HTML Overview

- HTML stands for **HyperText Markup Language**.
- Structured text with explicit markup denoted within < and > delimiters.
- Not *what-you-see-is-what-you-get* (WYSIWYG) like MS word.
- Similar to other text markup languages like `latex`.

Browser Technologies

HTML is just one of numerous *browser technologies*. Other notable technologies:

- **JavaScript**: used for scripting other browser technologies.
- **Document Object Model DOM**: API for accessing documents.
- **Cascading Style Sheets CSS**: used to style documents.
- **Uniform Resource Locator URL**: used to specify resources.
- **Fetch**: aynchronous fetching of resources.
- **Cookies**: permits browser and server to store key-value pairs within browser.
- **Storage**: allows storing of key-value pairs within browser.
- **Canvas** and **SVG**: allows graphics within browers.
- **WebGL**: provides access to local graphics hardware.

HTML Evolution

- HTML was designed as an application of IBM's *Standard Generalized Markup Language* SGML.
- HTML 1.0: used `href` for hyperlinks.
- Evolution added support for tables, client-side image maps.
- Evolution even added support for presentation elements like font, color.
- Modern HTML removes support for presentation elements; moved presentation into CSS.
- HTML documents are often sloppily marked up; standards define explicit behavior for some bad mark up.

HTML Markup

- A HTML document consists of a tree of HTML elements.
- A HTML element delimited between a start tag like <a> and an end-tag like .
- There may be text or other tags between the start tag and end tag. This is referred to as **element content**.
- The start tag may contain attributes, like <a `href="submit.cgi">`.
- The set of allowed attributes for any element are predefined with one exception: any element can have attributes with names starting with `data-`. Allows extensible attributes.
- A tag with empty content can be denoted as <br />; often simply use opening tag without closing tag; so simply <br>.

HTML vs eXtensible Markup Language XML

- The set of HTML tags and attributes are predefined with the notable exception of `data-` attributes. In XML, the set of tags and attributes are not predefined.
- XML documents must start with an xml declaration `<?xml version="1.0"?>`. HTML documents must start with a DTD declaration, currently `<!DOCTYPE html>`.
- XML documents must be **well-formed**: i.e., elements must be properly nested. That need not be the case with HTML.
- Empty XML elements must be denoted as either `<tag></tag>` or `<tag/>`. This need not be the case with HTML which permits empty elements to be denoted simply as `<br>`.
- The tags and attributes of an XML document can be constrained using an external specification language like **XML-Schema** or **RELAX NG**. If an XML document meets this external specification, then it is said to be **valid**.

HTML DTD's

The start of a HTML documents must contain a **declaration** which references a **Document Type Definition** or DTD. Some common declarations:

HTML 4.0 Never caught on.

```
<!DOCTYPE HTML PUBLIC
   "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
```

XHTML HTML as an XML document. Since XML must be well-formed, difficulties for web authors who were used to sloppy markup.

```
<!DOCTYPE html PUBLIC
   "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/
            xhtml1-strict.dtd">
```

HTML 5 Evolving modern HTML standard; simple DTD `<!DOCTYPE html>`.

Identifying and Locating Web Resources

- A **Uniform Resource Identifier** (URI) is an identifier for an abstract or physical resource.
- A **Uniform Resource Locator** (URL) is a URI with an access method which allows locating a resource.
- A **Uniform Resource Name** is a URI which uses specific sub-schemes and uniquely identifies a resource.
- Relative URLs relative to some base.
- Original RFC is quite readable.
- There is confusion about the above differences, URI and URL often used interchangeably; see this.

URI Components

Consider the URI
<http://zdu.binghamton.edu/cgi-bin/echo.pl?name=john&name=mary#label>

Scheme All URI's start with an identifier giving the specification it follows. This is followed by a `:` char. The example uses scheme `http`.

Authority Specifies the naming authority for the resource. Preceeded by a `//`. The example has the authority `zdu.binghamton.edu`, which corresponds to a hostname in the *domain-name system* (DNS). Can contains user-info (preceeded by an `@`), a host-name or IP address and a port number (preceeded by a `:`).

URI Components Continued

Path Separated from the authority by a / character. The example has the path `cgi-bin/echo.pl`.
It is terminated by a subsequent ? or # character.

Query Indicated by the first ? after the path and is terminated by a # character (or the end of the URI). The example has the query `name=john&name=mary`.

Fragment Identifies a secondary resource (relative to the primary resource). Follows a # character after the query. The example has a fragment `label`. This is not sent to the server.

URI Examples

```
https://zdu.binghamton.edu:8080/cgi-bin/hello.rb
  ?name1=fred&name2=john#label

http://128.226.116.131/

mailto:umrigar@binghamton.edu

file:///home/umrigar/cs580w/    #absolute paths only

urn:isbn:978-0596517748
```

Absolute and Relative URLs

- Absolute URLs are complete URLs containing scheme, hostname and path. Example: `href="https://developer.¬ mozilla.org/en-US/docs/Web/JavaScript"`.
- Relative URLs can omit parts of the URL which are filled in from the referring document:

  Use current scheme Use same scheme are current document. Example: `href="//developer.mozilla.org¬ /en-US/docs/Web/JavaScript"`.

  Use current scheme and host Same scheme and host as current document. Example: `href="/en-US/docs/Web/JavaScript"`

  Use current scheme, host and path Example: `href="Reference/Global_Objects/Array"` or `href="../HTML"`.

  Use current URL Example: different fragment in current document `href="#frag"`.

URI Encoding

- Encode characters which may have reserved meanings within a URI.
- RFC 3986 reserves special characters like /, ? and &.
- Special characters need to be escaped using *%hh* where *hh* is the ASCII code for the character.
  - Slash / represented as %2F.
  - Question-mark ? represented as %3F.
  - Ampersand & represented as %26.
- Alphanumerics, hyphen -, underscore _, period . and tilde ~ never need to be escaped.
- Characters need not be URI-escaped if used within a context where they are not special; for example, / does not need to be escaped within a query string.

JavaScript Encode URI Functions

`encodeURI(`*string*`)` Will encode only those special characters which do not have special use within a URI. So it will not escape characters like /, ?, #. Use to encode entire URI which does not contain special characters within contexts where they have special meaning. Decode using `decodeURI()`.

`encodeURIComponent(`*string*`)` Will encode all characters except -, _, ., !, ~, *, ', (and). Hence safe to use only on URI component. Decode using `decodeURIComponent()`.

JavaScript Encode URI Functions Examples

```
> uri = 'http://www.example.com?q=encode url'
'http://www.example.com?q=encode url'
> encodeURI(uri)
'http://www.example.com?q=encode%20url'
> encodeURIComponent(uri)
'http%3A%2F%2Fwww.example.com%3Fq%3Dencode%20url'
> decodeURI(encodeURI(uri))
'http://www.example.com?q=encode url'
>
```

Common HTML Attributes

`href`	Specifies absolute or relative URL to another resource.
`rel`	Specifies the relationship of the linked to resource from the linking resource.
`id`	Specifies an ID for element. The ID must be unique across the entire document.
`class`	Value consists of multiple space-separated identifiers. Element class can be used for attaching styling and/or behavior to the element.

`class` and `rel` attributes have been used to provide semantics to markup using microformats. Example microformats: *hCalendar* for events, *hCard* for contact information, *geo* for geographical information.

Page-Level HTML Elements

`<html>`	A single `<html>` element must be present enclosing entire content.
`<head>`	Contains meta-content like `<title>` (displays title in browser window bar), `<link>` for loading CSS stylesheets, `<script>` for loading JavaScript files.
`<body>`	Encloses actual document content.

Block-Level Markup

`<h1>`, ..., `<h6>` Headings at different levels.

`<section>` Delimits a section of the document. Usually followed by a <h*i*> element.

`<nav>` Used for delimiting content used for site navigation.

`<div>` Used for delimiting general block content. Usually used to attach style or behavior to a block using `id` or `class` attributes.

`<p>` Used for delimiting paragraphs.

List Markup

Unordered Lists Denoted using

```
<ul>
  <li>...</li>
  ...
</ul>
```

Ordered Lists Denoted using

```
<ol>
  <li>...</li>
  ...
</ol>
```

Definition Lists Denoted using

```
<dl>
  <dt>...</dt> <dd>...</dd>
  ...
</dl>
```

Table Markup

- Tables delimited using `<table>` tags.
- Rows within a table are delimited using `<tr>` tags.
- Table entries within a row are delimited using `<th>` tags (for heading entries) or `<td>` tags (for data entries).
- A table entry can span multiple columns (using attribute `colspan`) or multiple rows (using attribute `rowspan`).

Inline Markup

`<span>`	Simply used to delimit some content, similar to `<div>`. Example `<span class="keyword">while</span>`.
`<em>`	Emphasized text. Example: `<em>Important</em>`.
`<strong>`	Strongly emphasized text. Example: `<strong class="alert">Warning</strong>`
`<img>`	Used to embed an image Example: `<img src="smile-emoji.gif">`. Can also be used at the block level.
`<a>`	Hyperlinks. Example `See <a href="other-doc.html>other document</a>`.

Linking to Stylesheets

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

- End tag must not be present.
- Stylesheets are accessed synchronously. Content cannot be rendered until stylesheets available.
- `<link>` elements should be within the `<head>` section, though most browsers also allow within `<body>`.
- Various workarounds use JavaScript to load stylesheets asynchronously.

Linking to JavaScript: Traditional "scripts"

```
<script src="script.js"></script>
```

- End tag must be present.
- Script is accessed synchronously. Blocks the HTML parser while the script is downloaded and executed.
- For best efficiency, include after bulk of document body just before </body> tag.

Linking to JavaScript: Modules

```
<script type="module" src="module.mjs"></script>
```

- End tag must be present.
- Module is downloaded in parallel with HTML parsing (as though attribute `defer` is present within `<script>` tag).
- Module code is executed only after HTML parsing is completed.
- Module can `import` other modules.
- Module can `export` JavaScript objects.
- Module code is executed in `strict` mode turning off problematic JavaScript features.
- Can be included within `<head>...</head>` section.

For more details, see this gist.

Forms

```
<form action="http:www.google.com" method="get">
  Search: <input name="q">
</form>
```

- Forms need to be set up using `<form>` tags.
- `action` gives URI where form should be submitted.
- `method` can be `get` (default) or `post`. Other HTTP methods are **not supported**.
- `enctype` used when method is `post`. Default is `application/x-www-form-urlencoded`. Use `multipart/form-data` if uploading files. HTML5 allows `text/plain`.

Form Controls

- All form controls have a `name` attribute which gives the name by which that control is submitted.
- Usually form controls have to be embedded within a `<form>` element, but HTML5 allows using a `form` attribute specifying the `id` of any `<form>` element on that page.
- Form controls can be `disabled` which makes them inactive.
- Captions for form controls by putting control inside a `<label>` element or by specifying control `id` in the `for` attribute for `<label>`.
- Can group controls together using `<fieldset>`.

Form Input Control

`<input type="`*TYPE*`">` *Live example* from MDN. Less typing using *Local example*

- Main form input field.
- *TYPE* traditionally had values `button`, `checkbox`, `file`, `hidden`, `image`, `password`, `radio`, `reset`, `submit`, `text` (default).
- HTML5 added many more variants: `color` for color-picker, `date`, `datetime-local`, `time`, `month`, `week` for date-time, `email`, `tel` for contact information, `number`, `range` for numeric information, `url` for URLs.
- `autocomplete` attribute allows browser to fill in information previously saved by user.
- `pattern` attribute is a regex the entire value is matched against.

Miscellaneous Form Controls

`<select>`	Provides a menu of options using embedded `<option>` elements. Specify `multiple` attribute to allow multiple options to be selected. Can make an option selected by setting its `selected` attribute.
`<textarea>`	Multiline text input.

Escaping Special HTML Characters

HTML metacharacters can be represented by using *character references* with syntax inherited from SGML.

- Named character references `<`, `>`, `&quot;` and `&` can be used to represent the HTML metacharacters <, >, " and & respectively.
- Numerous other *named character references* in HTML like `&lambda;` and `&Delta;` to represent λ and Δ respectively.
- Numeric character references can be used to represent any unicode character using `&#`*nnnn*`;` `&#x`*hhhh*`;` where *nnnn* is its code point in decimal and *hhhh* is its code point in hex.
- Typically, one would depend on a library or framework to perform the escaping and unescaping.

Modern HTML

Separation of concerns:

- Presentation relegated entirely to CSS using **external** stylesheets.
- Behavior relegated entirely to *unobstrusive javascript*.
- HTML should specify content in as semantically meaningful a way as possible.
  - Do not use tables for layout, only for information which is naturally tabular in nature.
  - Use semantically appropriate HTML tag if possible, minimize use of semantically meaningless `div` and `span`.
  - Link to external semantics using itemscope and friends (a modern way of doing microdata) or *HTML + RDFa*.