HTML

HTML stands for Hypertext Markup language

Browsers translate HTML documents into viewable webpages

HTML was intended to facilitate content types

When designers want to do something new they write non-standard code to force browsers to do it

New standards are written to handle new requirements and browsers adopt the new standards

Client / server relationship

Servers:
Machines that hold shared resources
Always connected to the network

Clients:
Machines for personal use (laptops, phones, etc.)

LAN - local area network
WAN - wide are network

Request/response cycle

This is what happens when your computer (the client) requests a page and a server responds with the appropriate files

Uniform resource locator

URL - three parts:

- Protocol - how to connect
  - HTTP: Hypertext transfer protocol
  - HTTPS: secure hypertext transfer protocol
  - FTP: file transfer protocol

- Domain - the server*
  - Identifies the entity you want to connect to
    - Umich.edu, google.com, wikipedia.org
  - Each has different top-level domain
    - Determined by ICAAN
- (Optional) document -the specific file needed
  - Most pages are made up of multiple files

IP addresses
Internet Protocol version 6 (IPv6) is the communication protocol that identifies computers on networks.

Document
URLs can specify a specific document
http://www.intro-webdesign.com/ contact.html

If no document is specified, the default document is returned
Convention is index.html

The request:

Once the IP address is determined, the browser creates an HTTP request

Lots of hidden information in this request (header, cookie, form data, etc)

The response:

The server returns files, not "web pages"
- It is up to the browser to decide what to do with those files

If the server can't fulfill the request it will send back files with error codes: 404, 500 etc.

Creating and editing your file
1. Decide how you will organize your files
2. Decide on a naming convention
   1. Dash-names, camelcase
   2. No spaces, consistent capitalization
3. Decide on an editor
   1. Mac (textedit, textwrangler, sublime, VS code*)

Getting started
1. Open your editor
2. Select save or save as and name your file. You may need to create a new folder first.
3. Add Doctype, head, and body tags
4. Save file
5. Open in browser

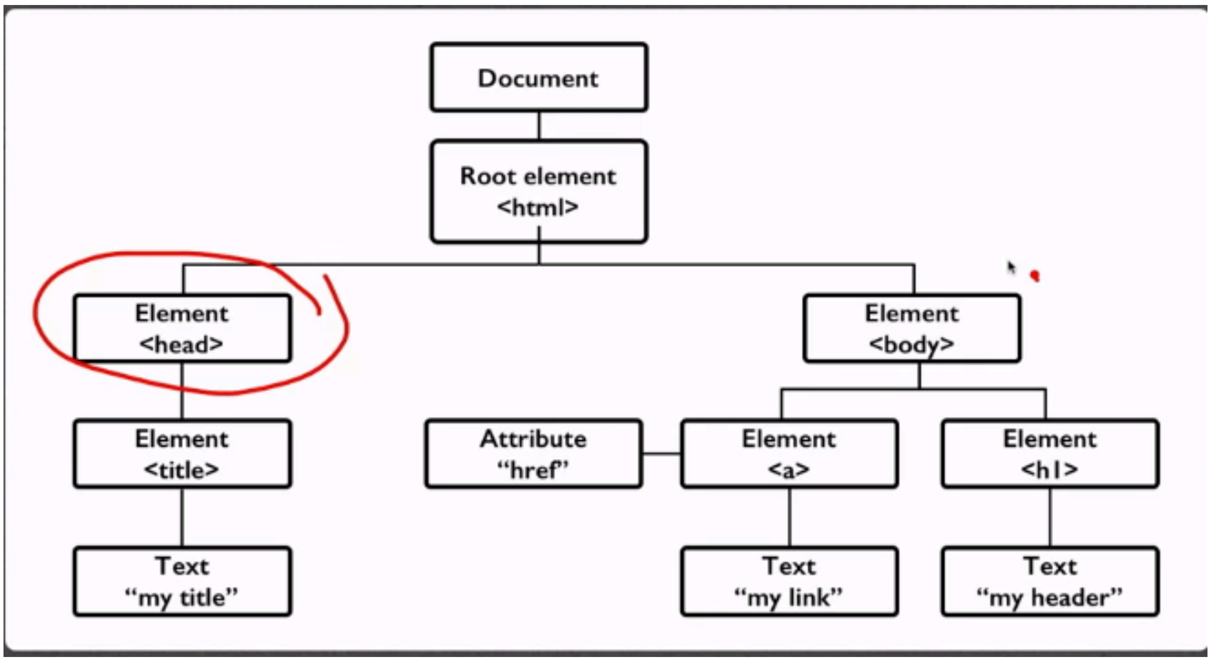
Troubleshooting

- My file opens in an editor instead of a browser
  - Right click and select 'open with"
- My browser shows my tags
  - Check that file extension is .html
- I changed my code, but my page looks the same.
  - Refresh your browser

  - Verify file name
- I get weird characters.
  - Try typing code in by hand not copy-and-paste

The document object model

DOM provides common tree-like structure that all pages should follow
Computer scientists love trees (the mathematical kind) because you can test them.



Three parts of a well-formed document
- Doctype
  - Version of HTML that you will be using
- Head
  - Metadata
- Body
  - Displayable content

Doctype: HTML5 <!DOCTYPE html>
Head:
Additional information used by the browser
Meta data - language, title
Supporting files - javascript, styling, add-ons
Other than title, meta-data is not displayed.

Body:
Bulk of your page
Important to write well-formatted (tree-like) code
Most of the content is displayed by the browser the there may be same meta-data too

HTML5
Tags have a beginning and an end
<h1> #start tag </h1> #closing tag
<img src="x.gif" /> #self-closing tag
Some tags have attributes (src, href.. etc)

Display
One of the most vital attributes of an element is its display.
- Block (can take width and height)
  - Newline is inserted before and after
- Inline (cannot take width and height)
  - Only uses as much space as needed to contain the element

Common Tags
- Headings (block)
  - <h1>,<h2>,<h3>,<h4>,<h5>,<h6>
  - These tags have syntax and semantics
- Paragraphs (block) - 用於編輯文本內容
  - <p>…</p>
  - Should only contain inline elements
- Divs (block) - 建立區塊
  - <div>.. </div>
  - Generic section that is larger than a paragraph
- Ordered lists

```
<ol>
   <li> item one </li>
   <li> item two </li>
</ol>
```

- Unordered lists

```
<ul>
   <li> item one</li>
   <li> item two</li>
</ul>
```

- Line breaks

<br>

Attributes
Attributes provide additional information about an element
Always specified in the start tag
Attributes come in name/value pairs

Some apply to any tag:
- Class - applies special properties to groups of elements
- Id - specifies a unique id to one element on the page
- Style - specifies a certain visual style (avoid this one!!!)
- Accesskey - a shortcut key to activate an element
- tabindex - the order elements will come into focus using the tab key

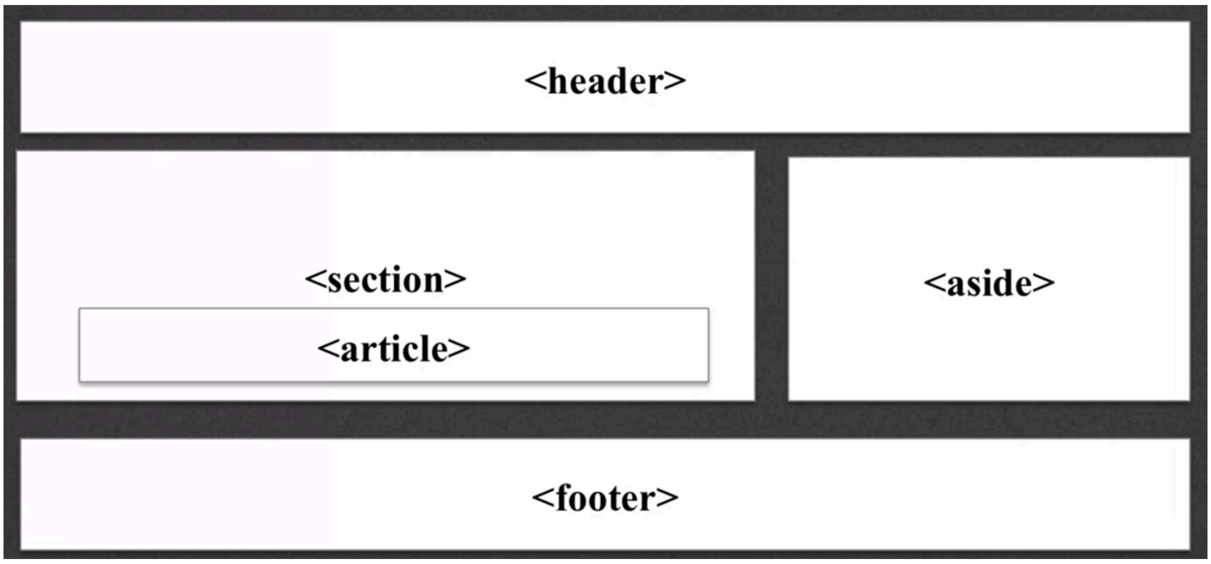
- Image (inline)

<img src = “myPicture.jpg” alt = “Image of Colleen”/>
- Image rarely work the first time
  - Show a broken link, too big, too small, etc.
- Save yourself heartache and size/carefully name your picture before you use it
- <img src = “logo.jpg” Image filename alt=“company logo” Info for screen readers, broken links title = “AAA1 LLC” displays on hover class = “thumbnail” extra formatting (height, width, position, etc.) />

Special entities
Tags always start with a bracket (<)
What if you want the browser to display a bracket, not start a tag?

Special entities
If you want < -> &it; > -> &gt; C -> &copy; blank space -> &nbsp; & -> &amp;

How to design
The most important step in web design is the design
You need a clear picture of what you want to create, before you can begin coding.



Using semantic tags

In the beginning (insert dramatic music of your choice ..)
There was div

<div> was a way to group related content together.

Divs almost always had special classes/ids associated with them
<div class = “header”> … </div>

<header>
A group of introductory or navigational aids: title, navigation links, etc.

```
<header>
   <h1> This is the Title </h1>
   <h2> the author is Colleen </h2>
</header>
```

Not to be confused with <head> or the different headings.

<nav>
A section of the page that links to other pages or two parts within the page.

```
<nav>
  <ul>
       <li><a href="#overview">Overview</a></li>
       <li><a href="#history">History</a></li>
  <ul>
</nav>
```

<footer>
A section that contains info such as copyright data, related documents, and links to social media

```
<footer>
  &copy; 2025 by colleen van Lent<br>
  <a href="http://www.intro-webdesign.com>Introduction to HTML5</a>
</footer>
```

Typically at the bottom of the page, but not required.

<figure>
More semantic than <img> can include:
Caption
Multiple multi-media resources

```
<figure>
  <img src="sunset.gif" alt = " Ashtabula sunse">
  <figcaption> A sunset over Lake Erie. Taken in Ashtabula ohio</figcaption>
</figure>
```

Other new tags
- Structural elements
  - Article, aside, main menu item, summary, section

- Form Elements
  - Datalist, keggen, output

- Input types
  - Color, date, email, list

- Graphics Elements
  - Canvas, sag

- Media elements
  - Audio, embed, source, track, video

Image - its more than the tag

- Many file types are widely supported
  - JPEG(.jpg and .jpeg), GIF, and PNG
  - SVG and BMP are additional options
  - File extensions must be included
- Every image must be downloaded, so size can be a factor
- Every image requires an HTTP request

Image sizes

- When you link to an image the browser displays the image as big (or small) as the file.
  - This size is rarely optimal

- Quick solutions change file, use width / height attributes

Using attributes
Always strive to keep style out of your HTML files but …
Some style may improve accessibility
<img> tag includes width and height attributes

Default image size
```
<figure>
    <img src = “imgs/Ashtabula.jpg” alt = “my house”>
    <figcaption> Default image size </figcaption>
</figure>
```

Using width in pixels
```
 <figure>
    <img src = “imgs/Ashtabula.jpg” width = “50%”  alt = “my house”>
    <figcaption> Relative image size </figcaption>
</figure>
```

Favicons
You can put /image/logo/icon next to the title of your page (in the tab)
Must go in <head> section
```
 <link rel = “icon” type =“image/png” href = “imgs/wd4elogo.png”>
```

Misuse of file extensions, filename, and file paths are often a problem.
For now, use width and height in HTML

Alternative text attribute
- Provide a textual alternative to non-text content
- Read by screen readers
- Displayed in place of images
- Provides semantic meaning for search engines

Creating Good alt text
- Be accurate
- Be succinct
- Don't be redundant
- Don't include "picture of …", "graphic of.."

Empty alt text
- It is okay to leave alt text empty (null)
  - Decorative images used for non-informative purpose
- Do not skip the alt attribute though!

Long alt text
- Some images (especially infographics) may require elaborate alt text
- Consider replacing alt text with link to separate page with full explanation

Finding usable images

Where can you find images for your site
Personal images
Images from image-sharing sites
Image with creative commons usage
Icons

Emojis and icons
A description of an emoji will be read by a screen reader, but not for an icon
Since icons are not images, they can't use the alt attribute
Instead, icons can use an aria-label attribute

```
<i class="fa-brands fa-pinterest" aria-label = "Pinterest"></I>
```

Images for impact
- Don't constrain yourself to the most common images
  - Include images of food from different cultures, athletes in adaptive sports, people of different body types.

- Using diverse images has the ability to dram more people to your site.

Hyperlinks

Anchor links
<a href = “http://www.umich.edu"> university of Michigan </a>

- The <a> tag stands for anchor link
- Needs a hyper-reference AND content
  - href: reference to location of new content
  - Content: the “clickable” part (text or image)

Type of link

- Absolute
  - <a #opening tag “href = http://www.intro-webdesign.com/" # where to go on click> Web design #clickable text </a> #closing tag
  - When to use -> I am not in charge of this page, somebody else is.
- Relative
  - <a href=“page2.html” #link to a local file in the same folder> second page</a>
  - <a href =“docs/page2.html” #link to a local file in a different folder called “docs”>second page </a>
  - <a href=“#history”#link to a different location in the same file >History section</ a>
  - When you’re developing your own website
  - Your link should NEVER have folders that are specific to your computer “c: \page2.html”
- Internal
- Graphical

Use image as the link
The clickable component doesn’t have to be text.
<a href=“http://www.redcross.org"><img src=“imgs/redcross-logo.png” alt = “Red Cross logo”/></a>

Usability issues

- Make sure the clickable component has an informative name
- Information in the images should be available to those who can’t see the image

Target

- Anchors can take a target attribute
  - _self - default action
  - _blank - open in new tab or window
  - _top and _parent

What does a “web accessibility coordinator” do?

- Helps guide policy and purchasing decisions
- Evaluates web interfaces for accessibility
- Assists those with disabilities to access online infrastructure.
- Keep pave with changing technology

W3C WCAG 2.0
W3C web content accessibility guidelines are principle-, not technology-based

The four principles (POUR)
- Perceivable
- Operable
- Understandable
- Robust

Validate your site

Three approaches
• Validate by URL
• Validate by filename
• Validate by direct input

Don't freak out
• Errors propagate, so always start at the top
• Search for solutions online and in forums

Accessibility validation
• You can use wave.webaim.org to validate accessibility
• You can install an extension to check local pages

CSS3 part

Browser default styling

The same html file may look different when viewed on different browsers.
- Some tags are supported, some aren't
- Browsers may have different default styles
In general, default looks are plain

Adding style: (X)

As styling tags were phased out of html, styling was done with style attribute
<h1 style = "color:blue"> styled heading </h1>

Cascading style sheet

CSS defined generic rules that can apply to multiple elements
```
Selector{
    Property: value;
}
```
Example:
```
h1{
Color: blue;
}
```

Rule syntax
- Brackets and semicolons are very important
- This is where a good editor can make a BIG difference

/* This is how comments are done */

Internal style sheet
- Styling is defined within <head>
- Rules are defined within <style>
- Styles are applied to all elements in that file
- don't forget to close the style tag!!

```
<head>
<title> Title here </title>
<style>
   h1{
         Color:blue;
}
</style>
</head>
```

External sale sheet
- You can put rules in an external file (don't use the style tag!!
- A link to the style sheet is put in the head section.

<link rel="stylesheet" href = "style.css">
- styles are applied to all elements in all files that links to the style sheet

Rule precedence
- What if one selector is defined in two external files?
  - The rules from the most recent file have precedence
- What if one selector has more than one rule in the same file?
  - The most recent rule has precedence.

! Important:
It is possible to override later rules, use !important

```
H1{
   Color:blue;
   Font-family: Arial !important;}
H1{
   Font-family: Times;}
```

Color conventions
- color names (blue, red, yellow, etc) work, but should be avoided
- Hexadecimal is common convention -> #0000FF, #FF0000, #FFFF00
- RGB -> (0,0,1), (0,1,0),(1,1,0)
- RGBA > (0,0,1,0.5)

Accessibility
- Appropriate use of color is critical to web accessibility

- Many more people are visually impaired or color blind than are legally blind.

What is color contrast
You intuitively know when something has poor contrast
There are tools that quantify the contrast between text and its background
http://wave.webaim.org/
http://webaim.org/resources/contrastchecker/

Styling your text
Many options for styling your text:
- Font (family, style, variant, size)
- Color and background
- Alignment
- Line - height

Font-family

- Font families are styles of text
- Examples: Helvetica, Courier, “courier New”, “comic sans MS”, cursive, Verdana
- Not all font-families supported by all of the operating systems, so you can provide alternatives.

```
h1{
      Font-family: courier, Impact, Arial;
}
```

- some fonts are not as user-friendly, use sans-serif when possible
- Custom fonts
- To expand beyond “web-safe” fonts use @font-face

```
@font-face{
      Font-family: mySpecialFont;
      Src: url(‘colleen.ttf’);
}
H1{
 Font-family: mySpecialFont;
}
```

Font-syle
- normal
- Italic
- Oblique

Font-variant
- Normal
- Small-caps

Font-size
- use pixel
- Use percentage

Color and background-color
- the color attribute is the color of the foreground
- The background-color is the color of the background

```
H1, span{
    Color:#0000FF; /* Blue*/
    Background-color: #B3B3B3; /*Grey*/
}
```

Text-align
Aligning text is simple!
Text-align
- Left
- Right
- Center
- Justify

Line-height
As you can guess, doesn't affect font
Adjusts the space between the lines of text

```
H1{
      Line-height: 50%;
}
```

Design larger projects on paper first!!!

Display is Key to Layout

- Every element is a box
- Display affects the layout of neighboring elements

Common values
- Inline: sits next to other elements
  - takes up "just enough" width and height
- Block: forces line break
  - default: take up all horizontal width and "just enough" height
  - rule can set width and height
- Inline-block
  - Same as inline, but accepts height and width
- None

Complementary properties

Float:
- reposition elements to the right or left.
- Elements are aware of one another and will not overlap.
- Values: left, right

Clear:
- used to keep floating elements away
- Values: left, right, both

Element overflow
What happens when you set a height/width and the content doesn't fit any longer?

Use overflow to determine access

Overflow:
Visible: can cause text to show up "on top" of other text
Hidden: hides anything that goes beyond bounding box
- this can cause problems since if the user increases font size. They may not be able to see content
Scroll: gives horizontal and vertical scrollbars
auto: Adds scrollbars as needed

New display properties are available, but not always supported:
Table
Sometimes you want to have table-like layout without using table structure, use display: table along with display: table-cell for elements

```
Body:{
Display: table
}
Div:{
Display:table-cell
}
```

Grid
Flexbox

Visibility:
Specifies whether or not element is visible
Options include: visible / hidden / collapse (only for table elements)

Grid:
You can use grid to place your content in columns
You need to define a parent element and give it children elements

Parent element
Step 1:
Set display to grid
Step 2:
Set grid-template-columns to number and size of columns
Step 3:
Set justify-content

One column grid

```
Div{
Display: grid;
Grid-template-columns: 500px;
}
```

Two column grid

```
Div{
Display: grid;
Grid-template-columns: 50% 50%;
}
```

Three column grid

```
Div{
Display: grid;
Grid-template-columns: 25% 25% 25%;
}
```

Justify-content
- You may want to adjust the default layout of the children with justify-content
- some of the possible values are:
  - Start, end, center, stretch, space-around, space-between, space-evenly
  - CSS tricks: justify-content

Modifying the child elements
- Best practice is to not hardcode the width of the children elements
- use a fluid measurement to make the most of the parent structure.

Positioning the children elements
- The children elements will automatically fall into the next available space
- You can move the element setting using
  - Grid-column-start
  - Grid-column-end

Flex / Flexbox
You can use flex if you want to let the browser resize your elements based on the screen size
You need to define a parent element and give it children elements

Parent element
Step 1: set display to flex
Step 2: set flex-wrap to wrap or no wrap
Step 3: set flex-direction to rows or column
Step 4: set the justify - items and/or align content

Default flex

```
div{
  Display: flex;
}
```

Using wrap

```
Div{
  Display: flex;
  Flex-wrap: wrap;
}
```

Changing the layout direction

```
Div{
 Display: flex;
 Flex-direction: column;
}
```

Adjusting the spacing

```
Div{
 Display: flex;
 Flex-direction: column;
 Justify-content: center;
}
```

Justify-content
You may want to adjust the default layout of the children with justify-content
Some of the possible values are
Flex-start, flex-end, center, space-around, space-between, space-evenly'

But
You use justify-content when the direction is row
If you are using direction column, you will want to use align-content instead.
Some of the possible values are:
Start, end, center, stretch, space-around, space-between, space-evenly

Styling links and lists

Anchor links
- link can take on all of the usual styles as well as text-decoration

```
A{
  Display: block;
  Font-weight: bold;
  Color: #ffffff;
  Background-color: #0006CC;
  Width: 200px;
  Padding: 4px;
  text-decoration: none;
}
```

“Button”
Many designers try to make their links look like buttons.
Be semantic, if you want a button use the <button> element instead.

<button> click me! </button>

States
- a: link —> a normal, unlisted link
- A:visited -> has been visited
- A:hover -> activated by mouse
- A:focus -> activated with the keyboard

Precedence of Rules
A: hover Must come after a:link
A: visited and a:active Must come after a:hover

Styling lists
- Number of properties beyond font, margin, etc.
  - list-style-type
    - ordered lists (lower-roman, upper-roman, decimal, decimal-leading-zero, upper-alpha, lower-alpha, hebrew, armenian

```
ul{
  List-style-type: upper-alpha;
}
```

      - override the default marker with circles, discs, or squares
  - list-style-image
    - use a custom image instead of traditional marker

```
ul{
  List-style-image: square url(‘icon.gif’)
}
```

  - list-style-position
  - list-style

Review: At this point you have learned how to write rules for the tags.
Embrace the many tools that are available to help you design your site

http://chrispederick.com/work/web-developer/
http://css3generator.com

Do web search for “Developer Tools”

Styling specific objects
- we have focused on type selectors
- What if don’t want to style all of the links, just some? Or just some of the lists?
- CSS gives you options

CSS selectors:
- some selectors follow the DOM
- Descendant selectors (nav a)
  - Style all of the anchor links inside a nav tag
- Child selectors (nav > a)
  - More constraining The anchor elements must be a child of the nav, no intermediate tags, e.g. paragraph
- Adjacent sibling (hl+ol)
  - Elements must be at same level and follow each other

Id selectors:
#id selector
- used to identify a single element in the DOM
- Was used extensively for <div id="header>, <div id="footer">, etc/
- There is a small movement to move the use of id OUT of CSS.

HTML:
```
<img src="logo.jpg" id="mainLogo" alt="logo"/>
```
CSS:
```
#mainLogo{
   border: 5px solid #0006CC;
   Margin:0 auto;
}
```

Class selector:
.class selector
 - Used to identify an element in the DOM that is part of a special class of items
 - think of thumbnail images, all of the links that are in the navigation, your social media images, etc…

Class in css -> syntax is "." -> class can be used multiple items
Ids in css -> syntax is "#" -> id should be unique

Think of images and navigation bars
- Format numerous (but not all) images the same way
- Visually signify the current page

Narrowing the scope
As you get more advanced pages, you will want to narrow the scope of the of action
p.main -> paragraphs using main class
Header img.special -> paragraphs inside header that use special class

Expanding the scope
You can combine element with a comma
- p, hl, #main,.special {rule to apply to all of them}

Review: What happens when there are multiple rules for the same selector?
- When there are conflicts, use the one processed most recently
- Unless a rule has !important

Attribute selectors

- Universal
  - * applies styling to every element on the page
  - Ackk!! Try this!
- Attribute selectors
  - a[href='info.html']
- PseudoClasses
- Pseudo Elements

You may want to search the DOM for certain elements that have an attribute you are looking for.

- All the image that use gif files
- All of the images that have empty alt text
- All of the links that go to government sites…

Using operators

Operators can be used to find those attribute values you are looking for
^: match the beginning exactly
  A[href^='http://umich']
$: match the end exactly
  img[src$='.png'] -> apply to .png images
*: wildcard
  a[href*='umich']

Review of steps
- Style the links: text-decoration, font-size
- Style the list items: background-color (padding, margin, and border)
- Style ul using the flex: flex-direction, flex-wrap
- Add the skip to main Content
- Style the current class

Background image

If an image is purely decorative you may want to add it as a background-image rather than using an image tag.

The syntax is background-image: url('file_path') - ('../images/…')

Complementary properties:
/* set a background color */
Background-color: black;

/* set a specified height */
Height:500px

/* center the image */

Background-position: center;

/* Resize the background image */
Background-size: cover;

Opacity: .5;

The opacity property specifies the transparency of an element
0 is completely transparent
.5 is the halfway mark
1 us the default opacity

When applied to an element it changes everything, not just the background image

Box model

Height and width

The default width of inline elements is the content
Elements that are not inline can take width and height properties - we saw this in the display lecture

Border:

Any element can have a border around it
Border property specifies style, width, and color
The border style Must be specified

```
Div{
    Border: solid 1px #CC00AA;
}
```

Border-style:
None, dotted, dashed, solid, double, groove, ridge, inset, outset, hidden

Border width and color

Width: set in pixels or thin, medium, or large
Color: Name - “blue” / RGB - rgb(0,0,255) / hex - #0000FF / transparent

Margin
Margin is additional space outside your border - between you and neighbor
Positive margin - element moves right / down
Negative margin - element moves left / upward

Padding
Padding is additional space between the element and its border
Positive padding - border moves outward from element

Margin and padding

Neither takes a color (transparent)
Can also be defined in 1-4 values like border

Centering an element
To horizontally canter an element use:
- margin: 0 auto;

But
- The element must display: block
- The element must not float
- The element must not have a fixed or absolute position
- The element must have a width that is not auto

Box sizing
Box-sizing takes some of the "math" out
Options:
Content-box: default additive
border-box: width takes content, padding and border into consideration

Measurements
Absolute - set to a specific size - px, mm, cm, pt, …
Fluid - sets size relative to surrounding elements
- %, vw, vh
- Em (for font): 1em is current size, .75 is 75% of the current size
- Rem (for font): 1 rem is current size of root element

Positioning

Position properties:

The four position properties are: static, relative, absolute, fixed

Static:
Default value for elements
Place in the next available position
Not affected by the top, bottom, left and right properties

Relative:
Positioned "relative to itself"
Take thee static position, but add offsets
The new positioning does not affect any other element. It is possible to move an element and leave a big hole where it would have been.
Relatively positioned elements are often used as container blocks for other elements.

Absolute
Element is removed from the document flow and positioned relative to it’s nearest ancestor (or the root)
Other elements behave as if element does not exist
Can end up on top or another element

Fixed position:
Positioned relative to the browser window
Think of popup boxes that won’t go away!!
Or a navigation bar that is always visible on the top

Pseudo classes ans elements
Elements that are dynamically populated on dependent on tree structure.
You have seen this before ..
```
A:hover{
}
```

Types of pseudo-classes
Link
:link, :visites
User Action
:hover, :active, :focus
Forms(interfaces)
:enabled, :checked, :disabled

Types of pseudo-classes

Structural / positional
- :first-child, :last-child, :nth-child(), :only-child
- :first-of-type, :last-of-type, :only-of-type

```
Li:first-child{ }
Li:nth-child(4){}
P:empty{}
Img:only-of-type{}
P:last-of-type{}
```

Textual
- :first-letter, :first-line

Positional / generated
- :before, :after

Fragments
- :selection

Pseudo-elements and classes are just one more way to add style to your page

Transitions
When elements transition from one state to another, you can alter their appearance.
If you hover over the link, change the color.
If an image comes into focus, change the size.

The properties
Transition-property:
- What is it you want to change? (Size, color, position, etc.)

Transition-duration
- How long should each transition last?

Transition-timing
- Should it be a smooth transition (linear)? Or different?

Transition-delay
- How long should the wait before the transition begins?

Setting up
1. Define your element
2. Choose the element for transition
3. Define the new values

- you must combine this step with a pseudo-class

Steps
```
Div{
    Color: #000000;
    Background: #2db34a;
    Line-height: 200px;
    Text-align: center;
    Width: 250px;
    Height: 200px;
    Border-radius: 6px;
    Transition-property: color, width, background, border-radius;
    Transition-duration: .5s;
    Transition-timing-function: linear;
    Transition-delay: .5s;
}
Div:hover{
   Color: #ffffff;
   width: 350px;
   Background: #2D31B3;
   Border-radius: 50%;
}
```

Using shorthands
If you have multiple properties transitioning, you can use shorthand:
Transition: background .2s linear, broader-radius 1s ease-in 1s;

Transitions can be used to create fun aspects to your page.
Playing with them will help you better understand element states (active, focus, etc.)
Accessibility, as always, is an issue.

Transforms:

2D transform options
Options: translate, rotate, scale, skew, matrix

Transform: translate(x,y);
- move x pixels to the left/right and y pixel up/down

Transform: rotate(deg);
- rotate/“spin” the element a certain number of degrees

Transform: scale(width, height);
- change the width and height of the element

Transform: skew (x-angle, y-angle);
- rotate the element a certain number of degrees along the x and y axis

Matrix() - combines all of the 2D transform methods into one

3D rotate

You can rotate along the x,y, or z dimension along a given degree

Transform: rotate3d(x,y,z)

Transforms are one more way to modify the look of your page.
Often combined with state changes
Will typically require browser prefixes

Navigation:

Navigation is a critical aspect of accessibility
Sighted users have tried and true visual cues to orient them on a page
- Banner
- Search box
- Main navigation box
- Content well

Blind and low-vision users rely on proper coding of page for orientation

What if you can’t see?

- Title of page lets you know what page you’re on when page loads.
- Proper heading placement and hierarchy conveys organization of page and allows SR users to skip navigation
- Link descriptions convey content of page and organization of site.

Proper heading hierarchy

Headings need to be properly nested to convey organization of the page

<h2> tags follow the <h1> tags, the <h3> tags follow the <h2> tags, etc.

Off-page headings

Useful when you want to give SR users a navigational aid without cluttering presentation

Use CSS to position headings off-page
```
.offpage
{
   Position: absolute;
   Left: -1000px;
}
```

Don't use {display:none} or {visibility: hidden}

Meaningful link text
- screen readers can find and list links
- Descriptions for the links must be meaningful out of context, via tabbing or presented in a list.
- Don't use "here", "click here", "read this" and "more"
- Don't use URL as a link description - will sound like gibberish, unless very short and intuitive.

Javascript:

What you can do with JavaScript:

JavaScript is a "real" programming language":
store variables, set decision points, loop, reuse code with functions

Get data from the browser
Manipulate the DOM that browsers use to create web pages

Variables: Store data and refer back to it later
Decision points: use control statements to decide which code to run under different circumstances.
Looping:
- Avoid writing the same (or similar) code over and over again.
- Determine at runtime how many times you want to run some code.

Functions: Reuse code multiple times, but only write it once.
Use code from others

Manipulating the DOM: Javascript can find, add and delete elements from the DOM.

Can also react to mouse clicks, page reloads and other actions.

Web pages are built upon the DOM

- Document Object model
- Structures documents like a tree
- Every node has one parent, and possibly many children
- Nodes have properties, methods, and events

The DOM and JavaScript

Page content is represented by the DOM
Scripting languages (JavaScript) use the DOM to interact with the document

How does it work?

Accessing the DOM is done with an API - Application programming interface
- no matter which browser, no matter which scripting language, the API is the same

The DOM objects/elements

- document - the root of the page
  - Document.URI, document.height, document.links, document.bgColor,

- element - a node in the tree
  - Returned by a member of the API

- nodeList - an array (group) of elements
  - Document.getElementsByTagName(‘p’) would return a set of nodes

- attribute
  - A node in the DOM though rarely used that way. Another way to manipulate / change the document.

Specific APIs
Document.getElementById(id)
Document.getElementByClassName(class)
Element.innerHTML
Element.style
Element.setAttribute(attribute, value)
Element.removeAttribute(attribute)

Selecting the first element

getElementById() takes a single parameter and that parameter must be an id selector.

querySelecotor() Method
Return first result of the given selector which could be anything - except pseudo-elements
Because the elector can be anything which is a valid CSS selector you must include the #,.,etc.

Selecting Multiple elements

The querySelectorAll() method is identical to the querySelectir() but returns all the found values

Again, while getElementsByClassName doesn't need the "." As part of the selector, querySelectorAll does.

Deciding on a method:

- Speed won't be an issue for you
- QuerySelector allows you to use any css selector
- I am less prone to typos in the method name.
- getElementById, getElementsByClassName, etc. have more mnemonic names.
- I am less prone to typos in the css selector.

Output:

Interactivity

- HTML5 and CSS3 are not really interactive.
- New elements and pseudo-classes can only go so far.

What can Javascript do?

- Read and write HTML elements
- Reacts ti events (mouse events, keyboard events, etc.)
- Validate data
- Detect the visitor's browser
- Create cookies

Javascript output
- JavaScript doesn't have a built-in print function
- Data is displayed via
  - An alert box using window.alert()
  - A prompt using window.prompt()
  - HTML output using document.write()
  - HTML element using innerHTML()
  - The browser console using console.log()

Alert()
In JS, an alert is a pop-up window that displays information:
  alert(“My Message Here”);

prompt()
Very similar to alert, but wants input.
  prompt(“enter your name:”)

Document.write()
- What if we want something permanent?
- Document.write() writes directly to the page
  Document.write(“Time to learn JavaScript”);
- Not usually recommended since it can easily be misused.

innerHTML
To change the contents of the DOM, use innterHTML combined with the element you want to change.

Element,innerHTML = “Time to learn Javascript”;

The element will come from the API

Console.log()
This option white the data to the browser console
Console.log(“Leave a secret message”);
The console is a place to see what is going on during the execution of your program.

The console:
You should be utilizing the console by now
Does more than take “print” statements, also provides debugging information for JavaScript, HTML and CSS

Debugging

Safari: preferences -> advanced check the show development menu in menu box
Google chrome: developer -> javescript console
Firefox: Tools -> console

Storing the data

Part of learning to program is learning to store data.
In JavaScript, data is stored in variables.
To use a variable, you have to declare it
Var name;

Variable Names:
Consist of letters, digits, underscores, and dollar sign ($)
Can not start with a digit

Are case-sensitive…
- name, Name, naMe, NAME are all different variables

Should be meaningful.

Variable assignments
It is silly to have a variable if you are never going to use it.
You can assign values using the = operator

Var name =“Alan”

Assignment statements
I like to refer to the LHS and RHS of statements
LHS - the variable being updated
RHS - the new value that will be stored in the variable

Using a Variable
Var name = prompt(“What is your name?”);
Document.write(name);
Var date = Date();
Document.write(date);
Var location = window.location;
Document.write(location);

Data types

In many programming languages, variables need to have a single type.
In JavaScripte, a variable can take on many different types.

Number:
Numerical values
- with or without decimals
- Var width = window.innerWidth;
- Var pi = 3.14;

String

- A String is a collection of characters (letters, numbers, punctuation, …)
- To create a string you put the value in quotes “…”

Boolean:
- In programming, a boolean value is one that is either true or false

Object
- sometimes the variables are more complex
  - A node in the DOM a good example
  - Var topic = document.getElementByID(‘myID’);
- Nodes are more than a single value, they have attributes.

Array:

How can a function return more than one value?

Var links = document.getElementsByTagName('a');

- Arrays store multiple value using a variable name, and an index for each element in the array.

document.write(links[0]);

Luckily in JavaScript you have a lot of flexibility with the types of data

Operators and expressions:

Statements:
- We have been using statements to execute our JavaScript code
- Statements often have expressions
- Expressions produce values.

X = 5%2 => 1

Boolean operators
Operator:
== / X==5 -> false
== / X==12 -> true
!= / x1=5 -> true

&& / (15>x)&&(X>5) both sides must be true
|| / (15>x) || (x>5) at least one side must be true
! /!(x == 12) false

Functions
Functions are bits of codes that you can reuse
Functions have a special syntax

```
Function functionName(parameters){
   Code you want to run
}
```

Function Declaration
```
Function welcomeMsg(){
   Alert("Welcome to JavaScript!};
}
```

```
Function welcomeMsg(msg){
  Alert(msg);
}
```

Function call
Declaring a function doesn’t actual do anything.
You need to call the function.
Calling a function changes the program flow.

Parameters:
Sometimes functions need some information in order to perform its “function”
The names of the parameter are not important, as long as you are consistent.

Return values:
Some function return values
These values can be used in assignment statements or conditional expressions.

```
Function welcomeMsg(msg){
   alert(msg);
   Var name = prompt(“what is your name?”);
   Return name;
}

Var firstName = welcomeMsg(“Hi”);
```

Whenever possible, use built in functions.
When you need to write your own function, try not to be too specific.

Code placement:

Where to place the code:
Now that you are going to start to write your own functions, it is easier to separate code from content.

JavaScript code can be placed in the body, head, or in an external file.

In the head: when JavaScript functions are declared in the head section they are separated from the content. - Use the script tag
Have access to all of the document information (ids, classes, etc.)

In an external file
When JavaScript functions are in a separate file it is possible to reuse the code in multiple files.
Don’t use the script tag.

Debugging your code:
As your code becomes more complex, make sure that you are using your debugger.
The console is your friend.

Folder structure
Web developers tend to organize their code into separate parts
- HTML
- CSS
- Images
- JavaScript

Linking from an HTML file
<link rel=“stylesheet” href=“css/style.css”>
<script src=“js/javaFunctions.js”></script>
<img src =“images/myPicture.jpg”/>

Linking from a CSS file
Background: url(“../images/holiday.png”);

Debugging
If a link isn’t working you want to check few things:
- Did you spell the file names correctly? (Case matters!!)
- Are files in the correct folder?
- Are you working on the correct file?

Events
Adding the interactivity!!
It has been up to us to decide when the functions should execute.
It would be better if the functions were called based on special “events”
The JavaScript API lets us add dynamic function calls!!

Onclick
- User clicks on an HTML element

Onmouseover
- User moves the mouse over an HTML element

Onresize
- browser window is resized

Onload
- browser finishes loading the page

How it works
- Any element can react to an event.
- You need to add the event to the tag and include what you want to happen

<div onclick = “message()”> clicking on the Div will invoke a JavaScript function</div>

Using Quotes
- You can use single quotes or double quotes for the event result.
- Double quotes make it easier if you want to pass String parameters

<div onclick = “message(‘Hi’)”>

- be careful of copying and pasting quotes!

Events changes the program flow
- some programs ran in a linear order (step-by-step)
- Events cause the program to “run continuously” since the DOM is always listening for events.

Mouse events
- onclick, ondblclick, onmousedown, onmouseenter, onmouseleave, onmousemove, onmouseout

Keyboard events
- onkeydown, onkeypress, onkeyup

Frame Events
- onload, onresize, onscroll, onerror,…

Comprehensive list:
https://developer.mozilla.org/en-US/docs/Web/Events

Adding the interactivity!!

It has been up to us to decide when the functions should execute.
It would be better if the functions were called based pm special “events”
The JavaScript API lets us add dynamic function calls.

Modify the DOM

HTML:
```
<h1> modify the DOM </h1>
<button onclick=“document.getElementByID(‘stuff’).innerHTML = ‘click First
Button’;”> First </button>
<button onclick=“document,getElementByID(‘stuff’).innerHTML = ‘click Second
Button’;”> Second </button>
<p> This code shouldn’t change. </p>
<p id = “stuff”> this will change </p>
<p> This code shouldn’t change either. </p>
```

Referring to Elements
A key to smart programming is using functions.
A common roadblock is figuring out how to set up functions for reuse.
- How do I avoid writing a different function for every different element?
- How can the function know which one I want to use?

“This” is a keyword that allows an element to reference itself
- every object in the DOM has an automatically generated “this”

Allows you to access an element’s info
- without “this” it would be difficult for the functions to know what data to use.

“This” is also used outside functions.

Storing lots of data at once
- The variables I have used to this point store a single piece of information
  - Number, String, Boolean or Object

- What do you of if you want multiple, related pieces of information?
- Store them in arrays

Declaring An Array:
Var grades = [80,87,94,82,62,98,81,81,74,91];
Var foods = [‘bananas’,’apples’, ‘pizza’];
Var images = document.getElementsByClassName[‘images’];
Var listItems = document.getElementsByTagName[‘li’];

Arrays: An array is a collection of values.

Each value is called an element
Elements are referenced by index grades[0] refers to the value 90 (1st value)

The elements in the array Don’t have to be all the same type.

JavaScript Arrays are Objects
- They have attributes and methods
- Grades.length
- Grades.sort()
- Grades.push(element)
  - grades[grades.length] = element

Using lightbox:
It is more common to use code from others than to write your own/
One of my favorite is
https://lokeshdhakar.com/projects/lightbox2/

Concepts to consider:
- Media queries - detecting the viewpoint size
- Flexible grid-based layout for relative sizing
- Flexible images

Examples of great design
- http://mediaqueri.es/

Benefits if Response design

Responsive web design (RWD) - fluid measurements, flexible grids, and vary CSS rules

Adaptive Design (dynamic serving) - returns one of multiple versions of a page based on the type of device.

Separate Mobile Site (.m) - a separate page URL for the mobile site.

Fluid measurement: