Unit 2: HTML5, JQuery and Ajax
XML Vs JSON & HTML5 (New Semantic Tags)

HTML5 semantic elements are here to help web developers to navigate though their web page with ease, since the semantic elements together forms a more structured layout.

The purpose of creating semantic elements in HTML5 is to give meaning to its traditional design layout. It helps browsers quickly and efficiently understand the structure of the layout, and two, it helps web developers to systematically arrange or design web pages and give meaning to each section of the layout. The elements are easy to remember, and fits where it needs.

Technically, do not have to use the DIV element as a container to define every section on the web page. Define the header section using the <header> element and the footer section using <footer> element and so on.

For a example,

Developers have used the DIV element (<div>) as a primary container, which served as header, footer, navigation etc, by identifying each element with unique ids or Class attribute.

Header using DIV

```
<div id="header">
   Header content
</div>
```

Footer using DIV

```
<div id="footer">
    Footer content
</div>
```

The above structure cannot be considered as meaningless, since it serves the purpose of separating two different sections as header and footer.

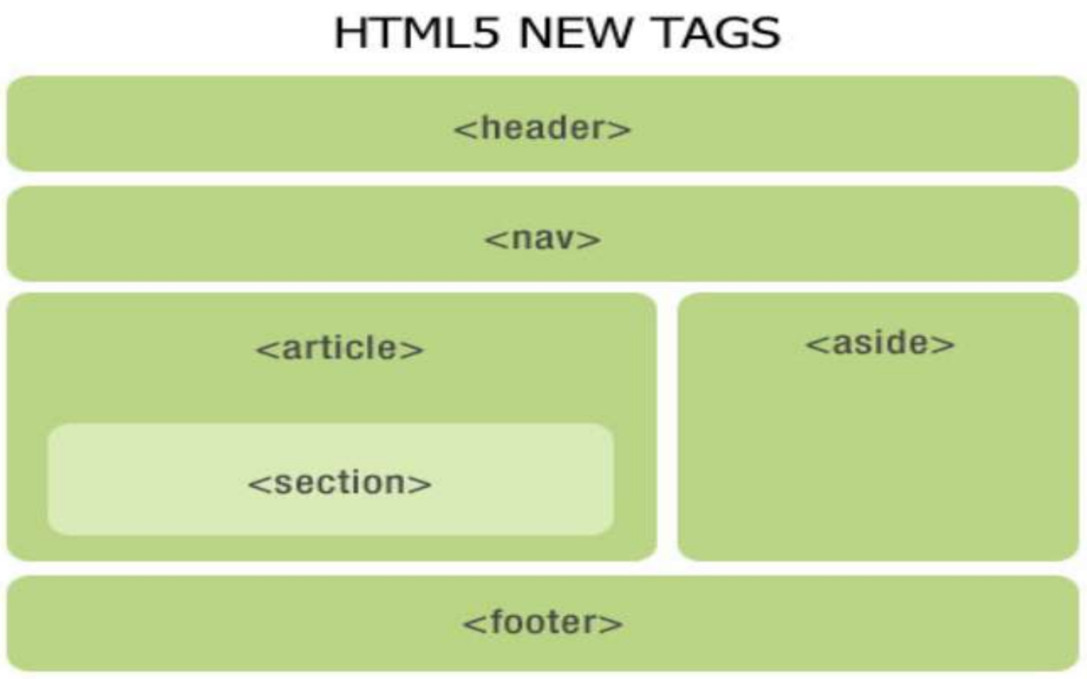
However, HTML5 has added a little meaning (semantic) and identifier to its elements, so web developers can use them wisely in their web pages.

HTML 5 introduces a number of new elements. Some of these are what I’ve termed “structural”—section, nav, aside, header, and footer. The dialog element is a kind of content element, akin to blockquote. There are also a number of data elements, such as meter, which “represents a scalar measurement within a known range, or a fractional value; for example disk usage,” and the time element, which represents a date and/or a time.

Example

Semantic <header> and <footer> elements

Prior to HTML5 semantics, the header and footer would have served as independent sections of a web page. We can define the new semantic <header> and <footer> elements as part of a section or sections.Each section can have its own <header> and <footer> element, allowing us to add multiple semantic elements in a single web page.



HTML 5 is the new major version of HTML. HTML5 brings a host of new elements and attributes to allow developers to make their documents more easily understood by other systems (especially search engines). In addition, HTML 5 will also include fancy APIs for drawing graphics on screen, storing data offline, dragging and dropping, and a lot more. Here are new HTML5 tags that will make it easier to write Web sites.

Doctype

In HTML 5, there is only one doctype. It is declared in the beginning of the page by <!DOCTYPE HTML>. It simply tells the browser that it’s dealing with an HTML-document.

<video> and <audio>

One of the biggest uses for Flash, Silverlight, and similar technologies is to get a multimedia item to play. With HTML5 supporting the new video and audio controls, those technologies are now relegated to being used for fallback status. The browser can now natively display the controls, and the content can be manipulated through JavaScript. Don’t let the codec confusion scare you away. You can specify multiple sources for content, so you can make sure that your multimedia will play regardless of what codecs the user’s browser supports.

<input> type attributes

The venerable <input> element now has a number of new values for the type attribute, and browsers do some pretty slick things depending on its value. For example, set type to “datetime” and browsers can show calendar/clock controls to pick the right time, a trick that used to require JavaScript. There is a wide variety of type attributes, and learning them (and the additional attributes that go with some of them) will eliminate the need for a lot of JavaScript work.

<canvas>

The <canvas> tag gives HTML a bitmapped surface to work with, much like what you would use with GDI+ or the .NET Image object. While <canvas> isn’t perfect (layers need to be replicated by using multiple canvas objects stacked on top of each other, for example), it is a great way to build charts and graphs, which have been a traditional weak spot in HTML, as well as custom graphics.

<header> and <footer>

The <header> and <footer> tags are two of the new semantic tags available. These two tags do not get you anything above and beyond <div> for the actual display. But they will reap long-term rewards for your search engine efforts, since the search engines will be able to tell the difference between "content" and things that are important to the user but that aren't the actual content.

<nav>

The nav-tag is used to contain navigational elements, such as the main navigation on a site or more specialized navigation like next/previous-links.

<article> and <section>

The <article> and <section> tags are two more semantic tags that will boost your search engine visibility. Articles can be composed of multiple sections, and a section can have multiple articles. Confusing? Not really. An article represents a full block of content, and a section is a piece of a bigger whole. For example, if you are looking at a blog, the front page might have a section for the listing of all the posts, and each post would be an article with a section for the actual post and another for comments.

<aside>

The aside tag is used to wrap around content related to the main content of the page that could still stand on it's own and make sense.

<output>

The new <output> tag is unique, in that it expects its content to be generated dynamically with JavaScript. It has a value attribute, which can be manipulated through the DOM with JavaScript to change what is displayed on the screen. This is much more convenient than the current ways of doing things.

<details>

It seems like every Web site needs to have an expanding/collapsing block of text. While this is easy enough to do with JavaScript or server-side code, the <details> tag makes it even easier. It does exactly what all have been doing for years now: makes a simple block that expands and collapses the content when the header is clicked. The <details> tag does not have widespread support yet, but it will soon.

<figure> and <figcaption>

<figure> is a container for content (typically images, but it can be anything), and <figcaption> (which gets put inside the <figure> tag) provides a caption or subtitle for the contents of the <figure> tag. For example, you could have four images representing charts of sales growth within a <figure> tag, and a <figcaption> with text like "Year-to-year sales growth, 1989 – 1993." The images would be shown next to each other with the text running below all four.

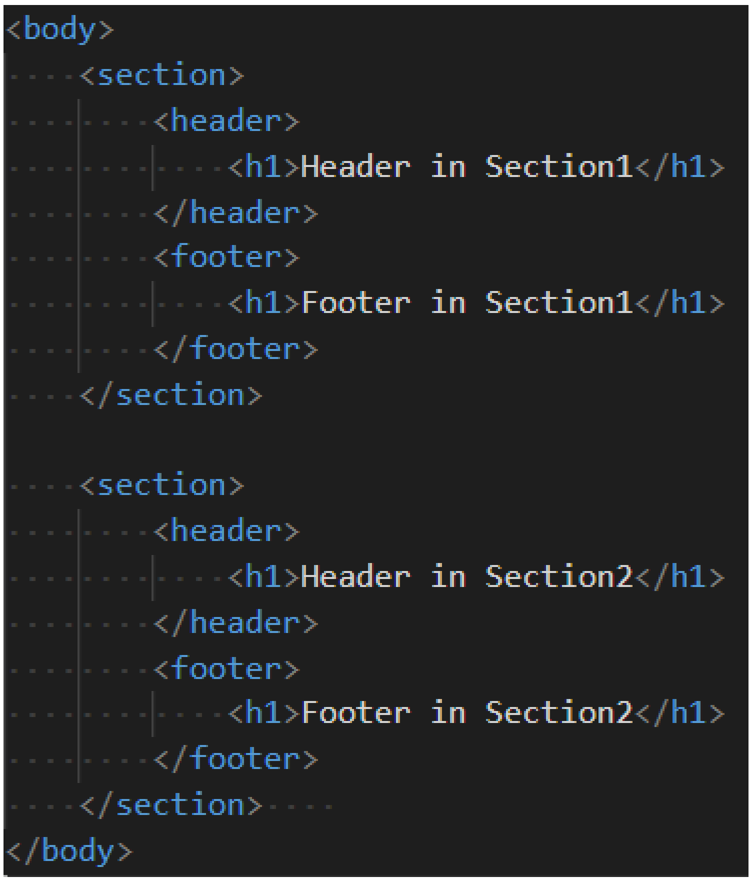
<progress>and <meter>

<progress> and <meter> are similar. You use <progress> for a task or a "measure how complete something is" scenario. It also has an indeterminate mode for something that has an unknown duration (like searching a database). The <meter> tag is for gauges and measurements of value (thermometers, quantity used, etc.). While they may look alike on the screen in many cases, they do have different semantic meanings.

<datalist>

The <datalist> tag acts like a combo box, where the system provides a pre-made list of suggestions, but users are free to type in their own input as well. There are tons of possible uses for this, such as a search box pre-populated with items based on the user's history. This is another one of those things that currently requires a bunch of JavaScript (or JavaScript libraries) to handle but that can be done natively with HTML5.

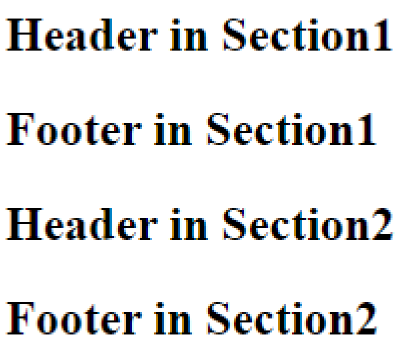
CODE: To add multiple semantic elements in a single web page.



```
<body>
    <section>
        <header>
            <h1>Header in Section1</h1>
        </header>
        <footer>
            <h1>Footer in Section1</h1>
        </footer>
    </section>

    <section>
        <header>
            <h1>Header in Section2</h1>
        </header>
        <footer>
            <h1>Footer in Section2</h1>
        </footer>
    </section>
</body>
```

Results:



Header in Section1

Footer in Section1

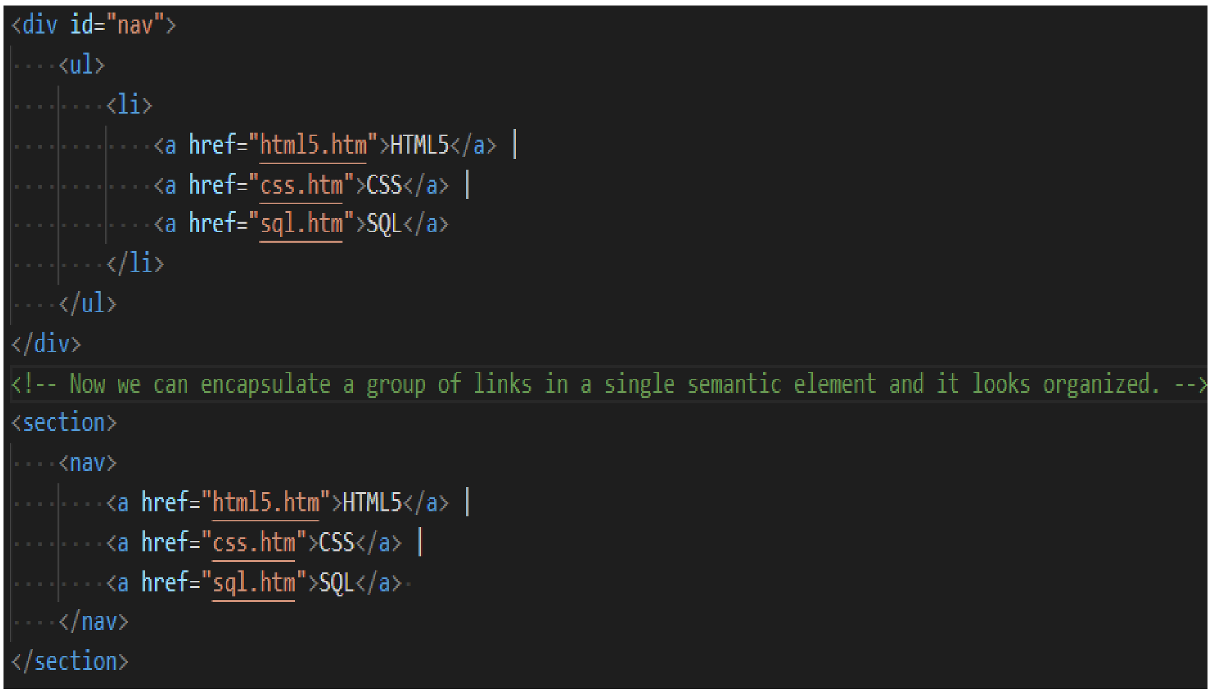
Header in Section2

Footer in Section2

CODE: Create navigation links using Semantic <nav> elements

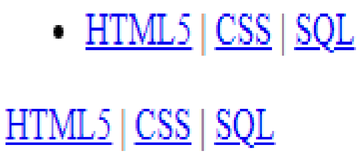
The <nav> element will allow us to group together a list of navigation links in a semantic manner.

A typical navigation link section would have looked like this before.



```html
<div id="nav">
    <ul>
        <li>
            <a href="html5.htm">HTML5</a> |
            <a href="css.htm">CSS</a> |
            <a href="sql.htm">SQL</a>
        </li>
    </ul>
</div>
<!-- Now we can encapsulate a group of links in a single semantic element and it looks organized. -->
<section>
    <nav>
        <a href="html5.htm">HTML5</a> |
        <a href="css.htm">CSS</a> |
        <a href="sql.htm">SQL</a>
    </nav>
</section>
```

Results:



- HTML5 | CSS | SQL

HTML5 | CSS | SQL

XML Vs JSON

JSON, XML, are Text-file formats that can be used to store structured data that can be handy for embedded and Web applications.

XML (Extensible Markup Language) has been around for more than 3 decades now and it is an integral part of every web application. Be it a configuration file, mapping document or a schema definition, XML made life easier for data interchange by giving a clear structure to data and helping in dynamic configuration and loading of variables!

JSON stores all of its data in a map format (key/value pairs) that was neat and easier to comprehend. JSON is said to be slowly replacing XML because of several benefits like ease of data modeling or mapping directly to domain objects, more predictability and easy to understand the structure. JSON is just a data format whereas XML is a markup language.

Difference between XML and JSON:

XML **(Extensible Markup Language)**	**JSON** **(JavaScript Object Notation)**
XML is a markup language, not a programming language, that has tags to define elements.	JSON is just a format written in JavaScript.

XML data is stored as a tree structure. Example –
<employees>
    <employee>
        <id>2001</id>
         <name>Varsha</name>
    </employee>
    <employee>
        <id>2001</id>
         <name>Varsha</name>
    </employee>
</employees>
	Data is stored like a map with key value pairs.
	Example –
	{“employees”: [
	 {"id":"2001", “name":"Varsha"},
	 {"id":"2002", "name":"Akash"}
	]}

Can perform processing and formatting documents and objects.	It does not do any processing or computation
Bulky and slow in parsing, leading to slower data transmission	Very fast as the size of file is considerably small, faster parsing by the JavaScript engine and hence faster transfer of data
Supports namespaces, comments and metadata	There is no provision for namespace, adding comments or writing metadata
Document size is bulky and with big files, the tag structure makes it huge and complex to read.	Compact and easy to read, no redundant or empty tags or data, making the file look simple.
Doesn't support array directly. To be able to use array, one has to add tags for each item. <subject>science</subject> <subject>maths</subject> <subject>computers</subject>	Supports array which can be accessed as – x = student.subjects[i]; where "subjects" is an array as – "subjects": ["science", "math", "computers"]
Supports many complex data types including charts, images and other non-primitive data types.	JSON supports only strings, numbers, arrays Boolean and object. Even object can only contain primitive types.
XML supports UTF-8 and UTF-16 encodings.	JSON supports UTF as well as ASCII encodings.
XML structures are prone to some attacks as external entity expansion and DTD validation are enabled by default. When these are disabled, XML parsers are safer.	JSON parsing is safe almost all the time except if JSONP is used, which can lead to Cross-Site Request Forgery (CSRF) attack.
Though the X is AJAX stands for XML, because of the tags in XML, a lot of bandwidth is unnecessarily consumed, making AJAX requests slow.	As data is serially processed in JSON, using it with AJAX ensures faster processing and hence preferable. Data can be easily manipulated using eval() method.

For parsing (converting string to object) and serializing (converting object to string) of XML and JSON, refer to the code examples.