But how do you find the file size, image size and resolution of your image? 2-3-7

You can find the file size and image size on your computer. For PCs, right-click on the image file, choose "Properties" and then the "Summary" tab. On a Mac, Ctrl+click on the image file and choose "Get Info". 2-3-8

Finding the resolution requires a more advanced photo program like Photoshop, but most basic image editing programs will automatically save images at a lower, web-friendly dpi. Pixlr's free web version presents your images at 72 dpi, and Canva lets you "save for web" which gives you a PNG at 92dpi. 2-3-9

Some pointers to keep in mind: 2-3-10

- Large images or full-screen background images should be **no more than 1MB**.
- Most other small web graphics can be **300KB or less**.

- If you're using a full-screen background, it is recommended that you use an image that is **2000 pixels wide**. 2-3-11
- If you have the option, **always "Save for web"** which will give your image a web-friendly resolution.

- You can make a large image smaller, but **it's very hard to successfully make a small image bigger.** As an example, if you have a 100x100 pixel image and you want to turn it into a 2560x1440 banner, the resulting image will become pixelated and blurry. If you think about it in terms of volume, you can't make a gallon of water fit into a swimming pool. 2-3-12

What if the file size is too large for your website? 2-3-13

If you have a digital camera, you might be taking photos that are several megabytes large - way bigger than you need for your website. Stock photos from high-quality sites tend to come with large file sizes too. If your file size is over 1MB, there are a few things you can do: 2-3-14

- **Resize the image.** If your photo is 5000 pixels wide, you can easily resize it to 1200 pixels wide, or even smaller depending on how you plan to use it on your site. This will significantly reduce the file size. When you resize, make sure ti keep the proportions the same so you don't distort your image. 2-3-15

- **Reduce the resolution.** Photo programs like Pixlr and Canva will automatically reduce your image resolution to a "web-friendly" size (72dpi and 92dpi, respectively). You can do this in Photoshop too with the "save for web" option. You can also "Save As" in many photo programs and then adjust the quality level from there. 2-3-16

- **Run your image through a free program like TinyPNG or TinyJPG**. Both will significantly reduce your file size without interfering with the quality. 2-3-17

Images on a web page will look better if you use consistent style and size/proportion. Consistency will also help when lining up your text, columns, and other information on your page. 2-3-18

Optimize images for SEO

Sections in this chapter:

1. Optimizing images for SEO

3-1. Optimizing images for SEO

Most people don't think about their file names. They may call a photo "Photo1.jpg" or "Screen Shot 2017-09-12 12:45:33". If this sounds familiar, take a moment to rename your files before you use them on your website. Why? Because doing so will give your SEO a boost. 3-1-1

Think about it this way. When Google scans your website, it can read your text but it can't see what's in your images. The file name provides information about what's in the image so that Google can interpret it correctly(think *eiffel-tower.jpg* rather than *DSC12345.jpg*). 3-1-2

The file name also becomes part of the image's url, so naming your file something in plain English will make your urls easier to navigate and interpret. 3-1-3

For consistency's sake, name your files with lowercase letters and numbers 0-9. Don't introduce punctuation or spaces. And it's better to use hyphens rather than underscores. 3-1-4

The reason for this is that Google will interpret underscores as concatenation, or joiners, so technical terms like *FTP_BINARY* will appear on search result pages. Spaces becomes *%20*. 3-1-5

People often forget to fill the additional information for their images, like the alternative text and the caption. Alternative text(or alt tags) won't be visible to your average visitor, but they give search engines a basic idea of what each image is about. 3-1-6

So once you've added an image to your site, be sure to fill in the alternative text with a phrase that describes the what the photo is showing, preferably with a targeted keyword. 3-1-7

Alternative text also helps visually impaired visitors navigate your site with audio-based software, so it's a nice way to improve your website's accessibility. 3-1-8

If you are posting a graphic or a photo with text over it, your alternative text can just be a repeat of what that text or title says. 3-1-9

File names and alternative text are especially important for the SEO of product pages. If it makes sense, also add a caption to your image, since people tend to read photo captions more than anything else. 3-1-10

Put your images near relevant text. Choose images that are related to what the text is saying, rather than completely out-of-the-blue. An image that is surrounded by relevant text(with related keywords) will rank better. 3-1-11

This tip also helps you avoid stock photo cliches. If your website is about education, you don't get any SEO boost from having a photo of an apple. Photos of teachers, students, and classrooms, though, will be more interesting for readers and more relevant to your subject matter. 3-1-12

Accessibility

Sections in this chapter:

4-1. Accessibility

What is web accessibility and why is it important? *4-1-1*

- Making sites and apps available for everyone
- Not just those with disabilities
- People with vision impairments, some people are partially or completely blind, some are using screenreaders

- People using assisitive technologies, some people navigate the web using only a keyboard or by using assistive tecnologies that mimick using a keyboard *4-1-2*
- People with cognitive issued, some have cognitive issues that affect learning, memory, perception and problem solving
- People with temporary issues, such as a broken arm or broken hand or a broken finger that prevent them from using the mouse

- People with changing abilities, due to age *4-1-3*
- People in urgent situations, they may have an emergency

Technology is for everyone *4-1-4*

Accessibility is about everyone *4-1-5*

It's our job to make it as easy as possible for this wide range of people to access the websites and applications that we design and build 4-1-6

What do we mean when we say disability? 4-1-7

Vision issues 4-1-8

- Blindness
- Low vision
- Color blindness

Hearing issues 4-1-9

- Deafness
- Hard-of-hearing

Cognitive issues 4-1-10

- Learning disability
- Distractibility
- Inability to remember or focus

Technical issues 4-1-11

- Slow connection
- Screen size
- Old technology

Changing abilities 4-1-12

- Effects of ageing

Temporary issues 4-1-13

- Broken bones
- Recovery

Crisis mode (Eric Meyer) 4-1-14

- Not thinking clearly
- Stressed

Planning for accessibility

Identify goals

- Get on the same page
- Set a clear target

Put someone in charge

- Everyone is responsible
- Ensures goals are met

Evaluate tools and frameworks

- Not accessible? Can't use it!

Research

- Best practices
- New tools

Create personas

- Keep accessibility in mind
- Commonality for discussion
- Create empathy

Why accessibility?

Accessibility is about making our software **usable** for:

- As many people as possible
- On as many devices as possible
- In as many circumstances as possible

It enables people with disabilities to **perceive, understand, navigate, interact with, and contribute** to the web.

This can include **people with different vision capabilities**: 4-1-23

- Color-blindness, low vision, etc
- Or just cannot look at the screen right now
- Accessibility is about us all

People with motor difficulties 4-1-24

- Needing larger selection areas
- Or needing assistive devices

People with hearing disabilities 4-1-25

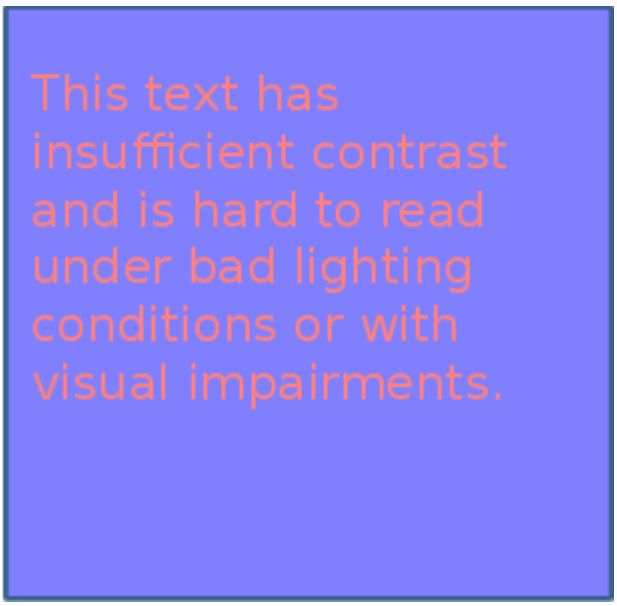
- Can't listen to video or audio on your website

Accessible design 4-1-26

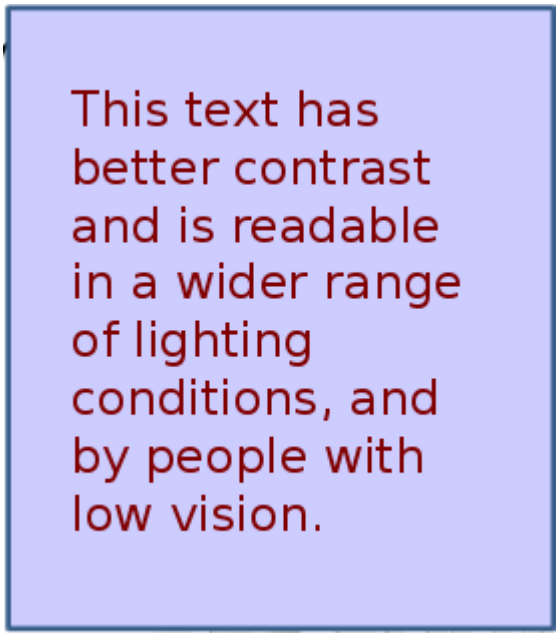
Designing for accessibility: color 4-1-27

Use moderate to high contrast for text

- Helps those with low vision
- Helps in bad lighting conditions too
- Choose a contrast ratio of at least 4.5



4-1-28



If using colors to indicate status add icons

- Helps those with color blindness
- Provides extra feedback for all users

Designing for accessibility: verbal feedback

Text controls are read by screen readers. Other controls require annotation.

- Like alt tags for images in HTML
- contentDescription
- labelFor, associates a text control with an input

Summary

Accessibility means:

- Better visibility in poor lighting
- Better use while performing other tasks
- Able to use without looking at screen
- Usable on wider range of devices

The same techniques that help people with disabilities out also produce more useful software even for the rest of us.

Responsive design and accessibility

4-2-1

Responsive design is an accessible design pattern

4-2-2

Other considerations

4-2-3

- Bright light makes it difficult to see
- Contrast can help make it easier to see

In a **busy environment**, you'll need an extremely **simple interface**

4-2-4

- Make it easy to understand what to do

Color and accessibility

4-2-5

Color blindness

4-2-6

- 1 in 12 men
- 1 in 200 women
- 5% of the population

How is color perceived by those wo are color blind?

4-2-7

Difficulty distinguishing between certain color combinations

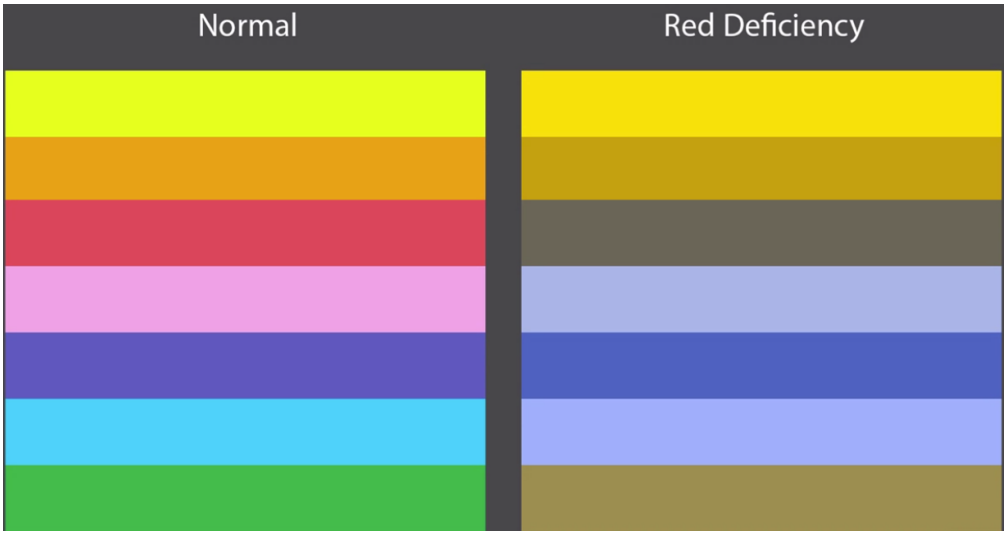
4-2-8

- The most common is red/green deficiency
- Makes it difficult to distinguish between red and green values, when they are of the same darkness

Red deficiencies

4-2-9

- Protanopia
- Protanomaly

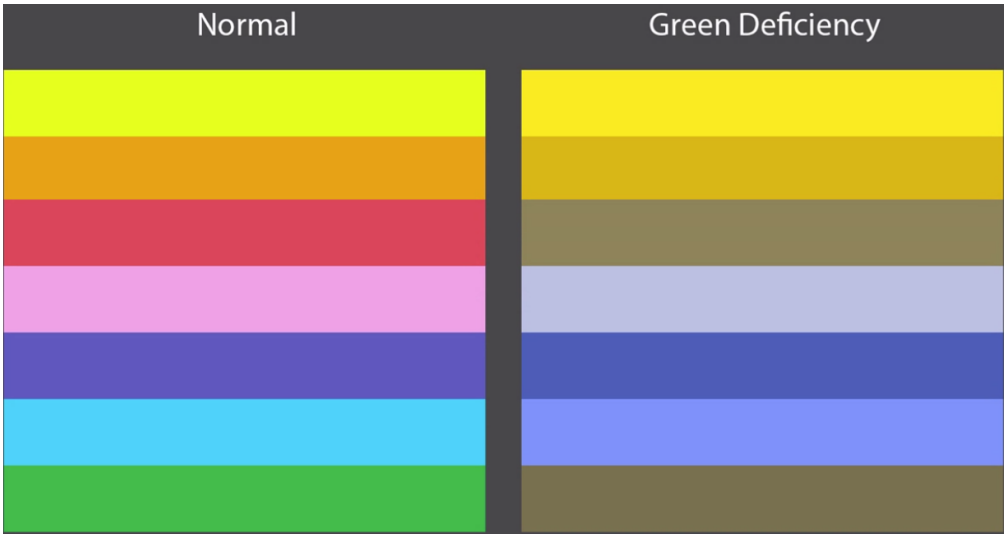


4-2-10

Green deficiencies

4-2-11

- Deuteranopia
- Deuteranomaly



4-2-12

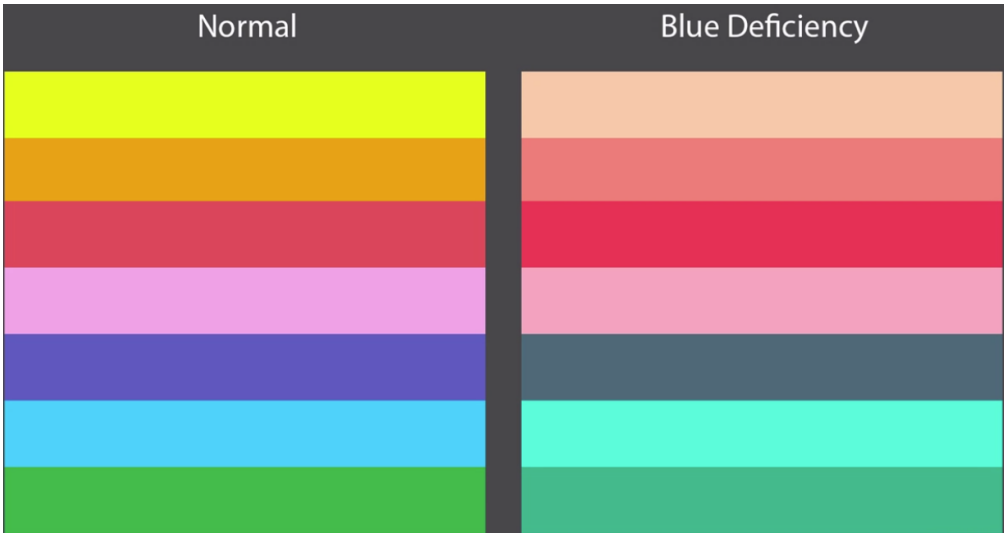
Similar to red deficiency, but the reds don't look as dark

4-2-13

Blue deficiencies

4-2-14

- Tritanopia



4-2-15

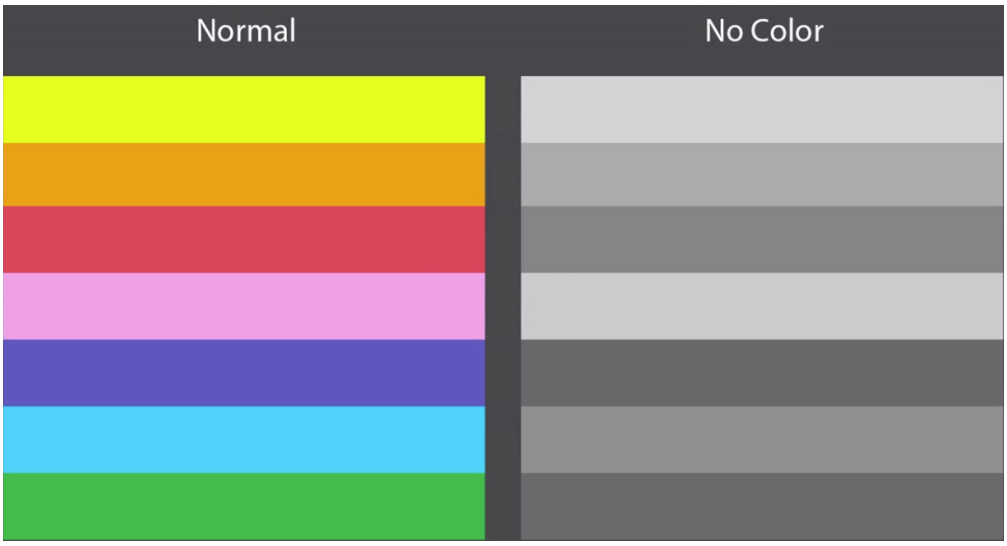
Must less common. Blue and green appear similar and yellow can disappear or possibly appear as red.

4-2-16

No color

4-2-17

- Rod monochromacy or Achromacy



4-2-18

Very rare. Can only differentiate between light and dark values and cannot see any color at all.

4-2-19

Don't rely **solely** on color. Icons may help convey functionality as well.

4-2-20

How color is displayed may vary between displays

4-2-21

- Some people may be viewing your site/app on a low-end mobile display
- Contrast helps
- Environment lighting can affect how color is displayed
- Again, contrast helps

Typography and accessibility

4-2-22

Font size

4-2-23

- Set a size where the characters are easily distinguishable without being obnoxiuosly large
- A healthy size is around 16px-20px

Although, it is better to use **relative units** like **em** and **rem**. These will **scale** better if the user has adjusted their **system font settings** for **larger text sizes**.

4-2-24

Contrast

4-2-25

- The contrast and size should be great enough that be extremely clear between heading levels and the body of text below it.
- Contrast between bold and normal text should be clear as well.

4-2-26

This is a heading, it should be larger

This is the content. Right now the content and the header are much too close in size to tell difference beteen the two

4-2-27

This is a heading, now it's much more clear

This is the content. Now the content and the header are much different in size, making it much easier to tell difference beteen the two

Line length

- In order to make content more readable, it is crucial to control line length
- This makes it easier for the reader to navigate from the beginning of the text to the end
- A good rule of thumb is to keep line lengths between 45 and 75 characters
- The ideal is a line length of 66 characters

Too Long

Far far away, behind the word mountains, far from the countries Vokalia and Consonantia, there live the blind texts. Separated they live in Bookmarksgrove right at the coast of the Semantics, a large language ocean. A small river named Duden flows by their place and supplies it with the necessary

Too Short

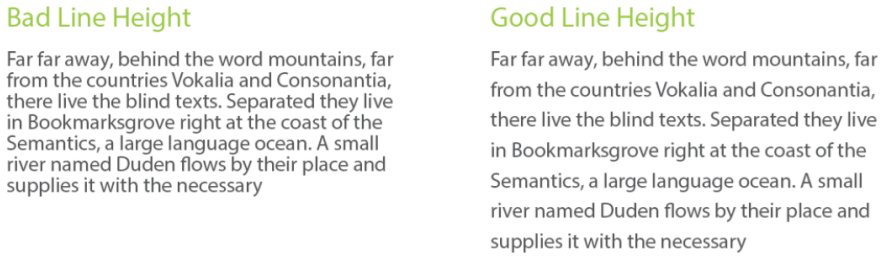
Far far away, behind the word mountains, far from the countries Vokalia and Consonantia, there live the blind texts. Separated they live in Bookmarksgrove right at the coast of the Semantics, a large language ocean. A small river named Duden flows by their place and supplies it with the necessary

Just Right

Far far away, behind the word mountains, far from the countries Vokalia and Consonantia, there live the blind texts. Separated they live in Bookmarksgrove right at the coast of the Semantics, a large language ocean. A small river named Duden flows by their place and supplies it with the necessary

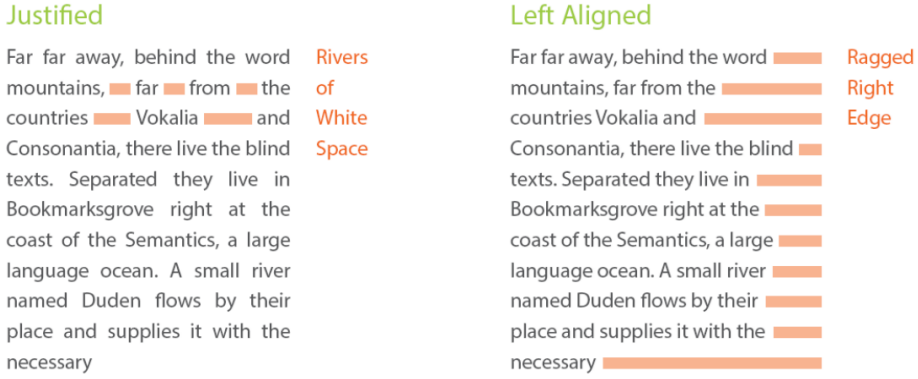
Line height

- A good rule of thumb is to set the line height to about 125% of the text size.
- This can however vary depending on the font face and text size
- The above are general suggestions



Text justification

- Should be avoided
- May cause "rivers of white space" throughout the words and paragraphs
- Makes it very difficult for those with learning disorders, like dyslexia, to read
- The "ragged right edge" of left adjusted text is easier to read



Take extra care with ALL CAPS

May be harder to read, and screen readers may **misread** them because they have trouble distinguishing between the characters

Accessible form design

4-2-38

Forms should be **clear, organized and easy to understand** how to engage

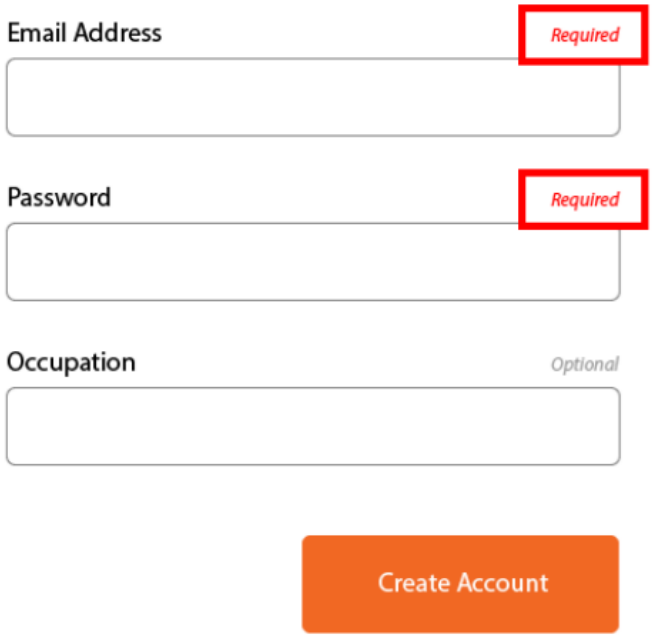
4-2-39

We need to clearly identify required fields

4-2-40

- Reduces the cognitive load
- Makes it easier for the user to scan for required information
- We should not solely rely on color, since the person accessing may have color blindness issues

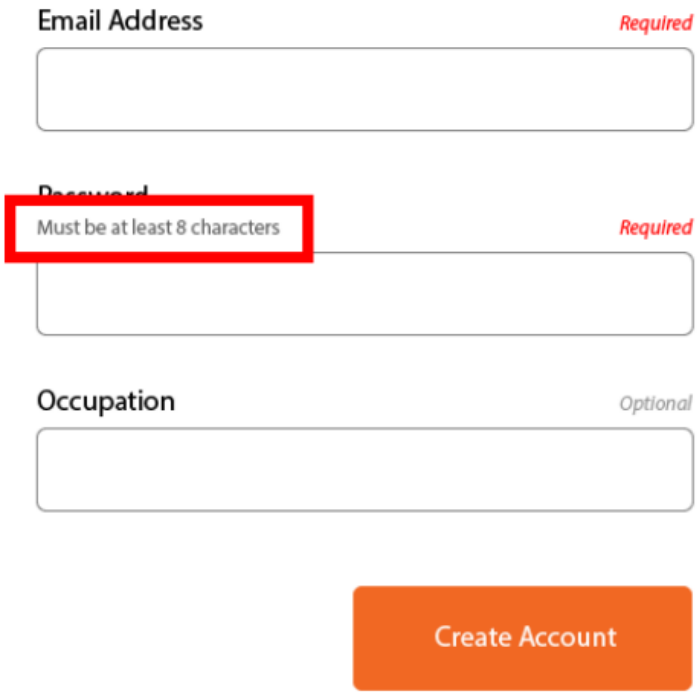
4-2-41



Identify special formatting requirements

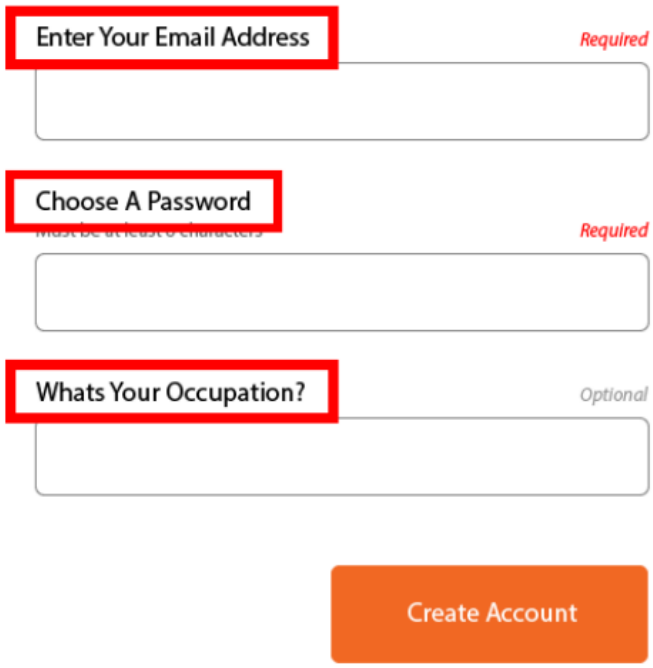
4-2-42

- Also reduces the cognitive load
- Prevents frustration by clearly outlining how the requested information should be provided



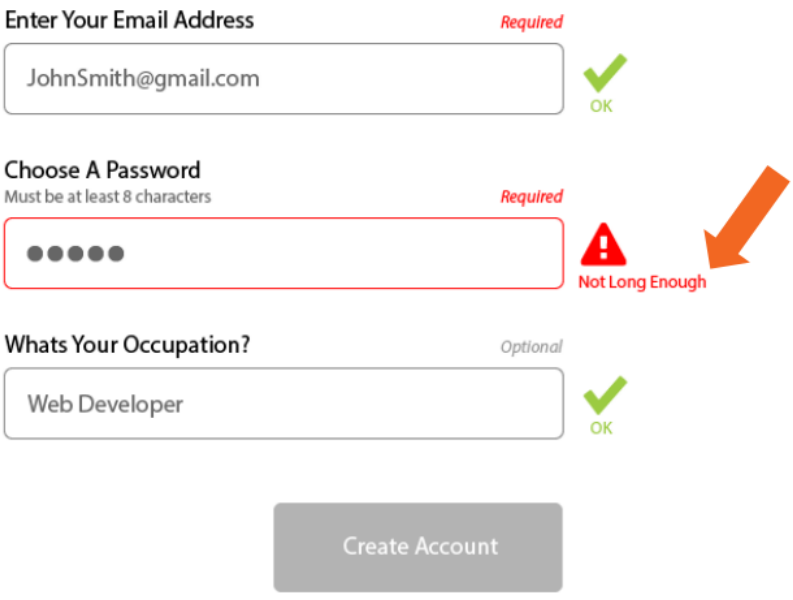
Add clear, descriptive form labels

- Every form field should have an associated label
- Place holder text is not a replacement for a label, since this disappear when the field is in focus and can make it difficult for the user to remember what they're supposed to be filling out



Provide clear feedback for errors and warnings

- Again, don't rely solely on color
- Add an error message also



Touch targets

4-2-48

Smaller touch targets require more accuracy

4-2-49

- Ensure that touch targets are larger than the finger tip
- Gives visual feedback to the user as they interact with the item
- Make it hard to make mistakes. Ensure that there is enough space between touch targets that the user will not accidentally select the wrong item
- User errors decrease as touch targets increase

Focus and active indicators

4-2-50

Some people with **mobile and dexterity disabilities** use **assistive technologies** that rely on the **keyboard** or **mimic keyboard functionality**. These user require **clear, onscreen indication**, in order to discern where they are on the page.

4-2-51

Focus/Active indicators

4-2-52

- Need to exist
- Need to be unobstructed
- Need contrast
- Do not remove with CSS or JavaScript
- Don't rely solely on color

Motion and vestibular disorders

4-2-53

Dizziness, vertigo, motion sickness and panic attacks are things most of us never think about

4-2-54

However, there are those who experience this **every day** while browsing the web

The effects can last for hours, and even days and can lead to things like **stomach aches and headaches**.

How can we prevent our users from experiencing this?

4-2-55

In most cases, this is related to motion

4-2-56

- Transforms
- Animation

Overall, the key is to keep it simple 4-2-57

- Every transition or animation should have a purpose
- Add in features to allow users to disable animations

Designing for crisis 4-2-58

Eric Meyer 4-2-59

- Influential part of the Web since the early 90's
- Has authored many books
- Presented at many conferences

While experiencing a **family crisis**, he discovered that many websites are designed with the **ideal user** in mind. 4-2-60

We should also design for **crisis driven personas**. The argument is that if someone in crisis can navigate our site or application then pretty much **everyone** will be able to as well. 4-2-61

We do this by 4-2-62

- Creating a diverse set of personas
- Provide information for different failure cases, this means providing detailed error messages when things go wrong, so users can figure out what to do next

Summary 4-2-63

- Accessibility concerns more than code 4-2-64
- Responsive design is a must
- Choose colors carefully
- Handle typography with care

- Forms should be clear, organized and easy to understand 4-2-65
- Touch targets should be easy to touch and provide visual feedback
- Keep it simple when using motion
- Design for those in crisis

- Proper document structure
- Use the correct HTML tags
- Don't use text in images
- Add alt descriptions to images in content

4-3-1

- Accessible from the keyboard
- Focus on performance

4-3-2

Proper document structure

4-3-3

A **proper document structure** should be written in a way, that if we **remove the CSS**, the page will still render in a **logical order** and with the **proper hierarchy**. It won't be pretty, but the content should be **organized** and the **hierarchy should remain clear**.

4-3-4

HTML headings, `<h1>` - `<h6>`, are a critical part of **accessibility**. They provide the **document outline** which provides **structure** and shows how sections of the page **relate** to one another. They also provide **targets** that some assistive technologies, like screen readers, can use to jump from section to section.

4-3-5

They should be used in the **proper order**.

4-3-6

`<h2>` headings should have a parent `<h1>` heading, `<h3>` headings should have a parent `<h2>`and so on.

No heading levels should be skipped.

4-3-7

```
<main role="content">
    <h1>I'm the Main Title</h1>
    <p>I'm the main description</p>
    <h2>I'm a Secondary Title</h2>
    <p>I'm a secondary description</p>
</main>
```

Use lists 4-3-8

`<ul>` Unordered

`<ol>` Ordered

`<dl>` Description

These are also critical to the **document hierarchy** 4-3-9

Unordered Lists 4-3-10

- Use when **no** order of sequence or importance is needed

```
<ul>
    <li>Milk</li>
    <li>Bread</li>
    <li>Lettuce</li>
</ul>
```

Ordered Lists 4-3-11

- Use when order of sequence or importance **is** needed

```
<ol>
    <li>First we need to do this</li>
    <li>Next we need to do this</li>
    <li>Lastly we need to do this</li>
</ol>
```

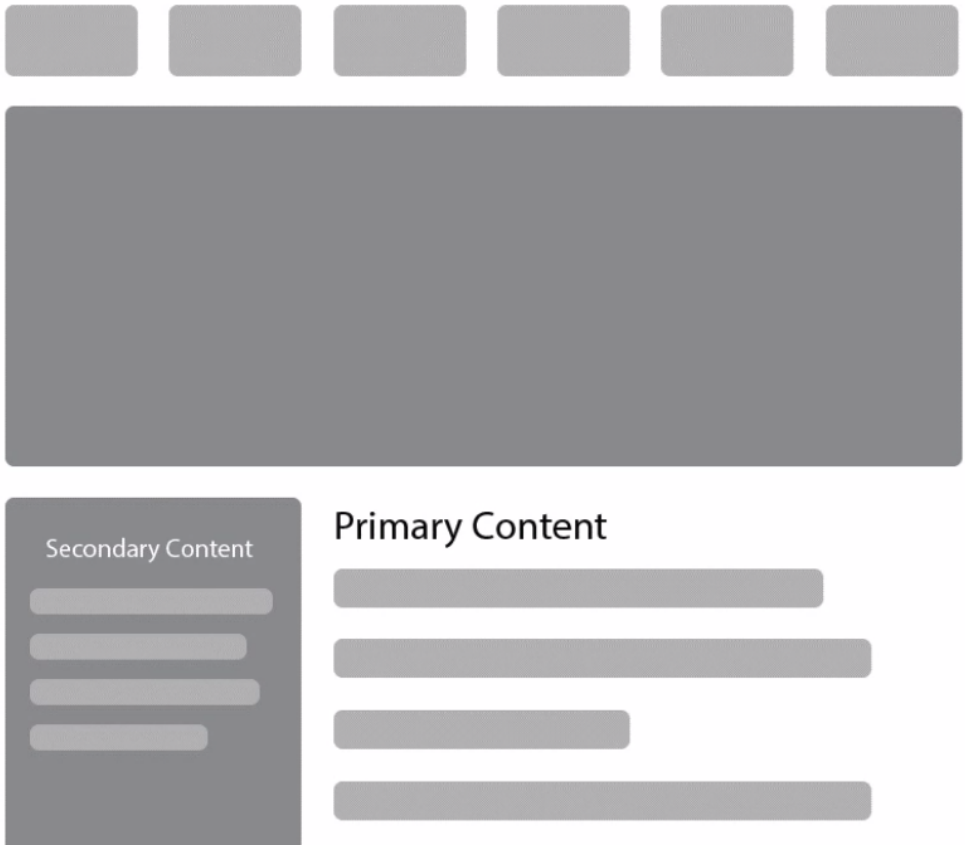
Description Lists 4-3-12

- Use when a description for each item is needed

```
<dl>
    <dt>Milk</dt>
    <dd>Fat free organic milk<dd>
    <dt>Bread</dt>
    <dd>Whole wheat sourdough</dd>
</dl>
```

We should write code that **facilitates keyboard navigation** in an easy and logical manner throughout the document. 4-3-13

For example, if you have a sidebar with secondary content on the left and your main content on the right: 4-3-14



You would still want your **main content** to come **before** your **secondary content**, in the **HTML**. This will ensure that users navigating your site with a keyboard will actually flow through in the **correct order**. 4-3-15

Images and Accessibility 4-3-16

We will primarily discuss images that are **inserted in content**. We are **not** concerned with images used for **decoration or look and feel**, because these images are either used as **CSS backgrounds** or do not require a description, since they only serve **aesthetic purposes**. 4-3-17

Images by default are **inaccessible** to those who cannot see them due to poor vision or blindness. 4-3-18

For this reason, we need to provide **descriptions** for anything that contains **informative content**. For those conveying simple information, a photograph for example, it can be appropriately described using an `alt` attribute. 4-3-19

Alternate text 4-3-20

- A succinct description of the content of the image

```
<img src="dog.jpg" alt="Brown Dog" >
```

For graphs or images containing a lot of information, the `longdesc` attribute can be used. 4-3-21

```
<img src="graph.jpg" longdesc="GraphDesc.txt" >
```

Long description 4-3-22

- Provides a link to the location of the long description content
- Can be a URL to a different page or can link to an ID on the current page
- However, browser support is almost non-existent

- It is instead recommended that you provide a link to the long description location 4-3-23

```
<img src="graph.jpg" aria-describedby="desc" alt="Graph" >
<a href="GraphDesc.html" id="desc">Get Graph Details</a>
```

To do this the right way, we need to use a specific **aria attribute**, `aria-describedby`. We will look more closely on ARIA in the next section. We use it to create an **association** between the image and the link to the long description. 4-3-24

We simply pass the `aria-describedby` attribute the `id` of the link. 4-3-25

If for some reason an image is added to the page and it **doesn't** require an explanation, if used for look or feel for example, we can give it an empty `alt` attribute, which will tell **screen readers** to ignore it. 4-3-26

No description needed? 4-3-27

- Add an empty `alt` attribute
- This tells screen readers to ignore it

```
<img src="image.jpg" alt="" >
```

Content and Accessibility

4-3-28

Most likely, it is our **content** that the user is visiting our site for.

4-3-29

It doesn't matter

4-3-30

- What device they are using
- If they have permanent or temporary disabilities
- If they are using assistive technologies

- If they are losing abilities due to aging
- If they vestibular or reading disorders

4-3-31

They all came for your content, and they should be able to get it

4-3-32

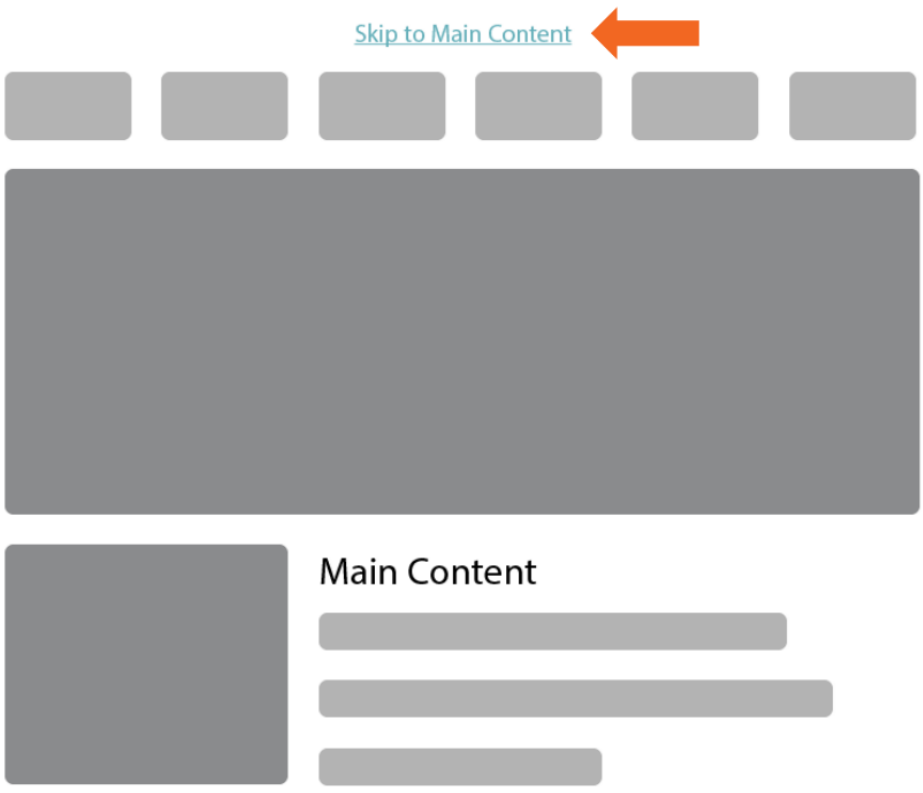
We can make our content more accessible by thinking of a few things

4-3-33

We should provide a **Skip navigation link**, that allows the user to jump right to the content when navigating with the keyboard or using assistive technologies that mimic keyboard functionality.

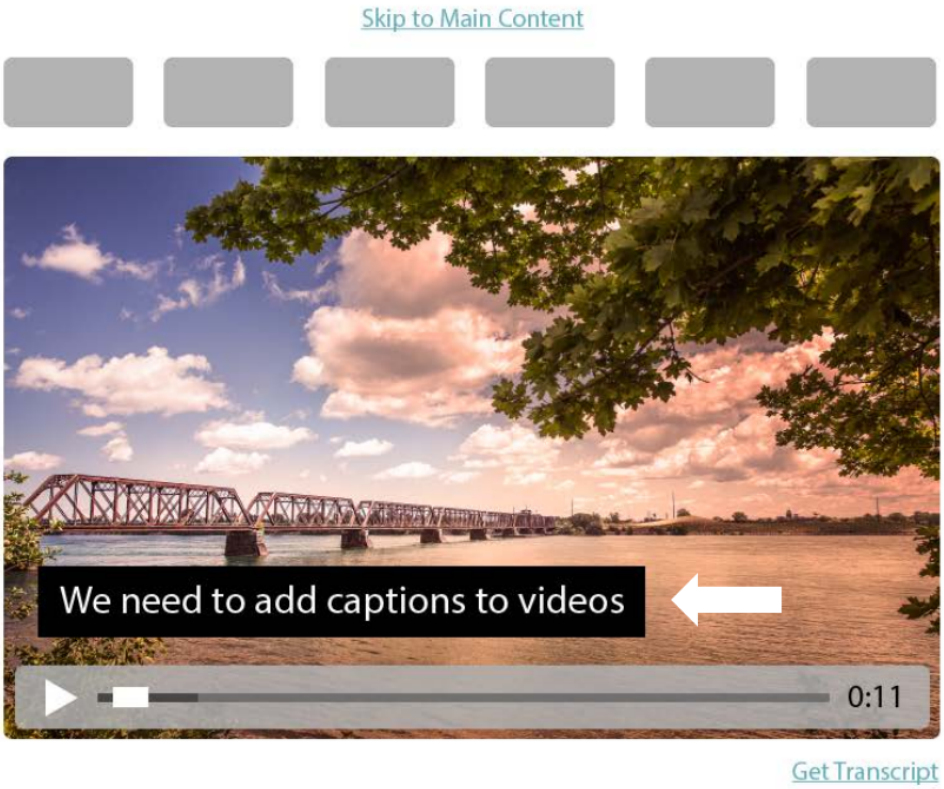
4-3-34

4-3-35



When providing **audio content**, it is important that we provide **transcripts** for those with hearing issues.

And when providing video content, we should provide **transcripts as well as captions** for those with vision and hearing issues.



We should also avoid making text as part of images where possible. 4-3-40

A fairly easy and common mistake to make is to add link text that **doesn't make sense out of context**. 4-3-41

The example most often used is the **Read more** link, when displaying truncated descriptions of a longer chunk of content. Many developers will simply put the text "Read more", directly within the link. 4-3-42

```
<a class=“about-link” href=“about.html”>
Read More
</a>
```

The problem here is that those **navigating with a screen reader** will hit these links and hear "Read more, Read more, Read more". Which is not very helpful. No **further information** about the section is available. 4-3-43

It is a better idea to use more **descriptive text**, like "**Read more about me**" or "**Read more about our products**". 4-3-44

Then, if it's critical to the visual design that it's a just "Read more", we can use **CSS** to hide this detail text and add in the "Read more" text with CSS generated content. 4-3-45

4-3-46

```
<a class=“about-link” href=“about.html”>
Read More About Me
</a>
```

```
.about-link:after {
content: “Read More”;
}
```

HTML Attributes and Accessibility 4-3-47

We should always provide the proper language attributes. 4-3-48

```
<html lang="en">
```

Language attributes

4-3-49

- Let the browser or assistive technology know what language the site is using

If you have a quote in french, you would set the `lang` attribute on that particular section:

4-3-50

```
<blockquote lang="fr">
    Si six scies scient six cyprés, six cents scies scient
    six cent cyprés.
</blockquote>
```

Just as a refresher, we need to provide **alternate text on images** so that those who cannot see them can still **understand** them.

4-3-51

```
<img src="dog.jpg" alt="Brown Dog" >
```

When adding forms we need to do a couple of things

4-3-52

We need to provide the `for` attribute to **associate labels** with the `id` of their respective **form fields**.

4-3-53

```
<label for="firstName">
    First Name
</label>
<input type="text" id="firstName" >
```

For Attribute

4-3-54

- Connect labels with their associated form fields

We will also want to add the `required` attribute for **required fields**.

4-3-55

```
<label for="firstName">
    First Name
</label>
<input type="text" id="firstName" required >
```

Required Attribute

4-3-56

- Provide more information for required fields

Tab indexes are **crucial** for those who use a keyboard to navigate, but we should only add it when needed. This would occur if we would have built some sort of functionality using **non-standard controls**. 4-3-57

Like if we built our own custom button using a generic element like a span for whatever reason. 4-3-58

```
<span class="button" tabindex="5">
    Submit
</span>
```

If we do this then we would need to add a `tabindex` attribute on it to insert it as an item that could be **tabbed to and triggered from the keyboard**. 4-3-59

Tabindex Attribute 4-3-60

- Forces non-standard controls into the tab order for keyboard navigation

Web Forms and Accessibility 4-3-61

In order to make **forms accessible**, we need to make them **simple**. We need to **structure** them in a **logical order**, grouping logical sections of fields together when it makes sense, with **fieldsets** and **legends**. 4-3-62

A form section that makes sense for this would be a billing address block. 4-3-63

```
<form>
    <fieldset>
        <legend>Billing Address</legend>
        <label for="billingAddress">Address</label>
        <input type="text" id="billingAddress">
        <label for="billingCity">City</label>
        <input type="text" id="billingCity">
</fieldset>
</form>
```

Billing Address

Address

City

Fieldsets and Legends

4-3-64

- Use them to logically group form fields

This provides a nice **visual que** to the user who can see while providing **more information** for screen readers and other assistive technologies.

4-3-65

We need to make sure that **form inputs** like **text boxes**, **textareas**, **checkboxes**, and others, have **associated labels**, and that they are **structured in a logical order** as well.

4-3-66

To do this, we use the `for` attribute with the `id` attribute of the field it is associated with.

4-3-67

4-3-68

```
<form>
    <fieldset>
        <legend>Contact us</legend>
        <label for="name">Your name</label>
        <input type="text" id="name">
        <label for="email">Your email</label>
        <input type="email" id="email">
        <label for="message">Your message</label>
        <textarea id="message"></textarea>
</fieldset>
</form>
```

Labels

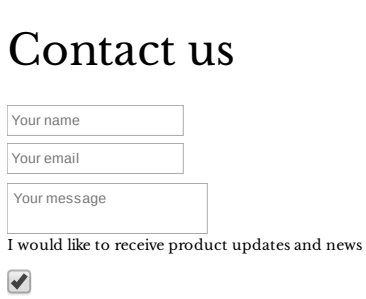
4-3-69

- Every field should have an associated label using the `for` attribute.

We need to make sure that we are **not** using placeholder text as a replacement for a label.

4-3-70

```
<form>
    <fieldset>
        <legend>Contact us</legend>
        <input type="text" id="name" placeholder="Your name">
        <input type="text" id="email" placeholder="Your email">
        <textarea id="message" placeholder="Your message"></textarea>
        <label>I would like to receive product updates and news</label>
        <input type="checkbox" id="updatesAndNews" checked>
</fieldset>
</form>
```



Placeholder text is not even widely supported across assistive technologies and therefore will not be reliable.

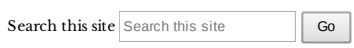
Placeholder text

- **SHOULD NOT** be used as a replacement for labels
- **SHOULD** be used to provide hints and tips

Hidden labels are ok sometimes, where it makes sense, but the label must still **exist as visually hidden text in the code**.

An example of where we might run into this sort of hidden label scenario would be in a **search box**, or something, where it's **visibly clear** what the form does.


```
<form>
    <label for="search" class="hidden">Search this site</label>
    <input type="text" id="search" placeholder="Search this site">
    <input type="button" value="Go">
</form>
```



So a label is **not** needed for those who can **see**. But the **literal** one is needed for those **who cannot and who are using a screen reader**.

Hidden labels

4-3-78

- It's ok to visually hide labels where it makes sense to do so

CSS used to visually hide the label

4-3-79

```
/* Visually hide content */
.hidden {
    position: absolute;
    left: -9999px;
}
```

We also need to be sure to add the **required** attribute to required fields along with a required field **identifier**, to make it clear that the fields is required at a glance.

4-3-80

Also, add **email**, **url**, **number**, **range**, **date** or **time** attributes to all form inputs of each type. This will provide more information about the types to screen readers and other assistive technologies.

4-3-81

Required

4-3-82

- This attribute should be added to all required fields.

We need to provide clear **instructions** for more complex fields that cam easily be filled out **incorrectly**.

4-3-83

4-3-84

```
<form>
    <label for="email">Enter your email address</label>
    <input type="email" id="email" required>
    <label for="password">Choose a password</label>
    <em>Must be at least 8 characters long</em>
    <input type="password" id="password" required>
</form>
```

Enter your email address

Choose a password

Must be at least 8 characters long

We mentioned **tab order** earlier in this section. The fields in a form are a part of that tab order. We need to make sure that they **follow the correct order** when tabbing through it. If we have the proper **markup structure** this will require very little attention.

4-3-85

Accessible Keyboard Control

4-3-86

Keyboard accessibility is one of the most **neglected web accessibility** areas, and it really is one of the easiest to test for. Simply stop using your mouse and see if you can navigate through your site.

4-3-87

It's a good idea to begin by considering who **can't** use a mouse.

4-3-88

Who doesn't use a mouse, or touch a screen?

4-3-89

- Blind people
- People with fine motor control issues

- People with little or no use of their hands
- People that don't have hands
- People that find it easier

4-3-90

Those of us that **cannot use a mouse** tend to either navigate the web using **only** the keyboard or using an assistive technology that mimics the behaviour of the keyboard. There are a few common problem areas to watch out for.

4-3-91

Focus and Active Indicators

4-3-92

- Help people see where they are

Browsers handle this by default, adding an **outline** to the focused element. Some developers will remove this because they don't like the way that it looks. Others may not realise that by using certain CSS properties they **remove this functionality**.

4-3-93

4-3-94

Using `display: none` for example will **disable** the default outline added by the browser. When no focus indicator exists, these elements are almost **completely inaccessible** to keyboard users. 4-3-95

Generally, you should either **avoid** using `display: none` on links and form controls or add your own visual indicator with focus styles. 4-3-96

4-3-97

```
input:focus {
    outline: solid 1px #98c741;
    box-shadow: 0 0 5px 0 rgba(152, 199, 65, 1);
}
```

Then there's the **navigation order**, or **tab order**. The flow of keyboard navigation should be **logical** and easy to follow. It should primarily follow the **visual flow of information**. 4-3-98

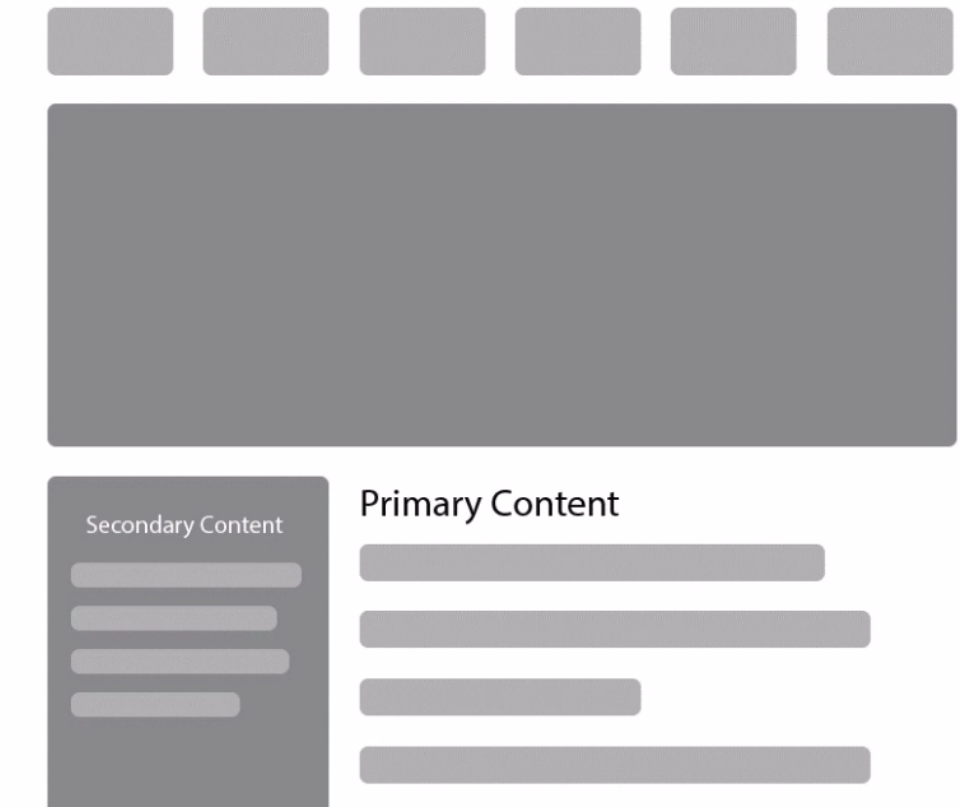
LTR for languages that read left to right 4-3-99

RTL for languages that read right to left 4-3-100

And then top to bottom 4-3-101

This is controlled primarily by the way that we've **structured our markup**.

4-3-102



4-3-103

```
<body>
    <main>
        <section>
            <header>Primary Content</header>
        </section>
        <aside>
            <header>Secondary Content</header>
        </aside>
    </main>
</body>
```

Main Content Comes First!

4-3-104

- Even though the sidebar looks like it comes first, the main content should come before the sidebar in the code.

These same concepts apply to **forms**. We should **flow through the form in a logical order**. And for the most part our markup should reflect that order.

4-3-105

Non-Navigable Items

4-3-106

Sometimes the developers will forego **standard HTML elements** like **links and buttons**, and instead **add** JavaScript functionality using things like `onclick` to **generic elements** like **divs or spans**. 4-3-107

When doing this, these items become **completely inaccessible** via the **keyboard** as it will **skip right over them**. 4-3-108

We should use the **standard HTML elements** like **links and buttons** wherever possible, but when it's not possible this scenario can be remedied by **adding** a `tabindex` attribute. 4-3-109

4-3-110

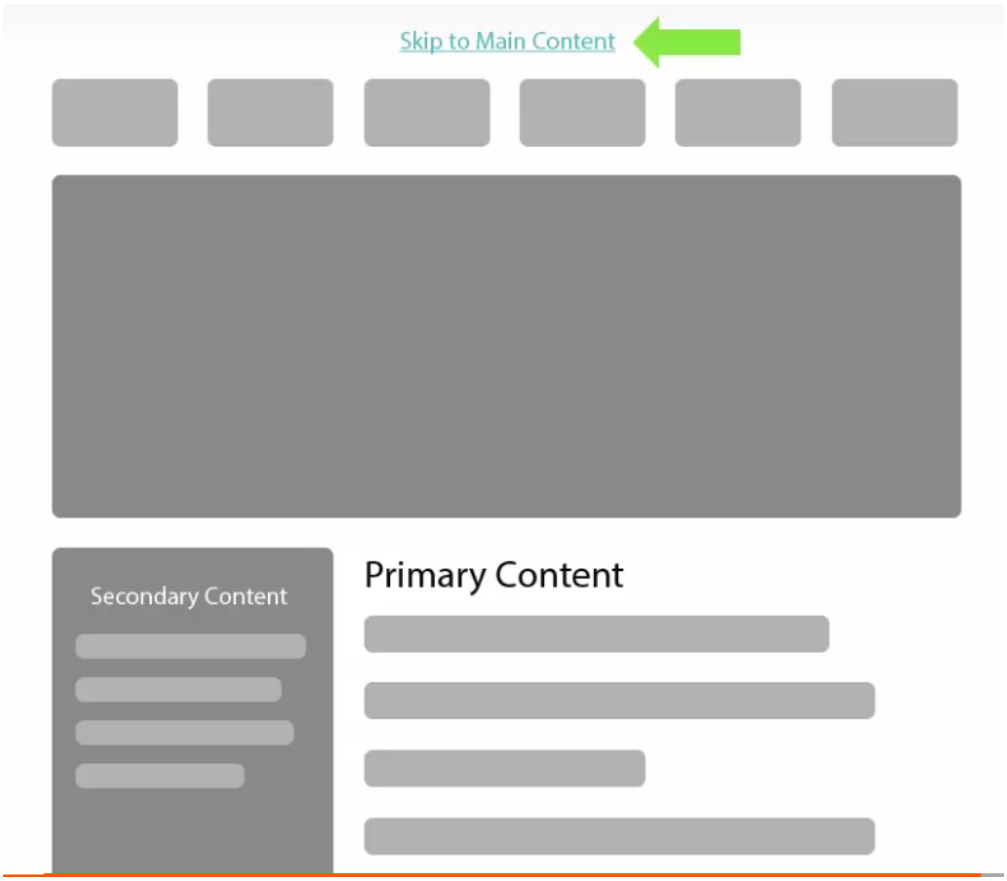
```
<span tabindex="0" onclick="document.getElementById('form-id').submit();">
    Submit
</span>
```

Non-Navigable Items 4-3-111

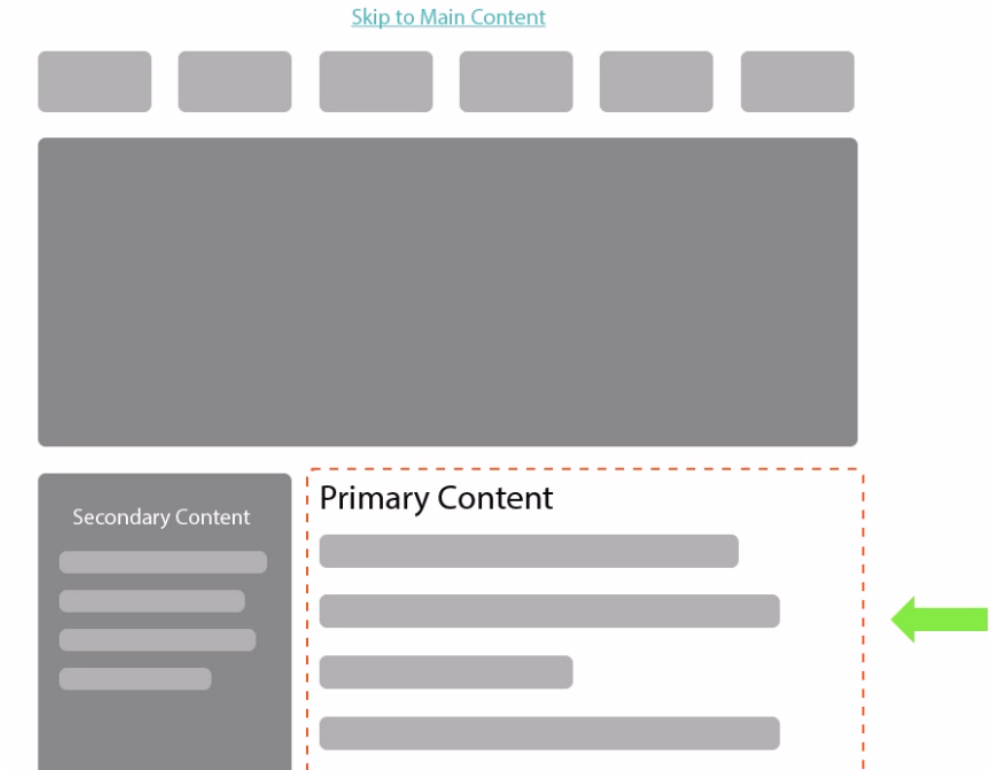
- Adding a `tabindex` value of zero will insert the element into the default navigation order.

We need to allow the **skipping of long navigation lists**. Keyboard navigation steps through all of the **links and form elements** on a page. This can be really painful when the main navigation menu has many links and submenus within it. 4-3-112

It's generally a good idea when this occurs to add a "**Skip to Main Content**" link before the menu. 4-3-113



4-3-114



This allows people to **bypass** the navigation and jump right into what they're interested in. 4-3-115

To do this, just **add a link above your menu** as one of the **first** items in the page. The target of this link will be the `id` of the section that you are **navigating** to. 4-3-116

4-3-117

```
<head>
    <title>Skip Link Example</title>
</head>
<body>
    <a href="#primary-content">Skip to Main Content</a>
    <header>
        <h1>Skip Link Example</h1>
    </header>

    <main>
        <section id="primary-content">
            This is my main content
        </section>
    </main>
</body>
```

Skip Navigation Link 4-3-118

- Make it one of the first elements in the document
- Use the `id` of the main content in the `href` of the link

It's recommended that it be **visually apparent**, meaning that it should **not** be small and hidden, as it may be missed by keyboard users. Many designers and developers may not like this because it can mess with the aesthetics of the site. 4-3-119

So another approach is to make the element **visually hidden**, a technique we'll discuss in the next section, and then make it visible on focus. 4-3-120

CSS and Accessibility 4-3-121

CSS Cascade Overview 4-3-122

- Default browser styles
- Our external CSS
- Style sheets embedded in the page
- Inline styles

Q But what about when a user changes their own accessibility settings to improve the overall experience specifically for themselves? 4-3-123

A Those styles will override everything else. The individual has the final say on how our pages will look. 4-3-124

CSS Cascade Overview 4-3-125

- Default browser styles
- Our external CSS
- Style sheets embedded in the page
- Inline styles
- **End user settings adjustments**

Many people with vision issues will **increase the text size settings** so that it's easy for them to read. Plenty of us alter our **color settings** and change them into **high contrast mode** so that we can more easily read and consume content. 4-3-126

Maybe it's because we have **color deficiencies**. Maybe not. Those with **reading disorders** may even change things like **fonts** to provide a better reading experience, tailored for their own needs. 4-3-127

The bottom line here, is that the **end user is ultimately the one in control of how our sites and applications will look for them**. So we need to build things with **flexibility** in mind and as we build out our interfaces should consider a few questions. 4-3-128

What happens if we increase font size? 4-3-129

If you provide **explicit height and width pixel values** for something, then it's likely to break if the user increases their system font size. 4-3-130

Another question would be what would happen if we change the font? 4-3-131

Again, if we have explicit pixel values used, it's likely things are going to break when this happens. 4-3-132

It's best for us to use **relative units**, like `em` and `rem`, as opposed to **pixel-based values** as they allow items to shrink and grow more flexibly. 4-3-133

So that when font size increases, the containing element will increase along with it. 4-3-134

Same thing if we change the font. 4-3-135

This is also a **key component of responsive web design**, so they go hand in hand. 4-3-136

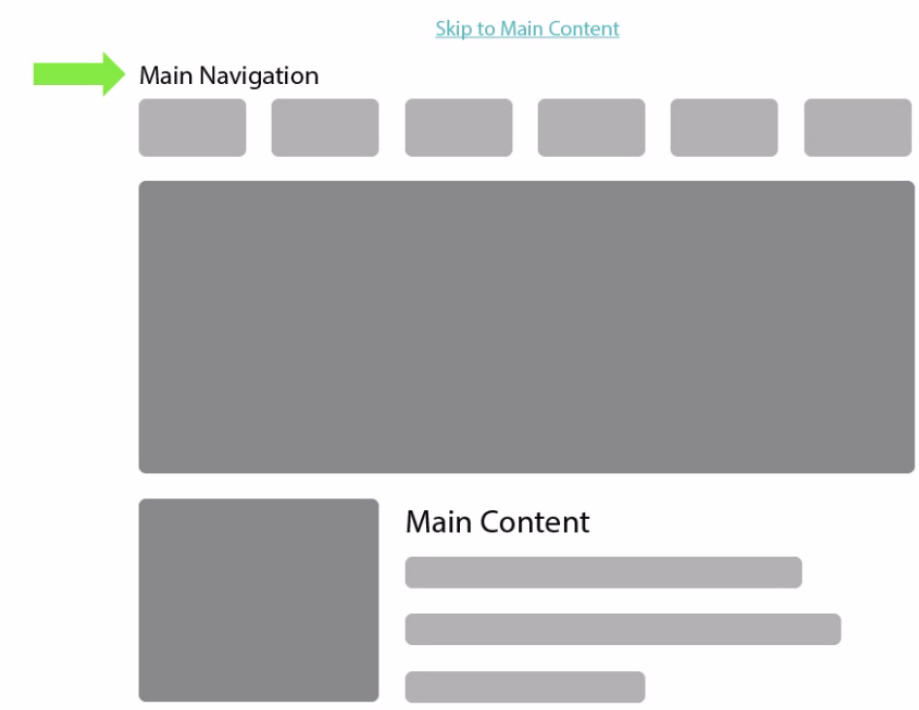
Many times when end users alter their size settings, they will actually trigger **media query breakpoints** too, so if we've built proper responsive sites, many things will just work, and will accommodate the user correctly. 4-3-137

CSS allows us the **flexibility to control the layout** and the way that things render in the end, but our underlying content needs to be **structured in a logical order** where it makes sense, without CSS. Because many assistive technologies will pretty much ignore it. 4-3-138

We **cannot rely solely on CSS alone to present meaning**. Instead we should use the proper **semantic elements** and then just layer on styling to make them look better for **all users**. 4-3-139

Sometimes we need to **hide** content that would'nt make sense or is not necessary visually but would make sense and is necessary for **assistive technologies** like **screen readers**. 4-3-140

For example, adding an `h2` with the text "**Main Navigation**" is very clear to a person who is having the content on the page **read** to them. 4-3-141



But for people with **no** vision impairments, this is just extra stuff that can actually get in the way. 4-3-142

So, we can hide this content but there a few things to consider in doing so.

4-3-143

```
<body>
    <header>
        <a href="#content">Skip to Main Content</a>
    </header>
    <nav>
        <h2 class="hidden">Main Navigation</h2>
        <ul>
            <li>Home</li>
            <li>About</li>
        </ul>
    </nav>
</body>
```

4-3-144

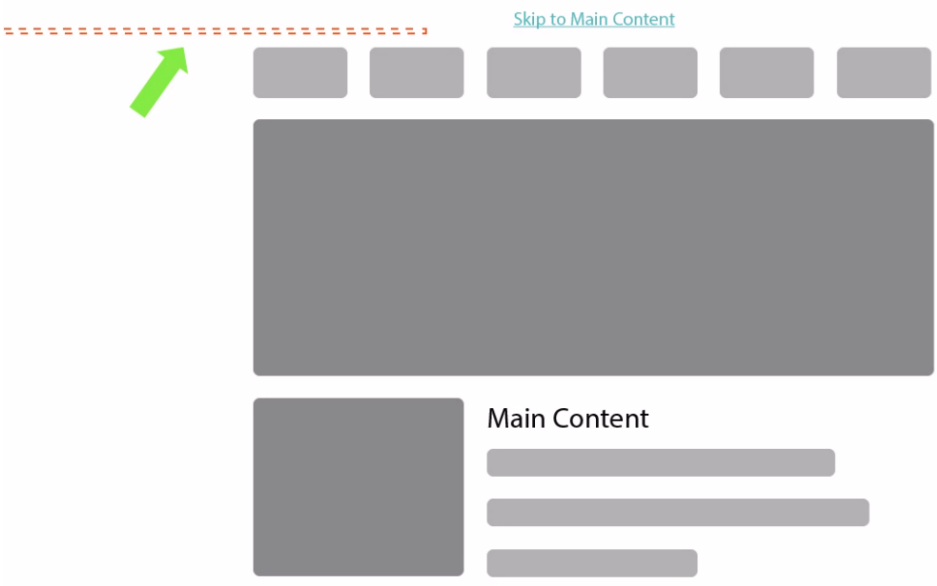
```
.hidden {
    height: 0;
    width: 0;
    display: none;
    visibility: hidden;
}
```

If using `visibility: hidden`, `display: none` or `height: 0, width: 0` we will actually prevent assistive technologies like screen readers from reading the content. This is a **big** no no.

4-3-145

```
.hidden {
    text-indent: -9999px;
}
```

If using the common **hidden text technique** `text-indent: -9999px;`, so that the text moves of the page, a screen reader will read it but some browsers will add a **focus indicator** that wraps the hidden text as well, providing a long rectangular box which may look pretty bad.



4-3-146

One of the best ways to create the visually hidden text is to use **absolute positioning** to position the text of the page and **outside** of the viewport.

4-3-147

```
.hidden {
    position: absolute;
    left: -9999px;
    top: auto;
    width: 1px;
    height: 1px;
    overflow: hidden;
}
```

We should note however that it's best to position of to the **left of the page** because when positioning a link or form control on the top of the page you could **force** the browser to **scroll** to the top of the page when that link becomes focused.

4-3-148

We can also use the CSS `clip` property. It's been gaining steam more recently and is a nice way to provide visually hidden text.

4-3-149

```
.hidden {
    position: absolute;
    clip: 1px, 1px, 1px, 1px;
}
```

The main argument for this method is that there have been some **performance concerns** around other methods that position hidden text of the page based on the way that browsers render content on the screen.

4-3-150

The `clip` method does **not** need to position anything of the page, so it avoids these scenarios.

4-3-151

Hiding content should be used sparingly. In most cases text should be appropriate for users with **and** without vision impairments. 4-3-152

CSS can also be used to establish or enhance **focus and active states** for keyboard users that are not vision impaired. 4-3-153

Using `active` and `focus` pseudo-classes we can customize the way that focus is applied to elements and make them consistent **across different browsers**. 4-3-154

```
:focus {
    outline: dotted 2px #f06924;
}
:active {
    outline: dotted 2px #39b54a;
    transform: scale(0.8);
}
```

Performance and Accessibility 4-3-155

A **slow** internet connection can make it much more difficult for people to **access** the web. Having a site with a ton of resources and bloated files can make it much worse for them. 4-3-156

We can make it much easier for them by **focusing on performance** in our development process. 4-3-157

There are some **key ideas and concepts** to focus on 4-3-158

- Keep files sizes small
- Limit HTTP requests
- Optimize images

We also want to take advantage of SVG graphics where possible, and where it makes sense. They work well with **responsive design** because they are scalable. 4-3-159

SVG Benefits 4-3-160

- Relatively small file sizes
- Can be embedded directly into HTML

- Can be styled with CSS
- Greater flexibility
- Scalable without loss in quality

4-3-161

Progressive Enhancement and Accessibility

4-3-162

"Not assuming with which technology a site is consumed is a key component in accessible web development, as assistive technology is so diverse."

4-3-163

```
- W3C
```

We tend to design for the **software and tools** we as designers and developers use, which tends to be on the **higher** end of the spectrum.

4-3-164

Without Progressive Enhancement

4-3-165

- Build for high-end devices
- Add in fallbacks to fill gaps

With Progressive Enhancement

4-3-166

- Start with the lowest common denominator
- Well-structured HTML
- Add in well-supported CSS
- Add in more advanced CSS
- Add in JavaScript to enhance the experience

If we build with **progressive enhancement** in mind, the process will ensure that our content will remain **accessible** to the **largest number of people**.

4-3-167

Summary

4-3-168

- Proper document structure makes sense without CSS
- Images are inaccessible by default and need descriptions
- The content is the most important piece and it should be accessible to everyone

4-3-169

- Various HTML attributes enhance accessibility
- Forms should be simple, logical, navigable, and well labeled
- Sites and applications should be navigable via the keyboard

4-3-170

- Certain CSS properties affect accessibility
- Performance enhancements help accessibility
- Progressive enhancements help accessibility

4-3-171

4-4. ARIA Roles and Attributes

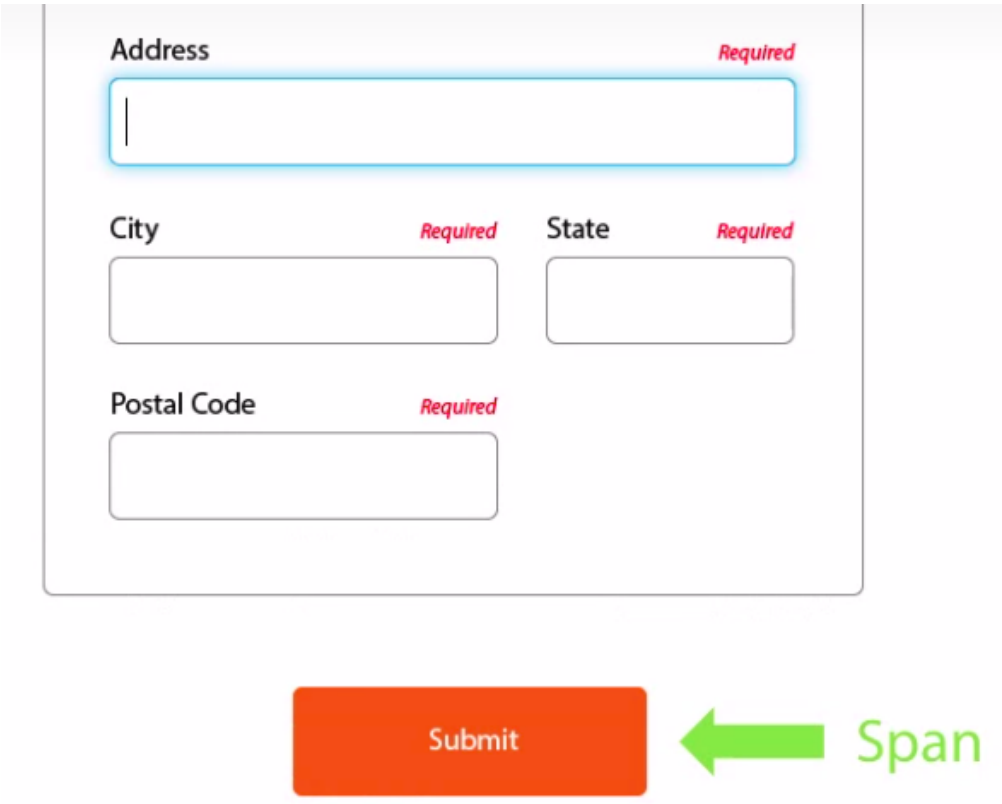
ARIA is short for **Accessible Rich Internet Applications**.

4-4-1

- A specification developed by the **W3C (World Wide Web Consortium)**.
- Helps bring accessibility to sites and apps using **AJAX**, **HTML**, and **JavaScript**.

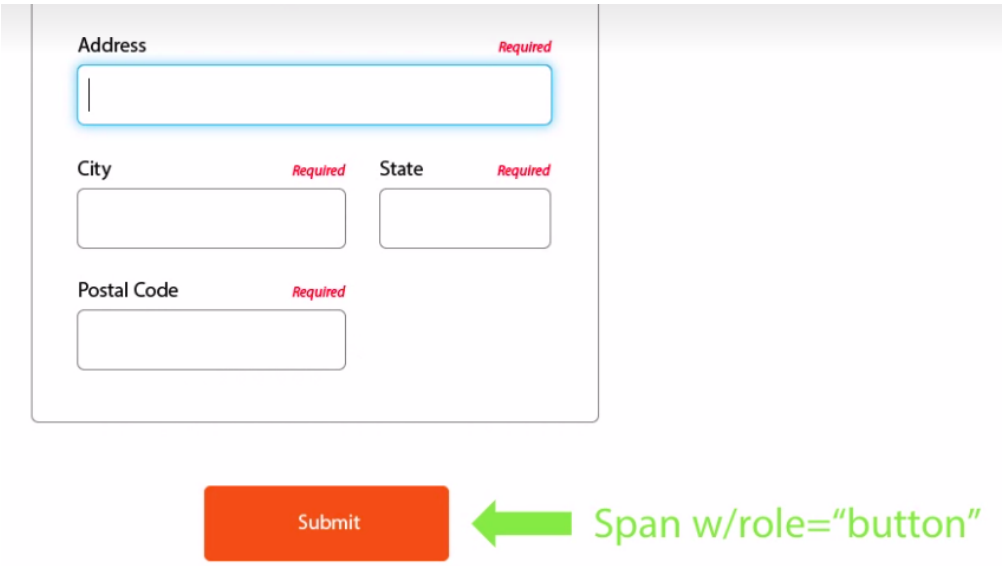
Without ARIA, assistive technologies interpret the elements in a page as **content**, not **functionality**, providing **no additional feedback** to the user, rendering them completely **inaccessible**.

4-4-2



4-4-3

With ARIA, these items become **widgets**, that can be interpreted by assistive technologies. This allows the user to **interact with them**. 4-4-4



4-4-5

We can add **ARIA attributes** that can provide more **accessibility and information**, and insert these items into the **tab order**, making them **navigable via the keyboard**. 4-4-6

It's important to note that it's a best practice to use **default HTML elements if possible**. 4-4-7

ARIA Categories 4-4-8

- Roles
- States and properties

ARIA Roles 4-4-9

Define what a widget is or does

- Menus
- Sliders
- Progress meters
- Modals
- Etc.

ARIA Roles *4-4-10*

Describe structure

- Headings
- Regions
- Tables

ARIA States and Properties *4-4-11*

Describes the current interaction state of a widget

- Checked
- Disabled
- Selected
- Hidden

ARIA States and Properties *4-4-12*

Properties are about the characteristics of a widget

- Drag and Drop
- Live Regions
- Modals

Use existing semantic HTML elements if possible! *4-4-13*

ARIA Roles *4-4-14*

ARIA roles are used to define what a widget is or what it does. These roles come in a few different types. *4-4-15*

Landmark Roles *4-4-16*

- Help navigate content
- Used to describe common regions
- Added with `role` attributes

`role="application"` *4-4-17*

Used if the region is considered a **web application**, and not a **web document**. Tells assistive technologies to switch to **application browsing mode**.

An application consists of many widgets, and when in **application browsing mode**, the user will navigate from widget to widget instead of links and form controls. *4-4-18*

`role="banner"` *4-4-19*

Generally the **header** of the site or application. Will likely contain the **name** of the site and the **title** of the page.

`role="navigation"` *4-4-20*

Contains navigation for the site or application

`role="main"` *4-4-21*

Contains main content

`role="search"` *4-4-22*

Added to the search form for the site or application

`role="complimentary"` *4-4-23*

Contains content that is complimentary to the main content for the document

`role="form"` *4-4-24*

Add to anything that contains a collection of fields that makes up a form

`role="contentinfo"` *4-4-25*

Generally found in the footer

Provides information about the parent document

Widget Roles

4-4-26

These are used to describe interactive elements that currently have no HTML counterpart

- `alert`
- `log`
- `menuitem`
- `timer`
- Etc.

More can be found at https://www.w3.org/TR/wai-aria/roles#widget_roles

ARIA State and Properties

4-4-27

ARIA states and properties are normally referred to as **ARIA attributes**.

4-4-28

ARIA States

4-4-29

- Describe dynamic stated
- Changed with JavaScript
- Prefixed with `aria-`

`aria-busy`

4-4-30

`aria-disabled`

`aria-hidden`

ARIA Properties

4-4-31

- Describe relationships between elements
- Rarely change

`aria-describedby`

4-4-32

`aria-haspopup`

`aria-label`

ARIA and Keyboard Control

4-4-33

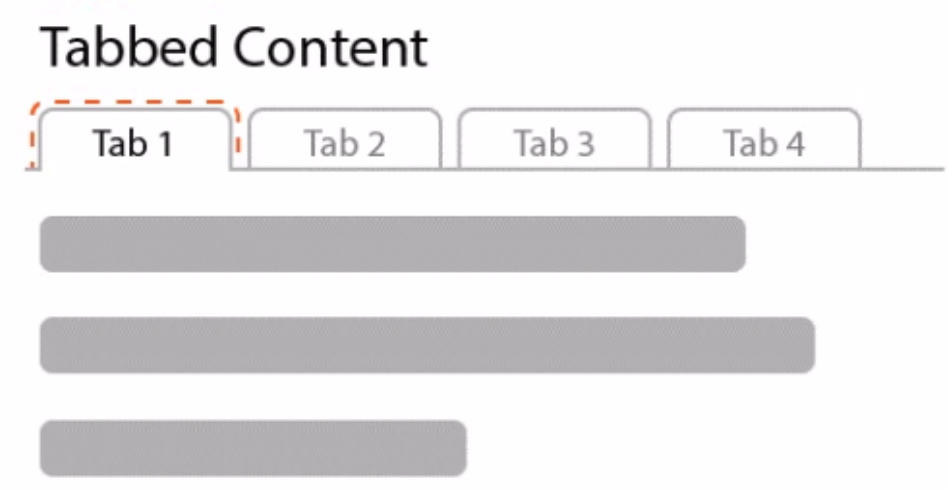
Many assistive technologies use keyboard like functionality to interact with the web 4-4-34

But developers often forget, or are unaware of this when building sites and applications 4-4-35

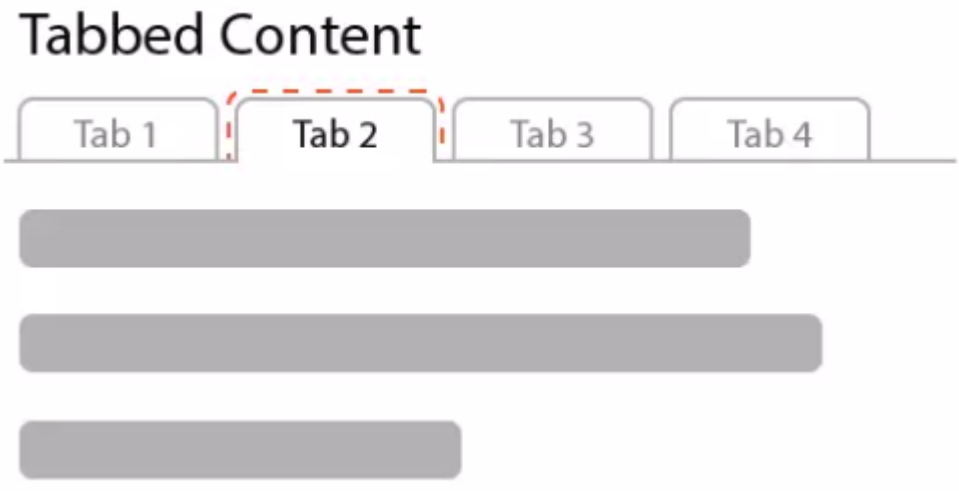
Generally, keyboard navigation on the web uses **tab** to navigate controls, and **shift** + **tab** to navigate backwards. 4-4-36

This can make the process slow and difficult for users navigating with a keyboard. The **tab** key should focus **individual widgets**. 4-4-37

For example, tabbing into a **tab set** would focus the first tab 4-4-38



Then the arrow keys would navigate **between** the tabs 4-4-39



For **forms**, the **spacebar** should activate the control. Then the **enter** key should **submit** the default action of the form. 4-4-40

Summary

4-4-41

- ARIA = Accessible Rich Internet Applications
- Developed by the W3C to bring accessibility and semantics to modern web applications
- ARIA roles versus states and properties

4-4-42

- Keyboard control is extremely important
- Only use custom elements ARIA if absolutely necessary

4-4-43

4-5. Accessibility Guidelines and Standards

Web Content Accessibility Guidelines(WCAG 2.0) is a technical standard intended to be a single set of accessibility guidelines that meets the needs of individuals, organisations and governments internationally.

4-5-1

The main goal of the WCAG is to **ensure content is accessible to as many people as possible**

4-5-2

WCAG is intended for

4-5-3

- Web content developers
- Web authoring tool developers
- Web accessibility tool developers

It provides **12 guidelines** that are organized into **4 categories**, making up the acronym **P.O.U.R.**

4-5-4

- **P**: Percivable
  Users must be able to perceive the information
  It cannot be invisible to all of their senses

4-5-5

- **O**: Operable
  Users must be able to operate all UI controls
  Cannot require interaction that can't be performed

4-5-6

- **U**: Understandable
  Users must be able to understand the information
  And the UI

4-5-7

- **R**: Robust
  The site should not be fragile
  It should work in a wide variety of user agents and assistive technologies
  Should be future proof

4-5-8

Summary

4-5-9

- There are standards and guidelines
- WCAG is the primary standard for now
- P.O.U.R

4-5-10

4-6. Accessibility Testing, Tools and Resources

Accessibility is all about the **human experience**, and the only way to know if we **truly** have provided experience for **every human** is to test with actual humans.

4-6-1

One of the easiest ways to test for accessibility is to simply **stop using your mouse**. Many users use only their keyboard, or assisitive technology that mimics keyboard functionality, to navigate sites.

4-6-2

Download and use a screen reader. For example the JAWS screen reader. We can downoad a test version to test our site.

4-6-3

Checklist for Web Content Accesibility Guidelines: How to Meet WCAG 2.0. This can be used to measure the accessibility of a site.

4-6-4

The **key** to remember with testing, it that we have to test fpr accessibility just like we test for everything else.

4-6-5

Accessibility Tools

4-6-6

There are **many** tools to help with accessibility.

4-6-7

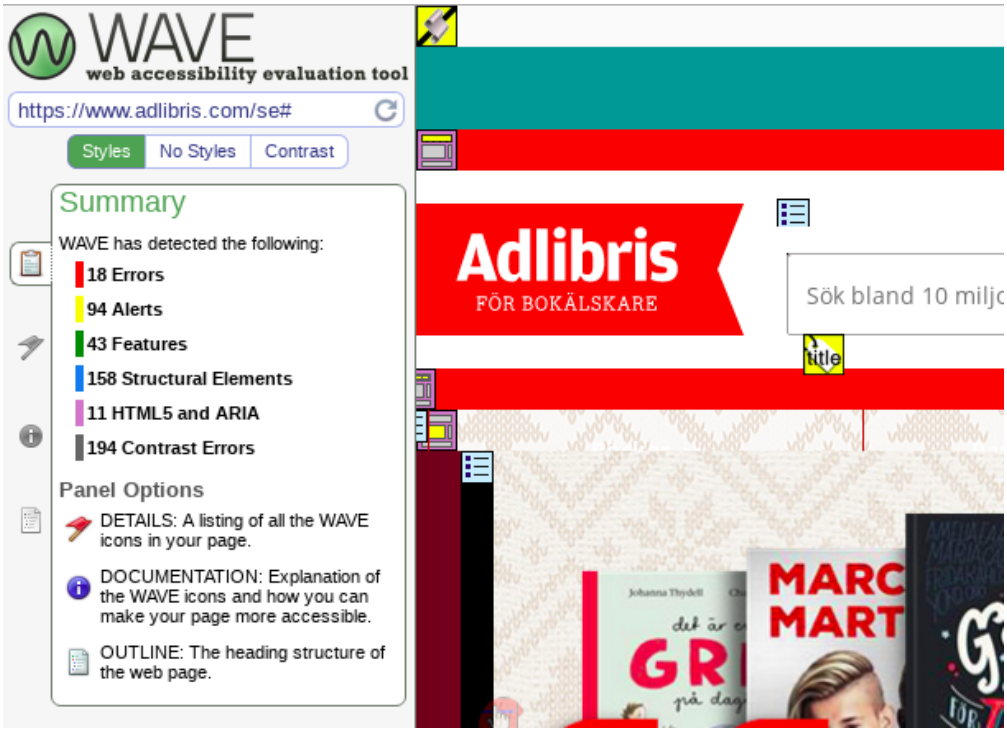
Wave Web Accessibility Evaluation Tool

4-6-8

https://wave.webaim.org/

We can pass in our URL, and it will give us a summary of errors and alerts for things that are missing or incorrect, along with features and notes about things that we are doing correctly.

4-6-9



Wave will check our **color contrast ratio** and let us know if we are passing or failing, and what levels of the **WCAG standards** that we are in compliance with.

4-6-10

We can also download the **Chrome** or **Firefox** toolbar, to make it easy to test while we are developing.

4-6-11

The **Color Contrast Analyser**: tool helps determine if your colors **pass or fail** against the **WCAG color contrast success criteria**. It can also be used to simulate different vision impairments. Link:
https://developer.paciellogroup.com/resources/contrastanalyser/

4-6-12

It also has a plugin for **Chrome** *4-6-13*

Accessibility Developer Tools Chrome Extension. This extension was created by Google to provide an accessibility audit for sites or applications. As part of the **Audits** tab, we can choose accessibility. *4-6-14*

It will run against the **given page** and it will give us **errors and warnings** where we have issues, and also let us know the tests that we have passed. *4-6-15*

However, **there is no replacement for human testing** when it comes to accessibility *4-6-16*

External links

1-1-2 PicJumbo: https://picjumbo.com/

1-1-2 Pixabay: http://pixabay.com/?rf=jimdo

1-1-2 Unsplash: https://unsplash.com/

1-1-6 99designs: https://99designs.com/go/ybbzjogz

1-1-6 Canva: https://www.canva.com/

1-1-6 PiktoChart: http://piktochart.com/

2-2-7 Zamzar: http://www.zamzar.com/

2-3-17 TinyJPG: https://tinyjpg.com/

2-3-17 TinyPNG: https://tinypng.com/

4-6-3 JAWS screen reader: http://www.freedomscientific.com/Products/Blindness/JAWS

4-6-4 How to Meet WCAG 2.0: https://www.w3.org/WAI/WCAG20/quickref/?showtechniques=11%2C111