



Graphic Era HILL UNIVERSITY

Established by an Act of the State Legislature of Uttarakhand (Adhiniyam Sankhya 12 of 2011)
University under section 2(f) of UGC Act, 1956

MOOC based seminar report

On

HTML5 AND CSS FUNDAMENTALS

BY : AMAN RAUTELA

ROLL NO: 1961011

COURSE: B.TECH

BRANCH : CSE

UNDER THE GUIDANCE OF

DR. NAVEEN TEWARI

ASSISTANT PROFESSOR

DURING THE YEAR 2020-2021



Graphic Era HILL UNIVERSITY

Established by an Act of the State Legislature of Uttarakhand (Adhiniyam Sankhya 12 of 2011)
University under section 2(f) of UGC Act, 1956

ACKNOWLEDGEMENT

I would like to give a special thanks to several people who made this all possible.

It is because of their tremendous support that I was able to finish this project with immense success .

I would like to thank each and everyone who helped me in the completion of this project, but firstly and fore mostly I would like to pour my heart out in gratitude and humility towards our MOOC coordinator Dr. Naveen Tewari who ostensibly supported me in this project whole heartedly.

I consider myself very fortunate and gratified that I was given an opportunity and once again would like to show my appreciation whole heartedly to everyone to make this possible.

rautelaaman7409@gmail.com

AMAN RAUTELA



Graphic Era HILL UNIVERSITY

Established by an Act of the State Legislature of Uttarakhand (Adhiniyam Sankhya 12 of 2011)
University under section 2(f) of UGC Act, 1956

BHIMTAL CAMPUS

THIS IS TO CERTIFY THAT MR **AMAN RAUTELA** HAS SATISFACTORILY PRESENTED
MOOC BASED SEMINAR. THE COURSE OF THE MOOC REGISTRATION **HTML5 and**
CSS Fundamentals IN PARTIAL FULFILMENT OF THE SEMINAR PRESENTATION
REQUIREMENT IN 3rd SEMESTER OF B.TECH (✓) / M.TECH () / BCA / MCA / BBA /
MBA DEGREE COURSE PRESCRIBED BY GRAPHIC ERA HILL UNIVERSITY BHIMTAL
CAMPUS DURING THE YEAR **2020-21**

MOOC – Coordinator

HOD

Name : Dr. **NAVEEN TEWARI**

Name : **DR. M. C. LOHANI**

Signature :

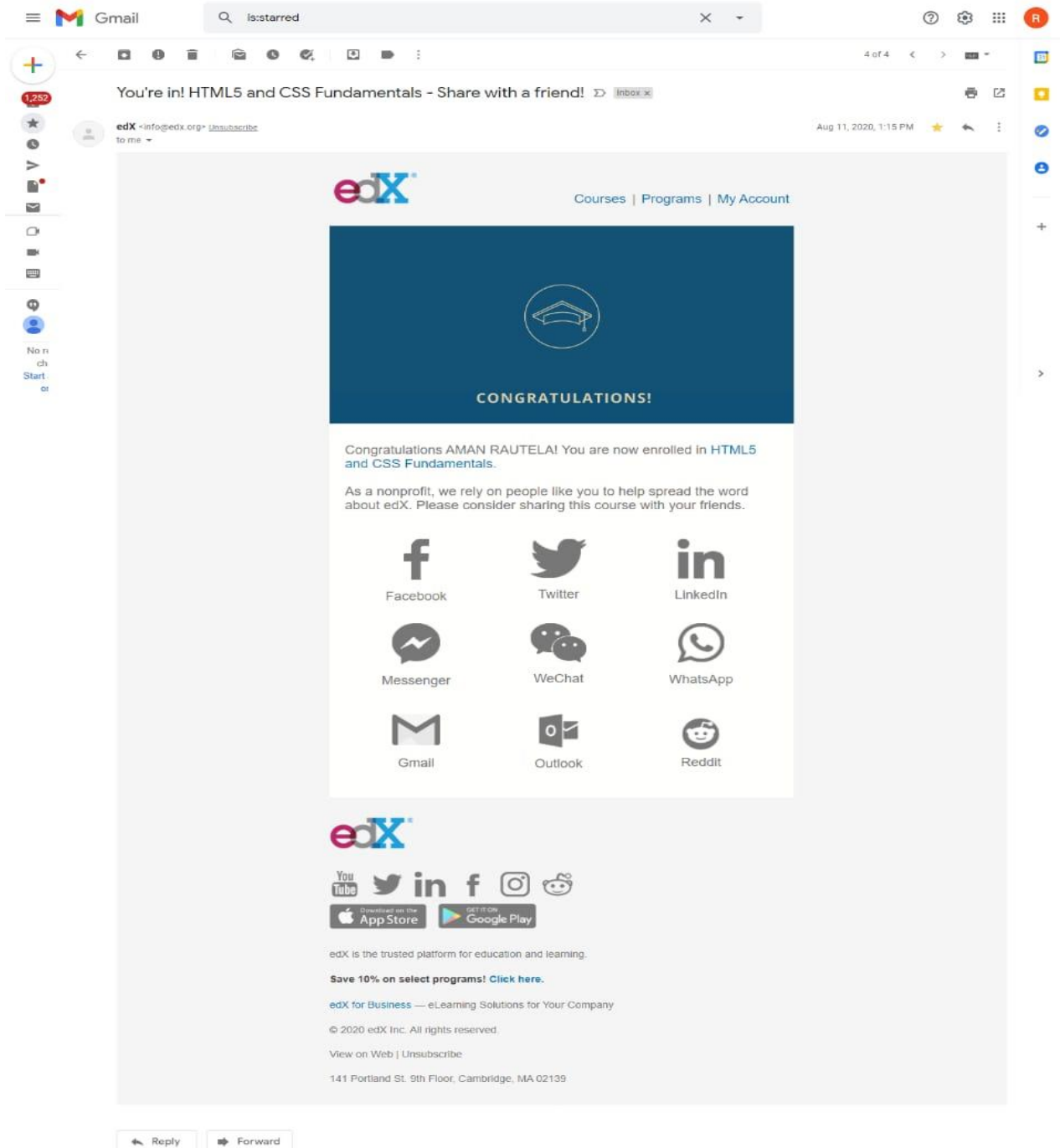
Signature :

ABOUT THIS COURSE :

This course is all about the basic understanding about how to build a Web Site using HTML 5 and CSS. I have enrolled in this course from a well renowned online learning platform Edx.

Enrollment date : 11-Aug-2020

Time Duration : 8 Week (approx. 4 to 5 hrs per week)



SYLLABUS :

This course is subdivided into 6 module as provided below:

Module 1: Web page

The big three: HTML5, CSS and JavaScript Elements, tags and attributes Character encoding Best practices

Module 2 : Debugging Attributes, images and links

Attributes Semantic meaning Images Hyperlinks

Module 3 : Adding style with CSS

CSS basic syntax CSS properties Lists and selectors

Module 4: Fixing and debugging

Debugging tools Debugging and the CSS box model
Debugging CSS precedence

Module 5: More HTML5 and CSS

Tables Multimedia Embedding content CSS tricks

Module 6: Basics of page layout

Concepts Flexbox Recipe project Where to from here

HTML5 and CSS Fundamentals

WEEK 1

HTML5

When people say 'HTML5', they usually mean a bit more than just the 5th version of the "Hyper Text Markup Language". Modern Web pages and Web applications are generally composed of at least three components, so what people often mean when they say 'HTML5' is the trio of languages: HTML5, CSS3 and JavaScript.

The 'HTML' part contains all the content, organized into a logical structure. This is the part that an author might be most concerned with: the words, chapter headings, figures, diagrams, etc.

While there have been numerous versions of HTML since its inception, our focus in this course is the most recent version, HTML5. HTML5 was developed to provide more powerful and flexible ways for developers to create dynamic Web pages.



CSS :

The 'CSS' part (version 3 being current) is all about the presentation or style of the page; what it looks like without too much regard for the specific content. We'll be going into more detail on that later in this

course, but for now, think of it as the way you might specify a "theme" in a word processing document, setting fonts, sizes, indentations and whatever else may apply to what it looks like.



JAVASCRIPT :

The 'JavaScript', or 'JS' for short, part is about the actions a page can take such as interaction with the user, and customizing and changing the page according to any number of parameters. This is what allows a Web page to be more than just a document, but potentially a Web application, with nearly unlimited possibilities. We will not be doing much with JavaScript in this course, but you should know that it is an important leg of the stool for modern Web pages.

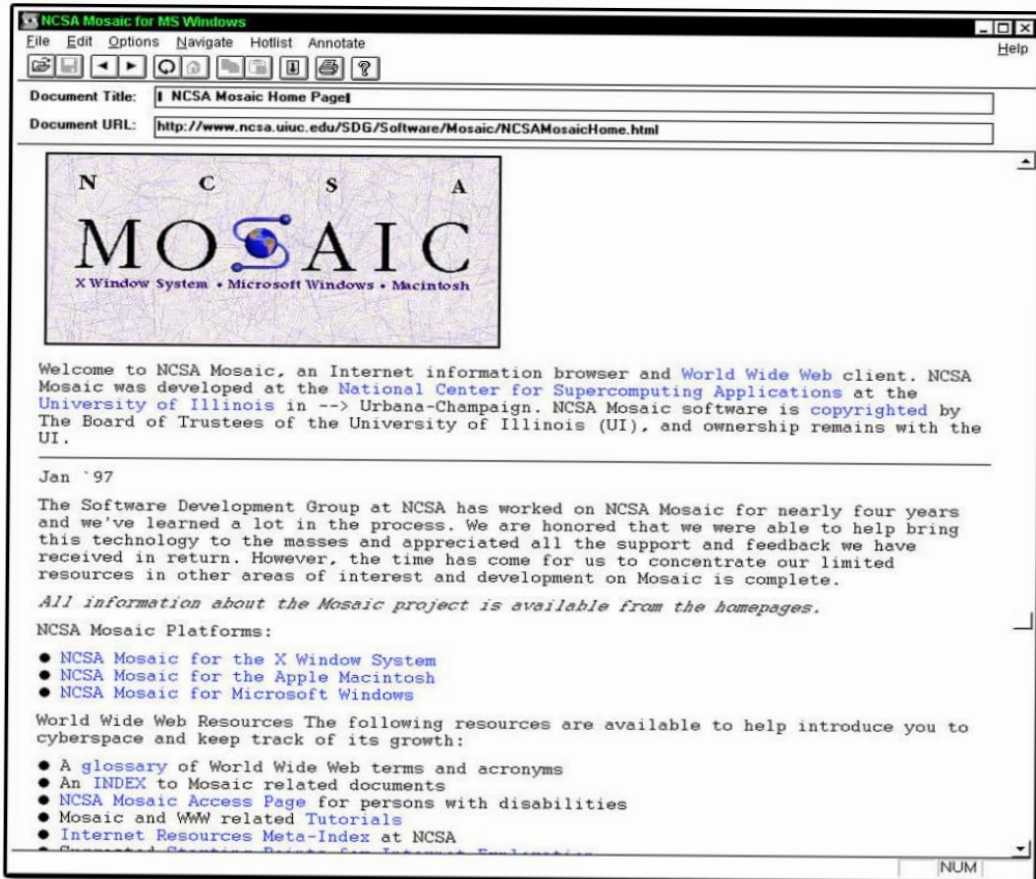
JavaScript enables interactive web pages and is an essential part of web applications. The vast majority of websites use it for client-side page behavior, and all major web browsers have a dedicated JavaScript engine to execute it. As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative programming styles. It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM). However, the language itself does not include any input/output (I/O), such as networking, storage, or graphics facilities, as the host environment (usually a web browser) provides those APIs.

JavaScript engines were originally used only in web browsers, but they are now embedded in some servers, usually via Node.js. They are also embedded in a variety of applications created with frameworks such as Electron and Cordova.

A large yellow square with the letters 'JS' in a bold, dark blue, sans-serif font. The 'J' and 'S' are positioned side-by-side, centered within the square. The square has a slight drop shadow, giving it a 3D appearance as if it's floating above a white surface.

THE BROWSER :

The Internet existed long before the Web came to fruition, and lots of organizations were connected to it, including schools, companies and government organizations. As things progressed through the 80's, the Internet was used for file transfers, newsgroups (a kind of open forum), email and other conveniences.



At the time there were a number of different programs like 'fetch', 'gopher' and 'archie' that were used to download, browse and search for files. Typically, you might use one tool to search for the location of files of interest, then another to copy that file to a local machine. Then, you still needed more tools to read that file. If it was text, you could use a text editor, if it was a formatted document you might need a word processor, if a picture you would need an image viewer and so on.

Marc Andreessen conceived of a solution that would put all the pieces together in one app, making it easy for users to browse all the different sorts of information and data on the World Wide Web. Together with others, he started the "Mosaic" project.

Though not technically the first browser, Mosaic was the first one that many people experienced and played a big part in popularizing the concept of the World Wide Web and the Web browser. It provided a simple graphical way to access and browse the various resources on the Internet. Instead of using different tools to download and view information on the Internet, a simple click on a link would present the information in a graphical window. In many ways, it is the ancestor of most modern browsers.

ELEMENTS, TAGS and ATTRIBUTES

ELEMENTS:

If you are sitting at a coffee shop next to a table of Web developers, you will probably hear three words quite a bit: 'Tags', 'Attributes' and 'Elements' (or sometimes 'DOM elements', same thing just more precise and wordy).

'Elements' are the pieces themselves, i.e. a paragraph is an element, or a header is an element, even the body is an element. Most elements can contain other elements, as the body element would contain header elements, paragraph elements, in fact pretty much all of the visible elements of the DOM.

Consider the figure above.

It contains a single 'html' element.

It turns out this includes within it the entire content of your html file. If you click on the "html" node, you'll find that it contains two components, a head and a body.

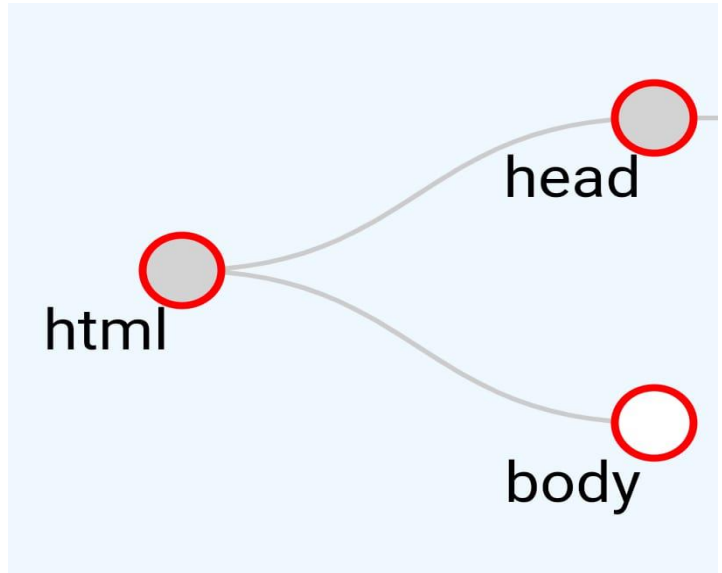
Clicking on each of those will reveal their respective contents.

This structure is what we computer scientists call a "tree". Any given element (except for the outermost

'html' element) is wholly contained inside another element, referred to as the "parent" element. Not surprisingly, the elements that a given element contains are its "child" elements. And, yes, children of a common parent are often referred to as "siblings".

Thus in the example above, the top element is the html element, which contains just two elements, the head and body. The head element contains a title element and the body contains an 'h1' element and a 'p' element. In a more typical example, the body would contain many more children, but for our purpose this is enough.

That may be a great picture, but how do we represent such a structure in a text file? Well, that's where "tags" come in.

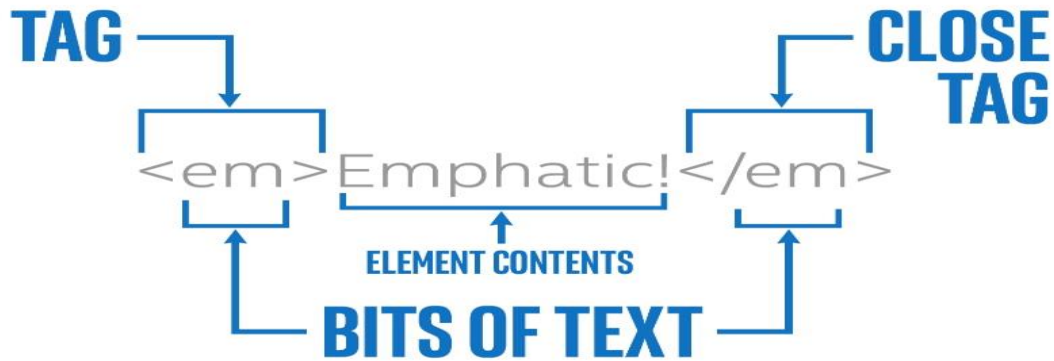


'Tags' are what we use to organize a text file (which is just a long string of characters) such that it represents a tree of elements that make up the html document. Tags are not the elements themselves, rather they're the bits of text you use to tell the computer where an element begins and ends. When you 'mark up' a document, you generally don't want those extra notes that are not really part of the text to be presented to the reader. HTML borrows a technique from another language, SGML, to provide an easy way for a computer to determine which parts are "MarkUp" and which parts are the content. By using '<' and '>' as a kind of parentheses, HTML can indicate the beginning and end of a tag, i.e. the presence of '<' tells the browser 'this next bit is markup, pay attention'. Whatever that tag (or 'open tag') does, it applies to the content following the tag. Unless you want that to be the entire rest of the document, you need to indicate when to stop using that tag and do something else, so '<' and '>' are again used. Since elements are typically nested within other elements, the browser needs to be able to distinguish between the end of the current tag or the beginning of a new tag (representing a nested element). This is done by adding a '/' right after the '<' to indicate that it's a 'close tag'. To indicate the beginning and end of a paragraph (indicated by the single letter 'p') you end up with something like this:

```
1. <p>This is my first  
    paragraph!</p>
```

The browser sees the letters '<p>' and decides 'A new paragraph is starting, I'd better start a new line and maybe indent it'. Then when it sees '</p>' it knows that the paragraph it was working on is finished, so it should break the line there before going on to whatever is next.

For example, the '' tag is used for element that needs Emphasis. The '<' and '>' indicate that this is a tag, and the "little bits of text" in between tell us what kind of tag it is. To completely describe the element, it needs an open and close tag, with everything in between the tags is the contents of the element:



Most tags have open and close versions, but there are a few strange ones. We'll learn more about these later, but we generally refer to the strange ones as "self closing" tags. Usually these tags represent an element that is completely described by its attributes, and thus there is no need for other content. So if you see something like this:

```
1. 
```

... then you should know that the slash at the end of the open tag is sort of a shorthand for a close tag, so you won't see any other indication that this element is now complete. There are also a few tags that don't even use the '/' at the end, they just don't have any close tag at all. This works because all of the information this tag needs is declared in an "attribute".

ATTRIBUTES :

Most of what we'll cover about attributes will come later, but I wanted to introduce the idea briefly. Basically, a given element on your Web page can be distinguished by any number of unique or common attributes. You can identify it uniquely with an 'id' attribute, or group it with a class of other elements by setting the 'class' attribute.

Attributes in HTML are written inside the opening tag like this:

```
1. <p id="paragraph-  
    1" class="regular-  
    paragraphs">  
2.     Call me Ishmael . .  
    .  
3. </p>
```

- The paragraph above has a unique identifier, "paragraph-1" and is part of a class of "regular-paragraphs". The letters inside the quotes have no meaning to the computer, they just need to be consistent. They are actually strings, which, as we will soon learn, suggest that if you want to have another paragraph in this class, it has to say "regular-paragraphs", not "regular" or "Regular-Paragraphs" or any other variation.
- Again, the fact that the computer does not care what we put in those strings (except for some restrictions) means we can use them to convey meaning to a human developer. I could just as easily have said id='x' and class='y', but anyone looking at that would have no hint what the significance of x and y are. Best practice is to name these things to increase clarity, consistency and brevity.

WEEK 2

SEMANTIC ELEMENT :

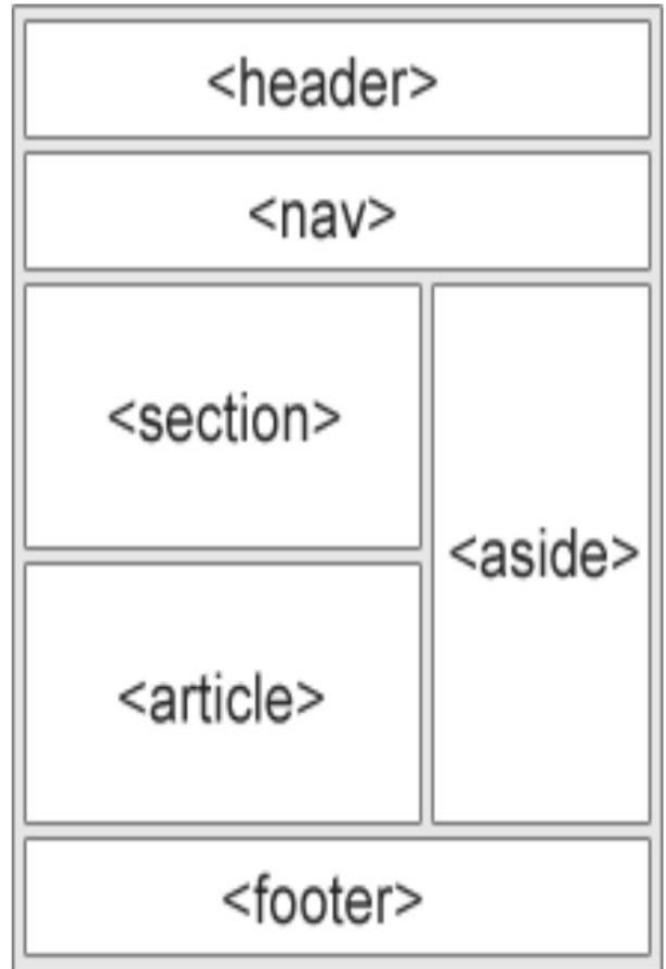
Elements such as <header>, <footer> and <article> are all considered semantic because they accurately describe the purpose of the element and the type of content that is inside them.

A semantic element clearly describes its meaning to both the browser and the developer.

Examples of semantic elements: <form>, <table>, and <article> - Clearly defines its content.

In HTML there are some semantic elements that can be used to define different parts of a web page:

- <article>
- <aside>
- <details>
- <figcaption>
- <figure>
- <footer>
- <header>
- <main>
- <mark>
- <nav>
- <section>
- <summary>
- <time>



WHY IT IS IMPORTANT????

HTML5's semantic elements help structure the code we create, making it more readable and easier to maintain. They help us think about the structure of our dynamic data, and to choose titles' hierarchy properly. They help us differentiate the semantic elements of our markup from the ones we use solely for layout.

INTRODUCTION TO IMAGES :

The tag :

In this age of visual culture, what is a Web page without images? Boring! Pictures and images make everything more interesting and engaging.

```
1. 
```

The image tag has several attributes out of which only src and alt are required. The rest are useful but optional attributes.

The source attribute from the tag tells us where to fetch the image from. There are two different types of URLs you can give for source.

1. Path to an image file within your Web site:

```
1. 
```

2. Path to an image file that resides elsewhere on the Web:

```
1. 
```

When using images in your HTML5, there are a few image format related information to be aware of.

Image data: most images, especially JPEG, contain a lot more data than is needed for a browser and are too often overly large and slow. You can reduce the size of the image using photo editing software that allows you to re-sample an image to reduce its pixel data and in turn reducing image size. However, once you re-sample an image, do not make change its size (height and width) to make it larger as it will become pixelated and blurry.

JPEG (Joint Photographic Experts Group) images compress well and are the standard for photos. But they don't support any sort of animation or transparency.

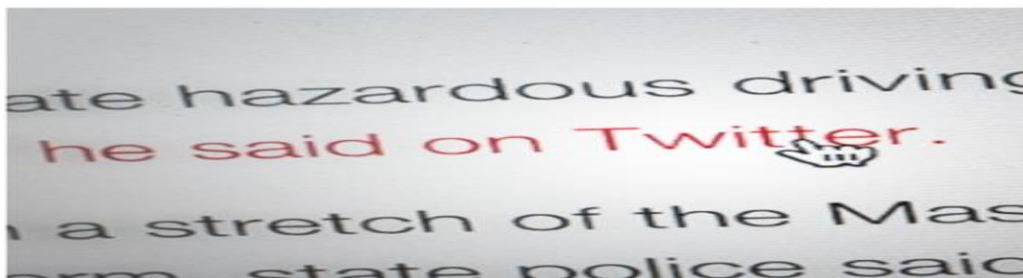
PNG (Portable Network Graphics) images support transparency and alpha channels. This makes them useful for non-rectangular images that may need to overlay different background colors or other elements on the page. To make PNG images, a user would need graphics editing software (like GIMP, Photoshop, or others). PNG is a W3C Web standard (this is the 2nd edition - the 1st edition was published in 1996!).

SVG (Scalable Vector Graphics) are defined mathematically and support animation. Also, since they are defined mathematically they scale to Logo Scalable Vector Graphics (SVG) any size without worrying about pixels, resolution or image data. This makes SVG images an excellent format to use, if possible. SVG is great for charts, graphs, maps, geometric shapes, and line based illustrations. SVG is also a markup language in its own right and is very similar to HTML. Typically, it is created with vector graphic software (like Inkscape, Adobe Illustrator, and others), but some people write the markup by hand.

INTRODUCTION TO HYPERLINK :

What are hyperlinks :

In a typical Web page, I am sure you have seen a sentence like this one:



Why you should use text over image links

When it comes to hyperlinks, try to use text instead of images when possible.

- Images are not as well understood or recognized as text.
- Text is better for accessibility.
- If you have text in an image like the 'Buy now' button, search engines do not recognize text in images.

Best practices

- Apply hyperlinks to short phrases. It is unusual to see the link tag used around a whole paragraph.
- Make link phrase meaningful. Avoid phrases like 'Click Here' or 'Read More'. 'Click here to get help' is redundant when you can create a link around the phrase 'Get Help'. The link is an indication that it should be clicked.
- Don't use short link text. It is easy to miss the 'blue' hyperlink if it is used on one word or character and is hard for users to click using touch screen.
- Appearance - links have a default appearance in most browsers, blue and underlined. Ensure no other text in your page is underlined so as to avoid confusing the user. They might get frustrated trying to click text that they think is a link.
- If you choose to have image links, it should have alternate text that describes the purpose of the link instead of the image used - describe the target link.

```
1. <a
    href="https://www.table-
    header-cells.com">
2.   
3. </a>
```

States of a hyperlink

If the link has not been clicked, it will be blue and underlined. Now, click on the link and you will see that a visited link looks purple and underlined. Apart from unvisited and visited links, there is also a status called active link. A link becomes active while the user is clicking on it. During this time, the link will be red and underlined.

1. Unvisited: blue + underlined
2. Visited: purple + underlined
3. Active: red + underlined

WEEK 3

CSS BASIC SYNTAX :

The <style> tag :

The best practice when working with CSS is to keep it in an external file using the <link> tag, however, when starting, it is simpler to merely place it directly into the document under edit.

To place CSS directly into an HTML document, we use the <style> tag. This tag can appear anywhere in an HTML document, however, the most common practice is to place it in the <head> section. Such as:

```
1. <!DOCTYPE html>
2. <html lang="en">
3.
4.   <head>
5.     <meta charset="UTF-
6.       8">
7.     <title>Style and
8.       link tags</title>
9.     <style>
10.      /* CSS will go in
11.        this area */
12.    </style>
13.  </head>
14.
15.  <body>
16.
17.  </body>
18. </html>
```

The <link> tag :

While <style> is convenient, the better practice is to put the CSS into a separate file. One of the key advantages of using a separate file is that the CSS styles can easily be re-used between your different .html pages. Many authors further divide their CSS up into different files (for example: one for text styles and another one for layout).

Simply put your CSS into a separate file. This file does not need any HTML markup (i.e., no `<style>` tag required). Use the `.css` file extension and use a `<link>` tag to bind it in. The `<link>` tag must appear in the `<head>` section. By convention, css files are kept in a directory named `css`. Use this `<link>` as a template:

```
1. <link rel="stylesheet"
    href="css/my_styles.css"
    >
```

Here is an example HTML document.

```
1. <!DOCTYPE html>
2. <html lang="en">
3.
4.   <head>
5.     <meta charset="UTF-
6.       8">
7.     <title>Style and
8.     link tags</title>
9.     <link
10.      rel="stylesheet"
11.      href="css/my_styles.css"
12.    >
13.   </head>
14.   <body>
15.   </body>
16. </html>
```

CSS PROPERTIES :

There are hundreds of CSS properties for you to use. The complete list is available on the W3C Web site (or also, see the CSS reference page on the MDN Web site).

Below we've gathered a more manageable list of the most useful and common CSS properties: font-size, line-height, text-align, text-decoration, font-weight, font-style and font-family.

Font Size :

font-size can be used to size the text of a tag. The value for the font-size has two parts: a number and a unit. Some of the most common units are: px, em, %, vh. For example:

```
p { font-size: 18px; }
```

```
q { font-size: .8em; }
```

```
blockquote { font-size: 10vh; }
```

These units are discussed below.

Additionally, font-size supports a more readable set of values that many authors prefer: xx-small, x-small, small, medium, large, x-large, xx-large

and relative sizing (relative to the text of the parent): larger, smaller. For example:

```
p { font-size: medium; }
```

```
q { font-size: small; }
```

```
blockquote { font-size: larger; }
```

LINE HEIGHTS :

Whereas font-size may drive the size of the text itself, the line-height property drives the height of the space it is drawn into. A large line-height will give the text more spacing. A small line-height will smash the text lines together.

For example, all of the Middlemarch text below has font-size:16px;

But on the left, we see line-height:0.5; and on the right, line-height:3;

line-height: 0.5;	line-height: 3;
Miss Brooke had that kind of beauty which seems to be thrown into relief by poor dress.	Miss Brooke had that kind of beauty which seems to be thrown into relief by poor dress.

The used value is this unitless <number> multiplied by the element's font size. The computed value is the same as the specified <number>. In most cases this is the preferred way to set line-height with no unexpected results in case of inheritance. Read more on the MDN Web site.

TEXT ALIGNMENT :

Anyone familiar with a text editor will be familiar with this property. It can be used to align the text **left, center or right**. There are additional possible values like justify and justify-all . It usually defaults to left. However, remember that you shouldn't use text-align unnecessarily.

Note that text-align may not work as expected if applied to elements that are the same width as their text, or whose width is determined by the text within them (i.e., inline elements). The tags ``, `<a>`, `<i>`, ``, `<q>` and others are considered "inline" because they do not receive their own new line when used. And text-align is often not useful on these tags.

But it is useful on block level text tags, such as `<p>`, ``, ``, ``, `<div>`, and `<blockquote>`

```
p { text-align: left; }
```

```
blockquote { text-align: right; }
```

Bear in mind, also, that you should only use text-align when the alignment really needs to be changed, since it can cause additional work to reverse all the values when translating into languages that use Arabic, Hebrew, Thaana scripts, scripts (the default alignment for those languages is right). The new values start and end are currently being implemented in browsers, and those will be a much better choice than left and right once Internet Explorer supports them.

LIST and SELECTORS :

Styling List :

The list markup tags (``, `` and ``) are some of the most frequently used specific purpose tags in HTML. There are a few CSS style properties that are available for lists.

List-style-type :

list-style-type governs the little list marker that is usually positioned to the left of any list item. For un-ordered lists (``), there are several popular values: disc, circle, square, and none.

```
li { list-style-type: disc; }
```

html	default	disc	circle
<pre> eggs milk bread </pre>	<ul style="list-style-type: none">eggsmilkbread	<ul style="list-style-type: none">eggsmilkbread	<ul style="list-style-type: none">eggsmilkbread

For ordered lists () you can choose different ways of having the numbers shown: decimal, decimal-leading-zero, lower-roman, upper-roman, lower-alpha, upper-alpha, as well as several of the worlds languages: armenian, georgian, simp-chinese-formal, and many others.

list-style-position

Besides choosing the type of marker applied to each list item, you may also want to govern how closely it is positioned to the list itself. The list-style-position property handles that. The two values are inside and outside. They govern whether the markers are positioned inside the box of the list, or outside. This is most evident if a border or background or similar is applied to the list. Below, we have put a blue border on the list.

outside	inside
<div><div>01. eggs</div><div>02. milk</div><div>03. bread</div></div>	<div><div>01. eggs</div><div>02. milk</div><div>03. bread</div></div>

list-style-image

The little markers on a list can also be customized to be an image of your choosing. This will require you to have a small image in a Web compatible format (PNG or JPEG recommended) and to know the path from the place where the CSS is being defined to the image. Image pathnames were covered in Module 2, and we'll be discussing them again in the background-image section.




```
li { list-style-image: url("my_triangle.png"); }
```

list-style-image

eggs

milk

bread

list-style-image
<div><div> eggs</div><div> milk</div><div> bread</div></div>

SELECTOR :

tag selector

We've already seen this one. A CSS selector that consists solely of a single tag (without punctuation or spacing) will be applied to any matching tag on the page.

```
li { list-style-type: circle; }
```

id selector

You may remember the id attribute (short for "identifier"). This attribute can be applied to an HTML tag to uniquely identify the element. Recall that the value for any given id attribute can only appear once in a document. No two tags are allowed to have the same id. You may also recall that the id cannot contain spaces, nor most punctuation, nor begin with numbers.

In the HTML below, there are two paragraph tags. So, to style them individually, we can apply unique id attributes to the paragraphs (id="p18" and id="p19"). In the CSS, we will use the id selector. The id selector is simply a hash sign (#) followed directly by the id.

CSS:

```
#p18 { color: blue; }  
#p19 { color: green; }
```

class selector

The class attribute is similar to the id. However, whereas the id must be unique and singular, the values of the class attribute can be shared by multiple tags. And, multiple classes can be assigned to a tag by simply separating them with spaces.

HTML:

```
<ul>  
<li class="bird flying">eagle</li>  
<li class="bird">ostrich</li>  
<li class="insect">ant</li>  
<li class="insect flying">moth</li>  
</ul>
```

The class selector is simply a period (.) followed by the class name itself.

CSS:

```
.bird { color: blue; }  
.insect { color: green; }  
.flying { text-decoration: underline; }
```

Result:

- eagle
- ostrich
- ant
- moth

WEEK 4

IDENTIFYING HTML ELEMENTS :

Remember that elements are the intangible parts of your Web page, which are described by the text in tags and are rendered on the screen of whatever device you're looking at your Web page with. The two things (the text code and the pixels on the screen) correspond to each other, but it's not always obvious which bit of the screen corresponds to which bit of text.

There are two opposite directions in which you might need to figure out in these two different things that both correspond to an element. You might have some HTML5 code that you've written and want to find out where on the Web page that code shows up. The other direction can be needed as well, i.e. Given a particular part of the page, what part of your code produced it?

When you hover over an element in the DOM Explorer window in your browser developer tools, the corresponding element on the displayed page is highlighted:

It is also possible to go the other direction, i.e. Click on a point on the displayed page and it will highlight the code in the source that corresponds to that element. This is helpful when you want to figure out where something came from and what might be affecting it's styling (size, color, font, any number of other characteristics). To do that in your browser developer tools, use the DOM explorer pane and the "select element" option, you can also right click on the section on your page you want to inspect and select "Inspect element" which will bring up the developers tools and highlights the HTML element or code for that section on the page.

MODIFYING HTML ELEMENTS :

Another handy feature of the developer tools is the ability to make temporary modifications to your code to try out different things and see what works the way you want it to. When you have a visible element selected in the DOM explorer tab, you can make style changes in the "Styles" panel, or use the "Computed" panel to see the values for each property and how they were determined.

It's possible to change things a few different ways. If you double click on an element in your HTML5 source code, you can change the source code. For example you could click on an attribute to modify it or it's value, or you can change the type of the tag or even the contents of an element.

You can use this same approach to add a "style" attribute to a particular element, which should override any other settings, there is also an easier way to do that. In the panel just to the right of the elements panel is the another panel with tabs including "Styles" and "Computed" and a few others. Most of the time we'll want the "Styles" tab activated. Once you do that, you can modify CSS properties of the current element by adding them to the "element.style" box at the top of the "Styles" panel.

Just click in between the two curly braces on the "Inline style" rule at the top of the Styles panel. After clicking you should see a little text entry box with which you can type property value pairs that will then effect the currently active element.

It's important to know that any changes you make in the developer tools will have no effect on the original Web page. They only affect that particular instance of that page during that debugging session. If you navigate to another page and come back, you'll need to make the same changes again if you want to get back to where you were. It's not that easy to break the Web!

CSS BOX MODEL :

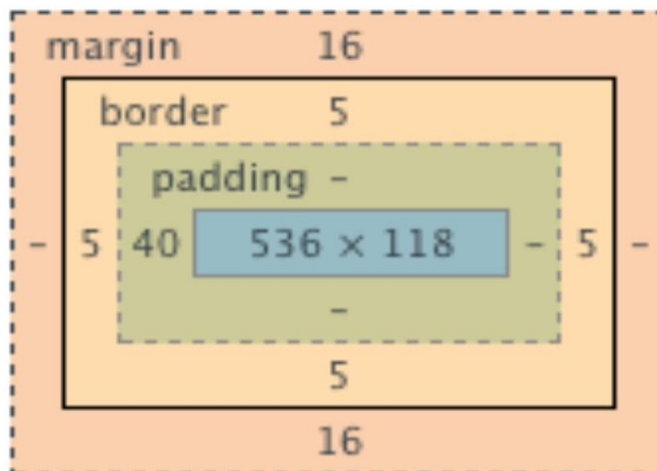
Before we get too far into debugging, it's helpful to understand a couple of things about CSS more deeply. The placement of elements on a Web page can be fairly complicated. One of the most basic features that influence where things go on a Web page is the CSS Box Model. The Box Model governs 3 important spacing features of CSS. We learned about margins previously as the space between elements. There are two other similar notions, padding and borders. Perhaps the best way to understand is with a picture. All elements in an html document end up being treated as rectangles somewhere in the window. The content of each rectangle corresponds to the innermost rectangle in the image below. Just outside the content is the padding. This is kind of like an internal margin, meaning that it separates the contents from the border. The border essentially traces the sides of the padding rectangle. It's important to note that the border goes around the content and the padding. There are sometimes visible things associated with an element that are not technically part of the content of the element. One such example is the list item:

- I'm in a blue box

The box does not include the bullets because it is outside of the content. Sometimes when you see that it might be a bit confusing, especially because it also affects padding (which is inside the border).

DEBUGGING WITH THE BOX MODEL :

In any browser's debugger, you will see a box model diagram. It looks like this:



This is an example of a diagram of the box model information for a selected element. The innermost box gives the dimensions of the element, outside of that is the padding, then the border around which is the margin. On each side of each corresponding rectangle is the width in pixels of that side, with “-” when it is 0 (essentially non-existent). Also, when you hover over one of the rectangles, that portion of element is highlighted on the rendered page, so you can see exactly where the margin, the border, the padding and the element are.

So, in the example above:

the centered blue box corresponds to the size of the inspected element: width is 536 pixels and height is 118 pixels

padding is only defined by padding-left which value is 40 pixels

there is a border of 5 pixels on each side

margin-left and margin-right are undefined (default value is 0 pixel)

margin-top and margin-bottom are equal to 16 pixels.

CSS PRECEDENCE :

As we learned in the last module, in order for the computer to decide which of several rules may apply to a given element in a particular context, there is a well defined “precedence” to define which rule should apply. Theoretically, we should be able to always figure out which rule applies by using the precedence rules, but in practice, it can be quite complicated, especially when conflicting rules are in different .css files.

Nevertheless, that can cause problems when you have different rules that could apply to the same element.

CLOUD IMAGE

Clouds sometimes take on shapes and forms of all matter of things, some obscure and others quite clear. The images you see in the clouds can foretell what’s to come. They can also give an indication of your current state of mind.

CHANGING TEXT SIZE :

In HTML, you can change the size of text with the tag using the size attribute. The size attribute specifies how large a font will be displayed in either relative or absolute terms. Close the tag with <

/font> to return to a normal text size.

WEEK 5

TABLES :

Using tables to organize information goes back a ways. A long ways. Three or four thousand years ago, Sumerians were using tables to calculate compound interest. Even so, tables are not obsolete, in fact HTML5 includes extensive facilities for describing/building tables.

Tables are used to arrange data in tabular format – rows and columns of cells. You can put a variety of data like text, images, forms, links and even other tables in your table.

You may hear experienced Web developers decrying the use of tables, giving the impression that tables should be avoided at all costs. It might seem off-putting at first, but they're not really talking about using tables for tabular information. In earlier days, when layout options were limited, many developers resorted to tables as a means of layout. There's really no need to do that anymore, there are plenty of layout capabilities in CSS3. You really don't want to use tables that way, but there absolutely nothing wrong with using them for their intended purpose – making tables.

Separating content and style

Look at this site that is laid out with CSS. You might be tempted to do this via tables instead. Or you might want to just make your whole Web page into one big table: site header in a table row, left navigation bar on the second row, left column, etc.

Bad idea! Here's why:

They are semantically incorrect for layout because they represent presentation and not content.

It puts presentation data in your content making your HTML larger. The user must download this unnecessary presentation data for every page they visit.

Accessibility: tables are not very screen reader friendly. Using tables for layout will clutter your HTML making it harder for assistive technology to parse your Web page.

Redesigns are harder when your HTML is cluttered with presentational code that should go into CSS. To change the layout of the page, you shouldn't be editing your content. Instead, you should just have to make CSS related changes.

Using CSS (one or two external stylesheets for your whole Web site) is easier to maintain consistency among pages.

Tables were not intended as a layout tool, so it is best to stick to them only for tabular data.

MULTI MEDIA :

What is Multimedia?

Multimedia comes in many different formats. It can be almost anything you can hear or see, like images, music, sound, videos, records, films, animations, and more.

Web pages often contain multimedia elements of different types and formats.

Browser Support:

The first web browsers had support for text only, limited to a single font in a single color. Later came browsers with support for colors, fonts, images, and multimedia!

Multimedia Formats:

Multimedia elements (like audio or video) are stored in media files.

The most common way to discover the type of a file, is to look at the file extension.

Multimedia files have formats and different extensions like: **.wav, .mp3, .mp4, .mpg, .wmv, and .avi.**

COMMON VIDEO FORMAT :

There are many video formats out there.

The **MP4, WebM, and Ogg** formats are supported by **HTML**.

The **MP4** format is recommended by **YouTube**.

COMMON AUDIO FORMAT :

MP3 is the best format for compressed recorded music. The term MP3 has become synonymous with digital music.

If your website is about recorded music, MP3 is the choice.

WEEK 6

TEXT BASELINE and DISPLAY PROPERTY :

When newbie developers are groping around CSS blindly, they often stumble upon a variety of CSS properties that could be used to alter the positioning or size of an element such as left, top, and margin.

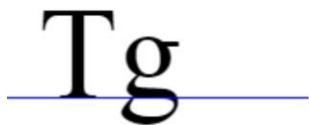
However, when using the properties, these developers get confused because the properties fail to behave consistently. Sometimes the properties work, sometimes they don't, sometimes they do the opposite of what they are doing in a different rule. We have not covered properties like left and top yet, but we have introduced margin and the intrepid readers may have already discovered in their exercises that margin can have some unexpected behavior. Why is that?

The answer has to do with the two CSS properties: display and position. The display property, in particular, has different default values for different tags. Some tags start with display:block, and others are display:inline. They behave very differently. These two properties (display and position) often change how an element responds to certain other layout properties. And when this is not understood, then it may seem random to a developer struggling to get stuff to work.

So, let's start with understanding a very important difference between block and inline display. And that begins with the baseline.

Baseline :

The text "baseline" is a key concept to understanding how the browser makes its layout decisions.



In the image above, we see two text characters placed next to each other, the blue line indicating the baseline. The baseline determines how and where the characters are positioned. Note that the tail of the "g" hangs below the baseline.

The baseline is never drawn by the browser, it is not exposed directly to you as a developer, and CSS only may have some properties related to it. However, the baseline governs the placement of all inline elements.

Display: box versus inline :

As the browser is rendering your page, every time it encounters the next tag it has a simple question: "Do I give this element its own line?" For example, every <p> tag gets a new line, but <a> tags do not. This is the key distinction between the "block" level elements (like the <p> tag) and the "inline" elements (like the <a> tag). Here is a quick table of the default values for some of the tags we've already learned.

Here are some differences between the block – level and inline elements.

Block	Inline
<ul style="list-style-type: none">• p• h1• div• blockquote• ul• ol• li	<ul style="list-style-type: none">• a• span• q• i• b

Block level :

The block level appears below and to the left of their block level neighbors (like a carriage return on a typewriter going to the next new line)

will expand to fill the width of the parent container by default

respects all margin properties

can have its width property set, which will make it narrower and cause its children to wrap, but not crop. (We'll cover this later)

takes on the height of all its children (pending certain exceptions) as long as its own height is unset. (We will cover setting the height later)

ignores the vertical-align property

Inline element :

Inline element proper simply appear to the right of their preceding inline neighbor. They do not drop to the next line unless they must "wrap".

By default, the width is simply the width of the content of the element, plus any padding

ignore top and bottom margin settings

ignore width and height properties

are subject to vertical-align property as well as CSS white-space settings

support padding, but any padding-top or padding-bottom does not contribute to the calculation of the height of the text line it sits upon

cleave to the baseline where they are being placed

The last bullet about inline elements is one of the most important to understand. Inline elements cleave to the baseline. This is very important to understand why inline elements are positioned vertically the way they are. It also contributes to why they ignore top and bottom margins. Note that making an inline element "bigger" with padding will certainly keep its neighbors away horizontally. But if there is a neighboring text line above or below, it can only be kept at bay with the line-height property, not margins or padding.

Inline Block :

The astute reader may have spotted an obvious omission from the table of block and inline elements above: ``. Is `` a block level element or inline? If you venture to experiment you may conclude “both”, and you will be right.

For historic reasons, the `` tag defaults to `display:inline` in most browsers. If you inspect using the browsers inspector, that’s what you will see. However, it does not follow the same rules as other inline elements. In fact, regardless of what the inspector says, images are special cased and are `inline-block`.

`Inline-block` elements still cleave to the text baseline of the line they are on. If top or bottom margins or paddings are used, then the entire line is adjusted to make room. (So the `line-height` does not need to be used.)

`inline-block` elements respect `margin-top` and `margin-bottom`

the vertical padding for `inline-block` elements contributes to the calculation of the height of the line it falls on

`inline-block` elements respect `width` and `height` properties

In some browsers, some of the form elements default to `inline-block` (like `<button>`, `<select>`, and `<input>`)

Display Property :

At long last we arrive at the display property. We have now seen three of its possible values: `block`, `inline`, and `inline-block`. There are others (like `none` and `flex`) and we will cover them later.

```
.name { display: inline-block; }
```

A key to not getting confounded by the display property is to have a grasp on which elements default to which display value and appreciating the differences between `block`, `inline`, and `inline-block` display.

POSITION PROPERTY :

The `left`, `right`, `top` and `bottom` property :

In CSS, there are four properties which can be used to adjust or set the position of an element. Those properties are: `left`, `top`, `right`, and `bottom`.

However, in one of the best jokes played by the authors of the CSS specification, using those properties by themselves will have no effect on any element. Surprisingly, most developers struggling with CSS don’t find this funny.

The reason these properties don’t work by default is that they only work when applied to positioned elements. And positioned elements are those that have had their position property changed from the default.

The ‘Position’ property :

The CSS position property governs how an element is positioned on the page and how it responds to the position adjusting properties (`left`, `top`, `right`, and `bottom`). Up to this point in this course, we have not used this property and so everything we’ve seen has been the default position, which is `position:static` for all elements.

As of today, the position property has four different values it can take: `static`, `relative`, `absolute`, and `fixed`. We will look at `static` and `fixed`. The options `relative` and `absolute` are more complex, they’ll be discussed in an optional advanced section for completeness, but we aren’t going to worry much about them because flexbox reduces their use cases.

CSS FLEXBOX :

Up to this point, we have covered quite a few different CSS layout concepts. Inline vs. Block level display, different position values, various positioning properties, six different sizing properties, plus countless details and interactions. So by this time, we should know enough to make a two-column page design, right? Sadly, we cannot. The intrepid among us might be able to cobble something together by creatively using inline-block, or maybe absolute or fixed positioning, but ultimately, any design built with just the topics we've covered so far will likely be brittle or unwieldy. Why is that?

All the layout properties we've looked at have all applied to an individual element. But performing layout tasks like columnar layout or anything responsive requires coordinating multiple elements. This is where the flexbox comes in. When working with flexbox layout, there are some CSS properties that are applied to a parent element (also known as the flex container) and other CSS properties that are applied to the direct children of that parent (also known as the flex items). The flex container will handle laying out of its children. And, best of all, the flex container will lay out its children smartly, making the best use of the screen size available to it, while still following the general guidelines you laid down for it. As a general rule, layout with flexbox is pretty easy and the results are great.

Sample flexbox example

First article

Tacos actually microdosing, pour-over semiotics banjo chicharrones retro fanny pack portland everyday carry vinyl typewriter. Tacos PBR&B pork belly, everyday carry ennui pickled sriracha normcore hashtag polaroid single-origin coffee cold-pressed. PBR&B tattooed trust fund twee, leggings salvia iPhone photo booth health goth gastropub hammock.

Second article

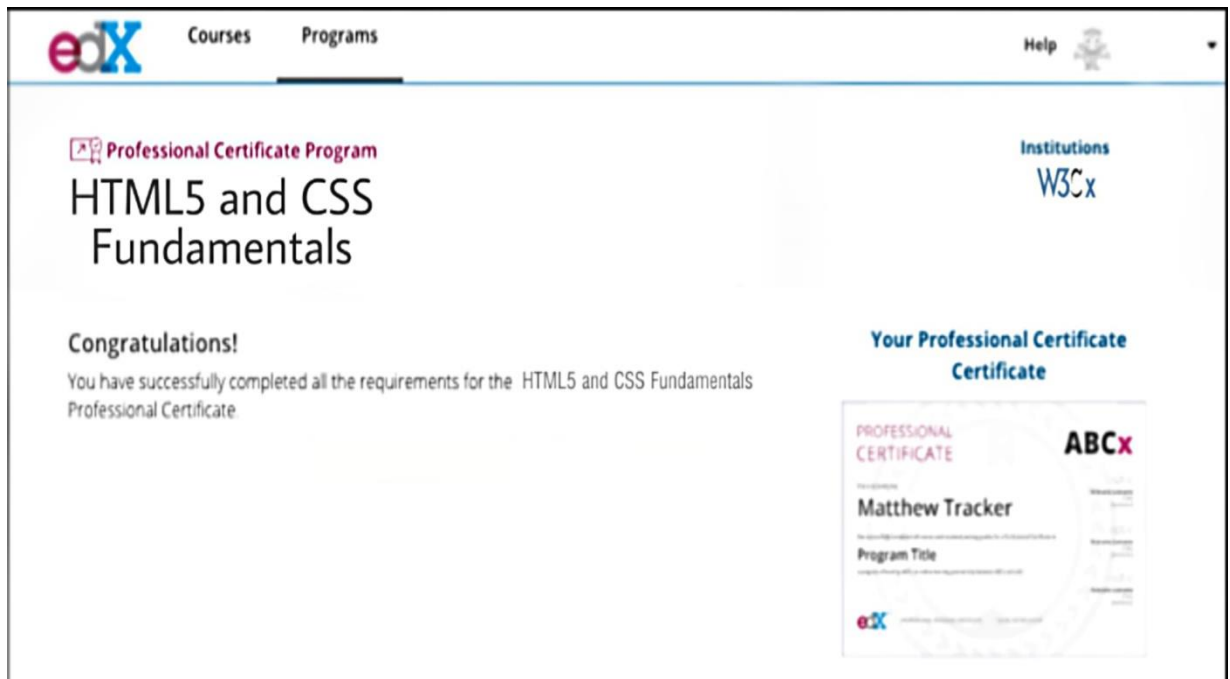
Tacos actually microdosing, pour-over semiotics banjo chicharrones retro fanny pack portland everyday carry vinyl typewriter. Tacos PBR&B pork belly, everyday carry ennui pickled sriracha normcore hashtag polaroid single-origin coffee cold-pressed. PBR&B tattooed trust fund twee, leggings salvia iPhone photo booth health goth gastropub hammock.

Third article

Tacos actually microdosing, pour-over semiotics banjo chicharrones retro fanny pack portland everyday carry vinyl typewriter. Tacos PBR&B pork belly, everyday carry ennui pickled sriracha normcore hashtag polaroid single-origin coffee cold-pressed. PBR&B tattooed trust fund twee, leggings salvia iPhone photo booth health goth gastropub hammock.

Cray food truck brunch, XOXO +1 keffiyeh pickled chambray waistcoat ennui. Organic small batch paleo 8-bit. Intelligentsia umami wayfarers pickled, asymmetrical kombucha letterpress kitsch leggings cold-pressed squid chartreuse put a bird on it. Listicle pickled man bun cornhole heirloom art party.

Proof of Completion of Course :



Web Site used for reference to complete this course :

W3Schools - <https://www.w3schools.com/>

EDX - <https://www.edx.org/>

Google – <https://www.google.com/>

THANK
YOU....