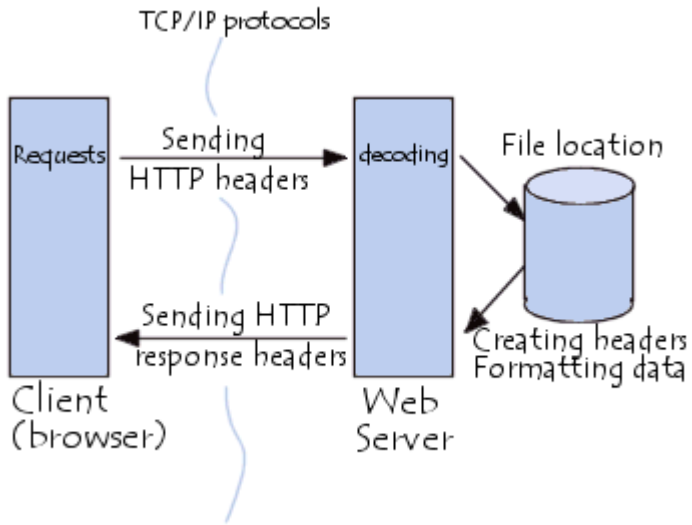
HTML and CSS

HTTP Requests and Responses

The basic premise behind all communication over the Internet is the HTTP Request Response cycle – An HTTP Request is made by the browser (or other client), and a HTTP Response is sent back to the client from the server.

That Response is the webpage, or other data that the server is set up to provide to its users.

All communication through this protocol is in text or binary form, regardless of what data is being transmitted.



Request types and GET vs POST

There are many types of HTTP Request that can be made: **GET**, **HEAD**, **POST**, **PUT**, **DELETE**, **TRACE, OPTIONS**, **CONNECT** to name a few. The most commonly used ones are **GET** and **POST**:

GET

Requests a specified resource from the server. In a **GET** request, all information is passed through the **URL**, and it can therefore be cached or saved as a bookmark. However, it has a limited length, and is completely unsecured. As such, it should never be used to deal with sensitive data.

POST

Submits data to be procced by a specified resource. A **POST** request is never cached, and is never reflected in the **URL**. However, in turn, they are not limited in data length or type, and are more secure than a **GET** would be. Unlike **GET** requests, **POST** requests can also pass binary data along.

HTML Pages

According to the current HTML standard – HTML5 – Web pages are built in three parts:

- HTML (HyperText Markup Language) – Defines the elements that exists on the page; the **content** that is shown on the page.
- CSS (Cascading Style Sheet) – Defines how things are displayed; the **styling** of the page.
- JS (JavaScript) – Defines scripts that run on the browser side of the page; the **behaviour** of the page.

HTML – Elements and Tags

Elements on a page are defined through HTML code which is a markup language consisting of heirarchially organized tags. Tags are either defined with a starting tag: **<element>** and a finishing tag: **</element>**; or as self-terminating tags: **<element />**.

```
<!DOCTYPE html>
<!-- This is a comment -->
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <!-- This is where the head -->
    <title>Page Title</title>
</head>
<body>
    <!-- This is where the body -->
</body>
</html>
```

In addition to their type, elements also have **attributes**, which handle additional information that is attached to them.

```
<a href="http://www.lexicon.se" class="link-button" id="link-lexicon">Lexicon</a>
```

In this instance, we define a **anchor** (clickable link, through the **a** keyword) element, that points towards the *http://www.lexicon.se* URL through the **href** attribute, has a style **class** of *link-button*, and an **id** of *link-lexicon*.

As a rule, tags denote ***what*** an element does, while attributes define ***how*** it does it.

HTML, HEAD and BODY

The HTML, HEAD and BODY tags are the most important ones in a html web page – the HTML tag defines the starting and finishing of the **entire document**, the HEAD element contains all the tags that provide **information to the browser**, and the BODY element contains all tags that defines content that is **visible to the user**.

Linking Resources

While their content can technically be entered into the HTML file itself, using the right tags, CSS and JavaScript can also be written in external files, which can then be linked into any HTML document using the <link> and <script> tags, respectively. This is generally preferrable to writing the code into the HTML file, since it can then easily be reused.

```
<link rel="stylesheet" type="text/css" href="theme.css">
<script type="text/javascript" src="myscripts.js"></script>
```

If for some reason you do need to put such code in the HTML itself however, you can do that with the <style> and <script> tags, respectively:

```
<script></script>      <style></style>
```

Useful HTML Tags

Containers
Container elements encapsulate other parts of the page. There is no functional difference between them, but there are conventions that dictate how and when one should use them:

<header> - Introductory content, the page ”header”, company logo etc.
<footer> - Information about the containing element, sitemap, related documents etc.
<nav> - Navigational links and elements.
<article> - Independent, self-contained content. Should make sense and be distributable on its own.
<aside> - Defines content aside from the content it is placed in, but related to it.
<section> - Defines sections in a greater document or article, such as chapters or other subdivisions.
 - A catch-all container tag for content that does not fit any above critera.

Text
Text can be written as is, but can be made into paragraphs with **<p>**. Text can also be surrounded with **** to control specifics about it, such as applying styling to it.

Images

Defines a image element on the page.

```
<img src="~/Content/Background.jpg" />
```

Lists
Lists consist of two element levels – the list itself, and its elements. **<ul>** can be used to create an **unordered** bullet list, while **<ol>** is used to create an **ordered** (numbered) one. List items are then defined with **<li>**.

```
<ol>
    <li>Coffee</li>
    <li>Tea</li>
    <li>Milk</li>
</ol>
```

```
<ul>
    <li>Coffee</li>
    <li>Tea</li>
    <li>Milk</li>
</ul>
```

```
<form action="/Home/Index" method="post">
    <input type="text" name="UserName" required />
    <input type="password" name="Password" required />
    <input type="checkbox" name="Remember" checked />
    <input type="submit" value="Submit" />
</form>
```

Forms
Forms are the normal way to pass information from one page to another, or from a page to a server. The form’s *action* defines where the information will be sent, and its *method* will determine how it is sent.

Each input tag defines a field in the form – a variable to be sent along. The *name* defines the handle by which it can be accessed, and the *type* defines how it will appear on the page. There are too many types to put them all in here, but the most common are: **text**, **number**, **password**, **checkbox**, **radio** (for radio buttons), **hidden** (for hidden values that gets passed along without the user seeing them) and **datetime**.

A form must always have a **submit**-type input, as this is the button that sends the form to its destination. It may also include a **reset**-type input, which will clear the fields of the form, letting the user start over.

Tables
When you want to show structured data on a webpage, tables are often useful. They are defined with **<table>**, with each row defined with **<tr>**. Each cell is then defined with either **<th>** for headers, or **<td>** for data.

You can control appearance and formatting through CSS, and you can also use the **rowspan** and **colspan** attributes to extend cells over multiple rows or columns.

```
<table>
    <tr>
        <th>header 1</th>
        <th>header 2</th>
        <th>header 3</th>
    </tr>
    <tr>
        <td>row 1, cell 1</td>
        <td>row 1, cell 2</td>
        <td>row 1, cell 3</td>
    </tr>
</table>
```

CSS – Cascading Style Sheets

Styling Rules
The appearance of HTML elements on a web page is controlled through CSS. CSS is defined as **style rules** that are applied to elements through **selectors**, that target one or more **element tags**, **classes** or **ids**.

```
body {
    padding-top: 50px;
    padding-bottom: 20px;
}
```

In this case, the **body** element is given a **padding** (distance between the edge of the element and the content it contains) of 50px from the top and 20px from the bottom.

As there are too many potential style rules to fit it into this sheet, I will not go into specifics here about all the various CSS properties. Look to http://www.w3schools.com for a CSS reference sheet.

Example Selector Patterns

type - selects all element of this **type,** * to select **all** types
.class - selects all elements with this **class**
#id - selects the element with this **id**
[attribute] - selects all elements with the target **attribute**
[attribute=value] - selects all elements with the target **attribute** is equal to **value**
type1, type2 - comma separator - selects **both** of these element **types** in this case
type1 type2 - space separator – selects all elements of type2 that are **within** a type1 element
type1>type2 - > operator – as above, but only if they are **direct child** elements
type1+type2 - + operator – select all type2 elements that are defined **immediately after** a type1
type1~type2 - ~ operator - select all type2 elements that are defined **at any point after** a type1
type:nth-of-type(n) - selects **every nth** element of **type** (replace n with 2 for every other one, etc.)
type:nth-child(n) - select every element of **type** that is the **nth child** of its parent.