HTML

Consider the following:

> Where the Sidewalk Ends Shel Silverstein There is a place where the sidewalk ends And before the street begins, And there the grass grows soft and white, And there the sun burns crimson bright, And there the moon-bird rests from his flight To cool in the peppermint wind.

Without any context, formatting or structure, the above collection of words is just that, a collection of words. We have no sure way of telling whether some of the words are to be read as a set, whether some words should be more emphasized than others, and so on. It requires extraneous cognition to answer questions about the text, such as *"Who is the author of the text?"* Because the human brain is capable of extrapolating based on past experience and poetry often adheres to a specific pattern of prose, some readers may recognize the text above as a poem by Shel Silverstein. However, a computer, without the complexities and sophistication of our neural pathways, would have a much harder time. If we were to break the poem into individual lines, we can provide a little more structure to our text:

> Where the Sidewalk Ends
> Shel Silverstein
> There is a place where the sidewalk ends
> And before the street begins,
> And there the grass grows soft and white,
> And there the sun burns crimson bright,
> And there the moon-bird rests from his flight
> To cool in the peppermint wind.

Given this new structure, we can say that the title of the poem is the first line, while the author of the poem can be found in the second line. For this specific collection of words, using this specific format, we could teach a computer how to extract information about this specific poem. However, our solution is not scalable. Some poems may have titles that span two lines. Others may have multiple authors. To make for more general problem, how do we even know that this is a poem.

With this structure, we might be able to write a program that would provide a computer with the ability to answer the question, *"Who is the author of the text?"* We could teach a computer that the first line of text is the title and the second line is the author. However, this isn't a scalable model as some poems may have titles that span two lines, multiple authors, etc.

To provide for a more structured (and scalable) approach, we can lean on the concept of a markup language. Markup languages allow us to provide structure, organization, and meaning to content.

```
<poem>
      <title>Where the Sidewalk Ends</title>
      <author>Shel Silverstein</author>
      <stanza>
            There is a place where the sidewalk ends
            And before the street begins,
            And there the grass grows soft and white,
            And there the sun burns crimson bright,
            And there the moon-bird rests from his flight
            To cool in the peppermint wind.
      </stanza>
</poem>
```

In the above example, we've surrounded our text with a set of "markup tags", marking the beginning and end of the poem, the beginning and end of the poem's title, the beginning and end of the author's name, and the beginning and end of the poem's only stanza. If we had a longer poem, we could mark multiple stanzas using the given structure.1 Markup languages bring order to chaos.

Markup Languages

All markup languages share the following common characteristics:

- **Provide additional semantic meaning**
  As shown in the previous example, markup languages can provide additional information about a given set of content, making the content more understandable and meaningful.
- **Are composed of a well-defined set of available elements**

Unlike the previous example, most markup languages have a specific set of what markup elements you can use. Using any undefined elements would result in an invalid document.

- **Document Type Definition (DTD) or schema**
  The details of what goes into a DTD would take a course of its own. It is sufficient for this course to know that the DTD provides a list of rules for a given markup language about what elements can be used, how they can be used, and what attributes a given element might have.

Some examples of markup languages are:

- **LaTeX**
  LaTeX is a markup language for the TeX typesetting program. It is most-commonly used in academic circles for writing papers and/or books.
- **XML**
  The goal of the eXtensible Markup Language (XML) can be used to make documents readable by both machines and computers. As the name implies, XML is very extensible and is used in a number of applications, including Microsoft Office and Apple iWork. There are also a number of markup languages based on XML.
- **HTML**
  HTML was proposed by Tim Berners-Lee in 1991. It was partially based on Standard Generalized Markup Language (SGML), another markup language in use at CERN, and specified a set of 21 markup elements.2 It has since grown considerably and will be the markup language that we'll be using for the remainder of the semester.

HTML stands for HyperText Markup Language. One of the core features of the web was HTML's ability to express hyperlinks, links that could be clicked on and would lead the user to another document. As previously mentioned, HTML and the WorldWideWeb were both adopted quickly. In the two years leading up to its formal recognition as a markup language3, programmers were fully embracing HTML, implementing support Tim Berners-Lee's original markup elements as well as proposing new elements as needs arose.

In an effort to advance the evolution of HTML, browser makers would sometimes implement their own tags as a solution to a given problem. For example, Marc Andreessen, the co-author of Mosaic, pushed for the tag for inline images. While others objected to this tag on the grounds that it wasn't general enough, Mosaic, the most popular web browser at the time, was already using it. It has become firmly ingrained in HTML specifications since.

The formal recognition of HTML in June of 1993 included 41 markup elements. By November of that same year, a new version of the HTML specifciation, named HTML+, was released. This added HTML forms, tables, inline images, and a handful of other useful tags. The addition of inline images to the HTML specification was largely driven by the way inline images were supported in the Mosaic web browser. Rather than waiting for the release of an official, updated HTML specification, the Mosaic developers had created their own HTML element, initially only supported in their browser. While their implementation of inline images would be adopted into the HTML+ specification, it was certainly a case of the tail wagging the dog.

While some would argue that this push for new tags was necessary for the development of HTML, the fear was that without some sort of oversight and canonical specification, web page designers would become overwhelmed by a barrage of HTML tags that only worked on some browsers.

The Mosaic browser was created in 1993, co-created by Marc Andreessen during his time at the University of Illinois. After graduating, Andreessen and Jim Clark, the former founder of Silicon Graphics, saw the fiscal potential of Mosaic and founded Netscape Communications. Using the Mosaic browser as a starting point, they developed Netscape Navigator, which quickly surpassed Mosaic, establishing itself as the leader of the web browser pack.

Working with Tim Berners-Lee, a computer scientist named Dan Connolly attempted to address this issue when he authored the HTML 2.0 specification. Connolly pulled together a list of all of the non-standard elements that browser makers had implemented and added them to the new version of the specification. To help further address the issue of loose HTML specifications, Tim Berners-Lee founded the World Wide Web Consortium (W3C). The goal of the W3C was to standardize web protocols and technologies. While this group was (and still is) successful at defining web protocols and specifications, browser makers weren't necessarily interested in using those standards. The problem was only going to get worse.

In 1995, Microsoft introduced Internet Explorer, a proprietary browser that initially could only be found on Windows computers. Due to the popularity of the Windows operating system, Internet Explorer quickly became the de facto web browser for many users, fighting Netscape Navigator for dominance of the web browser market. The developers of these two browsers viewed each other as direct competition. In order to compete, they would continually try to outdo the other by introducing new non-standard HTML elements. Novelty elements such as <blink> and <marquee> were added to the list of supported HTML elements, despite the fact that they weren't part of any HTML specification. In some cases, the developers would introduce an element using the same name but would have it provide different functionality. While viewed as trivial and/or silly by most serious web developers, these tags were often adopted, misunderstood, and misused by novices and people just learning HTML. The tag also caused issues for people with epilepsy, so there was that.

In January of 1997, the W3C and browser makers got together to publish a new version of the HTML specification. There was much back and forth between the groups, with negotiations over the specification proceeding like a divorce mediation. One browser maker would agree to get rid of a non-standard element, but only if one of the other vendors would get rid of one of their non-standard elements. In some instances, browser makers would push for a specific issue to be added to the specification, even though they were the only ones currently supporting it. The resulting specification ended

up being a combination of standard HTML elements from past specifications and a slew of formerly proprietary, browser-specific HTML tags. To make things worse, browser makers didn't stick to the specification, returning to their old ways of introducing non-standard tags ad-hoc.

This time of intense web browser competition is often referred to as the “Browser Wars”. At the height of the browser wars, the tendency of browser makers to implement their own tags and to interpret the use of some standard HTML 4.0 tags had reached the point that web pages would often look completely different depending on the browser the user was viewing the page on.

This led to some developers 'branding' their website with an image that said "Best viewed in _____", specifying what browser they had used to create the site. Others would try to reach a larger audience by developing multiple versions of the same web site, one for each browser they wanted to target.

This website is best viewed in

While most users were oblivious to the behind-the-scenes competition, the progress of HTML and the Web was quickly turning into a mess!

In 1997, the HTML 4.0 specification was published, broken up into three options. HTML 4.0 Transitional was to indicate that the web page was in the process of moving from HTML 3.2 to HTML 4.0. This allowed for the use of deprecated elements. HTML 4.0 Strict did not allow for deprecated elements. Lastly, HTML 4.0 Framesets allowed for the use of frames, a feature initially implemented by Netscape and later adopted by other browsers. To some, the inclusion of a “transitional” specification was an indication that the W3C was giving up on the thought that the HTML specification would ever be fully adhered to.

In the midst of all this chaos, a small group of dedicated web developers grew tired of having to worry about how things looked in one browser versus another and decided to take action. They began lobbying Microsoft and Netscape, trying to convince browser developers that they should adhere to HTML standards. Additionally, they began developing online material to help educate other developers on standards-compliant web development.

Despite the fact that the task at hand was a large one, the Web Standards Project was very successful in their efforts. They eventually convinced both Microsoft and Netscape to consider adhering to standards when developing their browsers. Most modern browsers today have followed suit. Many of the individuals who made up the Web Standards Project are still very active in the web development community and continue to provide education and insight as the field moves forward.

HTML was designed to be very flexible. This flexibility can be, in part, attributed to its quick adoption and popularity. For example, the original HTML specification by Tim Berners-Lee included an element named <NEXTID>. This was an elment specific to the NeXTSTEP operating system (on which Tim Berners-Lee was developing). In the write-up for the specification, he noted:

> <NEXTID> can be ignored by browsers, only needed for editors.

The ability that some elements may be ignored provided browsers with the ability to forgive novice developers as they made mistakes in crafting their HTML. For example, if someone accidentally typed

```
<text>This is my text</text>
```

browsers would display the words, “This is my text”, despite the fact that is not a valid HTML element. Unlike other markup languages, HTML also allowed for the following:

- Elements could be nested incorrectly.
- Elements did not have to include a closing tag.
- Elements could be specified using upper-case, lower-case, or a mix of cases.

In a move away from the flexibility provided by HTML 4.0, the W3C published the XHTML 1.0 Strict specification in January, 2000. If a document was using XHTML 1.0 Strict, much of the flexibility from HTML 4.0 was not to be found. Elements had to be nested properly. Elements must have a closing

tag, regardless of what element. Lastly, elements had to be named using lowercase only. These changes created an HTML specification that matched that of the more general XML specification. Some saw this as a step in the right direction, moving towards a more structured format with rules that made for easier parsing of content. Additionally, some believed that the move towards XHTML 1.0 would require a more in-depth understanding of HTML from developers. Others, however, argued that the change was necessary and that XHTML 1.0 Strict was a move away from the features that made HTML appealing in the first place.

XHTML 1.0 Strict is the HTML specification that we will be using during the first half of this course. The latest HTML specification, HTML 5, builds off of a combination of XHTML 1.0 and HTML 4.0 and will be discussed later in the semester.

HTML Element Structure

HTML elements are the building blocks of the web. Just as in our poem example above, HTML provides additional **structure, organization, and meaning** to text. There are two types of HTML elements: *standard* and *self-closing*. Standard elements are made up of an *opening tag*, followed by some content, followed by a *closing tag*. HTML opening tags all start with a less-than symbol (<), followed by the name of the element you're adding to the page, followed by a greater-than symbol (>). Similarly, a closing HTML tag start with a less-than symbol (<), **followed by a forward slash**, followed by the name of the element that you're closing, followed by a greater-than symbol (>). Below is an example of a generic standard HTML element:

<tagname> Some content here </tagname>

Self-closing HTML elements do not have an opening and closing tag. Rather, the element is created using a single HTML tag. Self-closing HTML elements start with a less-than symbol (<), followed by the name of the element you're adding to the page. The key difference between a standard opening HTML tag and a self-closing HTML element is that the self-closing element ends with a space, **followed by a forward slash**, followed by a greater-than symbol (>). Below is an example of a generic self-closing HTML element:

<tagname />

In some cases, an HTML element may require more information than just the element name. To do so, we rely on *HTML attributes*. HTML attributes provide additional information about a given element. Attributes are always specified in the opening HTML tag (in the case of a standard HTML element) and are always specified in the form `attr_name="attr_value"`. While attribute values can contain a wide variety of information, attribute names must always be lowercase.

<tagname attr_name="attr_value">Some content here</tagname> <tagname attr_name="attr_value" />

While the above descriptions are abstract, some examples of HTML should provide some clarity.

Creating HTML Documents

Below is an example of a simple web page:

HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
   <head>
      <title>My Page</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
   </head>
   <body>
      <p>
         This is my content.

      </p>
   </body>
</html>
```

While simple, there are still a number of components in the above example that are worth discussing.

Doctype

The first line of our example is the `DOCTYPE` used for XHTML 1.0 Strict. The `DOCTYPE` statement is what tells web browsers what type of document it is displaying. While the `DOCTYPE` statement has a fair amount of information included in it, a breakdown of each individual part is not required for our purposes. It is sufficient to say that you need to include this as the first line of every XHTML 1.0 Strict document you make in this course.

`<html>`

The second line of our example is the opening `<html>` tag. Everything between the opening and closing `html` tags constitutes our web page. To tell the browser what language we're using, we've included the `lang` attribute with a value of "en" (for English). Because we are using XHTML 1.0 Strict, we also must included the `xmlns` attribute.

`<head>`

Within our `html` element, you'll find the `head` element. The `head` of our web page contains information about the page but **does not contain any content that is displayed on the screen.** For example, the `title` element contained within the `head` (line 4) tells the browser what to display in the browser tab used to display the web page. The `meta` element contained within the `head` (line 5) tells the browser what character set we're using. Neither of these elements are displayed in browser. They're simply used to tell the browser information about the page.

`<body>`

After the closing `<head>` tag, you'll find the opening <body> tag. The `body` of our web page contains everything that will be displayed on the browser screen. In this example, the only content we've included is a single `p` element, indicating that we have a *paragraph* that should be displayed.

It is important to note that any 'white space' - that is, characters such as a space, a tab, or a line break, are treated as a single character when more than one of them occurs in a row. In other words, you can't use multiple spaces between words or elements in the HTML source code to create visual space in the browser. In our example, the space after our first sentance will not be displayed on the screen, nor will any line breaks. If we wanted to force a line break, we would have to include a `br` element

HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
   <head>
      <title>My Page</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
   </head>
   <body>
      <p>
         This is my content.<br />
         This is the second line.
      </p>
   </body>
</html>
```

Given the above HTML, below is the output that a browser would render to the screen:

Browser

This is my content.
This is the second line.

HTML Entities

In some instances, you might want to include a character that is reserved within HTML. For example, the less-than character is used to tell the browser that you are beginning an HTML tag. In the instance that you want to actually display a less-than symbol on your web page, you would need to use an *HTML entitity*. HTML entities allow us to display both reserved HTML characters as well as characters that don't exist on most keyboards.

The format for an HTML entity is an ampersand (&), followed by the entity name, followed by a semicolon (;). For example, given that the less-than HTML entity name is "lt", we would use the following HTML entity to include a less-than symbol on our page: `&lt;`.

See http://ista230.com/resources/htmlEntities for a full list of HTML entities.

<a> Elements and Hyperlinking

The `a` element provides us with the ability to include hyperlinks within our web pages. `a` elements require an `href` attribute in order to tell the browser where it should go when the user clicks on it. The text in between the opening and closing `a` tags is the text that is displayed to the user. For example:

HTML

```
<a href="http://www.google.com">Visit Google</a>
```

Browser

Visit Google

In the above example, we used the full URL to point the browser to Google's website. This type of URL is what is known as an *absolute* URL. Absolute URLs always start with the protocol being used (e.g., 'http') and always include the domain name. **Absolute URLs should be used when linking to other websites but not when linking to pages on your own website.**

Relative URLs, on the other hand, do not include the protocol being used, nor do they include the domain name. **Relative URLs should be used when linking to another page on the same web site.** Relative URLs are always crafted relative to the location of the current web page. Some examples follow:

```
<a href="http://www.google.com"> Visit Google </a>
```

The link above would direct the user to Google, **a different website than we're already on**, using an absolue URL.

```
<a href="/"> Visit our homepage </a>
```

Because the URL in the 'href' attibute for link above doesn't start with a protocol, we know that the URL is relative. The link above would direct the user to the root of our website. Since no file name was specified, the user would be taken to the file named index.html.

```
<a href="/resources/index.html"> View our resources </a>
```

The link above would direct the user to the resources folder on our website and then display the file index.html.

```
<a href="../files/report.pdf"> View our annual report </a>
```

The link above would direct the user up one folder, and then down into the 'files' folder where it would display the content of the file 'report.pdf'.

In addition to linking to other web pages, `a` elements can be directed to email addresses by including the text 'mailto:' at the beginning of the 'href' attribute, followed by an email address. On most computers, this will result in an email client (such as Microsoft Outlook) opening, ready to compose a message.

```
<a href="mailto:ORGANIZATION_EMAIL"> Email me! </a>
```

In some cases, users may want to link to a specific point within a web page. For example, Wikipedia entries often contain multiple sections, only one of which may be of interest for a given link. To accomplish this, we can use fragment identifiers within our `href` attribute:

```
<a href="somepage.html#middle"> Jump to the middle of some page! </a>
```

When used, fragment idenfiers link to a specific `id` attribute on the target page. `id` attribtues can be specified for any HTML element

ID values must start with a letter and be made up of the following:

- letters
- numbers
- underscores
- dashes

It is important to note that an 'id' can only be used once per page but can be used on different pages without any problem. For example, one could have multiple HTML files, each with an <h1> element with an 'id' attribute of 'logo'. However, one could **NOT** have multiple <h1> elements in a single HTML file with the same 'id' attribute of 'logo'.

In addition to linking to specific parts of other pages, we can also use fragment identifiers to move around the same web page. For example, when a web page contains a lot of text (e.g., articles, blog posts, etc), it is helpful to provide users with a convenience link for getting back to the top of the web page they're viewing:

HTML

```
<body id="top">

   <p>A little content here...</p>

   <p>And a little more content here...</p>

   ...

   <a href="#top">Jump to the top of the page</a>
</body>
```

Clicking on the link that says "Jump to the top of the page" would tell the browser that it should scroll up the page until the HTML element with an 'id' attribute whose value matches 'top' is displayed (in this case, the `body` element).

Headings

In the newspaper industry, headings provide newspaper editors with a way to indicate importance and structure within a large amount of text. The newspaper title is displayed with large, bold text, while article titles are displayed in a slightly smaller text. A tagline for an article might be smaller yet, while the text of the article is typically displayed in the smallest size font.

HTML heading tags provide us with a means to create a similar hierarchy within our web pages. This information can be semantically helpful in determining how your web page is structured and what the relationship is between certain elements. There are six levels of HTML headings: `h1`, `h2`, `h3`, `h4`, `h5`, and `h6`. `h1` indicates the highest importance, while `h6` indicates the least importance.

HTML

```
<body>
   <h1>Newspaper Name</h1>

      <h2>Section Name</h2>
      <p> Welcome to the Business section!</p>

         <h3>Article Name</h3>
         <p>
            This is the content of my article...
         </p>
</body>
```

In the above example, the <h1> element designates the name of the newspaper, the <h2> element is the section of the newspaper, and the <h3> is the title of the article. Using the appropriate heading levels, one can easily determine the hierarchy of the various elements.

When using HTML headings, it is important to not skip levels! While using `h1`, `h2`, and then `h4` is technically valid (because it uses a decending order), it will cost you points in this course.

HTML Comments

When crafting any document, it is often useful to leave "notes-to-self", small bits of information reminding you of what you still need to do, what you did, or why you did it a certain way. HTML comments are useful for documenting how your website is structured. They are not displayed to users when viewing your page in a web browser but are useful for other web designers as well as for yourself when editing or viewing the source of your web page.

HTML comments always begin with the text `<!--` . The comment then continues until the browser sees the text `-->`.

HTML

```
<!-- This is an HTML comment. "I don't show up!" -->
```

HTML

```
<!--
   This is another HTML comment.
   "I can span multiple lines."
-->
```

HTML comments can be used to specify where sections of your website start and stop. They can also be used to explain any HTML that might be confusing otherwise.

HTML

```
<body>
   <!-- Start newspaper content. -->
   <h1>Newspaper Name</h1>

      <h2>Section Name</h2>
      <p> Welcome to the Business section!</p>

         <h3>Article Name</h3>
         <p>
            This is the content of my article...
         </p>

   <!-- End newspaper content. -->
</body>
```

Make sure you close your comments correctly! If you don't, you may make your entire website into one large comment (as is the case in the example above).

HTML Emphasis

The purpose of HTML is to provide structure, organization, and meaning. However, this was not always a matra that browser makers adhered to.

In previous versions of HTML, one could use the <i> element to italicize text. While this was functional, it clearly was an HTML element that was used for display purposes, not to provide structure, organization, or meaning.

HTML

```
<i> This is some italicized text!</i> <!-- This is no good! -->
```

HTML

```
<em> This is some emphasized text!</em>
```

Browser

This is some italicized text!
This is some emphasized text!

While both examples above have the same effect on most browsers, the second is the proper way of using HTML, specifying that the text should be *emphasized*, not italicized.

Similarly, both the <b> and elements are displayed the same on most modern browsers, the second example is the proper way of using HTML, specifying that the text should be **strongly emphasized**, not bolded.

HTML

```
<b> This is some bold text!</b> <!-- Again, no good! -->
```

HTML

```
<strong> This is some strongly emphasized text!</strong>
```

Browser

This is some bold text!
This is some strongly emphasized text!

One might question, "If browsers display them the same, then why does this matter?" When thinking about users of your web page, it is important to remember that everyone brings with them a different set of abilities and experiences. While most people will be viewing your website, others may be listening to it. Users with vision imparement may be using a specific web browsing technology known as a screen reader. Screen readers will read the text of your website to a user. While things can be spoken "boldly", it is a little less clear how to speak something as "italicized". Conversely, "emphasized" and "strongly emphasized" are much more clearly interpreted.

Images

The ability to display inline images are one of the features that made the Mosaic browser stand out from other early web browsers. To include an inline image, we use the `img` element.

HTML

```
<img src="myImage.jpg" alt="About my image" />
```

You'll note that the `img` element is a self-closing element. `img` elements als have two required attributes:

The 'src' attribute should be used simliarly to the 'href' attribute for links. Specifically, you should use the same rules regarding images from external websites vs. images from your website. While there are rare cases that merit using an image from another website, you'll want to use relative images most of the time.

There are three types of image formats supported by all browsers on the web:

- JPG
- GIF
- PNG

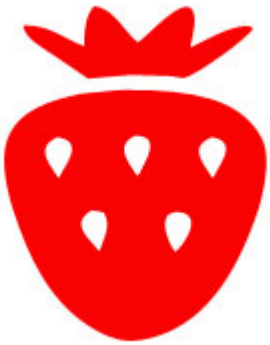
While there may be other image formats available, they are not necessarily supported by all browsers and should not be used when one of the three formats above is available instead. Below is a brief breakdown of the supported image formats and when they should be used:

JPEG - Joint Photographic Experts Group

- File extension: .jpg or .jpeg
- Best used for photographs or photo-like images
- Pixilation on some graphics and text

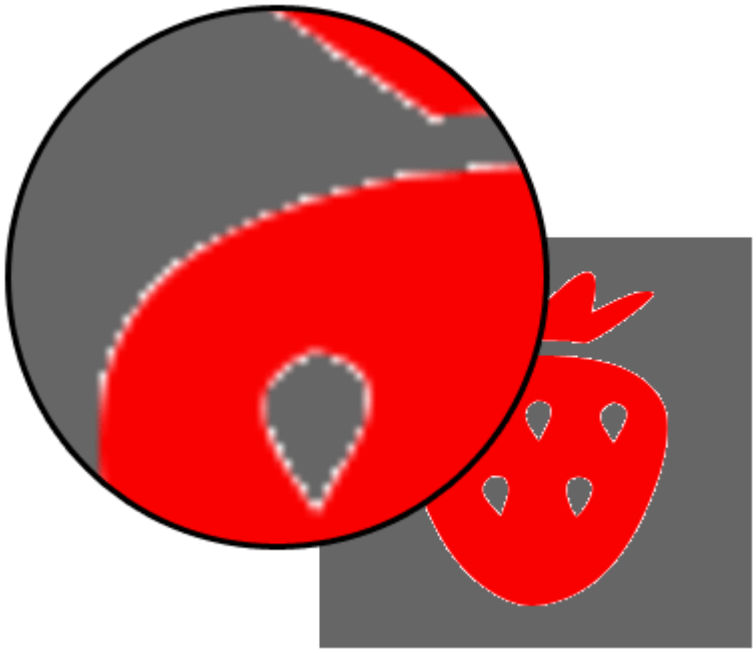
GIF - Graphics Interchange Format

- File extension: .gif
- Better for graphic images and text images
- Provides limited transparency

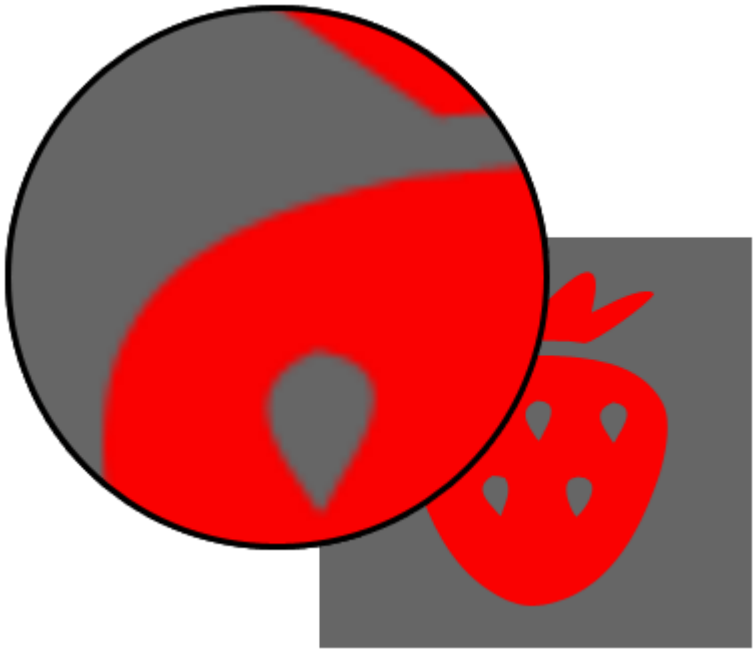


In the GIF image of the strawberry, we can specify that the white area around the strawberry is transparent, allowing the background color of anything underneath it to show through (see next slide).

While the transparency will work in some instances, there may be slight pixelation around the edges of the image. For this reason, GIF is often not the preferred format for transparent images.



- File extension: .png
- Better for graphic images and text images
- Provides smaller file sizes than GIF (typically)
- Displays more quickly on the screen
- Provides better transparency



PNG images don't usually suffer from the same sort of pixilation related to transparency and GIF images. Additionally, PNG images support alpha transparency, which allows us to not only specify that certain areas are completely transparent but that other areas are semi-transparent.

Given that PNG images provide such an improvement over GIF images, one might wonder why GIF images still persist. The answer is simple: GIF images support animation. While the use of GIF animations often can lead to an unprofessional feel, there are instances where they come in handy. For example, many of the 'loading' images used on websites are often animated GIF images.

Where to get images

- Make them - Adobe Photoshop, GIMP
- Google Images - Advanced Search
  - http://www.google.com/imghp?hl=en&tab=wi
  - Use advanced search to specify format
  - When searching for images on Google, it's important to be aware of copyright infringement. Unless otherwise stated, you can't just use someone else's images on your website. In the instance that no copyright statement or statement of use is available, your best bet is to assume that you **do not** have permission to use the image.
- Flickr - Advanced Search
  - http://www.flickr.com/search/advanced/
  - Creative Commons

    - In the advanced search on Flickr, you can specify that you only want to search for images that have a Creative Commons license. Within Creative Commons, you can search for images that are free to use without attribution, that are free to use commercially, and so on.
- Morgue File
    - http://www.morguefile.com/
    - Free to use/edit without attribution
    - These are images that have either fallen into public domain or images that have been shared for corporate or public use without any payment or attribution requirements.

Miscellaneous HTML Elements

The following sections cover a few miscellaneous HTML elements. While not as heavily used as those we've already discussed, it is good to know what they are and when they should be used.

Horizontal Rules

Horizontal rules can be used to provide an indication of a break in the content. By default, this displays as a horizontal line across the page.

HTML

```
<hr />
```

Preformatted Text

As previously mentioned, multiple spaces, tabs, and line breaks in the HTML source code are treated as a single space when displayed to the user. However, this can be problematic. In particular, when trying to display programming code, such as Python, where white space has a specific meaning, displaying each space can be necessary. To tell the browser that the text being displayed is "preformatted" (and that white space should be preserved), we can use the `pre` element:

HTML

```
<pre>
   This    is    some    text.
      This is some indented text.
   This    is    some    other    text.
</pre>
```

Browser

```
This    is    some    text.
   This is some indented text.
This    is    some    other    text.
```

When using the <pre> tag, all of the spaces, tabs, and line breaks in the element are displayed to the user. Additionally, the font used is a monospaced font, ensuring that all characters take up exactly the same width.

The <pre> element is one that can easily be abused. It should only be used for elements where it is important for the text to appear exactly as typed. For example, indentation and line breaks are important elements some programming languages. When trying to show a sample of these programming languages, one could use a <pre> element to ensure that the white space is displayed.

The <pre> tag would **NOT** be the optimal choice for displaying content that has lots of line breaks. While the element would allow you to skip the step of entering multiple <br> elements, it would also maintain multiple spaces and tabs, which is not typically needed.

Block and Inline Elements

All HTML elements fall into one of two categories: *block* elements and *inline* elements.

Block elements

- Have a line break before and after them automatically
- Can contain inline elements
- Can sometimes contain block elements*

* Unless otherwise stated, you can assume that any block elements that we discuss **cannot** contain other block elements.

Below are some examples of block elements that we've discussed thus far. Note that all of the elements listed below **cannot** contain other block elements.

- ...
- <h1> ... </h1>
- <h2>, <h3>, <h4>, <h5>, <h6>

Inline elements

- Do not have a line break before or after
  (
 is the exception)
- Can contain other inline elements
- **Must be contained by a block element**
- **Cannot contain block elements**

Below are some examples of inline elements that we've discussed thus far.

- ...
- ...
- ...
-

Semantically-neutral HTML elements

Of the elements that we've discussed this far, the one block element that would be used for most content would be a `p` element. That said, not everything is a paragraph. Since we want to use HTML to provide structure, organization, and meaning, we need something more generic than our `p` element...

HTML provides us with two semantically-neutral elements: `div` and `span`. These two elements provide us with elements that can be used when there isn't an HTML element that matches our need. Additionally, they provide us with a means to add additional structure/meaning as well as the ability to group related elements.

The `div` element is a generic **block** element that can contain both block and inline elements. This is useful when we have some content that needs to be in a block element but that doesn't semantically fit inside of another HTML element. For example:

HTML

```
<!-- Paragraph for holding an image (not really a paragraph). -->
<p>
   <img src="/images/slides/html/logo.png" alt="Company logo" />
</p>
```

```
<!-- div - a better containing block -->
<div>
   <img src="/images/slides/html/logo.png" alt="Company logo" />
</div>
```

We can also use the `div` element to group related elements into a single "container".

HTML

```
<h1>Article Title</h1>
<h2>Author</h2>
<p>
   This is the content of my article.
</p>
```

```
<!-- All article-related elements are grouped together -->
<div>
   <h1>Article Title</h1>
   <h2>Author</h2>
   <p>
      This is the content of my article.
   </p>
</div>
```

The span element provides us with a generic **inline** element. It can contain other inline elements but cannot contain block elements and must be contained within a block element itself. To see where a span element might be helpful, consider the following:

HTML

```
<p>
   My favorite book is
   Lord of the Flies.
</p>
```

In the above example, we want to find a way to distinguish the book title from the rest of the text. Unfortunately, there is not a <book> element that we can use...

```
<p>
   My favorite book is
   <span id="title">Lord of the Flies</span>.
</p>
```

We can use the generic element to encapsulate the book title. Using the 'id' attribute, we can provide some additional semantic information about the element as well.

Both and elements should only be used when there is no alternative semantic HTML element.

HTML

```
<!-- There's no need to do this... -->
<div>
   This is a paragraph in my article.
</div>
```

```
<!-- ... when this will suffice. -->
<p>
   This is a paragraph in my article.
</p>
```

Checking Your Markup

In this lecture, we have covered a fair amount of material. To help you check the validity of your HTML code, please use the ISTA 230 HTML Validator to check your work:

- ISTA 230 HTML Validator

