Front-end Web Development: Section 9 Pages

Notes - Class 1-2

(content below is from webplatform.org)

HTML

What is HTML

Most desktop applications that read and write files use a special file format. For example, Microsoft Word understands ".doc" files and Microsoft Excel understands ".xls". These files contain the instructions on how to rebuild the documents next time you open them, what the contents of that document are, and "metadata" about the article such as the author, the date the document was last modified, even things such a list of changes made so you can go back and forth between versions.

HTML ("HyperText Markup Language") is a language to describe the contents of web documents. It uses a special syntax containing markers (called "elements") which are wrapped around the text within the document to indicate how user agents (eg. web browsers) should interpret that portion of the document.

A user agent is any software that is used to access web pages on behalf of users. There is an important distinction to be made here—all types of desktop browser software (Internet Explorer, Opera, Firefox, Safari, Chrome etc.) and alternative browsers for other devices (such as the Wii Internet channel, and mobile phone browsers such as Opera Mini and WebKit on the iPhone) are user agents, but not all user agents are browser software. The automated programs that Google and Yahoo! use to index the web for their search engines are also user agents, but no human being is controlling them directly.

What HTML looks like

HTML is just a plain textual representation of content and its general meaning. For example:

```
<p id="example">This is a paragraph.</p>
```

The "** **" part is a marker (which we refer to as a "tag") that means "what follows should be considered as a paragraph". Because it is at the start of the content it is affecting, this particular tag is an "opening tag". The "** **" is a tag to indicate where the end of the paragraph is (which we refer to as a "closing tag"). The opening tag, closing tag and everything in between is called an "element". The **id="example"** is an attribute; you'll learn more about these later on. Many people use the terms element and tag interchangeably however, which is not strictly correct.

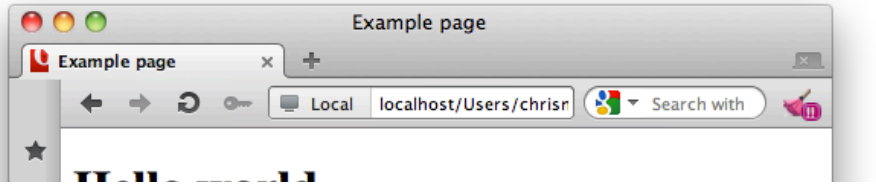
In most browsers there is a "Source" or "View Source" option, commonly under the "View" menu. Try this now - go to your favorite website, choose this option, and spend some time looking at the HTML that makes up the structure of the page.

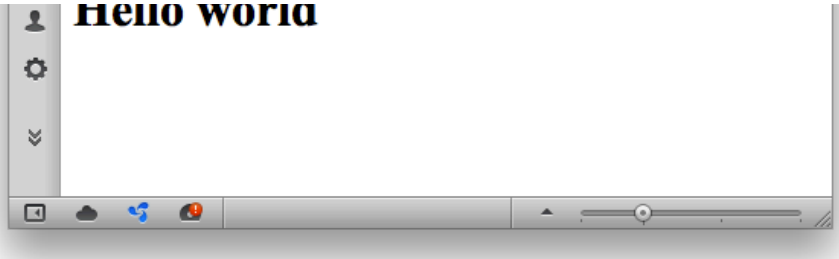
The structure of an HTML document

A typical example HTML document looks like so:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Example page</title>
  </head>
  <body>
    <h1>Hello world</h1>
  </body>
</html>
```

This looks like so when rendered in a web browser:





The document first starts with a document type element, or doctype. This mainly serves to get the browser to render the HTML in what is called "standards mode", so it will work correctly. It also lets validation software know what version of HTML to validate your code against. Don't worry too much about what this all means for now. We will come back to this later. What you can see here is the HTML5 doctype.

After this, you can see the opening tag of the **html** element. This is a wrapper around the entire document. The closing **html** tag is the last thing in any HTML document. The html element should always have a **lang**attribute. This specifies the primary language for the page. For example, **en** means "English", **fr** means "French". There are tools available to help you find the right language tag, such as Richard Ishida's Language Subtag Lookup tool.

Inside the **html** element, there is the head element. This is a wrapper to contain information about the document (the metadata). This is described in more detail in The HTML head element. Inside the head is the title element, which defines the “Example page” heading in the menu bar. Your **head** element should always contain a **meta** element with a **charset** attribute that identifies the character encoding of the page. (The one exception is when the page is encoded in UTF-16, but you should avoid that encoding anyway.) You should use UTF-8 whenever possible. Read more about character encodings.

After the **head** element there is a body element, which is the wrapper to contain the actual content of the page—in this case, only a level-one header (`h1`) element, which contains the text “Hello world.”
And that’s our document in full.

Elements often contain other elements. The body of the document will invariably end up involving many nested elements. Structural elements such as **article**, **header** and **div** create the overall structure of the document, and will contain subdivisions. These will contain headings, paragraphs, lists and so on. Paragraphs can contain elements that make links to other documents, quotes, emphasis and so on. You will find out more about these elements in later articles.

The syntax of HTML elements

A basic element in HTML consists of two markers around a block of text, and in almost every case elements can contain sub-elements (such as **html** containing **head** and **body** in the example above). There are some exceptions to the rule: some elements do not contain text or sub-elements, for example **img**. You'll learn more about these later on.

Elements can also have attributes, which can modify the behaviour of the element and introduce extra meaning. Let's have a look at another example.

```
<header>
  <h1>The Basics of
    <abbr title="Hypertext Markup Language">HTML</abbr>
  </h1>
</header>
```

This looks like so when rendered:

In this example a **header** element (used to mark up header sections of documents) contains an **h1** element (first, or most important level header), which in turn contains some text. Part of that text is wrapped in an **abbr**element (used to specify the expansion of abbreviations), which has a **title** attribute, the value of which is set to**Hypertext Markup Language**.

Many attributes in HTML are common to all elements, though some are specific to a given element or elements. They are always of the form **keyword="value"**. The value is often surrounded by single or double quotes: this is not necessary in HTML5, except when the attribute value has multiple words, in which case you need to use quotes to make it clear that it is a single attribute value, and not several attributes. Saying all this, I would

however recommend that you stick to quoting values for now, as it is good practice and can make the code easier to read. In addition, some HTML flavours you may work with in the future DO require quoting of attributes, for example XHTML 1.0, and it doesn't hurt to do so in flavours that don't.

Attributes and their possible values are mostly defined by the HTML specifications—you cannot make up your own attributes without making the HTML invalid, as this can confuse user agents and cause problems interpreting the web page correctly. The only real exceptions are the **id** and **class** attributes—their values are entirely under your control, as they are for defining custom meanings .

An element within another element is referred to as being a “child” of that element. So in the above example,**abbr** is a child of the **h1**, which is itself a child of the **header**. Conversely, the **header** element would be referred to as a “parent” of the **h1** element. This parent/child concept is important, as it forms the basis of CSS and is heavily used in JavaScript.

Block level and inline elements

There are two general categories of elements in HTML, which correspond to the types of content and structure those elements represent—block level elements and inline elements.

Block level means a higher level element, normally informing the structure of the document. It may help to think of block level elements being those that start on a new line, breaking away from what went before. Some common block level elements include paragraphs, list items, headings and tables.

Inline elements are those that are contained within block level structural elements and surround only small parts of the document’s content, not entire paragrahs and groupings of content. An inline element will not cause a new line to appear in the document: they are the kind of elements that would appear in a paragraph of text. Some common inline elements include hypertext links, highlighted words or phrases and short quotations.

Note: HTML5 redefines the element categories in HTML: see Element content categories. While these definitions are more accurate and less ambiguous than the ones that went before, they are a lot more complicated to understand than "block" and "inline". We will therefore stick with these throughout this course.

Character references

One last item to mention in an HTML document is how to include special characters. In HTML the characters **<**, **]]**and **&** are special. They start and end parts of the HTML document, rather than representing the characters less-than, greater-than and ampersand. For this reason, they must always be used in escaped form in content.

Other than for these characters, you should try to avoid using character references unless you are dealing with an invisible or ambiguous character. If you use the UTF-8 character encoding you can represent any character (other than the three mentioned above) without escaping.

One of the earliest mistakes a web author can make is to use an ampersand in a document and then have something unexpected appear. For example, writing “Imperial units measure weight in stones&pounds” could actually end up appearing as “…stones£s” in some browsers.

This is because the literal string “£” is actually a character reference in HTML. A character reference is a way of including a character into a document that is difficult or impossible to enter using a keyboard, or in a particular document encoding.

The ampersand (&) introduces the reference and the semi-colon (;) ends it. However, many user agents can be quite forgiving of HTML mistakes such as leaving out the semi-colon, and treat “&pound” as a character reference. References can either be numbers (numeric references) or shorthand words (entity references).

An actual ampersand has to be entered into a document as "&", which is the character entity reference, or as "&" which is the numeric reference. Web Platform Docs includes a Table of Common HTML Entities for reference.

For more information about working with character escapes see Using character escapes in markup and CSS.

****

CSS

What is CSS?

Whilst HTML structures the document and tells browsers the function of page element (Markup indicates a link to another page, or specifies a page heading), CSS provides rules that instruct the browser how to display a certain element — styling, spacing, coloring, etc. If HTML is the foundation and bricks that make up the structure of a house, CSS is the plaster and paint that decorate it.

CSS styles are applied using a system of rules (or rulesets). The exact syntax for CSS rules is described below. CSS rules:
Identify which HTML elements should have styling applied
Specify the "properties" (color, size, font, and other attributes) of the styled HTML elements
Contain the values that control the appearance of each property
For example, a CSS rule might state:

> *I want to find every* `<h2>` *element on a page, and change the color of text surrounded by these tags to green.*

or

> *I want to find every* `<p>` *element with a class attribute of* `author-name` *, set the background color to red, resize the text to twice the size of normal paragraph text, and add 10 pixels of spacing around each instance.*

CSS is not a programming language like JavaScript and it is not a markup language like HTML — actually there is nothing that can be compared to it. Technologies that defined interfaces before web development always mixed presentation and structure. As we've discussed earlier in the course, this is not a clever thing to do in an environment that changes as often as the Web, which is why CSS was invented.

Defining style rules

Style rules can operate on different sets of items, selected in different ways. For example, they can be directly applied to HTML elements (body, article, nav, list, p, em, strong, etc.), or they can be applied to any elements with custom made classes or IDs . This is the basic form:

```
selector {
  property1: value;
  property2: value;
  property3: value;
}
```

The pertinent parts are as follows:

- The selector identifies which HTML elements are affected by the rule, using actual element names, such as**<body>**, or other identifiers such as **class** attribute values. When applied to the actual element names, selector is simply replaced by the name of that block. For classes the syntax is .classname, for IDs, #idname. A more complete description comes later.
- The curly braces contain the property/value pairs, which are separated from each other by semi-colons; the properties are separated from their respective values by colons.
- The properties define what you want to do to the element(s) you have selected. These come in wide varieties, which can affect attributes such as text color, background color, the position of the element on the page, font type, border color and thickness, and many other appearance and layout controls.
- The values are the settings that specify details of each property applied to elements. The values are dependent on the property. For example, properties that affect color can use hexadecimal colors like #336699, RGB values like rgb(12,134,22) or color names like red, green, or blue. Properties that affect position, margins, width, height, and others can be measured in pixels, ems, percentages, centimeters, or other units.

Review this specific example:

```
p {
  margin: 5px;
  font-family: arial;
  color: blue;
}
```

The HTML element this rule affects is **<p>**, ie every **<p>** in the HTML document or documents that this CSS rule is applied to will display with these styles, unless they have more specific rules also applied to them, in which case the more specific rules will override this rule. The properties affected by this rule are the margins around the paragraphs, the font of the text inside the paragraph tags, and the color of that text. The margins are set to 5 pixels, the font is set as Arial, and the color of the text is set to blue.

A whole set of CSS rules are added to a CSS document to form a style sheet. This is the most basic syntax when writing CSS rules. Some rules are more complex, but not much — probably the coolest thing about CSS is its simplicity.

Whitespace in CSS

Note that whitespace in CSS works in exactly the same way as it does in HTML — excess whitespace is completely ignored by the browser that renders the CSS, so in most cases* you can add as much whitespace as you like to make the code easier to read. So this rule:

```
p {
  margin: 5px;
  font-family: arial;
  color: blue;
}
```

Is functionally identical to this rule:

```
p {margin: 5px;font-family: arial;color: blue;}
```

* There are a few exceptions where whitespace matters. For example, inside CSS function syntax, adding whitespace matters. So the property value **url(background-image.png)** would not work properly if you put a space between url and the rest of it, for example **url (background-image.png)**. But in

general, as long as you include the necessary curly braces, colons, and semi-colons to separate out different parts, the browser understands the values you apply to properties.

CSS comments

One thing to know early on is how to add comments in CSS. You add comments by enclosing them in **/*** and ***/**. Comments can span several lines. Browsers will ignore commented lines of text:

```
/* These are basic element selectors */
selector{
  property1: value;
  property2: value;
  property3: value;
}
```

You can add comments either between rules or inside the property block. For example, in the following CSS the second and third properties are enclosed inside comment delimiters, so they will be ignored by the browser. This is useful when you are testing the effect certain CSS rules have on your web page; just comment them out, save your CSS file, and reload the HTML page in a browser.

```
selector{
  property1: value;
  /*
  property2: value;
  property3: value;
  */
}
```

Grouping selectors

You can also group different selectors. If you want to apply the same style to **h1** and **p**, you can write the following CSS:

```
h1 {
  color: red;
}

p {
  color: red;
}
```

This however is not ideal, as you are repeating the same information. To remedy this, you can shorten the CSS by grouping the selectors together with a comma — the rules within the curly braces are applied to both selectors:

```
h1, p {
  color: red;
}
```

Basic types of selector

There are several different selectors, each matching a different part of the markup. The three types of selectors that you will encounter most often are:

Element selector

```
p {}
```

An element selector matches all the elements of that name on the page (**<p>** elements, in the case above). By specifying an HTML tag, you can affect all page elements that are surrounded by that tag.

Class selector

```
.example {}
```

A class selector matches all elements that have a **class** attribute with the value specified, so the above would match **<p class="example">**, **<li class="example">** or ** **, or any other element with a **class** of **example**. Note that class selectors do not test for any specific element name.

ID selector

```
#example {}
```

An ID selector matches all elements that have an **id** attribute with the value specified, so the above would match**<p id="example">**, **<li**

id="example"> or ** **, or any other element with an **id**of **example**. Note that ID selectors do not test for any element name. You can only have one of each ID per HTML document — they are unique to each page.

You can see the above selectors in action in the following examples. Notice that when you open the example in a browser the **warning** style is applied to both the list item and the paragraph (if the bullet disappears it is because you are setting a white text color on a white background).

example-selectors.html
selectors.css

Combining selectors

You can join selectors together to define even more specific rules:

- **p.warning {}** matches all paragraphs with a **class** of **warning**.
- **div#example {}** matches the element with an **id** attribute of **example**, but only when it is a **div**.
- **p.info, li.highlight {}** matches paragraphs with a **class** of **info** and list items with a **class** of**highlight**.

Note that this does not mean that you can use a shorthand for the definition of your elements in HTML. For example, your HTML paragraph will still have to be in the form **<p class="classname">**, but you can style it specially in your CSS with **p.classname {}**.

In the following example we are using these to differentiate between the different warning styles:

- example-specificselectors.html
- specificselectors.css

CSS shorthand

Another thing you'll come across regularly in this course is CSS shorthand. It is possible to combine several related CSS properties together into one property to save time and effort on your part. In this section we will look at the available types of shorthand.

For example, the **border** property is shorthand.

```
border: 2px solid black;
```

is equivalent to

```
border-width: 2px;
border-style: solid;
border-color: black;
```

Comparing individual and shorthand values

Consider the following margin rule:

```
div.foo {
  margin-top: 1em;
  margin-right: 1.5em;
  margin-bottom: 2em;
  margin-left: 2.5em;
}
```

Such a rule could also be written as:

```
div.foo {
  margin: 1em 1.5em 2em 2.5em;
}
```

These types of property can take less than four values too, as follows:

1. Same value applied to all four sides — **margin: 2px;**
2. First value applied to the top and bottom, second to the left and right — **margin: 2px 5px;**.
3. First and third values applied to the top and bottom respectively, second value applied to the left and right —**margin: 2px 5px 1px;**.

There are a number of properties that work like this, including **padding**, **margin** and **outline**. You'll find more out about these later on.

Making the choice to use a single property or a shorthand value

- Shorthand **margin** and **padding** properties tend to get the greatest share of use, though there are situations in which the shorthand properties are best avoided, or at least considered carefully, such as in the following situations:
- **When a single margin needs to be set.** In a situation where only one property needs to be set, the act of simultaneously setting multiple properties usually violates the KISS (Keep It Simple, Stupid) Principle.

- **The selector to which your properties apply is subject to many edge cases.** When this happens — which it will, sooner or later — the inevitable heap of shorthand values can become hard to follow when it comes time to repair or alter your layout.
- **The style sheet you're writing will be maintained by people with limited CSS skills (or spatial reasoning ability).** If you can get them to read this article you may not need to worry about this scenario, but it is best not to make any assumptions.
- **You need to supplant a value, to account for an edge case.** This requirement is often a signal of a poorly designed document or style sheet, but those are hardly unheard of.

Applying CSS to HTML

There are three ways to apply CSS to an HTML document: inline styles, embedded styles and external style sheets. Unless you have a very good reason to use one of the first two always go for the third option. The reason for this will become obvious to you soon, but first here is an overview of the different options.

Inline styles

You can apply styles to a specific element using a **style** attribute, like this:

```
<p style="background:blue; color:white; padding:5px;">Paragraph</p>
```

Inside this attribute you list all the CSS properties and their values. Each property/value pair is separated from the others by a semi-colon, and each property is separated from its value within each pair by a colon. Try viewing the source of this example.

If you open this example in a browser you will see that the paragraph with the **style** attribute is blue with white text and has a different size from the others, as shown in Figure 1.

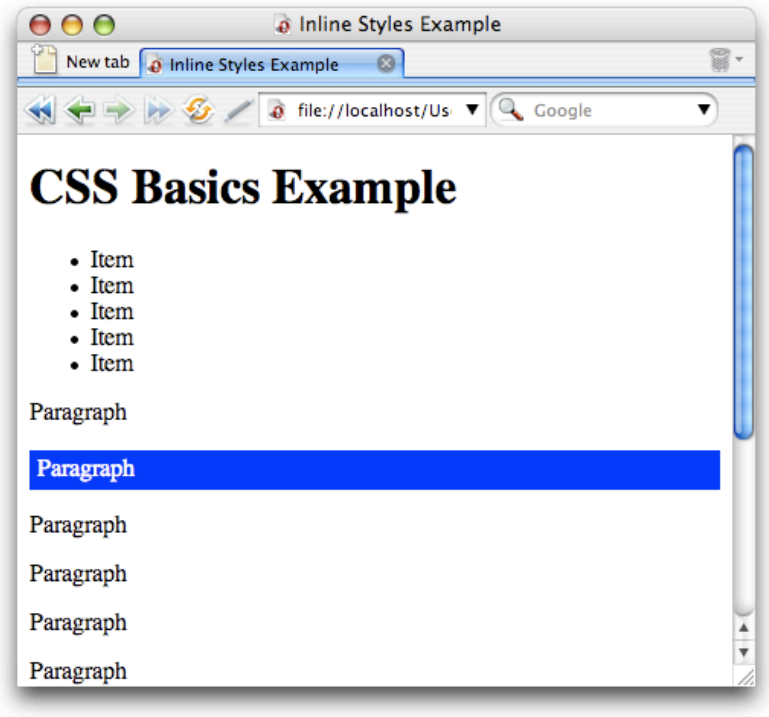


Figure 1: the paragraph with the applied inline styles differently from the others.

The benefit of inline styles is that the browser will be forced to use these settings. Any styles defined in other style sheets or even embedded in the **<head>** of the document will be overridden by these styles.

The big problem with inline styles is that they make maintenance a lot harder than it should be. Using CSS is all about separating the presentation of the document from the structure, but inline styles are doing just the opposite — scattering presentation rules throughout the document.

In addition to the maintenance issue you do not take advantage of the most powerful part of CSS: the cascade. The concept of the cascade is discussed in more detail later, but for now all you need to know is that using the cascade means you define a look and feel in one place and the browser applies it to all the elements that match a certain rule.

Embedded styles

Embedded styles are placed in the **<head>** of the document inside a **<style>** element as in this example:

```
<style type="text/css" media="screen">
  p {
```

```
    color: white;
    background: blue;
    padding: 5px;
  }
</style>
```

If you open the above link in a browser you will see that the defined styles get applied to all the paragraphs in the document, as shown in Figure 2. Also try looking at the example page's source to see the CSS inside the **head**.

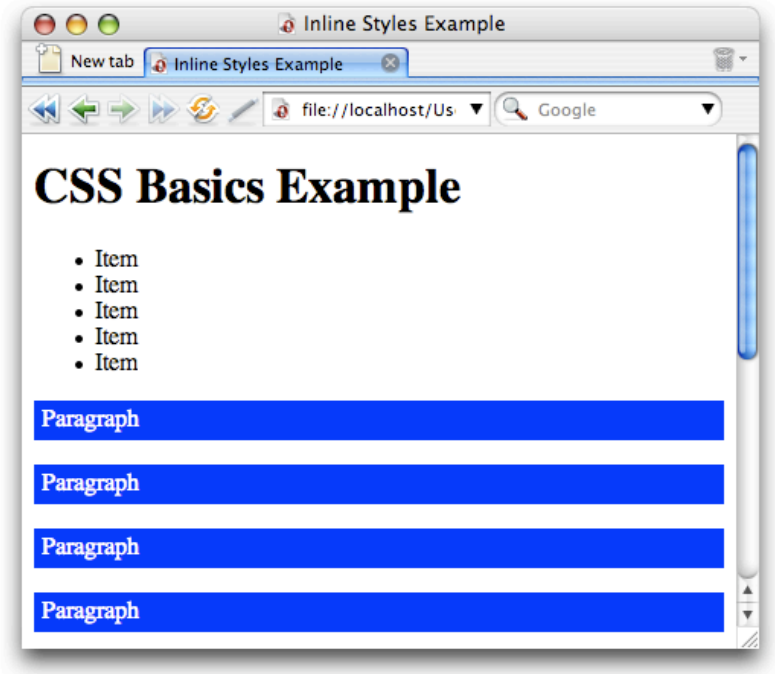


Figure 2: all paragraphs with the styles defined in the embedded style sheet.

The benefit with embedded styles is that you don't need to add a **style** attribute to each paragraph — you can style them all with one single definition. This also means that if you need to change the look and feel of all paragraphs, you can do it in one single location, however this is still limited to one document — what if you want to define the look of paragraphs for a whole site in one go? For this we use external style sheets.

External style sheets

External style sheets means putting all your CSS definitions in their own file, saving it with a file extension of **.css**, and then applying it to your HTML documents using a **<link>** element inside the document **<head>**. View source of our example page, and note that the **<head>** contains a **<link>** element that references this external CSS file, and verify that the styles defined in the external CSS file are applied to the HTML. Let's have a closer look at that **<link>** element:

```
<link rel="stylesheet" href="styles.css" type="text/css" media="screen">
```

We've talked about the **<link>** element before in this course. Just to recap: the **href** attribute points to the CSS file, the **media** attribute defines which media should get these styles applied to it (**screen** in this case as we don't want a printout to look like this) and the **type** defines what the linked resource is (a file extension is not enough to determine that).

Figure 3: the styles defined in the external style sheet when it is linked with a link element.

This is the best of all worlds: you keep your look and feel definitions all in one single file, which means that you can make site-wide changes by changing one file, and the browser can load that once and then cache it for all other documents that reference it, meaning less to download.

****

MISCELLANEOUS

linking to a stylesheet(s) from our .html pages:

at the top of each file representing a page of your website you need to specify a stylesheet you would like to use for that page, if you have kept styles for all your pages in one stylesheet then, just make sure that you are linking to that same file from all pages' files

```
<head>
...
    <link rel="stylesheet" href="css/normalize.css" type="text/css">
    <link rel="stylesheet" href="css/style.css" type="text/css">
...
</head
```

the order in which you list files matters because you override stylesheets with subsequent ones, above we have normalize.css followed by our own style.css, meaning that normalize will apply its styles to make all browsers show our content in a similar manner to begin with and then we override that with our style.css

type="text/css" is not compulsory, and if you omit it, browser will take it as that by default

***
line-height (CSS property):

Syntax:

```
Formal syntax:

line-height: normal
line-height: 3.5    /* <number> values */
line-height: 3em    /* <length> values */
line-height: 34%    /* <percentage values */
line-height: inherit
```

Values:

<number>
The used value is this unitless `<number>` multiplied by the element's font size. The computed value is the specified `<number>` times the height of the font-size used. In most cases **this is the preferred way** to set `line-height` with no unexpected results in case of inheritance.
<length>
The specified `<length>` is used in the calculation of the line box height. **em** is one of the units of `<length>` and it represents the calculated and inherited size of the element.
(Bear in mind that if you are setting the *line-height* in ***em*** it will rely on the *font-size* used previously in your stylesheet, and if that *font-size* was in turn set using ***em*** based on the font-size set even further up in your stylesheet this can make it harder to calculate the final *line-height*.
Someone in the class asked "em" vs "rem"? in terms of measure they are the same, the only difference is that em sets the size based on the first previous element's size, while rem sets the size based on the first element in such chain. "rem' is not supported by IE8, which is still in use.)
<percentage>
Relative to the font size of the element itself. The computed value is this percentage multiplied by the element's computed font size.
Percentage and **em** values may have unexpected results.
normal
Depends on the user agent. Desktop browsers (including Firefox) use a default value of roughly `1.2`, depending on the element's `font-family`.

Examples:

Examples:

```
/* All rules below have the same resultant line height */
div { line-height: 1.2;   font-size: 10pt }   /* number */
div { line-height: 1.2em; font-size: 10pt }   /* length */
div { line-height: 120%;  font-size: 10pt }   /* percentage */
div { line-height: 12pt;  font-size: 10pt }   /* length */
div { font: 10pt/1.2  Georgia,"Bitstream Charter",serif }
```

generally for paragraph font size of 1.2-1.5 em has good readability

***
CSS properties mentioned in class:
letter-spacing: 50px;
text-transform: uppercase
text-align: center; (US English spelling)
text-decoration: underline;
border-top: 1px solid black;
border-bottom: 2px dotted black;
display: inline-block;
margin: 0 auto;
(border style options: solid, dotted, dashed)

****
Couple of sites to use for reference on HTML, CSS and more:

http://www.webplatform.org/ - good for entry level reference, however is not as complete as MDN
https://developer.mozilla.org/ - a bit more technical language is used

***
when using selectors to target html elements in your stylesheet only be as specific as you need to, so use the least amount possible of selectors while making sure you are targeting what you want

***
selecting an element in your stylesheet can be done by typing the name of the element or referring to a class or id of that element

***
in html multiple classes for the same element are separated with space:
...
in css they are concatenated:
.intro.newsflash {....}

***
links have pseudo-classes, pseudo classes for links descrive their "state" examples:
a:link (targets only <a> tags which are links as opposed to the on-page anchor)
a:hover
a:active
a:visited
a:focus
(some other elements also have pseudo-classes, but those for links you will use most)

***
html elements come in 2 main types: block and inline. there is, however, also inline-block...

block takes up the whole width of the line, no matter if their own width is less than the width of the line
inline elements only take up just enough width for itself on the line

inline-block it's a way to make elements inline, but preserving their block capabilities such as setting width and height, top and bottom margins and paddings etc.

***
BOX-MODEL
http://www.hicksdesign.co.uk/boxmodel/ - gives a good visualisation of how padding, border and margin work together and the order in which elements are stacked
box-model hack was mentioned, you will come back to it in later classes

***
browser web inspector is accessed by right-clicking on any page element and selecting "Inspect element"
use web inspector a lot!!! you might want to use it on the sites which are of interest to you to see what is going on there

***
vertical centering is more difficult than horizontal centering

***
width can be specified in pixels or percent, when you specify it in percent the element will take width which is that specified percentage of the screen width's available.

***
margin:

```
margin: 5%;                 /* all sides 5% margin */

margin: 10px;               /* all sides 10px margin */

margin: 1.6em 20px;         /* top and bottom 1.6em, left and right 20px margin */

margin: 10px 3% 1em;        /* top 10px, left and right 3%, bottom 1em margin */

margin: 10px 3px 30px 5px; /* top 10px, right 3px, bottom 30px, left 5px margin */

margin: 1em auto;           /* 1em margin on top and bottom, box is horizontally centered */

margin: auto;               /* box is horizontally centered, 0 margin on top and bottom */
```

***
beware of the styles set by normalize.css, occasionally you will need to override styles defined there in your own style.css

styles overridden are shown as stricken-through in web inspector

***
semantic context for the on-page content is important, therefore using correct html tags to wrap your content in is essential. one of the reasons is for search engine optimization