Chapter 1: HTML

The HyperText Markup Language, or HTML is the standard markup language for documents designed to be displayed in a web browser.

The current version of has two standards

- WHATWG's HTML which is a living standard that is constantly being updated.
- W3C's HTML 5 Standard which is closer to a releases versioning standard. i.e., 5.1, 5.2, etc.

Generally people prefer the WHATWG's Living standard but because not all web browsers will constantly adhere to the the standard you should look at the capability table to see if the standard is implemented by the user agent/browser.

DOCTYPE

You need to tell the world that this is a modern HTML file. You must put a DOCTYPE at the top and enclose your content in the HTML tag.

```
<!DOCTYPE html>
<html>
...
</html>
```

XHTML

You can write HTML in XML format if you follow XML syntax: XHTML. Valid HTML is not valid XML, and valid XML is not valid HTML.

- For a while there was a drive to combine XML and HTML into "polyglot HTML" where you could write both at the same time, but this was abandoned in 2015.
- Browsers still support `<voidtag/>` syntax in HTML but it breaks other HTML parsers

XML is very unforgiving, a simple mistake could cause a parser to refuse to parse the document. Breaks `Document.write()`, the `<template>` tag, most entities. . .

Adds support for namespaces, HTML 5 XML has no DTD URL. Uses `Content-Type: application/xhtml+xml`.

Head/Header

The head tag is where we put information about the webpage. You will usually include a title here.

Meta tags contain meta information for browsers and other tools to help interpret.

```
<head>
  <title>Page Title</title>
  <meta charset="UTF-8">
  <meta name="description" content="Example
  ↪  Page">
  <meta name="author" content="Abram Hindle">

  <!-- proprietary extensions -->
  <meta name="apple-mobile-web-app-capable"
  ↪  content="yes" >
  <meta
  ↪  name="apple-mobile-web-app-status-bar-style"
  ↪  content="black-translucent" >
  <!-- mobile viewport information -->
  <meta name="viewport"
  ↪  content="width=device-width,
  ↪  initial-scale=1.0, maximum-scale=1.0,
  ↪  user-scalable=no">
  <!-- stylesheets / css -->
  <link rel="stylesheet"
  ↪  href="css/reveal.min.css">
  <link rel="stylesheet"
  ↪  href="css/theme/default.css"
  ↪  id="theme">
  <link rel="stylesheet"
  ↪  href="lib/css/zenburn.css">

</head>
```

Body

Is where all the content goes.

```
<body>
<p>Visible content goes here.</p>
<p>Consider that you should not directly
↪  specify layout information
  here but that you should mark up blocks
  ↪  with a class allowing you to
  apply layouts later. You can abstract
  ↪  layout</p>
</body>
```

Tags

All tags should be closed to allow for ease of reading and parsing, except void tags.

Start Tags are enclosed in < and >

End Tags are enclosed in </ and >

```
<tag>
  <enclosedtag>
  </enclosedtag>
</tag>
```

Void tags are enclosed in < and > like start tags, but don't have a matching end tag.

```
<tag>
  <voidtag>
  <br>
</tag>
```

Unless you are writing XHTML, in which case you must follow XML syntax, void tags typically look like `<voidtag>`. but sometimes are written like `<voidtag/>`.

This is against the HTML spec.

Tags can be any capitalization in HTML, but typically they are written in lower case. Stick to lower-case when writing HTML.

Paragraph Tag: <P>

```
<p>
  Now bears us onward one of the hard
  ↪ margins,
  And so the brooklet's mist o'ershadows it,
  From fire it saves the water and the dikes.
</p>
<p>
  Even as the Flemings, 'twixt Cadsand and
  ↪ Bruges,
  Fearing the flood that tow'rds them hurls
  ↪ itself,
  Their bulwarks build to put the sea to
  ↪ flight;
</p>
```

We can use CSS to style these text blocks. Whitespace in the <p> block doesn't matter. So formatting will most likely be off.

Now bears us onward one of the hard margins, And so the brooklet's mist o'ershadows it, From fire it saves the water and the dikes.

Even as the Flemings, 'twixt Cadsand and Bruges, Fearing the flood that tow'rds them hurls itself, Their bulwarks build to put the sea to flight;

Breakline Tag

We can make line-breaks explicit with `<br>`.

Image Tag <IMG>

We want to show images, we cannot embed them in HTML easily so we use hyperlinks (URIs) to reference them and include them.

Remember to include alt tags so they are machine and human-readable.

```
<img src="images/plasma.png" alt="Plasma
↪ Pattern">
<!-- scaling -->
<img src="images/plasma.png" alt="Plasma
↪ Pattern" width="30">
<!-- scaling without respecting aspect -->
<img src="images/plasma.png" alt="Plasma
↪ Pattern" width="30" height="60">
```

The `img` tags has certain attributes that are important. The `alt` tag should be used to describe the image using text. This is sometimes a requirement for accessibility requirements for people who have a visual impairment or for Search engine optimizations. Where most SEOs uses text, not images, although Google might have the capabilities for processing images.

The `width` and `height` attributes are used to set the size of the image. If one of the size attribute is specified then the aspect ratio is preserved otherwise it will follow both the height and width attributes specified by the developer.

Anchor Tag <A>

The anchor tag is used to make hypertext or text with hyperlinks.

Anchor tags let us link to other documents or locations. The `href` attribute links to a URL. The URL can be relative to the location of the current page. Anchor tags used to place anchors on pages, but in HTML 5 they just navigate to ID'd sections now.

```
<a href="http://slashdot.org">Slashdot.org:
↪ News for Nerds Stuff that
Matters</a><br>
<a href="http://cnn.com">T</a><a
↪ href="http://msn.com">a</a>
<a href="http://softwareprocess.es/">g</a><a
↪ href="http://ualberta.ca">s</a>
don't have to be very long.<br>
<a href="../">The a directory down!</a>
↪ Relative Link
```

You can also have images within the anchor tags to allow images to become buttons to another link. Elements within the anchor tags are often highlighted in blue to show that it's a hyperlink and a the URL of the link will show on the bottom right corner of the web browser.

Text Layout Philosophy

Let the user agent decide how to display the text. You don't need to control everything. But provide the appropriate semantics first. Then apply layout information to those semantics.

If you want something to show up in a certain make a CSS class and style a span or a div or a paragraph that way.

If you want a fancy layout use flexbox or grid layouts.

<DIV> and <SPAN> Tags

`<div>` and `<span>` tags have no particular meaning. They are used for separating content within the body and allow different style attributes to be applied. Using div and span as is will not change the content of style of the content.

- `<div>` will cause a line break before it is displayed.

- <span> will be inlined.

Example

```
<div>
<span style="background-color:#AAAAAA;
↪  font-weight:bold">
  Now bears us onward one of the hard
   ↪  margins,</span><br>
  And so the brooklet's mist o'ershadows
   ↪  it,<br>
  From fire it saves the water and the
   ↪  dikes.<br>
</div>
<div style="font-size:150%">
Even <span style="color:red">as the
↪  Flemings</span>, 'twixt Cadsand and
↪  Bruges,<br>
  Fearing the flood that tow'rds them hurls
   ↪  itself,<br>
  Their bulwarks build to put the sea to
   ↪  flight;<br>
</div>
```

Style

Tags can have attributes defined like so:

```
<tag attribute1="value1" attribute2="val2">
```

These attributes could affect the style of the content. The style attributes are Cascading Style Sheet properties.

Cascading Style Sheet (CSS)

A language to style an HTML document.

- Cascade - The children inherits properties of the "parent"
- Style - Properties refer to style properties such as layout, position, color, font, background, etc.
- Sheets - Apply to a page, change a page.

CSS can be applied on a per tag level, but can also be applied globally.

```
<!-- Include a file of CSS  -->
<link href="path/to/cssfile.css"
↪  rel="stylesheet">

<!-- Inline CSS -->
<style type="text/css">
.angry {
font-weight:900;
zoom: 3;
}
.angry:nth-child(odd)
{
color:green;
transform:rotate(7deg);
-webkit-transform:rotate(7deg); /* Safari and
↪  Chrome */
float: left;
}
.angry:nth-child(even)
{
color:red;
transform:rotate(-12deg);
-webkit-transform:rotate(-12deg); /* Safari
↪  and Chrome */
float: right;
}
</style>
<p style="color:orange">Apply style
↪  directly</p>
<div>
<div class="angry">HULK</div> <div
↪  class="angry">SMASH!</div>
</div>
```

Properties

Here are some of the properties to know

- color – color of the text or object
  - color:green
  - color:#abc (short form)
- font-family – the font
  - font-family:"Times New Roman"
  - font-family:"Verdana"
- font-size – the font size
- – font-size:10px;
  - font-size:10pt;
  - font-size:large;
  - font-size:200
- font-style – normal, italic, oblique
  - font-style:normal;
- background-color
- background-image

Selectors (classes)

A CSS selector selects the HTML element(s) you want to style.

Classes

CSS labels that starts with a period `.` denotes a class. We can use HTML attributes to label tags with classes that allow the tags to inherit CSS style information. The attribute name is `class`. E.x., `class="bg1"` for class bg1 or `.bg1`. You can also combine classes with HTML tag selectors; this example makes p tags with class="highlight" orange instead of yellow. You can override other CSS as well.

```
p.highlight { background-color:orange; }
```

```
<p class="highlight">ALERT in orange!</p>
```

ID

ID selectors are like class selectors except they aim at the one tag with the id="idtag" as an attribute

Element

Element selectors let you style entire HTML elements (tags). Important because you might want to theme all divs or imgs or links.

```
p { background-color:yellow; }
```

This will style all paragraphs with a yellow background.

Context

Selectors that behave depending on the context Can be chained with other selectors

- :hover - when the mouse is over
- :active - active link
- :first-letter - operate on the first letter
- :nth-child(2) - second child

Note

The index for nth-child starts at 1 not 0 unlike in most programming languages.

Positioning

The `position` attribute is used to position a element within a page. It use left, right, top, bottom attributes to control the position.

- `fixed` - stays on one spot in the browser
- `relative` - position relative to where it normally goes
- `absolute` - absolute positioning relative to the first parent that was positioned (often the page itself).
- `z-index` - can be used to order overlapping elements. The higher the number the close it is to the screen

Alignment

One big problem with CSS is how to center something! Margins can be used or alignment attributes can be used.

- `text-align` - specifies the horizontal alignment of text in an element
- `flex` - sets the flexible length on flexible items. Helps for big centering jobs, things with many elements, etc
- `align-content` - aligns the flexible container's items when the items do not use all available space on the main-axis (horizontally).
- `justify-content` - defines how the browser distributes space between and around content items along the main-axis of a flex container, and the inline axis of a grid container.

HTML for User Interfaces

The HTML Form elements let us accept input from browsers in a structured way and form HTTP GETs and POSTs.

<FORM> Tag

The form tag encloses a group of HTML input/UI widgets which can then be submitted as once.

- method - We can specify the HTTP method (POST/GET)
- action - We can specify the URI to GET or POST to.

<INPUT> Tag

The input tag can take in textual input, passwords, or act as submit button.

Type

The input could of many different types including but not limited to

- button
- checkbox
- color
- date
- datetime
- datetime-local
- email
- file
- hidden
- image
- month
- number
- password
- radio - Allow only one selection from a group of buttons
- range
- reset
- search
- submit
- tel
- text
- time
- url
- week

Name

The name of the data sent to the URI

Value

The default value Hidden types allow you to embed values to send along to help the request.

Example

```
<form action="http://webdocs.cs.ualberta.ca/~hindle1/1.py" method="get">
What is your name? <input name="name"/></br>
What is your quest? <input name="quest"/></br>
What is your favorite colour? <input name="colour"/></br>
<input type="submit"/>
</form>
```

<SELECT> TAG

Select tag is for a dropdown box of options.

```
<form action="http://webdocs.cs.ualberta.ca/~hindle1/1.py" method="get">
What kind of cake shall we have?</br>
<select name="caketype">
  <option value="angel">Angel Food Cake</option>
  <option value="devil" selected>Devil's Food Cake</option>
  <option value="cowpatty">Marie Antionette's Cake</option>
</select>
<input type="submit"/>
</form>
```

File Upload

You can upload files with the input tag but we need to switch the POST encoding.

```
<form action="http://webdocs.cs.ualberta.ca/~hindle1/1.py"
      enctype="multipart/form-data"
      method="get">
  Choose a file to upload!
  <input name="uploadFile" type="file" />
  <input type="submit"/>
</form>
```

CGI Diversion

Did you know you can write CGI in just about any language?

All it takes is printing HTTP Response headers, reading environment variables and reading from STDIN!

Here's a minimal PERL example.

```
use Data::Dumper;
print "Content-type: text/plain\r\n\r\n";
print Dumper(\%ENV);
my $line = 0;
while(<>) {
    print $line++ . " " . $_ . $/;
}
```

So you can use multiple languages.

I really really really recommend using the CGI library that comes with your language if you do. Mostly for parsing of GET and POST arguments.

```
import cgi
import cgitb
cgtib.enable()
# print your header
```

CGI Problems

There are many issues with CGI

- Slow

- 1 process per invocation
- Inefficient communication of requests
- Lack of object-orientation representation of a request
- Difficult to share state

Review

HTML.

- The language used for documents to be displayed on web browsers
- People prefer the WHATWG standard but not all web browsers supports it because it's a living standard
- Must have *<!DOCTYPE html>* at the beginning of the document.
- You can write HTML in XML format but the compatibility between the two ain't great.

Header.

- The header is where you put information about the webpage
- `<meta charset="UTF-8">` goes here.
- All `meta` tags goes in the header.
- Title goes here as well as the CSS style sheet under the `<link rel="stylesheet" href="path_to_css.css"`.

Body.

- The body contains the actual content of the page
- All tags must have an opening `<tag>` and a closing `</tag>`
- `<voidtag>` and `<br>` don't have a matching closing tag.
- `<p>` paragraph tags doesn't care about white space. Use `<br>` to breaklines explicitly.
- `<img>` Image tags shows images and you should set the `alt` property within that tag for text-to-speech on images and SEOs.
- `<a>` anchor tags are used to make hypertext or text with hyperlinks. Basically what makes Wikipedia great.

Text Layout.

- `<div>` will create a newline before it's content is displayed
- `<span>` will separate a div for column vertically.

CSS.

- The children will inherit properties of the parent node
- A css file can apply the style globally or on a per tag level.
- Style properties can be applied to certain selectors like
  - Tags like `div`
  - Classes that start with a `.` for example `.pokemon` will be applied to any tags with the class `<... class="pokemon">`
  - Tags that start with `#` for example `#yellowtag` will be applied to any tags with the id `<... id="yellowtag">`
  - Based on context with the `:` for example `:hover` apply when the mouse is over the element.
- You can set position using fixed, relative, absolute, and z-index for overlapping
- Alignment using text-align, flex, align-content, justify-content.

Forms.

- You can create user submittable forms using the `<form>` tag
- The input for the form is defined using the `<input>` tag and you have to define the type, the name of the input to be sent to the URI, and a default value.
- The `<select>` tag can be used to show a dropdown box of options.

CGI Diversion.

- You can write CGI in any language
- Basically you can create a program that will generate html content dynamically at runtime.
- Has some problems.