**HTML (**<https://www.w3schools.com/html/default.asp>**)**

**What is HTML?**

Hyper Text Markup Language (HTML) is the standard markup language for creating Web pages. It is NOT a programming language, but it is a language with its own syntax used to define the content of an HTML page, i.e., what is to be displayed. It is the building block of the Web. How the HTML elements are displayed on the browser screen (layout) is specified by CSS, which we will look at in Weeks 2 & 3.

Here is your first HTML document:

|  |
| --- |
| <!DOCTYPE html> <html>  <head> <title>Page Title</title> </head>  <body> <!-- This is a comment: Here starts the body of the page --> <h1>This is a heading</h1> <p>This is a paragraph.</p> <p>This is another paragraph.</p>  </body> </html> |

Look at 01-HTML/01-First/index.html

Let’s now analyze the anatomy of an HTML page:

* Every HTML document starts with the <!DOCTYPE html> declaration that defines this document to be HTML5 and represents the document type, which helps browsers display the document or page correctly. But even if you omit this, the browser will assume the document type to be HTML5 by default, and still parse the file correctly. There are other former HTML document types such as HTML4, XHTML etc., but we will not cover them in this document.
* After the document type, the rest of the file is the HTML page, which is nothing but a collection of HTML elements. An HTML element is the basic building block of an HTML page, and represents some information within page. It is typically defined by two HTML tags: A start or open tag, and an end or close tag. While some of these elements contain information that is to be displayed, some contain only semantic information, which does not get displayed but is there to give information to the browser. Here are the list of HTML elements in our example document:

<html>content goes here…</html>

<head>content goes here…</head>

<title>content goes here…</title>

<body>content goes here…</body>

<h1>content goes here…</h1>

<p>content goes here…</p>

* To start with, every HTML document must be enclosed within an “html” element, which is the root of the page.
* An HTML document consists of at least two sections: A header, enclosed within the “head” element, and a body, enclosed within the “body” element.
* The <head> element contains meta information about the document. What’s inside the head element is not displayed. They are just there to give some information to the browser.
* The <body> element contains the visible page content. Only the content inside the “body” element is displayed. To see the actual HTML document being displayed inside the browser, press Ctrl+U inside the browser.
* The <title> element, which is placed inside the head section, specifies a title for the document, which gets displayed as the name of the document on the browser.
* The <h1> element defines a large heading
* The <p> element defines a paragraph
* Comments are enclosed in between <!-- --> tags
* HTML tags are NOT case sensitive: <P> means the same as <p>.

**HTML Page Structure**

Below is a visualization of an HTML page structure. Notice that only the elements inside the “body” element (white area) get displayed. What’s inside the “head” element contains only meta-data information.



**HTML Elements**

An HTML element is the basic building block of an HTML document. That is, an HTML document is nothing but a collection of HTML elements. An HTML element is *typically* defined by an open and a close HTML tag. A tag is an HTML element name surrounded by angle brackets:

<tagname>content goes here...</tagname>

* HTML tags normally come in pairs like <p> and </p>. There are a few HTML elements that have only the start tag but no end tag, e.g., line break <br>, horizontal break <hr> elements.
* The first tag in a pair is the start tag, a.k.a., open tag, and the second tag is the end tag, a.k.a., the close tag
* The end tag is written like the start tag, but with a forward slash inserted before the tag name
* For more info, refer to (<https://clearlydecoded.com/anatomy-of-html-tag>)

**HTML Attributes**

Attributes provide additional information to HTML elements. They are put inside the start tag as **attributeName=**"**value**" pairs, where value is specified in double quotes. An element can have many attributes as the following examples show:

<a href="https://www.w3schools.com">This is a link</a>

<img src="img\_girl.jpg" alt="Girl with a jacket" width="500" height="600">

Look at 01-HTML/02-Attributes

For a complete list of HTML attributes and their usage, refer to: <https://www.w3schools.com/tags/ref_attributes.asp>

**id & class attributes**

The **id** attribute specifies a *unique id* for an HTML element (the value must be unique within the HTML document). The id attribute is mostly used to get access to the HTML element from within CSS style sheets or from within the JavaScript code (via the HTML DOM) to manipulate it.

<h1 id="header1">Hello World!</h1>

The **class** attribute specifies one or more class names for an element. While the **id** attribute must uniquely identify an element within the HTML document, more than one HTML element can belong to the same class. Thus it would be possible to manipulate all HTML elements that belong to the same class from within CSS style sheets and in JS code. For example, we may want to change the text color, font, size etc. of all HTML elements that belong to the same class using JS code.

<h1 class="header1">Header 1</h1>  
<p class="important">An important paragraph.</p>  
<p>A normal paragraph.</p>

<p class="important">Another important paragraph.</p>

<h1 class="header1 important">A header that belongs to two classes.</h1>

**style attribute**

The style attribute is used to specify the styling of an element, like color, font, size etc. When there are more than one styles specifies within the style attribute, they are separated by semi-colon. Specifying element styles using the style attribute is called in-line CSS and is NOT recommended. Instead, all element styles should be specified in a separate style sheet (CSS) document. We will look at CSS next week. Here is an example of in-line CSS:

<p style="color:blue">This is a blue paragraph.</p>

<p style="color:red;font-size:20px">This is a red paragraph. The font-size is 20px</p>

Look at 01-HTML/03-InlineCSS

**inline & block-level Elements**

Every HTML element has a default display value depending on what type of element it is. The two display values are: **block** and **inline**.

A **block-level** element *always starts on a new line and takes up the full width available* (stretches out to the left and right as far as it can). For example, a *heading* and *paragraph* elements are block level elements.

An **inline** element, however, does not start on a new line, but only takes up as much width as necessary within the line it is placed. For example, *span* is an inline element. Other inline elements are img, a etc.

<p>This is an inline span <span style="border: 1px solid black">Hello World</span> element inside a paragraph.</p>

For a complete list of inline and block elements, look at: <https://www.w3schools.com/html/html_blocks.asp>

Now let’s look at some of the most important HTML Elements:

Look at 01-HTML/04-Elements

1. **Headings**

There are 6 different heading types in HTML: h1, h2, h3, h4, h5 and h6.

|  |
| --- |
| <h1>This is a h1 heading</h1>  <h2>This is a h2 heading</h2>  <h3>This is a h3 heading</h3>  <h4>This is a h4 heading</h4>  <h5>This is a h5 heading</h5>  <h6>This is a h6 heading</h6> |

When you load this page up, you will see that h1 has the biggest font-size and h6 has the smallest font-size. There are also some default paddings around the headings. You can in fact change the font-size, font-family, padding etc. of these headings using CSS, which we will look at next week. Notice also that headings are block-elements. That is, each heading element occupies the entire row it is placed in. That’s why each heading element in the document occurs in a different line.

1. **Paragraph**

Defined by the “p” tag, which is a block-level element. If you open Chrome Developer Tools and look at the Elements, you will see all the CSS used to format this and other elements.

|  |
| --- |
| <p>Lorem ipsum, dolor sit amet consectetur adipisicing elit.  Another sentence in the paragraph.  </p> |

Notice that line breaks that you manually insert inside a paragraph are not preserved. If you want to have a line break to be displayed, you have to insert a <br> element yourself. Alternatively, if you think that the text you write should appear as is, i.e., verbatim, on the browser, then use a <pre> element rather than a <p> element:

|  |
| --- |
| <pre>  This is line 1.  This is line 2.  </pre> |

The above text will appear as two lines of text. But if you put this text inside a <p> element, both will appear on the same line.

1. **Some Inline Elements for Formatting Text**

Let’s now look at some inline-elements such as **strong**, which is used to make certain parts of a text bold-face by default. In the following example, “important” will appear bold-face.

|  |
| --- |
| This is a very <strong>important</strong> problem. |

You can grap onto this tag from within your CSS and change the default behaviour. For example, you can change the font-size, color, make it italic etc. The idea with strong is to specify that the text within the strong element must stick out somehow.

“em” is another inline element used to emphasize a piece of text. By default it makes the text italic, but you can change the default behaviour using CSS:

|  |
| --- |
| This is a very <em>important</em> problem. |

“abbr” is another inline element used to specify an abbreviation:

|  |
| --- |
| The <abbr title=”World Wide Web”>WWW</abbr> is awesome. |

Here are some of the other text formatting elements in HTML with their meaning:

<b> - Bold text

<strong> - Important text

<i> - Italic text

<em> - Emphasized text

<mark> - Marked text

<small> - Small text

<del> - Deleted text

<ins> - Inserted text

<sub> - Subscript text

<sup> - Superscript text

1. **Links**

Links allow users to click their way from page to page. You can click on a link and jump to another document.

<a href="*url*">*link text*</a>

<a href="http://google.com">Go to google</a>

When the user presses the link, the browser will overload google.com on top of the current tab. If you want the new page to open in a separate tab, you must specify a target=”\_blank” attribute as follows:

<a href=<https://google.com> target="\_blank">Go to google</a>

This way, the user never leaves your site, but the new page opens in a new tab. You should use target=”\_blank” if you are going to an external site. If you are diverting the user to another page in your own site, then you should not use it.

1. **Images**

Images are added using the “img” element, which does not have a closing tag:

<img src="img\_girl.jpg" alt="Girl in a jacket">

“img” is an inline element, so after displaying the image, the cursor stays on the same line. So the next element will appear to the right of the image. If you want the next element to appear on the next line, then you must move the cursor to the next line using <br> before putting the next element.

**Image Maps**

The <map> tag defines an image-map. An image-map is an image with clickable areas.

|  |
| --- |
| <img src="workplace.jpg" alt="Workplace" usemap="#workmap">  <map name="workmap">   <area shape="rect" coords="34,44,270,350" alt="Computer" href="#Computer">   <area shape="rect" coords="290,172,333,250" alt="Phone" href="#Phone">   <area shape="circle" coords="337,300,44" alt="Coffee" href="#Coffee"> </map> |

Shape can be “rect”, “circle”, “poly”, “default”, which means the entire image.

1. **Lists**

There are two types of lists: **unordered** list and **ordered** list.

An unordered list starts with the <ul> tag. Each list item starts with the <li> tag. The list items will be marked with bullets (small black circles) by default.

|  |
| --- |
| <ul>  <li>Unordered List item 1</li>  <li>Unordered List item 2</li>  <li>Unordered List item 3</li>  <li>Unordered List item 4</li>  </ul> |

You can change many of the default properties of lists using CSS styling. For example:

<ul style="list-style-type:circle;">

will change the discs that appear before each list item to a circle.

An ordered list starts with the <ol> tag. Each list item starts with the <li> tag. The list items will be marked with numbers by default:

|  |
| --- |
| <ol>  <li>Ordered List item 1</li>  <li>Ordered List item 2</li>  <li>Ordered List item 3</li>  <li>Ordered List item 4</li>  </ol> |

You can use the type attribute to define the type of the list item marker as follows:

**Type Description**

type="1" The list items will be numbered with numbers (default)

type="A" The list items will be numbered with uppercase letters

type="a" The list items will be numbered with lowercase letters

type="I" The list items will be numbered with uppercase roman numbers

type="i" The list items will be numbered with lowercase roman numbers

|  |
| --- |
| <ol type="A">   <li>Coffee</li>   <li>Tea</li>   <li>Milk</li> </ol> |

By default, an ordered list will start counting from 1. Use the start attribute to start counting from a specified number:

|  |
| --- |
| <ol type="1" start="50">  <li>Coffee</li>  <li>Tea</li>  <li>Milk</li>  </ol> |

What if you want to have a list of items as a (key, value) pair. For example, you want to have a list with items with keys being the terms and the values being the definition of these terms. Then you use the definition list <dl> with <dt> marking the term (definition term) and <dd> marking the definition. Here is an example:

|  |
| --- |
| <dl>  <dt>Coffee</dt>  <dd>Black hot drink</dd>  <dt>Milk</dt>  <dd>White cold drink</dd>  </dl> |

1. **Tables**

An HTML table is defined with the <table> tag. Each table row is defined with the <tr> tag. A table header is defined with the <th> tag. By default, table headings are bold and centered. A table data/cell is defined with the <td> tag.

|  |
| --- |
| <table>  <thead>  <tr>  <th>Firstname</th>  <th>Lastname</th>  <th>Age</th>  </tr>  </thead>  <tbody>  <tr>  <td>Ali</td>  <td>Kuscu</td>  <td>50</td>  </tr>  <tr>  <td>Ayse</td>  <td>Kaptan</td>  <td>28</td>  </tr>  </tbody>  </table> |

Notice that when this table is displayed, there are no borders between table elements. Also note that the table headers are centered, while table data items are left-justified. But you can change this behavior using CSS styles.

You can have cells that span more than 1 column using “colspan” attribute:

|  |
| --- |
| <table>  <tr>  <th>Name</th>  <th colspan="2">Telephone</th>  </tr>  <tr>  <td>Ali Kuscu</td>  <td>55577854</td>  <td>55577855</td>  </tr>  </table> |

Similarly, you can have a table where a cell spans multiple rows using “rowspan” attribute:

|  |
| --- |
| <table>  <tr>  <th>Name:</th>  <td>Ali Kuscu</td>  </tr>  <tr>  <th rowspan="2">Telephone:</th>  <td>55577854</td>  </tr>  <tr>  <td>55577855</td>  </tr>  </table> |

You can add a caption to the table using the <caption> tag, which must be inserted immediately after the <table> tag.

|  |
| --- |
| <table>    <caption>Monthly savings</caption>   <tr>     <th>Month</th>     <th>Savings</th>   </tr>   <tr>     <td>January</td>     <td>$100</td>   </tr>   <tr>     <td>February</td>     <td>$50</td>   </tr> </table> |

In the old days, people used to use tables for HTML page layout. But this is NOT recommended anymore. Instead you use CSS for page layout. We will look at the HTML page layouts in Week 3.

1. **Forms (**<https://www.w3schools.com/html/html_forms.asp>**)**

The HTML <form> element defines a form that is used to collect user input. Here is the general structure of a form element:

<form>  
*form elements*  
</form>

Since forms are used to collect data from the user, it has many from elements ranging from input elements such as buttons, text fields, radio buttons, check boxes, etc. to other from elements such as drop-down lists, text areas, etc. Let’s look at the most common input element, which is a text field:

<form>

<label>First name: </label>

<input type="text" name="firstName">

<br>

<br>

<label>Last name: </label>

<input type="text" name="lastName">

<br>

</form>

Notice that both label and input are inline elements. That’s why they appear on the same line, one after the other. To place the second label and text-field on the next line, we can put line-break <br> elements to move the cursor to the next line. The second <br> in the above markup is to separate the two text fields. Alternatively, you can enclose the label and the input text field inside of a div container, which is a block-level element:

<form>

<div>

<label>First name: </label>

<input type="text" name="firstName">

</div>

<br>

<div>

<label>Last name: </label>

<input type="text" name="lastName">

</div>

</form>

The name attribute is the name of the variable to be used when sending the form element values to the server. The general convention for variable names is to use camelCase. That is, the name of the variable starts with a lower case character, and then every other word in the variable name starts with a capital letter. It is also possible to specify a default value for the text field using the value attribute as follows:

<form>

<label>First name: </label>

<input type="text" name="firstName" value="Ali">

<br>

<br>

<label>Last name: </label>

<input type="text" name="lastName" value="Kaya">

<br>

</form>

After the user enters all the necessary information, s/he must submit the form element values to some service for further processing. The submission is done using the “submit” element, which displays a Submit button, as follows:

<form>

<label>First name: </label>

<input type="text" name="firstName" value="Ali">

<br>

<br>

<label>Last name: </label>

<input type="text" name="lastName" value="Kaya">

<br>

<br>

<input type=”submit” value=”Submit”>

</form>

After the user enters all the required data, s/he should be able to submit the form input values to some form-handling service for processing. This service is usually a remote Web server, which would process the input values and generate a response. The place where the form element values will be submitted is specified in the form’s **action** attribute. If the action attribute is omitted, the action is set to the current page. That is, the form input values are submitted to the current page if the action attribute is omitted.

You also specify the type of HTTP method to be used for submission, GET or POST, in the method attribute. The default submission method is GET.

<form action="process.php" method="POST">  
*form elements*  
</form>

The difference between GET and POST method is that GET specifies the form element values as key/value pairs in the URL separated by &, which is not secure at all as the values are directly readable by anyone looking at the URL. For example, if we submit our form using GET, then we will see that it gets submitted to process.php with firstName and lastName specified in the url as key/value pairs separated by & character as follows:

/process.php?firstName=Ali&lastName=Kaya

Notice that “firstName” is the name of the variable specified in the name attribute. Similarly, “lastName” is the name of the variable specified in the name attribute. We should also note that the length of a URL is limited to 2048 characters. Using the GET method is OK for search requests, where you do not care if other people see your query string. For example, google search query is sent as a GET request as follows:

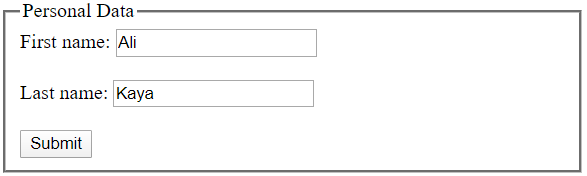
https://google.com/search?q=javascript&num=5&cr=countryTR

With the POST request, the key-value pairs are put inside the body of the HTTP message, which makes sure that the key/value pairs cannot be seen by other people assuming that we use the https protocol.

The <fieldset> element is used to group related data in a form. The <legend> element defines a caption for the <fieldset> element.

|  |
| --- |
| <form>  <fieldset>  <legend>Personal Data</legend>  <label>First name: </label>  <input type="text" name="firstName" value="Ali">  <br>  <br>  <label>Last name: </label>  <input type="text" name="lastName" value="Kaya">  <br>  <br>  <input type=”submit” value=”Submit”>  </fieldset>  </form> |

Here is how this looks on the browser:



**Other Input Elements**

In addition to text field, there are many other form input elements. The complete list can be found here (<https://www.w3schools.com/html/html_form_input_types.asp>). Let’s look at one example with checkbox and one example with radio button:

|  |
| --- |
| <form>  <input type="radio" name="gender" value="male" checked> Male<br>  <input type="radio" name="gender" value="female"> Female<br>  <input type="radio" name="gender" value="other"> Other  </form> |

When this form is submitted, the submitted key/value pairs would be gender=”male” assuming Male is selected by the user.

Here is a checkbox example.

|  |
| --- |
| <form>  <input type="checkbox" name="vehicle1" value="Bike"> I have a bike<br>  <input type="checkbox" name="vehicle2" value="Car"> I have a car  </form> |

Here is how we create a button inside of a form:

|  |
| --- |
| <form>  <input type="button" onclick="alert('Hello World!')" value="Click Me!">  </form> |

There is also a “button” element to create a button that allows added functionality that is not possible with input type=”button”. Specifically, it is possible to put “img” elements inside buttons created with the “button” element, whereas this is not possible with the input type=”button” elements.

|  |
| --- |
| <form>  <button type="button" onclick="alert('Hello World!')">Click Me!</button>  </form> |

HTML5 added some extra input elements including email, number, date etc. These allow type checking without the need for JS.

|  |
| --- |
| <form>  Email: <input type="email" name="email"><br><br>  Number: <input type="number" name="number" value="10"><br><br>  Date: <input type="date" name="date"><br><br>  <input type = "submit" value="Submit">  </form> |

**Other Form Elements**

The <select> element defines a drop-down list. The <option> element defines an option that can be selected:

|  |
| --- |
| <select name="car">   <option value="volvo" selected>Volvo</option>   <option value="saab">Saab</option>   <option value="fiat">Fiat</option>   <option value="audi">Audi</option> </select> |

The <textarea> element defines a multi-line input field (**a text area**):

|  |
| --- |
| <textarea name="message" rows="10" cols="30"> The cat was playing in the garden. </textarea> |

Finally, the form elements have many other attributes than “name” and “value” illustrated here. For a complete list of form attributes and how they are used, please refer to: <https://www.w3schools.com/html/html_form_attributes.asp>

**The <div> element**

The <div> is a block-level element that defines a container for other HTML elements. It is widely used to define a part of your page for special CSS styling.

|  |
| --- |
| <div style="background-color:black;color:white;padding:20px;">   <h2>London</h2>   <p>London is the capital city of England. It is the most populous city in the United Kingdom, with a metropolitan area of over 13 million inhabitants.</p> </div> |

**HTML5 Semantic Tags**

HTML5 defines several semantic elements to help search engines and browsers identify the correct Web page content. The semantic elements have no role in defining Web content or layout. They are there just to provide extra descriptive information to the search engines.

**Why Semantic Elements?**

Before HTML5, developers used their own id/class names to style elements: header, top, bottom, footer, menu, navigation, main, container, content, article, sidebar, topnav, etc. This made it impossible for search engines to identify the correct web page content. HTML5 defined several such semantic elements (<header> <footer> <nav> <section> <article>) to make this easier. According to the W3C, a Semantic Web: "Allows data to be shared and reused across applications, enterprises, and communities."

A semantic element clearly describes its meaning to both the browser and the developer. HTML5 offers the following semantic tags to define and describe different parts of a web page:

|  |  |
| --- | --- |
| * <article> * <aside> * <details> * <figcaption> * <figure> * <footer> * <header> * <main> * <mark> * <nav> * <section> * <summary> * <time> | HTML5 Semantic Elements |

Just to stress it again, you do not have to use any of these semantic elements and they provide no information as far as the Web content or layout is concerned. They are there just to give extra descriptive information to the search engines about the content of the Web page. So it is good practice to enclose different parts of your web page inside these semantic tags. Let’s now look at how we can use these semantic tags:

**HTML5 <section> Element**

The <section> element defines a section in a document. According to W3C's HTML5 documentation: "A section is a thematic grouping of content, typically with a heading." A home page could normally be split into sections for introduction, content, and contact information.

|  |
| --- |
| <section>   <h1>WWF</h1>   <p>The World Wide Fund for Nature (WWF) is....</p> </section> |

**HTML5 <article> Element**

The <article> element specifies independent, self-contained content. An article should make sense on its own, and it should be possible to read it independently from the rest of the web site. Examples of where an <article> element can be used:

* Forum post
* Blog post
* Newspaper article

|  |
| --- |
| <article>   <h1>What Does WWF Do?</h1>   <p>WWF's mission is to stop the degradation of our planet's natural environment,   and build a future in which humans live in harmony with nature.</p> </article> |

**Nesting <article> in <section> or Vice Versa?**

The<article> element specifies independent, self-contained content. The <section> element defines section in a document. Can we use the definitions to decide how to nest those elements? No, we cannot! So, on the Internet, you will find HTML pages with <section> elements containing <article> elements, and <article> elements containing <section> elements. You will also find pages with <section> elements containing <section> elements, and <article> elements containing <article> elements. Example for a newspaper: The sport <article> in the sport section, may have a technical section in each <article>.

**HTML5 <header> Element**

The <header> element specifies a header for a document or section. The <header> element should be used as a container for introductory content. You can have several <header> elements in one document. The following example defines a header for an article:

|  |
| --- |
| <article>   <header>     <h1>What Does WWF Do?</h1>     <p>WWF's mission:</p>   </header>   <p>WWF's mission is to stop the degradation of our planet's natural environment,   and build a future in which humans live in harmony with nature.</p> </article> |

**HTML5 <footer> Element**

The <footer> element specifies a footer for a document or section. A <footer> element should contain information about its containing element. A footer typically contains the author of the document, copyright information, links to terms of use, contact information, etc. You may have several <footer> elements in one document.

|  |
| --- |
| <footer>   <p>Posted by: Hege Refsnes</p>   <p>Contact information: <a href="mailto:someone@example.com">   someone@example.com</a>.</p> </footer> |

**HTML5 <nav> Element**

The <nav> element defines a set of navigation links. Notice that NOT all links of a document should be inside a <nav> element. The <nav> element is intended only for major block of navigation links.

|  |
| --- |
| <nav>   <a href="/html/">HTML</a> |   <a href="/css/">CSS</a> |   <a href="/js/">JavaScript</a> |   <a href="/jquery/">jQuery</a> </nav> |

**HTML5 <aside> Element**

The <aside> element defines some content aside from the content it is placed in (like a sidebar). The <aside> content should be related to the surrounding content.

|  |
| --- |
| <p>My family and I visited The Epcot center this summer.</p>  <aside>   <h4>Epcot Center</h4>   <p>The Epcot Center is a theme park in Disney World, Florida.</p> </aside> |

**HTML5 <figure> and <figcaption> Elements**

The purpose of a figure caption is to add a visual explanation to an image. In HTML5, an image and a caption can be grouped together in a <figure> element. The <img> element defines the image, the <figcaption> element defines the caption.

|  |
| --- |
| <figure>   <img src="pic\_trulli.jpg" alt="Trulli">   <figcaption>Fig1. - Trulli, Puglia, Italy.</figcaption> </figure> |

**The “meta” tag in the head section**

Before we finish our discussion on metadata tags, let’s talk about the “meta” tag that gives meta data information about the document and is placed inside the “head” element.

|  |
| --- |
| <meta name="description" content="HTML5 Cheat sheet: Examples of HTML5 elements">  <meta name="keywords" content="HTML5 tags, Web design"> |

The meta tag gives information about the document such as description and keywords, which are used by search engines in correctly indexing the page. Again, these do not get displayed to the user nor do they have any functionality as far as the content or layout of the page is concerned. But they give information to the search engines so that your page get indexed correctly.

**Using HTML5 Semantic Tags: An example blog post page**

Assume we have blog post page consisting of several posts. Let’s see how we can design such a page.

We will typically have a main header for the document. We can enclose this inside a header semantic element. We also give the head an id of “main-header” so that we can grab onto this header from within CSS for styling.

|  |
| --- |
| <header id="main-header">  <h1>My Blog Webpage</h1>  </header> |

Then we can the main part of page, which can be enclosed within a “section” semantic header. Within the main part of the page, we can have several posts, each enclosed within an “article” semantic header. We also give each post a class of “post” so that it can be grabbed and styled from within CSS.

|  |
| --- |
| <section>  <article class="post">  <h2>Post 1</h2>  <p>Posted on January 14, 2020 at 09:45am</p>  <p>Lorem ipsum dolor sit amet consectetur!</p>  <a href="post.html">Read more</a>  </article>  <section> |

We can then have a sidebar with different category of posts. This can be enclosed inside an “aside” semantic header and the categories can be enclosed inside a navigation, i.e., “nav”, semantic header:

|  |
| --- |
| <aside>  <h3>Categories</h3>  <nav>  <ul>  <li><a href="#">Category1</a></li>  <li><a href="#">Category2</a></li>  <li><a href="#">Category3</a></li>  </ul>  </nav>  </aside> |

Finally, we can have a main footer at the bottom of the page. We will also give this footer a unique id of “main-footer” so that it can be grabbed and formatted in CSS:

|  |
| --- |
| <footer id="main-footer">  Copyright &copy; 2020, John Doe.  </footer> |

Notice that we had no CSS, i.e., styling, in this page. That’s why it looks very ugly. Next week we will look at how we can use CSS to format such HTML pages.

Look at: 01-HTML/05-Blog

**Document Object Model (DOM) (**<https://www.w3schools.com/js/js_htmldom.asp>**)**

When a web page is loaded, the browser creates a Document Object Model (DOM) of the page. The DOM is a tree of objects. For example, consider the following HTML document:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>My title</title>  </head>  <body>  <h1>My header</h1>  <a href="...">My link</a>  </body>  </html> |

Here is the DOM that corresponds to this HTML page:



Each object in DOM has attributes and events. Given a DOM, it is possible to dynamically manipulate DOM using a client-side programming language such as JavaScript. JS can:

* add/change/remove HTML elements
* add/change/remove HTML attributes
* add/change/remove CSS styles
* react to HTML events
* add/change/remove HTML events

In other words, it is possible to dynamically manipulate the entire DOM using JS. But for JS to manipulate the DOM, it must have access to each and every HTML element, i.e., object, in DOM. Here are couple of ways to do this:

* Get HTML elements by id, e.g., var myElement = document.getElementById("intro");
* Get HTML elements by tag name, e.g., var x = document.getElementsByTagName("p");
* Get HTML elements by class name, e.g., var x = document.getElementsByClassName("intro");
* Get HTML elements by CSS selectors, e.g., var x = document.querySelectorAll("p.intro");
* Get HTML elements by HTML object collections, e.g., document.anchors, document.forms, document.links, etc.

**Changing HTML Elements**

After you get a hold of an element, you can do the following:

|  |  |
| --- | --- |
| element.innerHTML = “new html content” | Changes the inner HTML of an element |
| element.attribute = “new value” | Changes the attribute value of an element |
| element.setAttribute(attribute, value) | Changes the attribute value of an element |
| element.style.property = “new value” | Changes the CSS property of an element |

We will look at dynamic manipulation of DOM using JS starting from Week6.

**More Resources to learn HTML**

Here are a couple of videos where you can learn about the HTML:

1. Jen Simmons (HTML): <https://www.linkedin.com/learning/html-essential-training-4/what-is-html>
2. Net Ninja (HTML): <https://www.youtube.com/playlist?list=PL4cUxeGkcC9ibZ2TSBaGGNrgh4ZgYE6Cc>
3. Net Ninja (HTML & CSS): <https://www.youtube.com/playlist?list=PL4cUxeGkcC9ivBf_eKCPIAYXWzLlPAm6G>
4. Traversy Media (HTML): <https://www.youtube.com/watch?v=UB1O30fR-EE>

Documentation:

1. Mdn Docs: <https://developer.mozilla.org/en-US/>
2. HTML Living Standard: <https://html.spec.whatwg.org/>