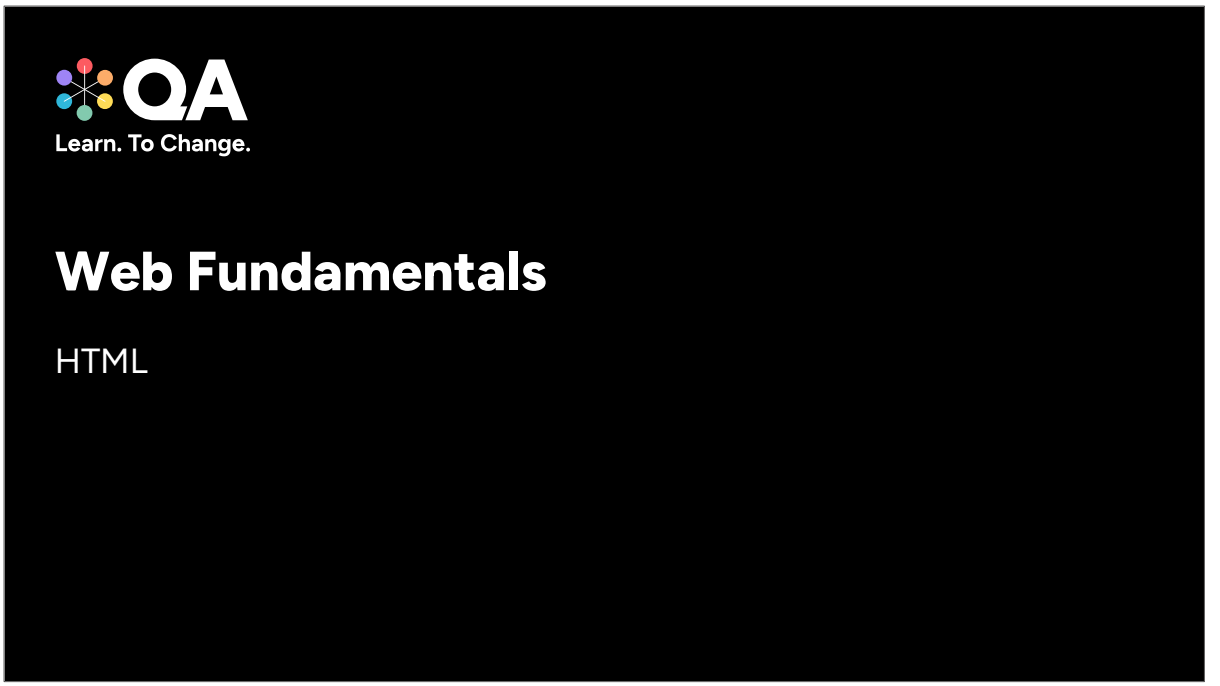

QA
Learn. To Change.
Web Fundamentals
HTML

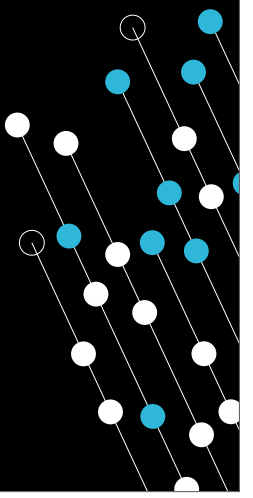
QA

Learning Objectives

- **How The Web Works**
- **Basic HTML**
  - HTML History and Syntax
  - Structural HTML
  - Hyperlinks
  - Lists
  - Tables
  - Forms
  - The <head> tag
  - The DOM

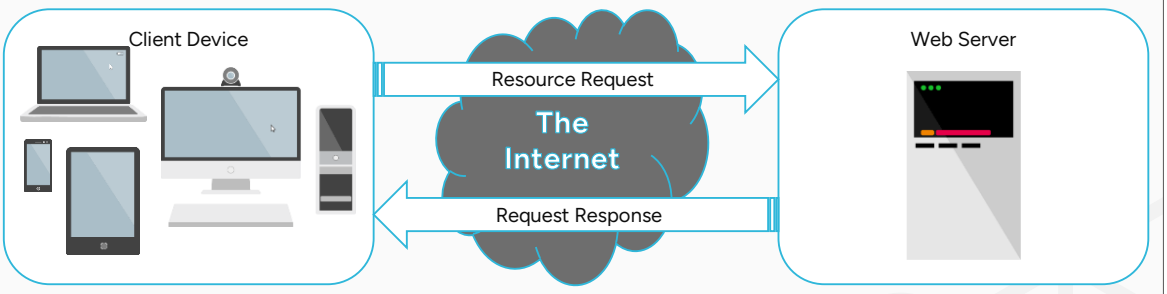
How the Web Works

- Clients and Servers
- URLs
- HTTP, HTTPS and SSL



QA

How the web works...



QA

Client Devices

- Need some form of browser to make requests
- Most commonly:
  - Chrome, Safari, Edge, etc
- Can also be
  - Smart Devices (Televisions, home appliances, etc)
- Makes request using a Uniform Resource Locator (URL) to specify where request is made to
- Uses HyperText Transfer Protocol (HTTP) to actually make the request

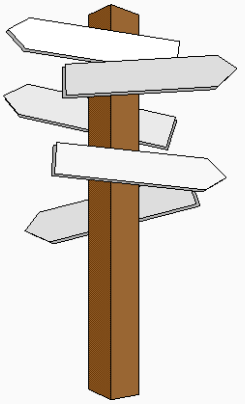
QA

Web Servers

- Needs to be running some Web Server software
- Most commonly:
  - Apache
  - Nginx
  - Microsoft Internet Information Server (IIS)
- Handles HTTP requests
- Dispatches response to requests
- Are addressed by URLs converted to IP addresses by Domain Name Servers (DNS)

QA

Introduction to URLs



- **Uniform Resource Locators (URL)**
  - Identifies location and protocol to access a resource
  - URLs are a form of Uniform Resource Identifier (URI)

Uniform Resource Locators or URLs are at the heart of the Web's Hypertext system. A URL describes both the linked resource in terms of the server address and the location within the server's file-system and the protocol to be used for retrieval. They can be embedded in HTML documents and are illustrated on the browser's display by highlighting a piece of text or image. They can also be entered directly into a special field on the browser window.

The term resource is used in preference to file or document because it is more general and encompasses all kinds of data. URLs are an implementation of the standard Uniform Resource Identifier scheme described in RFC1630.

A URL is divided into two parts separated by a colon. The basic syntax is:

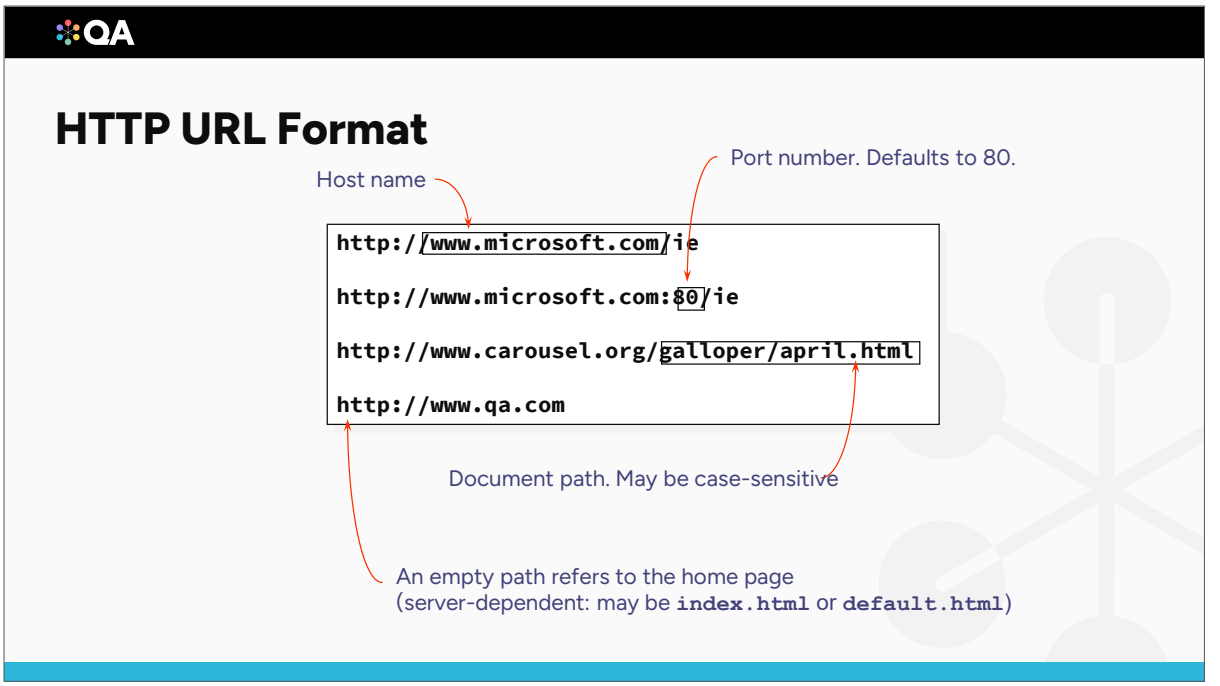
Scheme:Path

The scheme specifies the protocol to be used to retrieve the document (e.g. ftp, http, gopher) and the path specifies the location of the server and the resource. The path format is dependent on the scheme used.

Some examples of valid URLs

http://www.microsoft.com/ie
ftp://ftp.demon.co.uk
news:comp.infosys.www.announce
mailto:billg@microsoft.com

Text strings without a scheme, such as www.bbc.co.uk, are not really valid URLs because the scheme is required. However, most browsers will accept such URLs if they are typed in by the user and will assume a scheme of http these cases.



The Web uses HTTP to retrieve documents, the URL format is:

http://Hostname[:Port]/Document-Path

The hostname is the textual name of the machine on the internet or Intranet. This name is usually resolved by a domain name server (DNS) into a numeric address. Numeric addresses can be used although this is less flexible as it doesn't allow the server's address to be changed without changing all the URLs which reference it.

Hostnames often begin with **www** although this is by no means essential. It's just a convention which many sites adhere to.

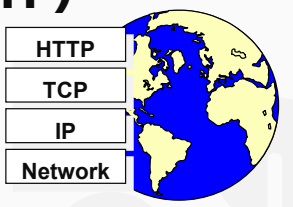
The host name can be followed by a colon and a port number. Web servers normally use the "well-known" HTTP port number 80. This is the default value if no port is specified. A common alternative port number is 443, which is used by several security systems. Port number 8080 is another popular alternative as it allows a normal Unix users (not the administrator) to run a Web server.

The document path is the location of the resource under the Web server's directory tree, the root of which may differ from that on the local machine. Posix (Unix) backslash characters are used as directory separators and the names are case-sensitive where this is important to the server's file-system. Unix servers have case-sensitive file servers but Windows NT server's don't.

If the document path is omitted the URL is taken to refer to the home page for that site. The home page is very server-dependent and configuration-dependent, but often refers to a file such as **index.html** or **default.html**.

QA

HyperText Transfer Protocol (HTTP)



Application-Level Protocol

- Technical information at http://www.w3.org
- TCP-based
- Current version is 3.0 (2022 release)

Lightweight

- Easy to implement clients and servers
- Stateless: each request is independent from the others
  - Other technologies required in order to enable e-commerce, online banking, etc

Request/response paradigm

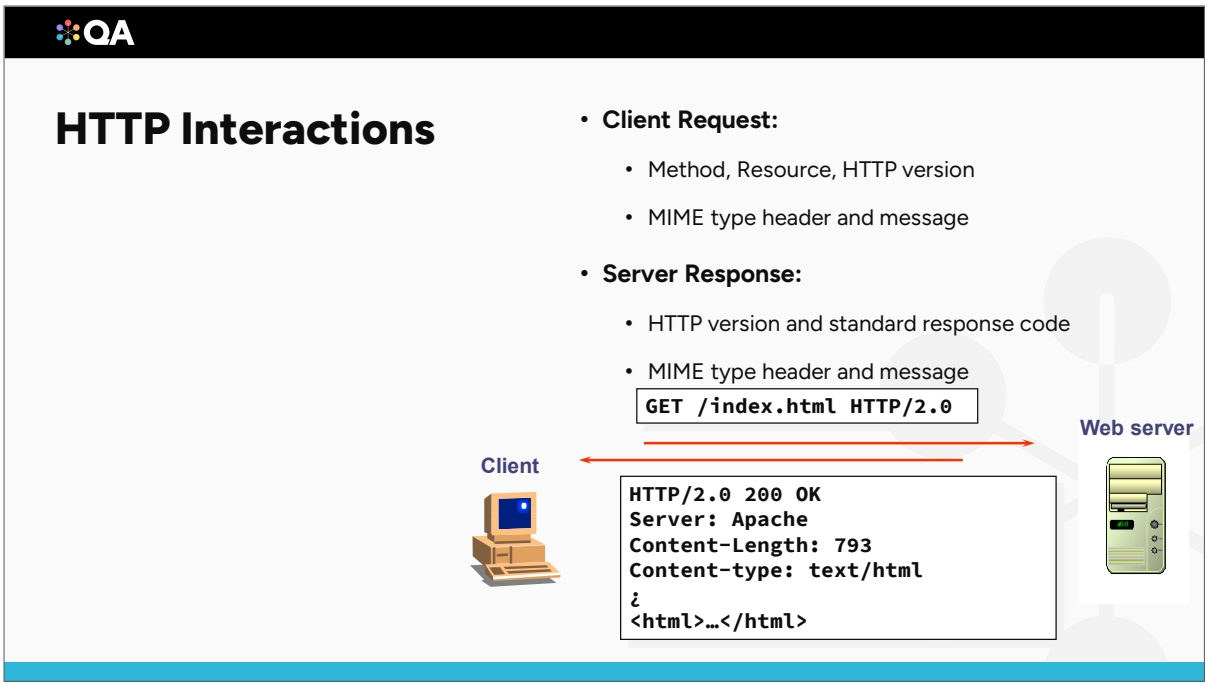
The core of the World Wide Web is the HyperText Transfer Protocol (HTTP). This is a lightweight application-level protocol which can be used to build hypermedia (e.g. Web) systems. It has been in use on the Web since 1990 and the current version is 1.1.

HTTP builds on a number of other Internet standards. Version 1.1 is not a rigid specification but more a documentation of current usage by today's Web community. The specification is available from the W3 organisation at **http://www.w3.org**, along with other documentation of interest.

HTTP is based on TCP, a connection-based part of the TCP/IP stack, even though interactions follow a single request and single response mechanism which might appear better suited to datagram-based techniques such as UDP. The reason that TCP is used is because the response often carries a large amount of data, perhaps a document or even an audio or video clip, and TCP has the capacity to transfer large amounts of data reliably without complicating HTTP itself.

The request/response may pass through proxies or tunnels (e.g. SOCKS) en-route. It may even be gatewayed onto networks which are not TCP/IP based.

The HTTP protocol is said to be 'stateless', each request stands alone, with no information being retained about the previous request either in the browser or server. This is fine for simple document requests but is not ideal for a whole range of applications where a user's identity must be tracked. There are a range of ad-hoc solutions to this problem.



HTTP is one of the many Internet protocols based on ASCII messages. Although not very efficient in terms of transmission bandwidth and machine parsing they are easy for humans to read and understand, a useful feature when testing servers.

Most communication is initiated by the client, a user agent such as Internet Explorer. The client establishes a connection with a server and sends a request consisting of a Method, a Uniform Resource Identifier (URI) and the HTTP protocol version, this is followed be a MIME-like message containing request modifiers and other client information. Note that a Carriage Return/Line Feed (CRLF) pair must always separate the MIME header from the body, even where both are null.

The server responds with a status line including the protocol version and an Internet standard error code, a MIME-like message then follows and contains the requested resource.

Readers should note that Web servers are sometimes referred to by the more generic name of HTTP servers because they really service HTTP requests as opposed to HTML based documents.

QA

HTTP Client Request

- **Method**
  - Action to perform on resource - GET, HEAD, POST
- **Uniform Resource Identifier**
  - Identifies a networked resource
  - Absolute URI used with a proxy server
  - Request URI used with an origin server
- **HTTP Version**
  - Major.minor version - Default (no version given) is 0.9
  - Version 2.0 now the most popular
  - Huge shift from 2015/16 onwards
- **MIME-like message - Contains request modifiers and forms data**

A method specifies the action to be performed on a resource. Three methods are commonly supported: **GET**, **HEAD** and **POST**. A **GET** method means retrieve whatever information is identified by the Request-URI. The Request-URI may be a script or program. The **HEAD** method gets header information about the Request-URI. It is used by proxy servers to determine whether the cached copy of a document should be updated. The **POST** method submits a stream of data to the resource identified by the URI. The action performed by the server is dependent on the URI but it can, for example, be used to submit HTML forms data to a server script.

URIs are formatted strings that identify a networked resource. HTTP servers can address a multiplicity of different resource types and this term is therefore more appropriate than file. Absolute-URIs are used with proxy servers and include the hostname or IP address of the origin server (the Web server where the resource is actually located). Request URIs are used with origin servers, a TCP connection is opened to the server and only the absolute path of the resource is transmitted, e.g.:

GET /pub/WWW/index.html HTTP/1.0

HTTP version numbers consist of a major and minor part. In general a greater minor number implies the addition of some field values which do not change the general message parsing algorithm. Major numbers are changed when the format of the message is altered. Versions: 0.9, 1.0 and 1.1 are currently found on the Web with 0.9 being the default and 1.1 now being the most popular.

HTTP version 0.9 interactions are also called simple-requests, only the GET method is supported and MIME is not used. This form of request is discouraged as it doesn't permit the server to identify the content type of the resource.

QA

HTTP Server Response

- **Simple Response/Full Response**
- **Status line**
  - HTTP version
  - Standard status code
  - Reason phrase
- **MIME like message**
  - Generated by Web server or by backend script
  - Header fields describe the requested resource
  - Modified using HTML <meta> tag
  - Requested data
  - Header and Data are separated by CRLF pair

After the server has received and interpreted the client request message, it returns an HTTP response message. If the client sent a simple-request only a simple-response will be returned. A simple-response will also be returned where the server supports only HTTP version 0.9.

The first line of a full-response (HTTP version 1.x) is a status line. This identifies the version number and includes a status code and phrase which follow the usual three letter Internet server format. The reason phrase is a textual representation of the code.

A MIME-like message follows. This consists of various header fields separated from the message body by a Carriage Return/Line Feed (CRLF) pair. The MIME message is normally generated by the Web server, the header information describes aspects of the document such as an expiry date. The message body contains the requested resource. Where the resource is an HTML file the header information can be modified using the **<meta>** tag.

The MIME message can also be generated dynamically where the resource is a backend server script. These scripts often use some information supplied by the client and may interact with other programs running on the Web server. They must generate a valid MIME message as a response, complete with appropriate header fields. It is therefore necessary to have some knowledge the MIME message format to write scripts.

QA

MIME And HTTP

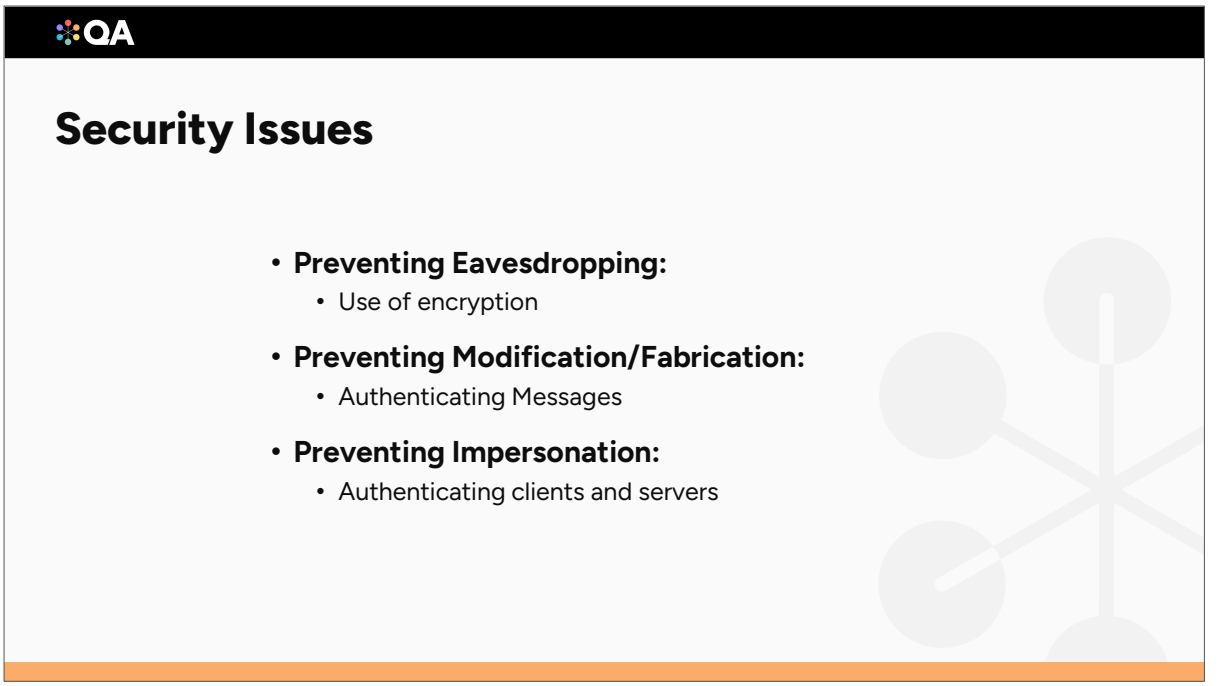
- **Multipurpose Internet Mail Extensions**
  - Based on Internet Mail (RFC 822)
  - MIME is defined in RFC 1521
  - HTTP usage differs from RFC 1521
- **Transmission of Multimedia Objects over Internet**
  - Header consists of colon-separated fields
  - Data contains requested object
  - Content-Type field describes object
- **Object Types**
  - Defined by IANA (Internet Assigned Numbers Authority)
  - Consist of type/subtype
  - Unofficial types preceded by x- (x-world/x-vrml)
- **Multipart Messages**
  - Multiple MIME messages each containing a header specifying the type of body data.

Rather than build its own multimedia messaging capabilities, HTTP augments the Multipurpose Internet Mail Extensions (MIME) defined in RFC 1521. MIME is itself based on the Internet mail message format defined by RFC 822. MIME is a flexible message format devised for sending multimedia objects over Internet mail. HTTP has a few features that are different from those described in RFC 1521. In particular there is no need to transmit the message body as 7-bit ASCII data as required by some Email systems.

The MIME message consists of a header, this is made up of a number of colon separated fields. The simplest document consists of nothing more than a Content-Type line followed by a CRLF pair and the message body. The Content-Type header line identifies the data in the body and consists of a type and subtype field such as **Content-Type: text/html.**

This field is used by the browser to select the appropriate application to display the returned data. Official content types are defined by the Internet Assigned Numbers Authority (IANA) and the list is growing all the time. Experimental content types are normally preceded by the letters x- although as browser and server can negotiate acceptable types this is not enforced.

MIME defines a multipart message type. The message body may then consist of multiple MIME messages each containing a header specifying the type of body data. HTTP usage differs from RFC 1521 in that each sub-message can contain a full set of HTTP header fields, not just a content field. Netscape used multipart messages as part of the client pull/server push feature for streaming data.



Why do we require security?

There are three basic points that we need to consider:

Eavesdropping - Network communications are typically not secure, particularly not in an Internet environment, where any number of unknown and unaudited networks could be transporting data between the client and server.

Modification and Fabrication - Even with encrypted data it is still possible that someone might insert new data into the communications stream between a client and server and attempt to subvert our communications.

Impersonation - Hacker's or malicious users might impersonate a different system or user in order to gain access to resources that they are not entitled to reach.

QA

Security - HTTPS

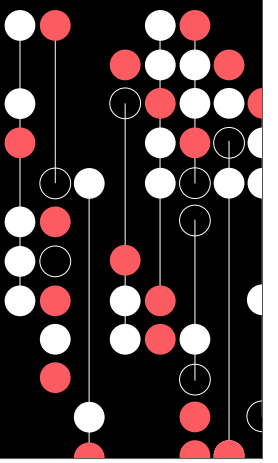
- **Essentially works in same way as HTTP**
  - Uses Secure Socket Layer (SSL) to encrypt data being passed
- **Lots of websites and development libraries and frameworks will work with or require HTTPS**
- **Stops 'eavesdropping' on data transfers between client and server**

QA

Security - SSL

- Secure protocol for sending information securely over the Internet
- Encrypts data transmitted between client and server
- Mostly uses 256-bit encryption in response to customer fears
- Requires web server to hold a valid certificate

HTML Basics



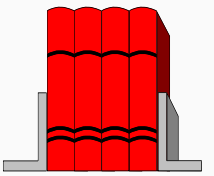
QA

Basics

- HTML History and Standards
- Basic HTML syntax, tags and errors
- Special Characters
- Hyperlinks
- Images
- Lists
- Tables
- Forms
- The Head Element
- The DOM

QA

HTML Standards



- **HTML 2.0**
  - Application of Standard Generalized Markup Language (SGML)
  - Attempted to update the standard

HTML 3.0 (Obsolete)

- Netscape and Microsoft ad-hoc extensions

- **HTML 3.2**
- **HTML 4.01**
  - Includes Cascading Style Sheets(CSS), DOM, scripting, XML and many other features
- **HTML 5**
  - The current W3C standard
  - 5.3 now a recommendation available at http://www.w3.org/TR/html5/

HTML describes the structure of the document and has the ability to make links to other resources. Structural markup differs from presentational markup embodied by languages such as PostScript in that the meaning and importance of elements within a document is specified, as opposed to their page location, font type and size. With HTML the actual rendering decisions are left to the display software which will use whatever resources are available locally, be that a high resolution bit mapped screen, text only or even audio or Braille output.

Publishers have long used a similar technique with documents to be sent to printers. To assist this process a language called SGML (Standard Generalized Markup Language) has been developed. HTML is defined in terms of an SGML Document Type Definition (DTD). SGML applications can use this definition to validate HTML documents.

All modern browsers boast broad support for HTML5, but we have legacy browsers to support and vary degrees of support across the different browsers for the various HTML5 APIs.

QA

A Simple HTML Page

Three sections to an HTML document

- Prolog(document type declaration), head and body

HTML Tags appear in angle brackets (< and >)

- Many tags appear in pairs, acting as containers
- Names are not case sensitive (convention is lower-ca
- Carriage returns are not significant

```
<!DOCTYPE html>
<html>
<head>
     <title>Success!</title>
</head>

<body>
Congratulations on completing
your QA Training course.
</body>
</html>
```

HTML distinguishes between text and tags. Tags can be easily seen because they are enclosed in angle brackets (**<** and **>**). Many (not all) tags appear in pairs which act as containers. The terminating tag uses a forward slash (**/**) character immediately after the opening angle bracket.

Tag names are not case sensitive, so **<BODY>** and **<body>** are exactly equivalent although lower-case is now expected. Any text not in tags is literal text which the browser will interpret as a document's content, appearing on the browser's screen. The non-tag text in the document is displayed without change, except that carriage returns in the HTML are treated like spaces. If you want to insert a line break in the document, there are tags designed for this purpose, one of which is **
**.

An HTML document is divided into three sections: the prolog, the head and the body. The Prolog identifies the HTML specification that the document conforms to. This is done with the **!DOCTYPE** directive. The outermost pair of tags, after the prolog, will be **<html>** and **</html>**. Within this section the document has the head and body, which, respectively, defines document header information and document content. The head section must contain at least a title, using the **<title>** container.

QA

HTML versions & DTDs

The different versions of HTML have their own DTD

- The appropriate DTD should be included in the HTML prologue

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/html4/strict.dtd">
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
"http://www.w3.org/TR/REC-html4/loose.dtd">
```

```
<!DOCTYPE html>
```

HTML 5

QA

Attributes of Tags

Many tags have additional attributes

- Some are required

```
<img src="……" />
```

- Some are optional

```
<h1 title="……">……</h1>
```

Attribute values should be enclosed in quotes

- Not necessary if one word attribute value
- Good practice

Many tags have additional attributes which specify various options. Some attributes are required. Many are optional and have default values if not supplied.

Attribute values are usually enclosed in quotes. Quoting is not usually necessary unless the attribute value contains more than one word. In later technologies, such as XML (see later), attribute values must be enclosed in quotes at all times.

QA

Comments

- Use <!-- and --> to enclose comments
- Not displayed by browser
- Can be used to hide scripts/styles from older browse

```
<html>
<head><title>Success!</title></head>
<body>
<!--A demo that comments can be used
to clarify HTML -->
Congratulations on completing
your <strong>QA Training</strong>
course.<br>
</body>
</html>
```

A document can be annotated with the Comment element, denoted by <!--. Comments can appear in any section. Text within the comment is generally ignored by the browser. The --> string is treated as the end of a comment, therefore comments cannot be nested.

Comments may be as long as you wish, but bear in mind that they are downloaded by the server to the client even though they are not displayed on screen. Long comments make for wasted bandwidth.

Containers may be nested but may not overlap. For example, this is correct:

** This text is a paragraph containing bold and italic text. Some words are both bold and italic. **

This is incorrect, because the closing **** tag occurs in the middle of a **** ... **** block.

** The quick brown fox jumps over the lazy dog. **

QA

Special Characters

- How can we express characters like "<" and ">"?
- Use character entities

Character	HTML Name
<	<
>	>
&	&
"	"
(non-breaking space)	

```
If the relative density &gt; 1.0 then
&quot;you have a problem!&quot;
```

```
If the relative density > 1.0 then "you have
a problem!"
```

Since the angle brackets (**<** and **>**) have a special meaning to HTML, how can they be displayed as part of a document's contents? HTML provides an escape mechanism for specifying special characters. These escapes are called character entities.

These begin with an ampersand, this is followed by the decimal index in the character set or a name and terminated by a semicolon.

The most common (and standard) entities are:

< **<**

> **>**

& **&**

" **"**

There are also a number of character entities for expressing special or uncommon characters outside the conventional ASCII 7-bit character set. This is necessary because different operating systems encode these characters in different ways, and HTML strives to be portable. The US ASCII or ISO-8859-1 character set is used. This has characters for most European alphabets. Where the keyboard doesn't provide a character, or where the character may be interpreted as part of the markup, a character entity must be used. For example the "e acute" character (é) is entered as **é** or as **é**

Windows character map can give you the codes for special characters so you don't have to remember them.

QA

Syntax Errors and Non-Standard HTML

Browsers never say “syntax error”

- May respond in different ways to incorrect HTML

Browsers ignore unrecognised tags and attributes

Allows updating of HTML standards

Non Standard HTML:

- Blink

```
<blink>Blinking text</blink> can be annoying,
```

- Marquee (Still works in many browsers)

```
<marquee>This message will scroll</marquee>.
```

- Not recommended as they could be removed at any time

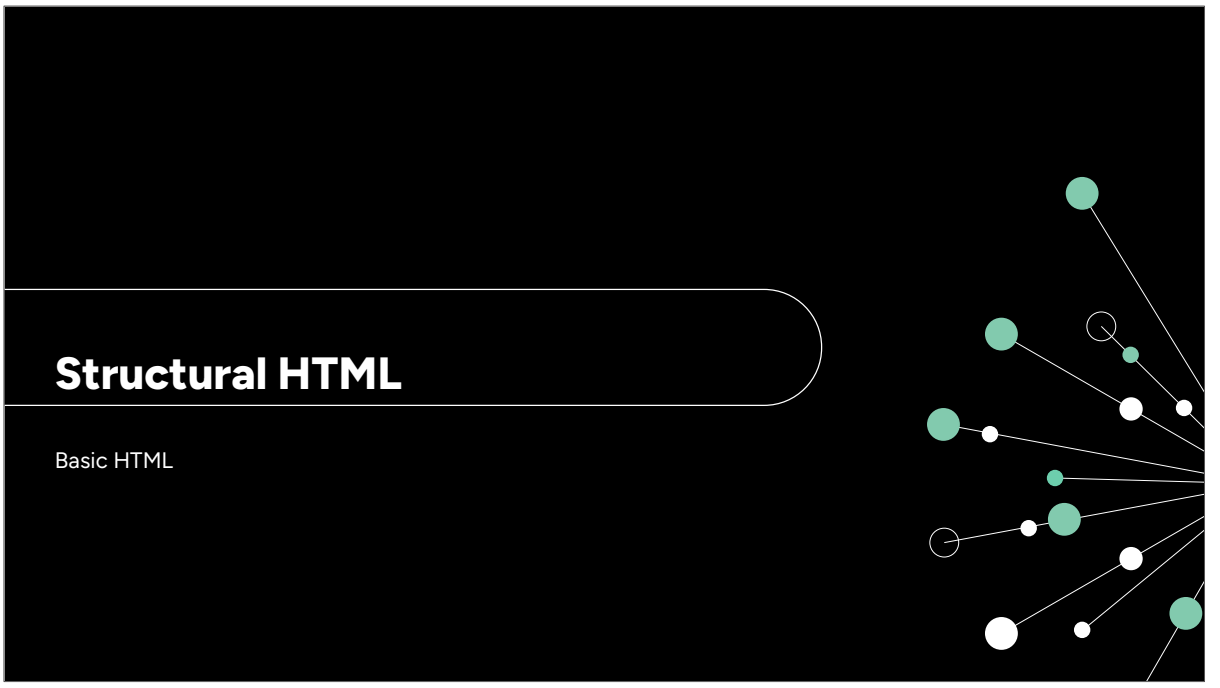
A browser will try to make sense of HTML no matter how many syntax errors it has, although they may have different ways of recovering from incorrect HTML.

Browsers are designed to ignore HTML which they do not understand. Unrecognised attributes on an HTML tag are also ignored, although text which is not within tags is always displayed.

This enables future enhancements to be made to the HTML markup language without having to upgrade all the browser tools across the Internet or a company’s intranet. Of course, old browsers won’t be able to make use of the new features that they don’t understand, but at least the rest of the page content will be correctly displayed.

There are some other text formatting tags which are non-standard. Firefox supports the **<blink>** tag which, as its name suggests, makes characters flash. Microsoft supports the **<marquee>** tag which creates a scrolling display on screen.

Not generally recommended, as they are not universally supported, and also because they don’t look particularly good on screen.


Structural HTML
Basic HTML

QA

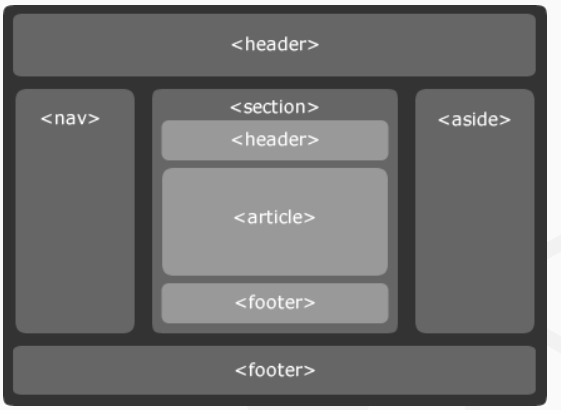
HTML5 Structural Elements

HTML5 has a series of structural elements

- To create a more semantically structured page

The main building blocks of HTML5 are:

- <header>
- <nav>
- <section>
- <main>
- <article>
- <aside>
- <footer>



If you look at some of the more obscure HTML 4 tags like <kbd>, <samp> and <var> you can see the scientific roots of HTML 4. HTML5 is a child of its time for a mainstream, information-based web. If you consider your own pages they are probably littered with tags with semantically useful id attributes such as header, foot, nav and content. To save us from the swarm of tags more suitable structural elements have been added.

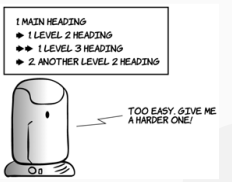
As previously discussed, HTML5 introduces a new sectioning set of tags. These allow us to provide semantic description to a page and place the document into a more human readable and better structured document. Over the next pages we will examine these elements and understand how to use them effectively and appropriately.

QA

Headings and sectioning elements

HTML5 documents have an outlining algorithm

- Generates a table of content based on the section and heading you've use
- Important for assistive technology and search engine optimisation



Sectioning elements create a logical structure in the document outline

```
<body>
  <h1>Main heading</h1>
  <p>Some text</p>
  <h2>Level 2 heading</h2>
  <p>Some more text</p>
  <h3>Level 3 heading</h3>
  <p>A bit more text</p>
  <h2>Another level 2 heading</h2>
  <p>The last bit of text</p>
</body>
```

The document outline is the structure of a document, generated by the document's headings, form titles, table titles, and any other appropriate landmarks to map out the document. The user agent can apply this information to generate a table of contents, for example. This table of contents could then be used by assistive technology to help the user, or be parsed by a machine like a search engine to improve search results.

The sectioning elements <section>, <article>, <aside> and <nav> can all help to create a more logical structure in the document outline.

QA

The <header> element

- **Normally the first element of the document**
  - Should act as container logos, links back to home etc.
  - Will usually contain a `<h1>` to `<h6>` to denote level of header
- **There can be one `<header>` per sectioning bloc**

```
<header>
  <a href="/">
    <img src="logo.png" alt="QA Home">
  </a>
  <h1>My Main Title</h1>
  <h2>My Sub title></h2>
</header>
```

Assuming we are using the <h*n*> tags as is appropriate in modern mark-up development, the main heading will normally have a <h1> element within.

The specification says it should be used to represent a group of introductory or navigational aids. It can contain an optional hgroup element. It can also be used to wrap a section's table of content, search form or similar.

Essentially when a <h*n*> element on its own is not suitable, there is other related information that makes up the block you should use a <header>.

Each block element, be that the <html> root or an <article> element for example can contain a <header> element. You may not have more than one <heading> per sectioning block.

The optional <hgroup> may only contain <h*n*> elements its purpose is to take a heading element, which would normally appear in the outline, out of it!

QA

The <nav> element

- `<nav>` is used to mark up navigation
- Should be limited to links within the page and site
  - Not sponsored links for instance
- Links normally surrounded by `<li>` within a `<ul>`
- Multiple `<nav>` allowed, each should contain a related categ

```
<nav>
  <h2>Main site navigation</h2>
  <ul>
    <li>
      <a href="/">Home</a>
    </li>
    <li>
      <a href="/aboutsus.html">
        About Us
      </a>
    </li>
  </ul>
</nav>
```

Not all groups of links on a page need to be in a nav element – the element is primarily intended for sections that consist of major navigation blocks.

It is common for footers to have a short list of links to various pages of a site, such as the terms of service, the home page, and a copyright page. The footer element alone is sufficient for such cases; while a nav element can be used in such cases, it is usually unnecessary.

<nav> elements can be nested inside other sectioning elements, <footer> and <header> most commonly.

QA

The <footer> element

- **Many `<footer>` elements may occur on a page**
- **Appear at the end of a sectioning element**
  - blockquote, body, div etc.
- **Contains information about the section or document, e.g. the author**
- **The `<footer>` element requires no heading element**
  - Unique in the sectioning element
  - It is optional and can be added
- **The above code is a 'fat footer'**
  - The **`<small>`** tag represents small print in HTML5
  - Add a **`<nav>`** element within if links required

```
<footer>
  <small>
    This tag has been redefined
  </small>
</footer>
```

The footer element represents a footer for its nearest ancestor sectioning content or sectioning root element. A footer typically contains information about its section, such as who wrote it, links to related documents, copyright data, and the like.

When the footer element contains entire sections, they represent appendices, indexes, long colophons, verbose license agreements, and other such content.

A common use of the <footer> element is to create a so-called 'fat footer'. "Fat Footer" are footers that contain a lot of material, including images, links to other articles, links to pages for sending feedback, special offers... in some ways, a whole "front page" in the footer.

QA

The <article> element

- **Represents a self-contained composition on the page**
  - A blog entry
  - Comic strip
  - Video
- **Articles represent indivisible units of work**

```
<article>
     <h2>Yesterday</h2>
     <p>Some stuff goes here</p>
</article>
<article>
      <h2>Today</h2>
      <p>Some more stuff goes here</p>
</article>
```

The <article> element represents a unit of work that makes up a significant section of data. This may include a <video> element, a blog entry or news story, or results from a dynamic server page.

As is common with all of our sectioning elements the first tag inside an <article> must be a heading.

QA

The <aside> element

- **`<aside>` provides tangential information to a block**
  - It should assist but not be essential to the main document
- **For example nested within an `<article>`**
- **Or used at a page level to denote 'sidebar' conter**

```
<article>
   <h1>My Blog Post</h1>
   <p>...</p>
   <aside>
     <h1>Glossary</h1>
     <dl>... </dl>
   </aside>
 </article>
```

```
<aside>
    <h2>Blog roll</h2>
    <ul>
      <li><a href="#">Link 1</a></li>
      <li><a href="#">Link 2 Friend</a></li>
    </ul>
 </aside>
```

The <aside> element is an often misused tag in HTML5. Its purpose is to provide data that is tangential to the main block to assist with related but non-essential information. In the first example above it is used to enclose a glossary.

With the new definition of aside, it is important to remain aware of its context. When used within an article element, the contents should be specifically related to that article (e.g. a glossary).

While in the second example which would exist as a child of the <body>, the contents relate to the site (e.g. a blogroll, groups of additional navigation, and even advertising if that content is related to the page).

QA

The <section> element

- **`<section>` is used to break up semantic elements**
  - Different parts to a news story or a group of links
- **Should be used in conjunction with a heading**
  - Otherwise the **`<section>`** is untitled in the HTML outline
  - Most generic and easiest to abuse semantic element
  - Do not use as a pure stylistic container

```
<body>
...
  <section>
    <h2>
      heading level = section nesting level
    </h2>
    rest of the content
  </section>
...
</body>
```

The section is used to break semantic articles like the <article> or <nav> into smaller chunks. There are a few simple rules in their use:

- Do not use it as a container for styling to scripting the element should be used
- Most generic and least meaningful of the other sectioning elements use them when more appropriate
- The section must be followed by a <h*n*> element

<sections> unlike provide a child level in the document outline, they are a structural sub level in the hierarchy. A common mistake is new developers to HTML5 think the tag is obsolete. As we will discuss shortly when you are using a <Tag> as a container presentational rather than informational structure the <section> element should not be used.

QA

<Div> Is Still Valid HTML5

- **You can still use the `<div>` tag no change to it in HTML5**
  - I.e. A generic element for structuring a page
  - Has no semantic meaning except via attributes
- **Used if no semantic alternative or as a CSS wrapper**

```
<body>
     <div id="wrapper">
          <header>…</header>
          <nav>…</nav>
          ...
     </div>
</body>
```

The new semantic elements allow us to replace a lot of 's old functionality. It is not a tag to discount in this new semantic world. The use of is perfectly appropriate if no suitable semantic definition can be found. Its most common purpose is for stylistic functionality where we want the rendered mark-up to look different but not apply meaning.

The same is true of the element. There are may more suitable semantic definitions such as <mark> but its use as a generic content wrapper is still very useful.

QA

When to use what

Element	Typical Content	Typical Parent Element	Typical Child Element
<header>	Title, logo, banner, Introductory information	Body, Section, Article	Nav, Section
<nav>	Primary navigation content	Body	Section, Nav
<section>	Generic page section	Body	Article, Header, Footer, Aside, Nav
<article>	Story, subsection, blog post	Body, Section	Section, Header, Footer
<aside>	Sidebar content, tip, quotation	Body	Section, Article
<footer>	Footer, summary, copyright, info, secondary navigation	Body, Section, Article	Nav, Section
<main>	Unique content, central to the topic of the document	Body	Section, Article, Aside (not repetitive content like nav, header or footer)

QA

Restructuring A Blog With HTML5

- Each blog entry would be an article
- Providing clear semantic intent for the page

```
<article>
        <header>
            <h2>My HTML5 Blog</h2>
            <p>March 15th 2010</p>
        </header>

        <p>Much less divitius occurs!</p>

        <footer>
            <address>
            <a href="..">Posted in</a>
            </address>
        </footer>
</article>
```

Here we have implied some intent meaning that the correct semantic tag is an <article> specifying a single unit of information. The introductory mater is encapsulated in a header and the footer contains closing navigation. While the main body of the article can go back to using <p> for what it was intended!

With the right use of CSS selectors we can easily style these specific articles (more on this later).

Still, there are issues with the article. The data is in a <p> for example which does not emphasis the intent of the data being displayed.

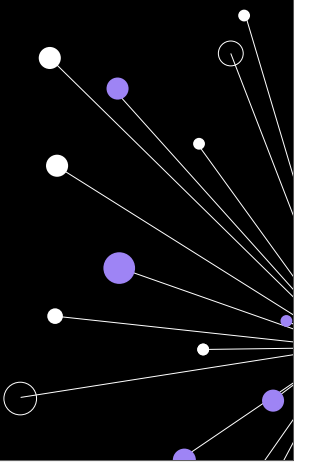
QA

Quick Lab 1 – Structural HTML

Convert an HTML document into a semantically tagged document and test the outcome

Hyperlinking

Basic HTML



QA

Hyperlinks

- **Hyperlinks connect related pieces of information**
  - Can point to separate documents located on different Web servers
  - Hyperlink text is shown by colour change or underlining
  - Linked document is loaded by clicking text/object with mouse
- **`<a>` … `</a>` tags define the hyperlink**
  - The HREF attribute takes a URL as a value

```
If you want to find out about
our courses, you could try
<a href="http://www.qa.com/index.html">
QA Training's home page</a>
or telephone us on
+44 (0) 1793 696000.
```

If you want to find out about our courses, you could try QA Training's home page or telephone us on +44 (0) 1793 696000.

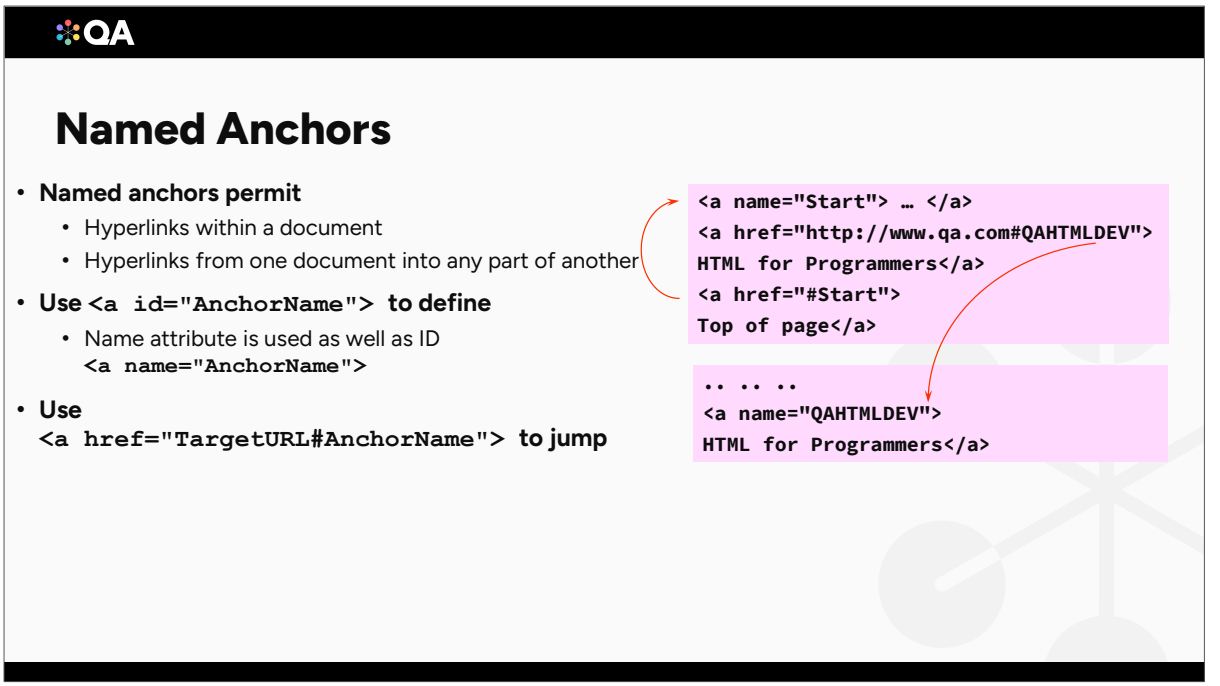
Structural markup is extremely useful, as it allows a designer to specify the layout of the page. However, the real power of HTML lies with the ability to include hypertext links (hyperlinks) in the document. A hyperlink is a piece of text or image which when clicked loads some linked resource. The link can point to another section of the same document (as in a table of contents) or another resource, located on the same server or a different server. The linked resource can be another HTML document or an image to be displayed directly by the browser, or even some external format such as a movie clip. As far as users are concerned they are simply navigating around related pieces of information and may be entirely unaware of the location of the server on the Internet or the actions taken to download and display files.

The anchor container element, **<a>** … ****, marks the start and optionally, the end of a hyperlink. In its simplest form, the **<a>** tag has an **HREF** attribute which consists of a URL.

The hyperlink can contain other elements and text; these are usually highlighted by the browser using a different colour from the default or by underlining, although you can add additional formatting if you wish.

For example:

QA Training home page



Named anchors permit hyperlinks to jump around within a document, and also hyperlinks that lead from one document into any position in another document.

To define a named anchor use the **<A>** tag with the **NAME** attribute.

** ... **

HTML4 supports the use of the **ID** attribute for the named anchor, as follows:

** ... **

To jump to a named anchor within the current document, use the **<a>** tag with the **HREF** attribute and the name of the anchor.

** ... **

To jump to a named anchor in a different document, use the **HREF** attribute again, but combine the named anchor with the URL, separated by a **#** symbol.

** Don't click me!.**

QA

Relative and Absolute URLs

```
<!-- Absolute pathname -->
<a href="http://www.qa.com/index.html">... </a>

<!-- Relative pathname: find 'index.html'
   relative to current page on this server -->
<a href="index.html">... </a>

<base href="http://www.myserver.com/backissues">
<!-- This is a relative pathname, but is interpreted
   relative to the setting in <base> above.
   Therefore this relative URL always points to the
   page on the server. -->
<a href="jan13.html">... </a>
```

URLs in hyperlinks can be specified in two ways: as *absolute* and as *relative* (or *partial*) addresses.

An absolute address includes a full server name and path. It would have exactly the same meaning if copied into another HTML document.

A relative (partial) URL gives the resource name without specifying the Web server. The name is relative to the current location. Partial URLs are normally used when specifying documents on the same server. They are useful because they are economical (the page is a little shorter) and also because groups of documents can be moved from server to server without extensive editing.

In order to interpret a relative URL, the browser needs some context. This is usually the current document. The problem with this scheme is that hyperlinks in such a document cannot be followed if the document is being read out of context (for example, if the document has been copied into the local file system of a PC). The context information can be explicitly defined using the **<base>** tag.

QA

Hyperlink Targeting

The `target` attribute of a hyperlink tag can have one of these predefined values:

- `_blank` A new blank window or new tab in a tabbed browser.
- `_parent` Immediate parent of the document the link is in.
- `_self` The same window the link was clicked in.
- `_top` The full body of the window.

```
<a href="circont.htm" target="_parent">...</a>
<a href="circont.htm" target="_top"  >...</a>
<a href="circont.htm" target="_blank" >...</a>
<a href="circont.htm" target="_self" >...</a>
```

The **TARGET** attribute of a hyperlink tag can also have one of these predefined values:

_blank A new blank window. This window is not named.

_parent Immediate parent of the document the link is in.

_self The same window the link was clicked in.

_top The full body of the window.

QA

Mailto: links

Changing the scheme from **`http://`** to **`mailto:`** tells the browser to execute it's default e-mail behaviour.

This usually opens a new e-mail for composing

```
<a href="mailto:instructor@qa.com">Contact Instructor</a>
```

Can also append a subject

```
<a href="mailto:instructor@qa.com?subject=Hello Instructor">Contact Instructor</a>
```

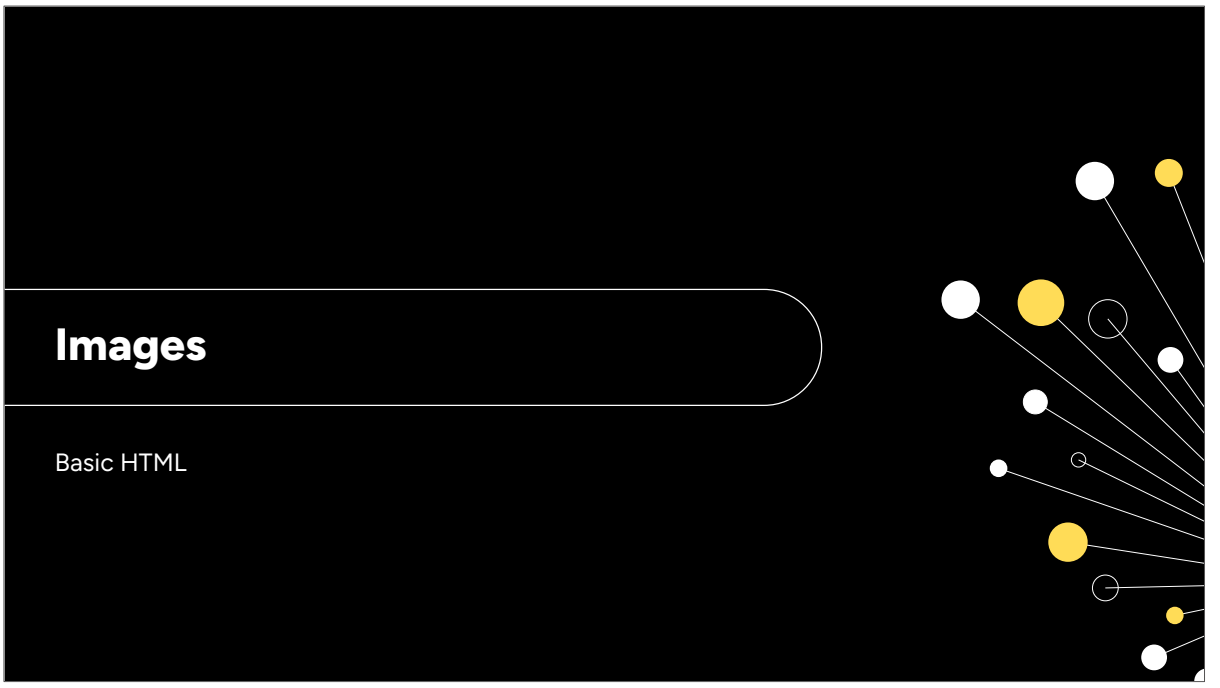
And even some content for the body of the e-mail

```
<a href="mailto:chris.bruford@qa.com?subject=Hello Instructor&body=Just dropping a line to say hello">Contact Instructor</a>
```

QA

Quick Lab 2 – Hyperlinks

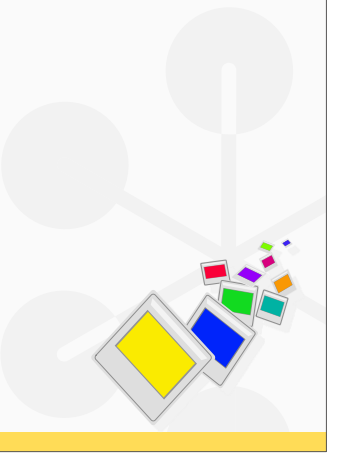
Add some links to HTML and ensure that they work


Images
Basic HTML

QA

Image File Formats

- **A very large number of graphics file formats**
  - BMP, GIF, PCX, JPEG, TIFF, CGM ...
- **Most browsers support a few well-known formats**
  - GIF (Graphics Interchange Format)
  - JPEG (Joint Photographic Experts Group)
  - Text-only browsers (e.g., Lynx) ignore all images
  - Usually possible to disable images to reduce bandwidth
- **Some browsers support other image file formats**
- **PNG (Portable Network Graphics) format**
- **WebP – modern image format that provides superior lossless and lossy compression**



In the world of graphics there are nearly as many file formats as there are utilities and tools available to process them.

Most browsers support the GIF (Graphics Interchange Format) and JPEG (Joint Photographics Experts Group) file formats directly.

Remember there is no compulsion for a web browser to display images at all. Lynx is a text-based web browser which does not attempt to display images, and most of the popular browsers allow the user to ignore images in order to speed up page load times and reduce bandwidth.

Some browsers, such as Microsoft Internet Explorer, support native Windows file formats such as BMP. For the purposes of this chapter we will assume that the browsers support GIF and JPEG as a minimum.

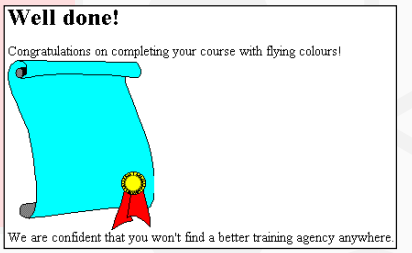
A standard known as PNG (Portable Network Graphics) has also been defined and works well in most current browsers.

QA

Inline Images: the <img> Tag

```
<html><head><title>Success!</title></head>
<body>
<h1>Well done!</h1>
Congratulations on completing your
course with flying colours!<br/>
<img src="scroll.gif" alt="Scroll" />
<br/>
We are confident that you won't find a
better training agency anywhere.<br/>
</body>
</html>
```

This text is shown when images are unavailable or disabled



The **IMG** tag is used to insert an inline image in the document. You must always provide a **SRC** attribute which is the URL (relative or absolute) of the image you wish to insert. This tag is not a container, so there is no matching **</img>** tag.

The **ALT** attribute of the **IMG** tag specifies text that will be displayed in place of the picture if the browser is not capable of displaying the image, or if the browser is normally capable of showing picture but this feature has been disabled. This option is important, because text-based users will be unable to make much sense of your web page without it. This is particularly true when dealing with hyperlinked graphics, as will be seen later.

QA

Image Height and Width

- **A picture's size is usually found from data in the file**
  - Can change this with WIDTH and HEIGHT attributes
- **Do not use WIDTH and HEIGHT attributes to reduce the display size of an image**
  - The browser will need to download the entire image

```
<!-- Force the image to be 100x100 -->
<img
  src="scroll.gif"
  alt="Scroll"
  width="100"
  height="100" />
```

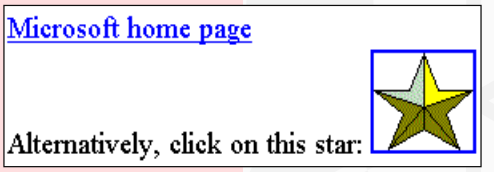
A picture's size is usually found from data in the file when it is downloaded. This can be overridden using attributes of the **<img>** tag.

The **WIDTH** and **HEIGHT** attributes specify the size at which the picture is drawn. If the picture's actual dimensions differ from those specified, the picture is stretched to match what is specified. Information is not strictly necessary, as the browser should be able to determine the dimensions of the image from the file, but supplying dimensions in the HTML file will enable the browser to draw a placeholder of appropriate size for the picture before it is loaded.

QA

Graphical Hyperlinks

```
<html>
<head><title>Graphical links</title>
</head>
<body>
<p>
<a href="http://www.microsoft.com">Microsoft home page</a>
<br/>
Alternatively, click on this star:
<a href="http://www.microsoft.com">
 <img src="star.gif" alt="MS Home Page“/>
</a>
<br/>
</p>
</body>
</html>
```



Images can be hyperlinks as well as just inline pictures. To achieve this, place the **<img>** tag within **<a>** and **** tags in the same way that you would enclose text for textual hyperlinks.

Most browsers will put a hyperlink border around the image. Sometimes the border around the image is a little obtrusive. In such cases, use the **border:0** CSS property of the **<img>** tag to remove it.

It is very important to put in **ALT** text for the image to accommodate browsers which are not able to display the image; there is nothing less helpful than a hyperlink with no clue as to where it points.

QA

Thumbnail Images

Images can be large

- An issue if the users' connection is slow

For larger images, provide a thumbnail image

- Acts as a preview
- Leads on via a hyperlink to the full image

Create thumbnail as separate image

Don't use WIDTH and HEIGHT attributes to shrink

```
<a href="pictures/monalisa.gif">
      <img src="thumbnails/monalisaSmall.gif“ alt="Mona Lisa“/>
</a>
```

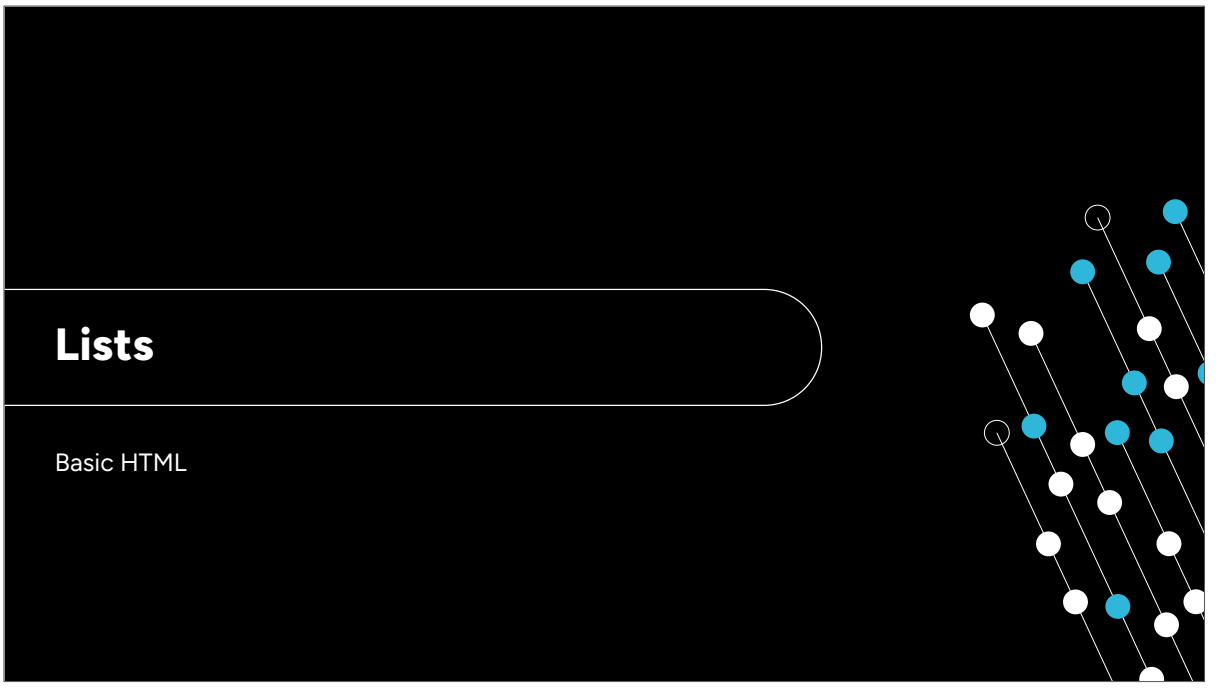
Remember that images can be large, and this is an issue for browsers connected through a slow or busy link. Small images (up to around 100x100 pixels) can be included inline without any problems, but for larger images it is considered courteous to provide two versions of an image: a "thumbnail" image at small size and with a small palette which previews the picture. This image can then lead on, through a hyperlink, to the full image. This allows users to download the images they want after having seen the preview.

The correct way to generate a thumbnail image is as described above: to create a separate image file from the original image. There is another way which might seem easier but is not efficient. The **HEIGHT** and **WIDTH** attributes of the **<img>** tag can be used to specify the size at which the picture is drawn. If these attributes are different to the actual size of the image, the browser will load the whole image and reduce or enlarge it to suit. This does create thumbnail images if the **HEIGHT** and **WIDTH** values are smaller than the original image but is very inefficient because the whole image has to be downloaded first. This rather defeats the object of having the thumbnail image in the first place!

QA

Quick Lab 3 – Images

Add some links to images to an HTML page and view them


Lists
Basic HTML

QA

Lists in HTML – Ordered and Unordered

List Items are inserted into lists using the `<li>` tag

Ordered lists (numbered, etc) are created by surrounding list items with `<ol></ol>`

- Start point can be defined using the start attribute

```
<ol start="5">
  <li>List item 1</li>
  <li>List item 2</li>
</ol>
```

5. List item 1
6. List item 2

```
<ul>
  <li>List item 1</li>
</ul>
```

- List item 1

QA

Lists in HTML – Description Lists

Description lists are created by using the **<dl>** tag

<dt> tags are used to define a term or name in a description list

<dd> tags are used to describe a term or name in a description list

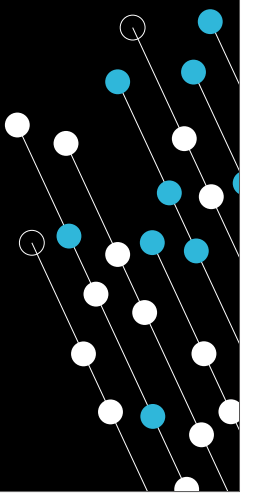
- Can contain paragraphs, line breaks, images, links, lists, etc

```
<dl>
  <dt>Summer</dt>
  <dd>Season when it is warm</dd>
  <dt>Winter</dt>
  <dd>Season when it is cold</dd>
</dl>
```

Summer
    Season when it is warm
Winter
    Season when it is cold

Tables

Basic HTML



QA

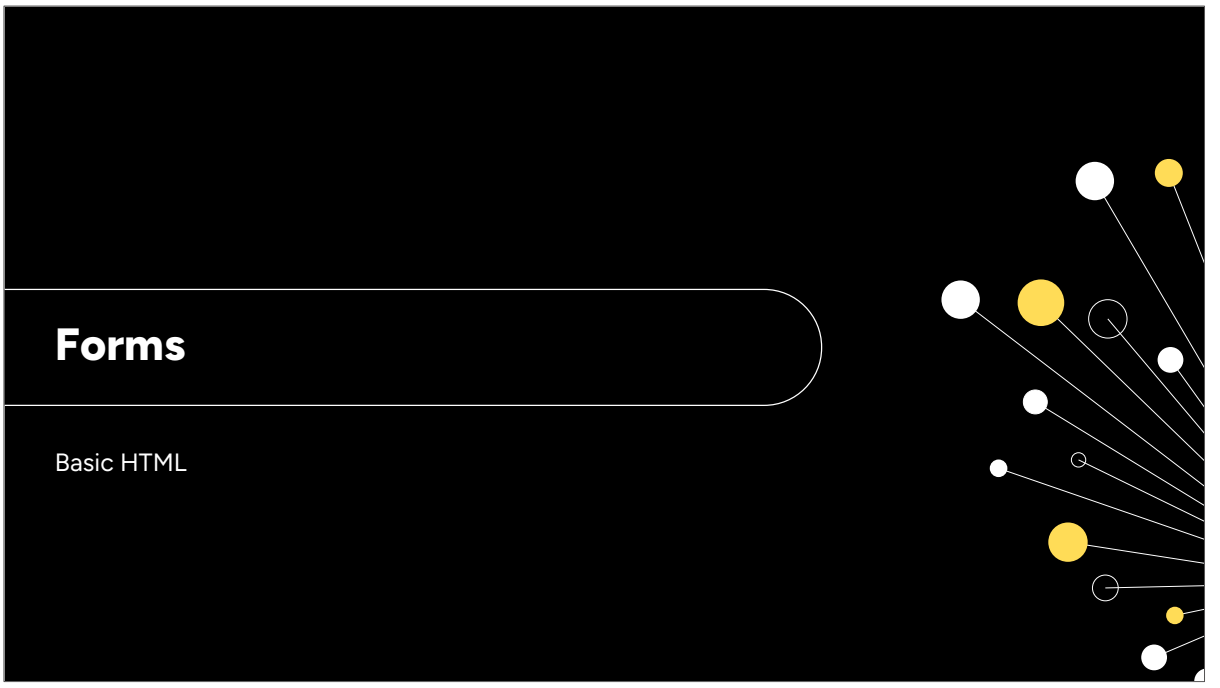
Tables

Tag	Description
<tr>	Table row
<td>	Table data (i.e. a normal cell)
<th>	Table heading (i.e. a header cell)
<caption>	Table caption
<thead>	Groups header content in a table
<tbody>	Groups body content in a table
<tfoot>	Groups footer content in a table
<colgroup>	Specifies one or more columns for formatting
<col>	Specifies column properties for each column within

QA

Quick Lab 4 – Tables

Display some data in a table on an HTML page


Forms
Basic HTML

QA

The <form> tag

Defines a form in HTML

Attributes

- **action**: URL to execute when form is submitted
- **method**: how the information is passed to server (GET or POST)

```
<form method="post" action="cgi-bin/subscribe.pl">
…
</form>
```

The **<form>** tag is used to denote a form in HTML.

The **ACTION** attribute specifies the URL of the script that is to be executed to carry out the action of the form.

The **METHOD** attribute indicates how the form data should be sent to the server. It can be either **GET** or **POST**.

GET Appends the arguments to the action URL and opens it as if it were an anchor.

POST Sends the data via an HTTP post transaction.

The **TARGET** attribute defines the window into which the browser is to display any information returned by the script. If no frames are involved, the **TARGET** attribute is not usually required.

The **<form>** tag may have other attributes for use with browsers which can handle client-side scripting language such as VBScript and JavaScript. These will be dealt with in the client-side scripting chapter.

QA

Text and push buttons

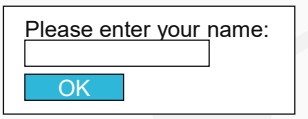
Kinds of push button

- **submit**: send the form information to the server for processing
- **reset**: reset all form fields
- **button**: generic push button

Single line text field

- **type="text"** attribute of **<input>** tag
- **type="password"** as text, but text is not displayed on screen
  – browser default hidden used

```
<form
  method="post"
  action="scripts/subscribe.pl">
  Please enter your name:
  <br/>
  <input type="text" size="20" name="UsrName"/>
  <br/>
  <input type="submit" name="OKBtn" value="OK" />
</form>
```



A pushbutton is a button on a form that causes an action to occur. There are three kinds of pushbutton, and each one is defined by an **<input>** tag with an appropriate **TYPE** attribute. The **SUBMIT** button is used to submit the form to the server script. A **RESET** button is used to reset the form to its initial state: blanking out the text fields and setting selection lists and radio buttons to their default settings.

There is also a third type of push button, **BUTTON**, which is if you want to insert a generic button to which you can assign a client-side script. This kind of button is discussed in the client-side scripting chapter.

For single-line textual input there are two kinds of control. By using **<input>** with an attribute of **type="text"** one can define a single line text input field. A field with **type="password"** has the same meaning as the **type="text"**, except that text is not displayed as the user enters it.

All **<input>** fields have a **VALUE** attribute. For textual controls, this specifies the default (initial) value of the control.

QA

Multi-line text input

<textarea> .. </textarea>

- Initial text can also be supplied
- Browser will supply scroll bars if necessary

```
Address: <br/>
<textarea
  name="UsrAddr"
  rows="7"
  cols="24”
>
  Enter address here
</textarea>
```

For multiple lines of text, the **<input>** tag is not used. Instead, the **<textarea>** .. **</textarea>** tag is used to define an area of text which can have more than one line. This tag has **COLS** and **ROWS** attributes to set the size of the field in terms of rows and columns respectively. These size parameters only set the size on the screen: the actual control can hold more text than can be shown in a confined area and the browser will add scroll bars where necessary.

The text between the **<textarea>** and **</textarea>** tags defines the initial value of the multi-line text field.

QA

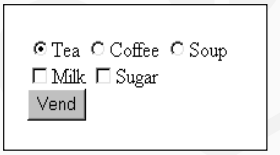
Check and radio buttons

Radio buttons: select from one of a group

- **name** attribute groups buttons together

Check buttons: independent yes/no value

```
<form …>
  <input type="radio" checked name="RadioDrink" value="Tea"/>Tea
  <input type="radio" name="RadioDrink" value="Coffee"/>Coffee
  <input type="radio" name="RadioDrink" value="Soup"/>Soup<br/>
  <input type="checkbox" name="CheckMilk" value="Yes"/>Milk
  <input type="checkbox" name="CheckSugar" value="Yes"/>Sugar<br/>
  <input type="submit" name="OKButton" value="Vend"/>
</form>
```



Radio buttons are used in groups: within each group, only one can be checked (selected) at any one time; in this respect they are used to represent mutually exclusive options, like pre-set tuning buttons on an in-car radio. Radio buttons are created using an **<input>** tag with a **TYPE** attribute of **radio**. Radio buttons are arranged in groups by the **NAME** attribute, and the browser will ensure that if a button in a particular group is selected, all other buttons in that group - which is to say all the buttons having the same **NAME** - will be unselected.

Check boxes are used for independent Boolean (yes/no) choices. Check boxes are created using an **<input>** tag with a **TYPE** attribute of **checkbox**. Each check box in the form has a different name.

Here is an example from a form:

```
<input type="radio" checked name="RadioDrink" value="Tea"/>Tea
<input type="radio" name="RadioDrink" value="Coffee"/>Coffee
<input type="radio" name="RadioDrink" value="Soup"/>Soup<br/>
<input type="checkbox" name="CheckMilk" value="Yes"/>Milk
<input type="checkbox" name="CheckSugar" value="Yes"/>Sugar<br/>

<input type="submit" name="OKButton" value="Vend"/>
```

Note that the first radio button in this example has the **CHECKED** attribute, which indicates to the browser that this button is initially selected. Without this attribute a button is considered to be initially unselected.

Both radio buttons and check boxes have a **VALUE** attribute which is a textual string that is returned when the button is selected.

QA

<select> and <option>

Select one from a drop-down lists

<select> tag defines the list

- **size** attribute is 1 for drop down list, > 1 for scrolled list
- **multiple** attribute specifies multiple selection list

<option> tags define the contents

```
<select name="Drop1" size="1">
  <option>Cookies</option>
  <option>Tortillas</option>
  <option selected>
    Poppadoms (my favourite)
  </option>
</select>
```

Puppadoms (my favourite)

```
<select name="Drop2" size="3">
   <option value="2">High</option>
   <option value="1">Medium</option>
   <option value="0">Low</option>
</select>
```

High
Medium
Low

List boxes and drop down lists are created using the **<select>** and **<option>** tags which define a list and its contents respectively. If the **SIZE** attribute of **<select>** is 1, it is typically shown as a drop down list box with only one item showing. If **SIZE** is greater than one, it is a list box with scroll bars. **MULTIPLE** attribute specifies multiple selection list.

By default the value returned by the list box will be the text in the option item which was selected when the form was submitted. If you want to display one piece of text but submit a different value, you can use the **VALUE** property of the **<option>** tag, thus:

```
<select name="Drop2" SIZE="3">
   <option value="2">High</option>
   <option value="1">Medium</option>
   <option value="0">Low</option>
</select>
```

If you want to set up an initial (default) value, an **<option>** tag can have the **SELECTED** attribute.

```
<select name="Cars">
   <option>Red</option>
   <option selected>Blue(myfavourite)</option>
   <option>Green</option>
</select>
```

QA

HTML5 input elements

- **The HTML5 spec started with forms**
  - Opera and Safari are the driving force and most complete
- **13 type options to date**
- **Mostly extend the `<input>` tags with additional type values**
  - If a browser does not understand the extension rendered as:

```
<input type="text" />
```

- **No requirement in the spec for how browsers present**
  - Different browsers show different UI and error messages
  - Browsers that do not understand the new types treat them as text
  - JavaScript defence is needed for legacy browsers

Consistent rendering is important and if any browser fails to understand the date type it will drop back to treating it as a default input="text" (it is actually optional to use the input attribute just typing <input> will give you a text box).

This gives us some consistency in handling browsers which do not yet implement the data types we are about to explore. If a browser fails to support the HTML5 types support we can defend against this and revert to javascript:

```
<script>
    var i = document.createElement("input");
    i.setAttribute("type", "date");
    if (i.type == "text")
    {
        // No native date picker support
    }
</script>
```

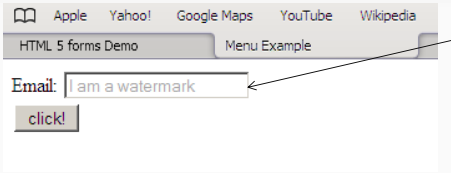
QA

Placeholder attribute

The placeholder attribute offers default text

- Gives the user example or instruction for the field
- Sometimes called a watermark
- Can only be used for text values
- Is not a default value

```
<input type="text" placeholder="I am a watermark" />
```



Text disappears as soon as you tab into the box or start typing

The first improvement HTML5 brings to web forms is the ability to set placeholder text in an input field. Placeholder text is displayed inside the input field as long as the field is empty and not focused. As soon as you click on (or tab to) the input field, the placeholder text disappears.

QA

Required Fields

```
<input type="text" autofocus="true" required />
```

- You can force a field to be mandatory on the client
  - Appears in:
    - Safari 6+
    - Firefox 4.0
    - Opera 9+
    - Chrome 9+
    - IE10+
- On a submit action an error message may appear
- Message will appear differently in each browser

QA

Autofocus attribute

It is common to have the first field of a form to focus

- To have the cursor flashing ready to type

Previously achieved with JavaScript

- The markup representation is faster
  - Part of the page rendering rather than code execution

Supported in all browsers other than IE9 and less:

- Use JavaScript to support legacy browsers

```
<form>
  <input name="q" autofocus="true">
  <input type="submit" value="Search">
</form>
```

HTML5 introduced an autofocus attribute on all web form controls. The autofocus attribute does exactly what it says on the tin: as soon as the page loads, it moves the input focus to a particular input field. But because it's just mark up instead of script, the behaviour will be consistent across all web sites.

If the browser does not support this functionality

- Add the autofocus attribute to your HTML markup
- Detect whether the browser supports the autofocus attribute, and only run your own autofocus script if the browser doesn't support autofocus natively

```
<form name="f">
  <input id="q" autofocus>
  <script>
   if (!("autofocus" in document.createElement("input")))     {
      document.getElementById("q").focus();
   }
  </script>
  <input type="submit" value="Go">
</form>
```

QA

Email input type

Add type value of email

What happens in the client is not consistent

- HTML5 spec does not demand it
  - Opera and safari provides submit validation
  - Firefox provides client validation on blur
  - Safari mobile changes the input keyboard
  - IE 9< does nothing
- Form will not submit until the error is solved
- Provides a simple input mask to check input
  - e.g. boffin@qa.com

This is where the different browsers become key and an interesting part of the standard, at this point, demanding more of an interface approach than a solid functionality requirement. Different browsers behave in different ways. If we are a Firefox user right now we have form validation without the developer doing a jot of code. If we are an < IE10 user we see nothing different. If you are a mobile user the touch screen of an iPhone for instance loads up a different keyboard suited to emails.

It is an easy win for a small change!

QA

Web address input type

Denotes the input must have schema prefixing an address

- e.g.: http://www.qa.com or ftp://ftp.mysite.com

Different browsers have different functionality

- More of an issue than previously due to different functionality
  - Chrome and Firefox 4 force user to add schema
  - Opera prefixes an address with http://
  - Safari mobile provides a different keyboard

To use effectively combine with a placeholder

- Users don't really understand schemas

The input of this in the major browsers is a bit more of a scrap at the moment with IE, with IE10 the first IE browser to support. In this environment Safari and Firefox 4 force the user to enter a http:// address to pass the validation. While Opera instead adds the http prefix upon submission of the form.

While the iPhone / iPad brings a different virtual keyboard up if you have set the field accordingly

QA

Number input type

```
<input type="number" min="1" max="12" step="2" value="6" />
```

Numbers often need to be constrained by range

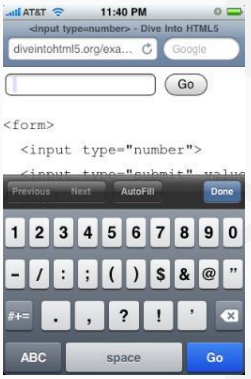
New number type provides this functionality

Four attributes:

- **min** - lowest range
- **max** - upper range
- **step** - what value the control enumerates by
- **value** - default value

Browser support issues

- Firefox, Chrome and Opera display these as 'spinboxes'
- No increment/decrement buttons in IE
- iOS UI widget doesn't consider step, min or max values



Once we get universal support to this control it will be an exciting and very useful resource. Windows forms programmers have been used to a numeric up down field for a number of years but in HTML we have had to do some JavaScript jiggery pokery to get this to work. At this point only Opera and Chrome support the functionality correctly in their desktop browsers. Mobile browsers again fair a great deal better here and once again the iPhone gets a different virtual keyboard displayed to assist the user.

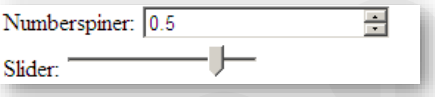
QA

Range input type

```
<input name="r" type="range" min="1" max="11" value="9">
```

Creates a slider bar in Chrome, Firefox, IE10 & Opera

- In other browsers you need JavaScript to display the value
  - Obtain a pointer to the slider
  - Subscribe to its change event
  - Add the output to a span



Has the same attributes as the Number type

Only IE10 displays the current slider value

The range type has the same properties as the number type.

QA

Date input type

A popup calendar is standard for date selection

- Normally requires a JavaScript framework
- Around 10% of web users do not support JavaScript

HTML5 defines six date time types

- Use UTC in the same way as time element

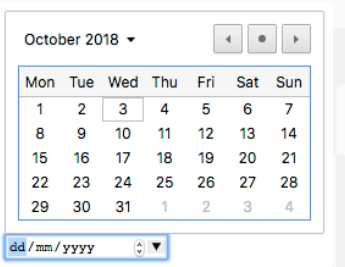
No support in IE or Safari as of Oct 2018

Will not require JavaScript enabled

- Native support is the ultimate aim

ECMAScript5 allows you to create dates from UTC

Formatting is dependent on browser defaults



Within forms constraint of users input and providing a user with effective JavaScript selection is a UI developers bread and butter. In recent years this has often involved a library like jQuery to quickly create rich UI functionality such as calendar widgets. The forms section of the HTML5 spec expects, for full support, there will be widget functionality provided with no code required, just set the input type.

Input Type	Definition
date	Reflects a UTC date
month	Choice for only month and year
week	Select only a specific week
time	Time only in UTC format
datetime	Reflects a UTC date and time
datetime-local	As datetime but assumes local timezone to be correct

QA

Search input type

- Provides a semantic definition for search input
  - Keep to one per page
- Need to set a name for the search field otherwise nothing will be submitted
- Most common name is `q`

```
Search Google:
<input type="search" q="googlesearch"/>
```

QA

Color input type

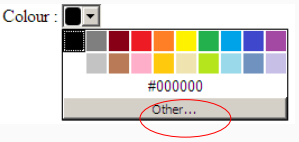
Currently implemented in Chrome, Firefox and Opera

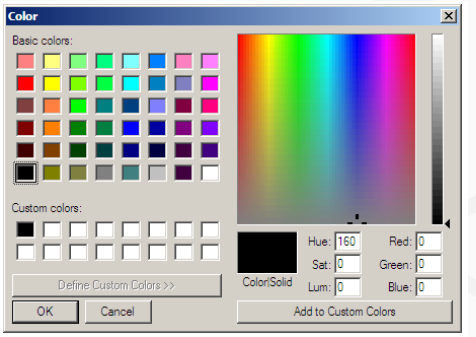
Returns a six digit hex value

- In other browsers should be verified as a hex value

On MacOS and Windows returns a colour picker

- On Linux only the basic colour picker so far





No support in IE, Edge, Safari so will need a fallback

QA

Pattern

The pattern attribute allows use of regular expressions

Pattern works with the following input types:

- **text**
- **search**
- **url**
- **tel**
- **email**
- **password**

```
<input type="text" pattern="[0-9]{13,16}" name="CreditCardNumber" />
```

Ensure the user understands the regular expression

- Support with a **placeholder**

The pattern attribute specifies a regular expression that the <input> element's value is checked against.

Note: The pattern attribute works with the following input types: text, search, url, tel, email, and password.

Useful pattern generation website:

http://html5pattern.com/

QA

Datalist

The **<datalist>** tag provides a list of pre-defined options

- Use the **<input>** element's list attribute to bind it to the **datalist**

Provides an "autocomplete" feature on **<input>** elements.

- Users see a drop-down list of options as they input data.

```
<input list="browsers" />
<datalist id="browsers">
      <option value="Internet Explorer">
      <option value="Firefox">
      <option value="Google Chrome">
      <option value="Opera">
      <option value="Safari">
</datalist>
```

The <datalist> element provides a list of options into an input element providing as set of options for a user to select. The implementation is sufficiently buggy that we can't currently rely on this behaviour.

QA

Form validation

As we have seen some browsers ship with validation

- IE offers no UI implementation in any version
- Firefox and Opera often the most complete implementation
- Chrome is pretty good and Safari will get better
  - Some controls have silent errors, not enough UI feedback

These are JavaScript free client validation

Uneven support may be more trouble than benefit

- You can tell a browser to switch it off
- Still benefiting from the semantic types

```
<form novalidate>
   <input type="email" id="addr">
   <input type="submit" value="Subscribe">
</form>
```

Time to look at this honestly, many of these new input types are exceptionally cool. When we have the big guys like Chrome and IE supporting these fully, and there is sufficient browser penetration in the market, then simple JavaScript validation will be a thing of the past. Until we get there you may be wondering is it worth the headache of supporting the two workflows of user validation.

If you choose to use JavaScript as you can switch off the HTML client side support and go back to pure JavaScript for validation.

The other option is to check for support as we discussed at the beginning of this chapter and using JavaScript to dynamically rewrite the page for non supporting browser.

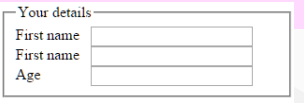
Even if you choose to disable the JavaScript free validation and go for your own controls the new semantic types are useful. With them we can define a clearer sense of meaning to the form creators intentions.

QA

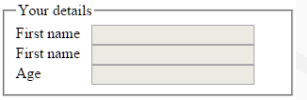
<fieldset>

- Group related form elements together
- Meaningful legend provides accessibility
- Can disable all contained elements

```
<fieldset>
        <legend>Your details</legend>

        <label for="fname">First name</label>
        <input id="fname" type="text">

        <label for="sname">First name</label>
        <input id="sname" type="text">

        <label for="age">Age</label>
        <input id="age" type="number">
</fieldset>
```



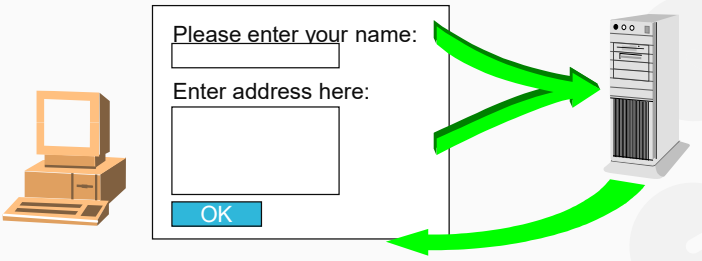

```
<fieldset disabled>
        …
</fieldset>
```



QA

Submitting the form

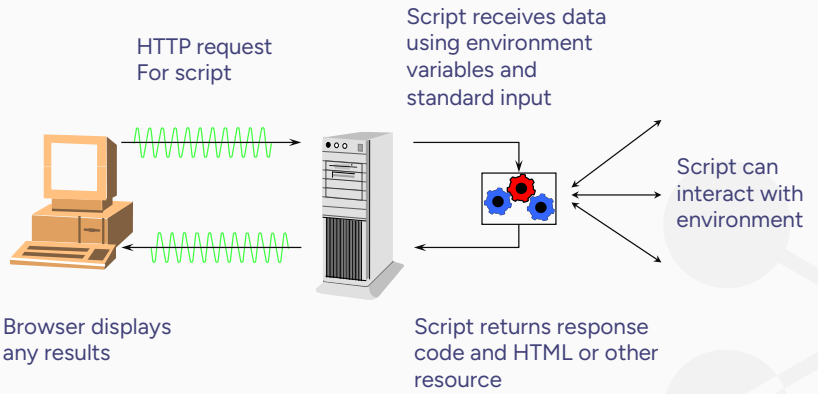
- The "submit" button triggers the script
- The values of the form's fields are sent to the server
- Two different mechanisms: GET and POST



Having information in a form is no use unless it can be submitted to a server script, so this next section examines the way in which information entered into the form by the user is transferred to the server.

When the submit button is pressed, the browser takes all the information in the form's **<input>** and **<textarea>** fields and packages the names and their corresponding values into a combined string. The exact mechanism depends on whether the **ACTION** attribute specifies the **POST** or **GET** mechanisms.

Server scripting mechanism



Server scripting simply defines the environment in which an extension application runs. This consists of the data that is available to the application and how it is received, together with how the application returns results back to the web server. The server is responsible for providing the data to the application and receiving the results back from it.

The browser triggers a request for a script in exactly the same way as for a normal page, through the HTTP **`GET`** command. The browser neither knows nor cares that the requested resource is a script; that is for the server to decide.

The inputs available to the script are the URL string which the browser requested, and, if the request was made via a form, the values of the various fields in the form.

This information is provided to the application as a set of environment variables and (in some cases) through the standard input stream of that application.

The script must, as a minimum, return a valid HTTP response code. This consists of HTTP headers and an optional body; the body is the part which is normally displayed by the client browser. The response need not be an HTML page; it could be any kind of resource, such as GIF file.

Between reading the input and creating the results, your application can do anything you like. It could update a database, subscribe to a mailing list, or order a product.

QA

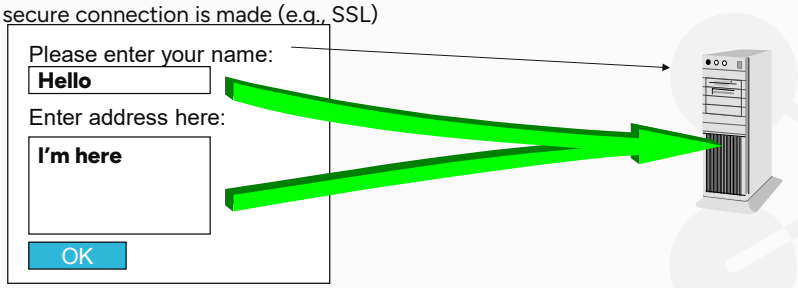
Form security issues

Scripts should check data received from a form

- Validate all data carefully!

Data sent from a form is not normally encrypted

- Unless a secure connection is made (e.g., SSL)



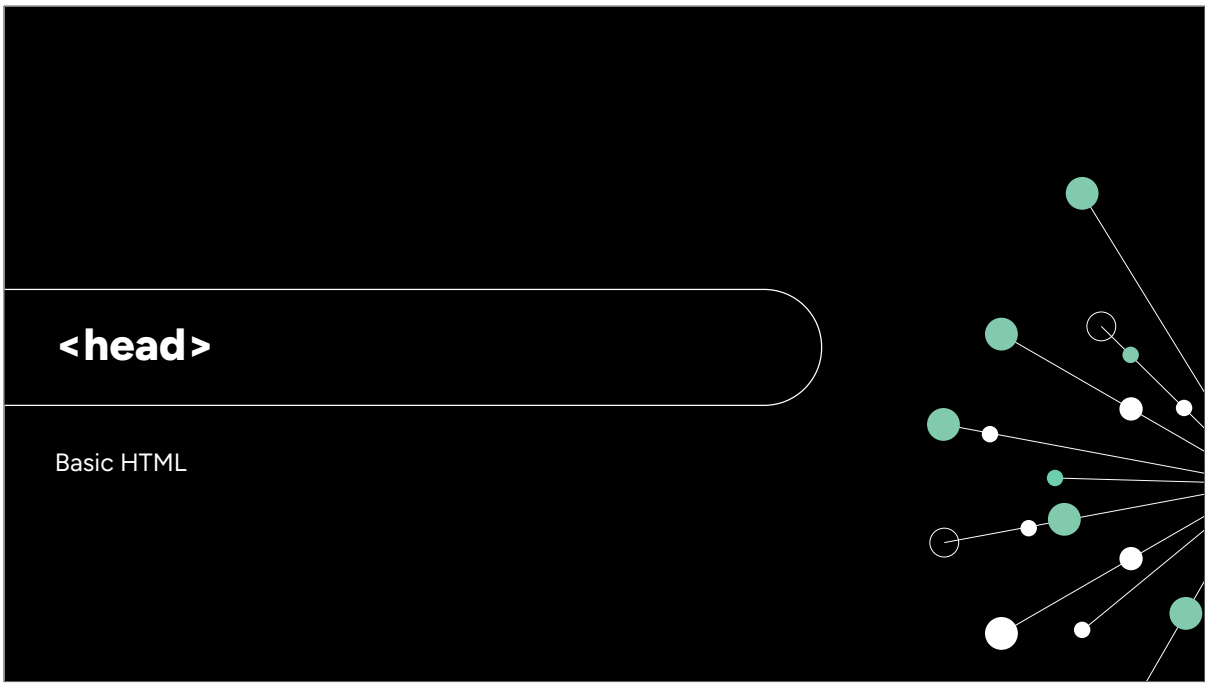
Forms are a security risk because users can type in just about anything into a form. Back-end programs should always be very careful when processing data submitted by a form. In particular check that the parent directory '..' command cannot be used to access files outside of the Web server environment and remember that these can also be specified by their character entity equivalents. Unverified strings should never be executed by the back-end script.

Web site developers should be aware that forms data is transmitted as plain text unless some lower level mechanism such as the Secure Sockets Layer (SSL) is used. Without some form of security, anyone with a TCP/IP analysis package could look at the data as it is being transmitted from the browser to the server.

QA

Quick Lab 5 – Forms

Create a form on a web page using different input types and in-built validation


<head>
Basic HTML

QA

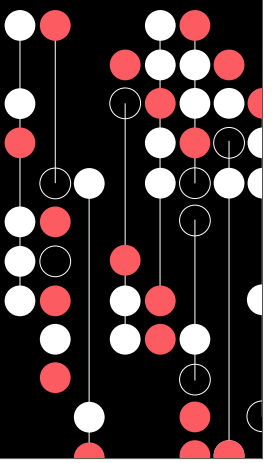
The <head> tag

- A container for metadata usually placed between the <html> and <body> tags
- Metadata is defined as data about the data and is not displayed

Tag	Purpose
<title>	Defines title of document for browser tab, favourites and search engine results
<style>	Defines style information about this document
<link>	Defines where external style sheets should be linked from
<meta>	Specifies the page's character set, description, keywords, author, viewport, etc
<script>	Defines client-side JavaScript (although more commonly included as last element of **<body>**)
<base>	Specifies base URL and target for all relative URLs in a page

The DOM (Part 1)

Basic HTML



QA

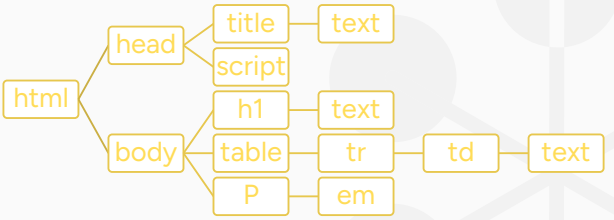
What is the Document Object Model?

HTML documents have a hierarchical structure that form the DOM

- Every element, except <html> is contained within another
- Creating a parent/child relationship

A DOM tree contains two types of elements

- Nodes.
- Text.



HTML documents have what is called a hierarchical structure. Each element (or tag) except the top <html> tag is contained in another element, its parent. This element can in turn contain child elements. You can visualise this as a kind of organisational chart or family tree.

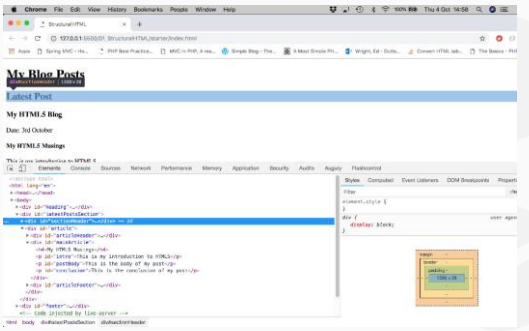
HTML has valid rules for how this DOM fits together and renders the markup delivered from the server into a client side DOM. For all intents and purposes, the DOM is a complex array.

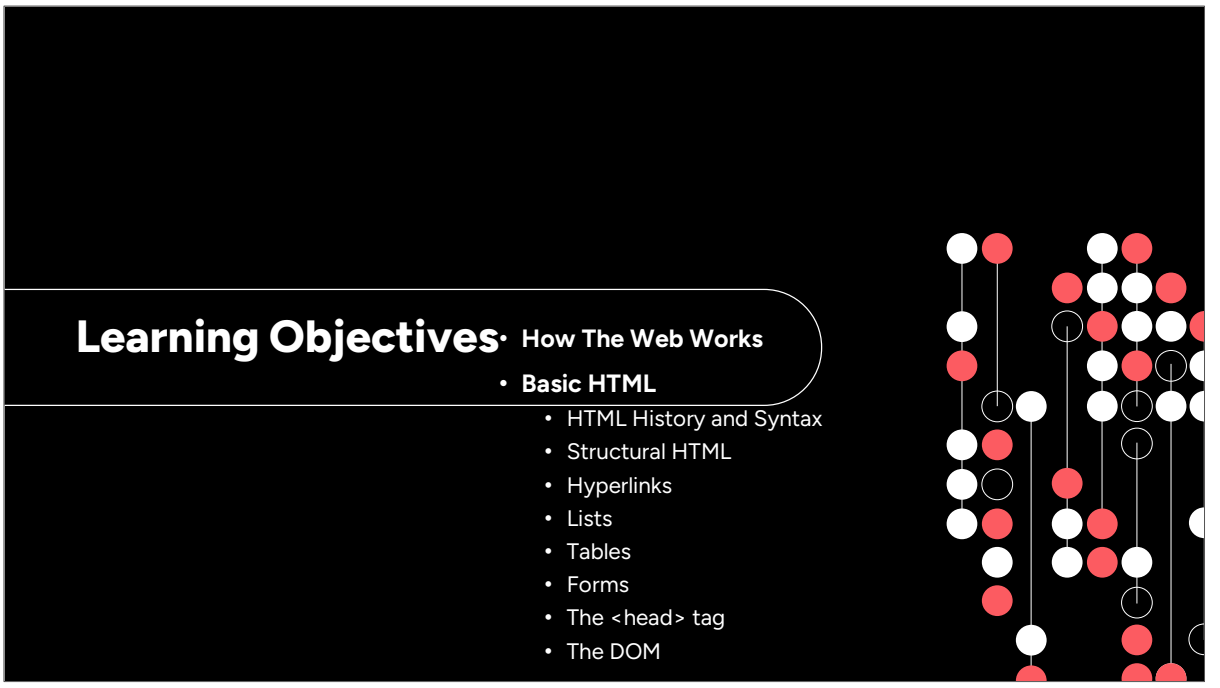
JavaScript provides a series of inbuilt functions to manipulate the structure of the HTML in your browser. It is fair to say that nothing rendered in the page is safe from your grubby little mitts as you start to traverse the page with JavaScript. You can manipulate existing items and create new ones.

QA

Inspecting HTML in the Browser

- Opening the Developer Tools in your browser will allow access to the Elements tab that shows the HTML.
- Selecting markup in this window highlights the area displayed because of it (as shown)




Learning Objectives
How The Web Works
Basic HTML
HTML History and Syntax
Structural HTML
Hyperlinks
Lists
Tables
Forms
The <head> tag
The DOM


Appendix
Web Fundamentals - HTML

QA

Appendix I

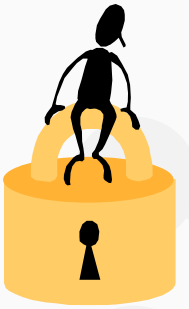
How The Web Works

- **Encryption**

QA

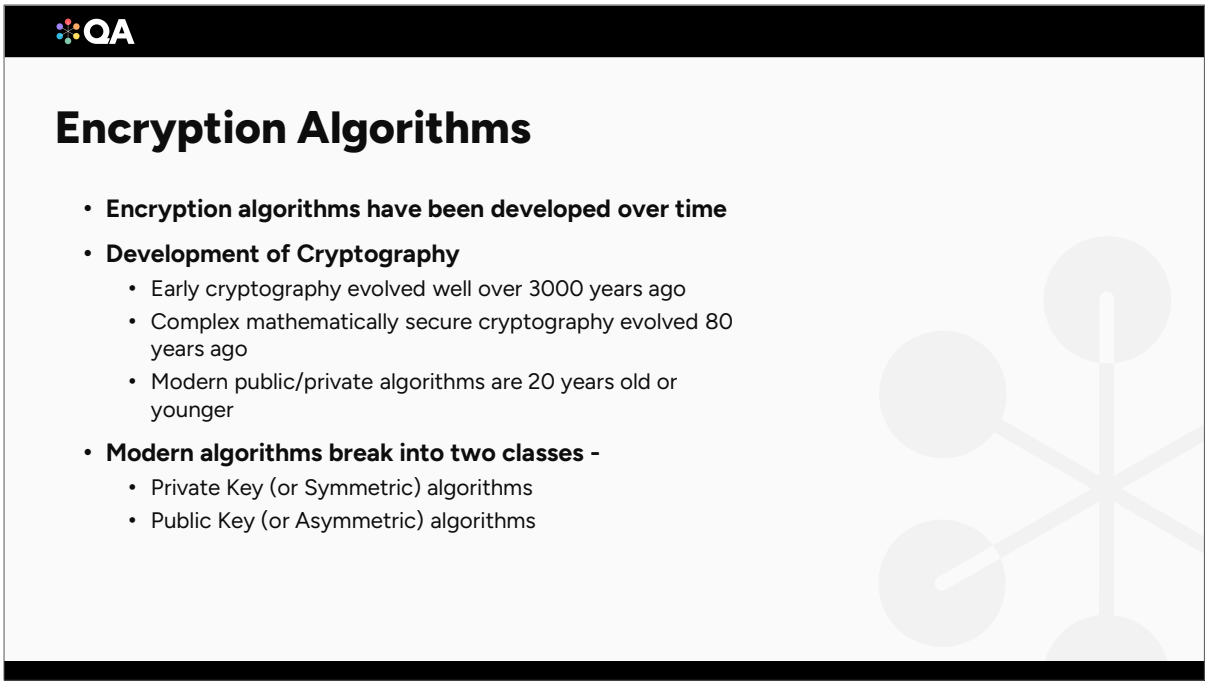
Protection through Encryption

- **Encryption can provide protection from network data attacks**
  - Internet links are not secure
  - Intranet links can be insecure
  - Encryption provides protection from Data Snooping



Encryption is used to protect data from being viewed by those who are not entitled to do so. Encryption can be applied both on a system level basis (e.g. to files and directories) or at a network level (e.g. to data flowing between a web server and web client). We will look at encryption from the network aspect, although the principles are the same in both cases.

In the case of networking encryption is primarily being used to protect our data from data snooping, in almost all modern broadcast based networks (Ethernet, Token-Ring, FDDI, etc.) it is a relatively simple matter to set-up a system which collects every single packet of data flowing on that network. These packets can then be disassembled and their contents analysed, in this way your data becomes visible to another party.



The development of encryption technologies has come about over a long period of time, an appendix at the rear provides an interesting timeline for the development of cryptology. All the most commonly used cryptographic algorithms are 20 or less years old, in many cases the mathematical principles which underlie their operation are similarly young. Modern ciphers break into two groups - the private key (or symmetric) algorithms and the public key (or asymmetric) ciphers. We will take a look at the operation of these ciphers and the differences between them, we will not be analysing how they work in detail since it is a highly mathematical process. We will however, look at comparisons between them and how their working principles affect their usage.

For any good algorithm it should be impossible to recover them by any means other than trying all the possible values. This should hold true no matter how much ciphertext and plaintext the attacker has captured.

QA

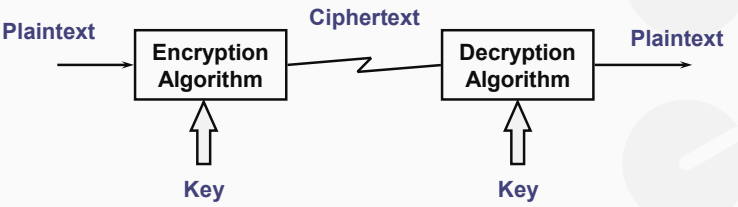
Private Key Algorithms

Private Key (or Symmetric Algorithms)

A single key is used for both encryption and decryption

Security depends on algorithm and key size

- Usually, the bigger the key size the better
- But larger key sizes impact on performance



The private key or symmetric algorithm is the quickest and simplest to use encryption algorithm in widespread use today. The term symmetric refers to the fact that the same key is used for encryption and decryption (using a encryption notation this means that K = K-1), hence only one key is required in order to encrypt and decrypt data. We use the following terms when talking about encryption -

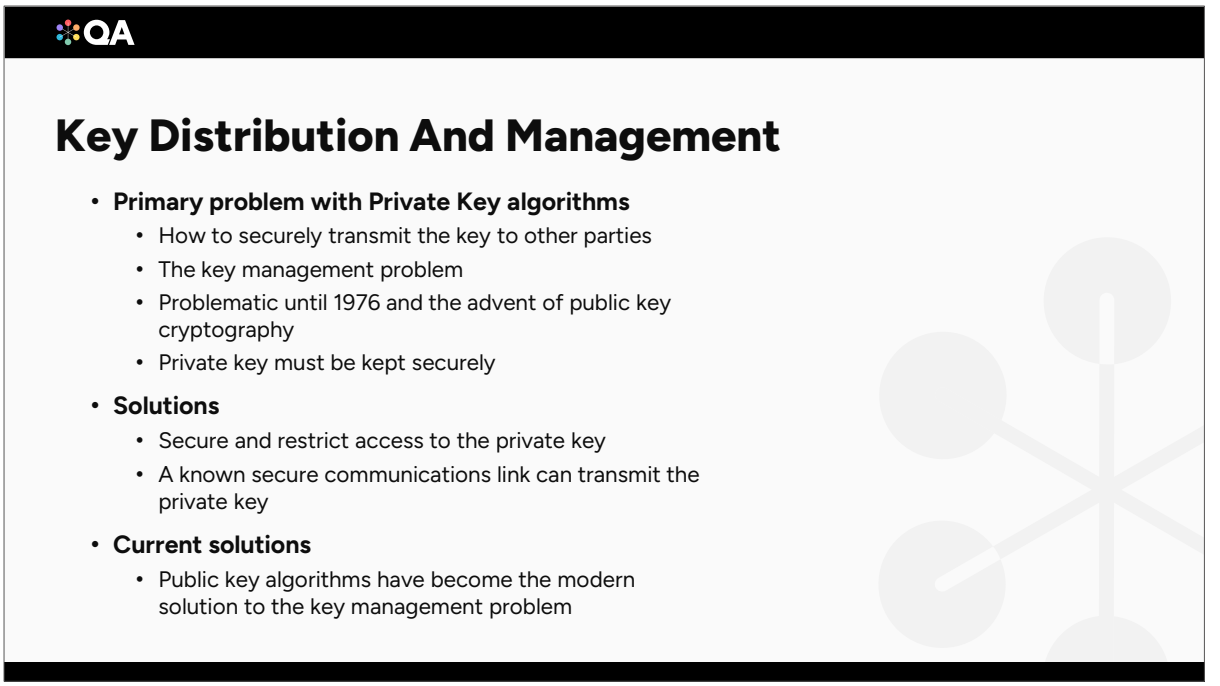
Plaintext - the initial unencrypted data or the encrypted data after it has been decrypted. Ciphertext - the plaintext after is has been encrypted and is no longer readable. Key - the value applied to the plaintext and encryption algorithm in order to achieve encryption. Conversely the value applied to the ciphertext and decryption algorithm in order to achieve decryption.

Encryption - the process of converting the plaintext to ciphertext.

Decryption - the process of converting the ciphertext back to plaintext.

The different algorithms in use have various means of achieving the encryption and decryption but they all have the same general goals -

Secure Encryption - It should not be practical to decipher the ciphertext without the key. Key secrecy - It should not be feasible to deduce the key from the ciphertext. The key point here is the word “practical” in practice it is almost always mathematically possible to decipher the ciphertext without a key. The aim is to make it so difficult (usually in terms of required computing power) that it is not practical to achieve this decryption - within a week, year, decade or whatever timescale is required. This is typically achieved by varying the key size, longer keys take longer to break, but require more computing power to encrypt and decrypt with in the first place.



The primary problem with private key systems, and the whole reason for the rise of public key systems is the so called "key management problem". The problem is that of securely transmitting the private key to the intended recipient so that you can use it to encrypt the data flowing between you. You need a secure link to transmit the private key across, but how do you secure the link without having already having a private key that both parties know in place? Conversely if you already have a secure link to transmit the private key across what do you need the private key for?

Traditionally these problems were dealt with through methods such as having media containing the private key sent to the remote site through some secure means - such as a secure courier service. This is of course time-consuming and does not lend itself to real-time communications. Often a master key, session key design is used where a master key is couriered between sites, it in turn is only used to encrypt session keys which can then be changed more readily and securely transmitted for as long as the master key is secure.

QA

Public Key Cryptography

- **Each entity (user etc) has two mathematically related keys**
- **The private key is told to *no-one***
- **The public key is made freely available**
- **Data encrypted with one key can be decrypted with the other**

The public key cryptography differs from conventional (private key) cryptography in that there are two keys – one for encryption and one for decryption. Because the two keys are different, it is possible to make one of them – the encryption key – public. If someone wants to send me a private message, they encrypt it with my public key. However, it cannot be decrypted with my public key, only with my private key. As its name implies, my private key is kept secret from everyone.

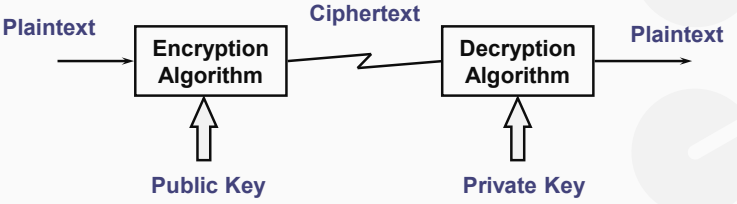
In some algorithms, not only can the private key be used to decrypt data encrypted with the public key, but the reverse also holds. Thus it is possible to use the private key to digitally sign data. I encrypt a block of data with my private key before sending it. Anyone who has my public key can decrypt it. Since only I know my private key, the data must have come from me.

Note that since my public key is just that – public; signing a document in this way only ensures authenticity; it does not provide privacy. If I require privacy in addition to authentication, then after signing the message (encrypting it with my private key) I must encrypt it again with the public key of the person I am sending it to. The recipient decrypts it with their private key, before decrypting it again with my public key to check the signature.

QA

Public Key Algorithms

- Public Key (or Asymmetric Algorithms)
- A pair of keys are used for encryption and decryption
- One key is kept private and used for decryption, the other is made public and used for encryption
- The private key cannot (practically) be derived from the public key

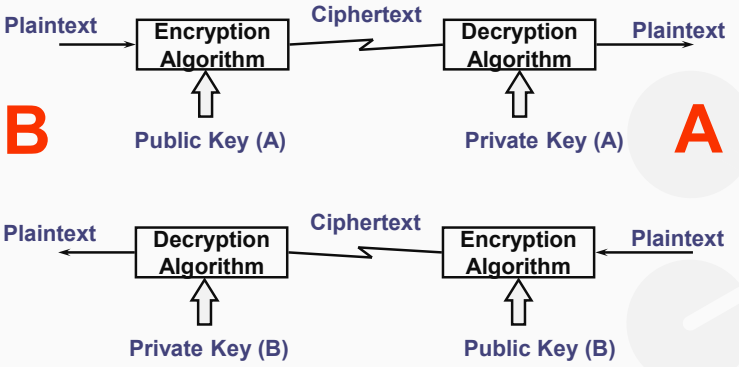


The concept is very simple. We first generate a pair of keys and transmit one of them (the public key) which we will use for encryption to the party with whom we wish to communicate with. They then encrypt data using the key we gave them and we use the key we kept to decrypt that data. The public key cannot be used to decrypt and so no third party can decipher our data. The algorithm we use should be designed such that it is impossible to deduce the value of our private key from the value of the key we gave to the sender of the data. In practice no-one has ever designed an algorithm where it is actually impossible to deduce one key from the other since they are mathematically related. However, in practice it is possible to design an algorithm where it is so difficult to compute one key from the other that it becomes practically impossible. This is the principle that modern public key cryptosystems work under.

QA

Public Key Algorithms (2)

- By using two pairs of keys, one for each party, a secure two-way channel can be established...



Our previous example illustrated a one way relationship where we could give our public key to someone and they could then use it to transmit data that only we could decrypt.

Getting a two way relationship is very simple, both parties generate a public/private key pair, hang onto their private keys and transmit the public key to the other party. Each party now uses the others public key to encrypt the data which is being sent to them. So we now have a fully secure two way link.

QA

Public Key Algorithms (3)

- **Advantages**
  - Solves the key management problem
- **Disadvantages**
  - Slow performance (Symmetric algorithms are up to 1,000 times faster than Public algorithms)
  - Needs large key sizes (512 bits is weak, 1024 is reasonably secure)
  - Vulnerable to certain types of attacks (plaintext and timing attacks)

RSA (Rivest-Shamir-Adelman) is the most commonly used public key algorithm. It was invented in 1977 shortly after Diffie and Hellman had first proposed the idea of public key cryptosystems. RSA can be used both for encryption and for signing, though in this chapter we will consider its use purely for encryption.

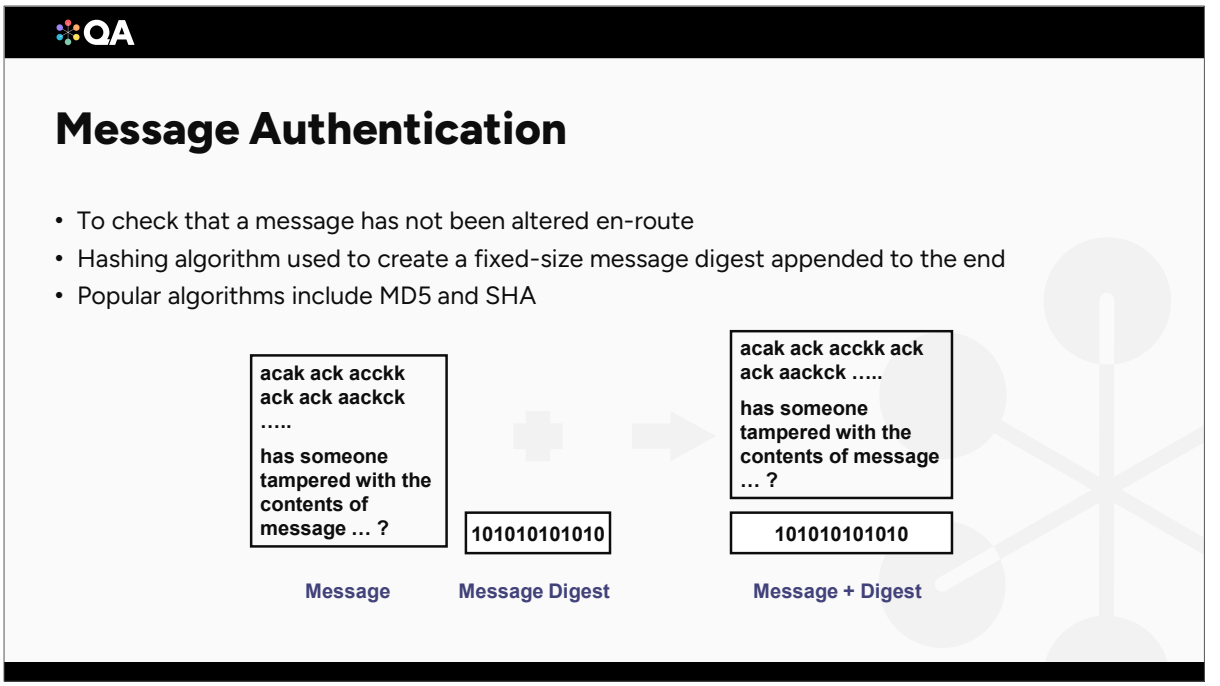
RSA is the most important public key algorithm because it is the technology that underlies well known systems such as SSL and PCT, as well as many of the firewall and network security products currently available. RSA is generally considered to be secure when sufficiently long keys are used (512 bits is insecure, 768 bits is moderately secure, 1024 bits is secure and 2048 bits should remain secure for the foreseeable future). This need for very long keys is common to all public key cryptosystems, unlike pure private key systems they require much large key sizes if they are to remain secure. A 128 bit private key system is more secure than a 1,024 bit public key system - despite the apparent advantage of the large key size for the public key system. The security of RSA relies on the difficulty of factoring large integers, these large integers are the mathematical relationship between the public and private keys. If a mathematician were to make sudden dramatic advances in techniques for factoring large integers it would immediately make it much easier to break RSA. RSA is very vulnerable to chosen plaintext attacks. There is also a new timing attack that can be used to break many implementations of RSA. The RSA algorithm is believed to be safe when used properly, but one must be very careful when using it to avoid these attacks. You might ask why we don't simply dispense with private key algorithms and use public key algorithms for everything. The problem lies in performance, the RSA algorithm is fairly quick for a public key algorithm but nonetheless is anywhere between 1,000 and 10,000 times slower than an typical private key algorithm. Hence it simply isn't capable of moving data at near network speeds, while a private key algorithm is. So in practice we end up using a combination of public and private key techniques to achieve the secure links with acceptable performance.

QA

Protection Through Authentication

- **Encryption provides us with the ability to hide data from third parties - but ...**
  - Do we know the data came from who we think it came from?
  - Do we know that it hasn't been tampered with?
  - Do they positively know who we are?
- **Authentication provides the answers...**
  - Individual messages can be protected from tampering
  - We can positively authenticate parties who connect to us
  - We can positively authenticate ourselves to other parties

Encryption gave us a sound basis from which to protect data from being viewed by those who are not entitled to do so. However, we now have another need we want to be sure that those parties we are communicating with are who they claim to be and we want them to be sure we are who we say we are. Additionally when we are communicating we to be sure that no-one is tampering with our data and altering it in transit.



If we want to send a message across the network in such a way that the recipient can be sure that it came from us, we need to sign it in some way. We have discussed a method of doing this which involves encrypting the message with the sender's private key. Unfortunately, this has two problems:

Encrypting a message with a private key is very slow, particularly for a long message. In general it is considerably slower than encrypting it with a public key.

How can we be sure that a message has not been altered after it has been signed?

The solution to both of these problems lies in a "message digest". This is a short fixed length string which is computed from the original message using some form of hashing algorithm. If we can discover a hashing algorithm with properties such that it is impossible to find a message which hashes to a given value, then we can use that to validate that the message to which the hash is attached on receipt is the same as that to which it was attached when it was sent. We have to make sure that the message digest itself has not been modified in transit. This can be done by signing it using the private key from our public/private key set. The recipient of a message decrypts the message digest using the sender's public key. They then recomputed the message digest of the attached message. If the result does not match that attached to the message, then either the message has been modified in transit or it is not from who it purports to be from. This means that we can now both sign and authenticate a message simply by signing its message digest. The most commonly used algorithms for generating message digests are MD5 and SHA. Each of these three algorithms generates a 128-bit digest.

QA

Digital Signatures

- **How can we be sure that the contents and the message digest are both unaltered … ?**
- **We use "Digital Signatures"**
  - Public key encryption in reverse
  - Data encrypted with a private key can only be decrypted using the corresponding public key
  - Recipient is already in possession of a public key they know belongs to you
  - If the message can be decrypted using public key, it came from you
- **In practice we combine digital signatures with message digests …**
  - Public key encryption is too slow to use on the whole message
  - So, we just sign the message digest so that cannot be tampered with

We've seen how we can use message digests to authenticate the contents of a message and prevent the message from being altered - however this leaves us with another problem - how do we prevent both the message and message digest from being tampered with ? If an attacker can alter the message they can also generate a new message digest which matches with the new contents for that message. The solution is to use "Digital Signatures", these are a clever addition to the functionality we already have provided by public key cryptography.

That is to say if we run the algorithms in reverse swapping the private and public keys we can carry out the same encryption/decryption process. This means that if we know for sure that a particular public key belongs to a particular party and we receive and message that can be encrypted using that public key to provide a sensible message then we know that the message came from them - since only their private key could have generated ciphertext that would work with the public key we have for them. This does of course assume that they managed to keep their private key secret! Building on this you might assume that we should just use this to sign the whole message - however we still have a performance problem with public key encryption and using the algorithm in this way is even slower than using it the normal way. Hence we try to minimise the amount we are going to sign by applying a digital signature and only signing that. If the signature is tampered with it will no longer match up to our known public key or the message, similarly if the message is altered it will no longer match up to the signed message digest. Hence we have fully secured authentication.

QA

Digital Certificates

- **Certificates bind a public key to a particular entity**
- **Certificates contain information about an entity…**
  - Certificate information, e.g., system name and public key
  - Server's digital signature
  - The most commonly used certificates are based upon the ITU-T X.509 standard
  - X.509 certificates are those used in SSL and other common authentication mechanisms
- **Certificates are issued by a trusted body**
  - The Certification Authority
  - You trust them to check that the certificate really does belong to whoever claims to own it

Certificates are design to solve the problem of how to authenticate the public key. What is to stop someone from pretending to be you and telling your bank that their own public key is yours? If your bank believes them, then the bank will now send data to "you" encrypted with "your" public key. The impostor will be able to read this without any problem; more alarming though is that the impostor can now sign things with his private key, and the bank will believe them to come from you because it can decrypt them with "your" public key. In this particular case it is easy to see how a bank might require further proof before accepting that the given key is mine. But what about in the wider context of Internet commerce? How is it possible for me to be sure that www.acme-travel.com is indeed Acme Travel Co before I send them my credit card details? Encrypting the details with the public key they gave me only ensures that no-one else will be able to read them. The answer to this lies in a Certificate, sometimes called a Digital ID. A certificate is a structure which contains the distinguished name (e.g. a legal name such as Acme Travel Co) and the public key of its owner. This structure is then signed by a Certificate Authority (CA) using its own private key. If you have the public key of the CA then you can verify that they signed the certificate, and if you trust them you can now believe the person who has the certificate is who he says he is. This still does not eliminate the problem of obtaining trust-worthy public keys: you still need to be sure of the public key for the CA. However, the problem is now substantially reduced; you just need a few keys for the CAs, rather than the public keys of everyone with whom you might ever do business. In practice, browsers have the public keys of reputable CAs built into them, so obtaining the keys is not normally a problem for the end user. Of course then you have to worry about whether or not this is a genuine unaltered copy of the browser, or if someone somewhere might have modified it.

QA

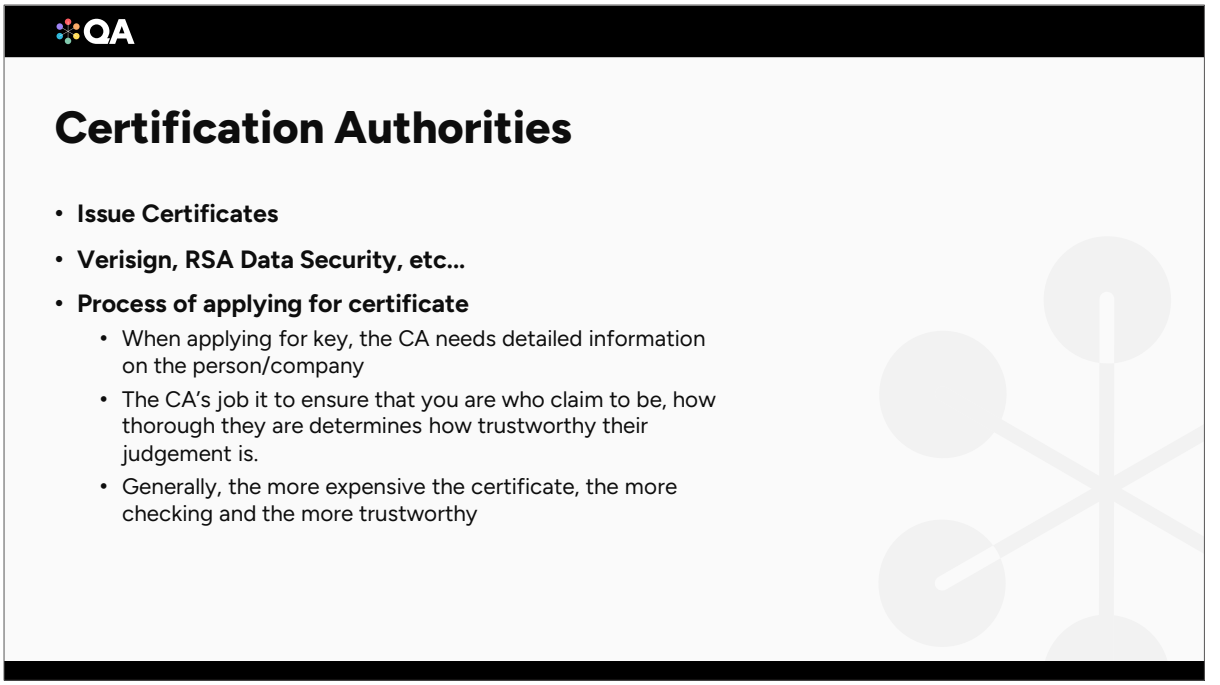
Certificates and Digital IDs

- A certificate contains your "distinguished name" and public key
- It is digitally signed by a "Certificate Authority"(CA)
- If you have obtained the CA's public key from a reputable source, then you can verify that they issued this certificate
- If you trust the CA, then you can be sure that the holder of the certificate is who he claims to be

The certificate contains the following information:

- Version
- Serial Number
- Signature Algorithm ID
- Issuer Name
- Validity Period
- Subject (User) Name
- Subject Public Key Identification
- Issuer Unique Identifier
- Subject Unique Identifier
- Extensions
- Signature for the above fields

In typical use for Internet applications the key fields are the "Subject Name" and "Signature" fields. The latter proves through a digital signature from the Certificate Authority that the other fields within the certificate have not been tampered with. The "Subject Name" or "Distinguished Name" field is typically used to identify the system to which the certificate belongs via a fully qualified domain name (FQDN - e.g. myhost.company.com). The entity checking the certificate such as a browser would connect check that name against the known FQDN for the host it is connected to, if they don't match it considers the connection suspect and warns the user or automatically prevents connection.



The certificate mechanism relies on a series of trusted source from which certificate can be obtained, these are known as Certificate Authorities.

(The following taken from RFC1507)

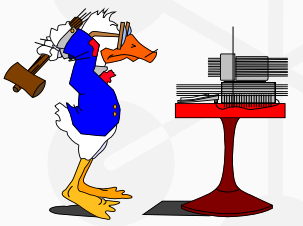
A certification authority is a principal trusted to act as an introduction service. Each principal goes to the certification authority, presents its public key, and proves it has a particular name (the exact mechanisms for this vary with the type of principal and the level of security to be provided). The CA then creates a "certificate" which is a message containing the name and public key of the principal, an expiration date, and bookkeeping information signed by the CA's private key. All "subscribers" to a particular CA can then be authenticated to one another by presenting their certificates and proving knowledge of the corresponding secret. CAs need only act when new principals are being named and new private keys created, so that can be maintained under tight physical security.

In practice this means that you apply to the CA for a certificate, they will then check that you are who you say you are. The exact amount of checking varies depending on how trustworthy the certificate needs to be, some bodies for example give out free trial certificates - but these certificates are clearly marked as such. At the other end of the scale commerce class certificates from Verisign are very expensive and involve detailed checking of your company and its security. This is the level of security the banks and similar institutions will require when proving their identity to users intending to carry out financial transactions with them using these authentication mechanisms.

QA

Breaking Public Key Cryptography

- **Obtain the private key by deception / bribery / etc**
- **Deduce private key by "brute force"**
  - Try every possible key in succession
- **Larger key size gives greater protection against brute force attack**
  - More possible keys
- **Standard "maximum strength" key is 128 bits**
- **SSL2.0 uses 256 bit private keys**

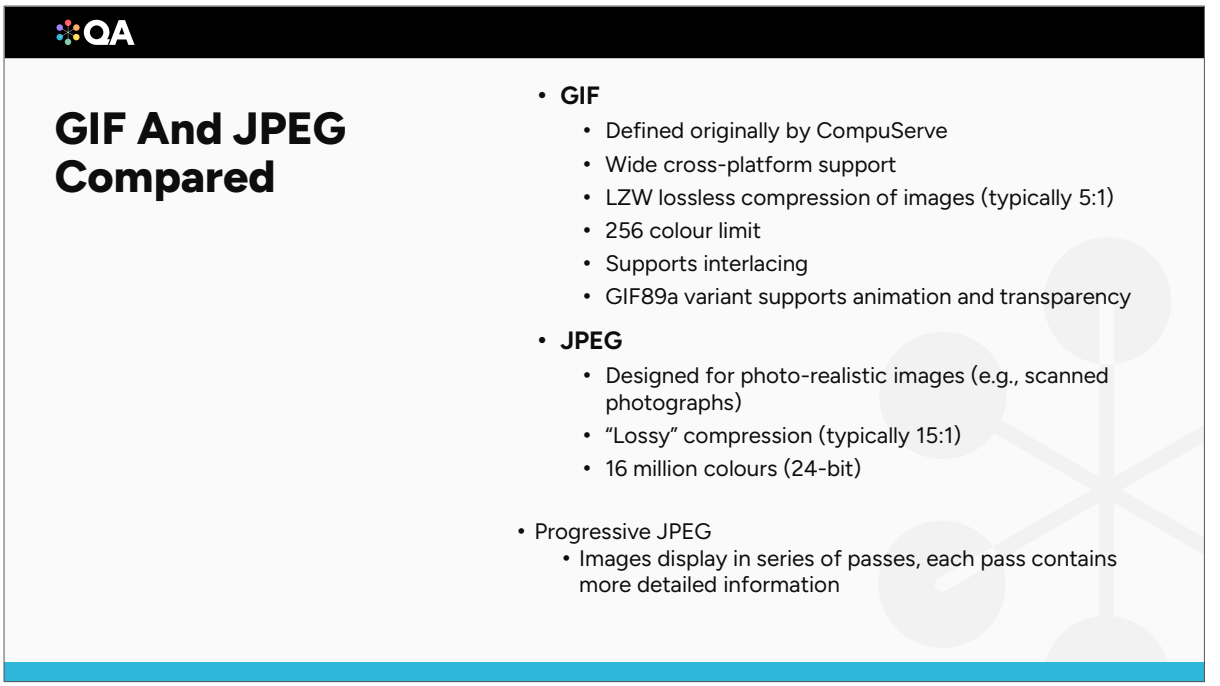


If we are going to depend upon public key cryptography to protect our data, then we need to understand how secure or otherwise it is. There are a few different algorithms, the best known of which is RSA. This forms the basis of PGP, and is also the one most commonly used with SSL. This means that there are only two possible ways to crack it: obtain the private key by some dastardly means, or deduce the private key by brute force.

The brute force method means trying every possible key in succession until the right one is found. Clearly, the more possible keys there are, the longer a brute force attack will take. We normally measure the key size in number of bits; for a key with n bits there are 2n possible keys. For maximum security, a minimum key length of 128 bits is recommended.

US export control used to treat cryptography as arms, and prohibited the export of software which used key lengths greater than 40 bits – Since December 1999 this rule has been relaxed for most of the world. As a consequence, software which originates in the US (such as IIS and IE) can now use larger key lengths.


HTML
Appendix II
Image Types



GIF was originally defined by CompuServe as a standard way to distribute graphics files though its online service. As a result, the format is robust and well-understood by a number of applications across a wide variety of platforms. GIF has built-in compression using the LZW (Lempel-Ziv-Welch) compression algorithm which typically gives 5:1 compression ratios (compression is particularly useful when bandwidth is tight as transfer times can be reduced). The GIF format also supports interlaced images which allow users to see a graphic build up in horizontal sections as it is downloaded.

There are two variants of the GIF file format. The original format is GIF87a, and the revised format known as GIF89a. The latter format also allows for transparency and animated images. A specification for GIF89a can be found at the WWW consortium's web site: **http://www.w3.org/pub/WWW/Graphics/GIF/spec-gif89a.txt**. One major limitation of GIF is that it is limited to 8-bits per pixel, allowing only 256 colours in any one image. The 256 colours can come from an arbitrary palette of 24-bit colours, but only 256 can exist in one image.

JPEG is a format designed for "real world" photographic images with continuous tones rather than graphics and schematics. It supports a lossy compression scheme which discards pixels that don't contribute to the perceived quality of the picture and can therefore achieve ratios of around 15:1. It supports high colour depth, using 24-bit colour representations. JPEG files don't support interlacing or animation.

GIF is probably the best choice for schematic images, graphics and small images, whereas JPEG is usually better at handling large, complex and colour-rich images such as scanned photographs.

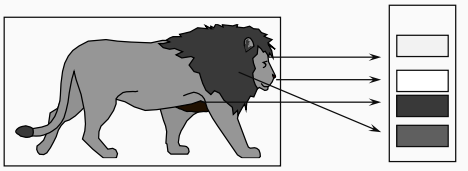
Neither GIF nor JPEG file formats are vector formats: they are bitmaps and therefore cannot be scaled without loosing detail.

QA

Transparent GIFs

All colours in a GIF image are palette-relative

One colour may be the transparency colour



All colours in a GIF file are defined relative to a palette which is part of the file alongside the pixel information. There are 256 entries in this palette. Each palette entry defines a colour in terms of red green and blue components.

It is possible to mark one colour in the palette as transparent. When the image is drawn the browser will then ignore the transparency colour and instead draw the image as a cutout, using the browser background colour or fill in place of the transparency colour. This technique works best when the GIF graphic has a solid background.

This is a way of overcoming the rectangular shape which is inherent in the GIF file format. However, note that the shape may be perceived as non-rectangular but text flow around the graphic is unaffected.

QA

GIF and JPEG Tools: A Sample

Adobe Photoshop

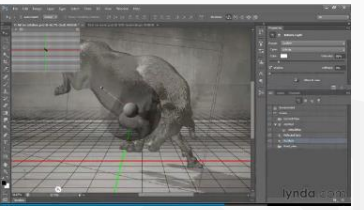
- Widely used by graphic designers
- Mac and PC variants
- Many "bells and whistles"
- Plug-ins for expandability
- http://www.adobe.com

Corel Paintshop Pro

- http://www.corel.com

LView Pro

- http://www.lview.com/index.htm



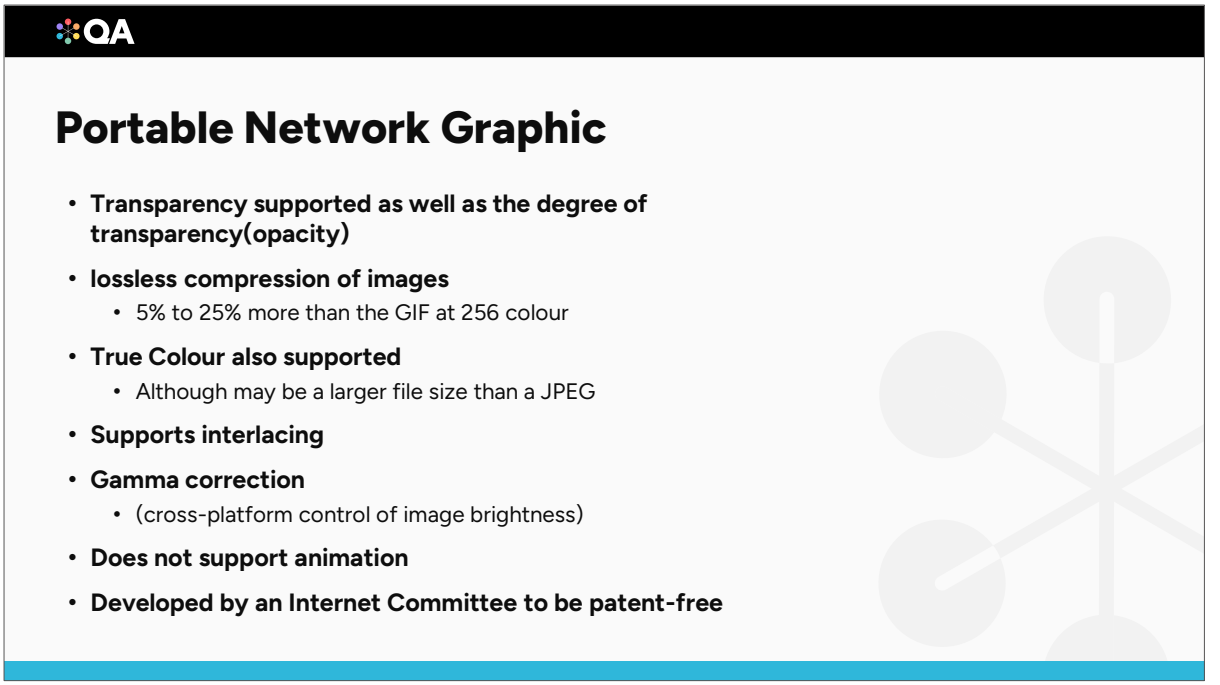
There are a number of image editors and conversion packages available. It will be sufficient to mention a few of those available for the Windows environment. The distribution files for these items of software can be found in many archive sites across the world.

Adobe's Photoshop is a very important tool and is used extensively by graphic designers. It is a comprehensive image editor with many "bells and whistles". For example: alpha channel support for transparency/translucency, anti-aliasing of text, image scaling. It is expandable though the use of "plug-ins", many of which are available commercially. Adobe's web site is at **http://www.adobe.com**.

Paintshop Pro is available from Corel. This product can cope with a very wide variety of file formats including GIF and JPEG. It is a very comprehensive package that includes image retouching and enhancement filters. Corel's web site is at **http://www.corel.com**.

Lview Pro is also a painting package that can deal with most of the standard bitmap file types. It is marketed by CoolMoon Corp who can be contacted at their web site **http://www.lview.com/index.htm**

.



PNG (pronounced ping as in ping-pong; is a file format for image compression that, in time, is expected to replace the GIF. Owned by Unisys, the GIF format and its usage in image-handling software involves licensing or other legal considerations. (Web users can make, view, and send GIF files freely but they can't develop software that builds them without an arrangement with Unisys.)

The PNG format, on the other hand, was developed by an Internet committee expressly to be patent-free. It provides a number of improvements over the GIF format. Like a GIF, a PNG file is compressed in lossless fashion (meaning all image information is restored when the file is decompressed during viewing). A PNG file is not intended to replace the JPEG format, which is "lossy" but lets the creator make a trade-off between file size and image quality when the image is compressed. Typically, an image in a PNG file can be 5 to 25% more compressed than in a GIF format.

The PNG format includes these features:

- You can not only make one colour transparent, but you can control the degree of transparency (this is also called "opacity").
- Interlacing of the image is supported and is faster in developing than in the GIF format.
- *Gamma correction* allows you to "tune" the image in terms of colour brightness required by specific display manufacturers.
- Images can be saved using true colour as well as in the palette and gray-scale formats provided by the GIF.

The PNG format doesn't support animation since it can't contain multiple images. It is described as "extensible," however. Software houses will be able to develop variations of PNG that can contain multiple, scriptable images.

QA

WebP

- **Modern image format**
- **Provides superior lossless and lossy compression for images on web**
- **Allows creation of smaller, richer images to help make downloading faster**
- **Lossless images approximately 26% smaller than PNGs**
- **Lossy images are 25-36% smaller than comparable JPEGs**
- **Lossless supports transparency at cost of 22% additional bytes**
  - Typically 3 x smaller files than PNG

APPENDIX III

GIT Essentials

QA

Git Configuration – Setting the User

Git keeps track of who performs version control actions

- Git must be configured with your own name and email

To configure Git with your name and email we use the following commands:

```
% git config --global user.name "Your Name"
% git config --global user.email "mail@example.com"
```

The values can then be accessed using:

```
% git config user.xxx
```

You can list all the configurations with:

```
% git config --list
```

QA

Git User Configuration – Ninja Lab

- **Use the command line to configure Git with your user name and email**
  - Check the settings to ensure these are set

QA

Git Commands – Entomology

Git commands all follow the same convention:

- The word 'git'
- Followed by an optional switch
- Followed by a Git command (mandatory)
- Followed by optional arguments

```
git [switches] <commands> [<args>]
```

```
git –p config –global user.name "Dave"
```

The -p switch paginates the output if needed.

QA

Getting started with Git

When you first launch Git you will be at your home directory

- ~ or $HOME

Through the Git Bash you can then move through and modify the directory structure

Command	Explanation
ls	Current files in the directory
mkdir	Make a directory at the current location
cd	Change to the current directory
pwd	Print the current directory
rm	Remove a file (optional –r flag to remove a directory)

QA

Git Bash making a directory – Ninja Lab

- **Launch your git command line**
  - Ensure you are at your home directory
  - Find the corresponding directory using your GUI file explorer
  - Create a directory called gitTest
  - Navigate within the directory using the command line
  - Create a sub-folder
  - Navigate back to home
  - Remove the gitTest directory
  - Once you have completed all other tasks type history

QA

Git Help

```
git help
```

QA

Git Help – Ninja Lab

- **Use your command line to access git help find out about:**
  - help
  - glossary
  - -a
  - config
  - -g

QA

GIT Key Concepts - Repos

GIT holds assets in a repository (repo)

- A repository is a storage area for your files
- This maps to a directory or folder on your file system
  - These can include subdirectories and associated files

```
$mkdir firstRepo
$cd firstRepo
$git init
```

The repo requires no server but has created a series of hidden files

- Located in .git folder

```
$git status
```

QA

Adding Files

You can see the status of your git repository

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be
                                committed)
  (use "git checkout -- <file>..." to discard changes in
                               working directory)

      modified:   DG_02_Git.pptx

no changes added to commit (use "git add" and/or "git commit -a”)
```

QA

Adding Files

To add a new file, use the 'add' argument

```
# a single file
$ git add specific_file_name.ext

# To add all the files
$ git add .

# add changes from all tracked and untracked files
$ git add --all
```

Git status will show the newly added file

```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   DG_02_Git.pptx
```

QA

Committing files

To commit the changes, use the “commit” argument

- You can specify the author with the –a flag
- The –m flag is used to set a message

```
$ git commit -m "More content for DG02 - git"
[master 93e8300] More content for DG02 - git
 1 file changed, 0 insertions(+), 0 deletions(-)
```

This is now saved to the local version of your repository

QA

Creating A Git Repo – Ninja Lab

- Create a folder at the ~ called firstRepo
- Initialise it as a Git repository
- Check the status of the repo
- Use the touch command to create a file called myfile.bat
- Check the status of the repo

QA

Attack of the Clones!

Cloning makes a physical copy of a Git repository

- It can be done locally or via a remote server, e.g. GitHub
- You can push and pull updates from the repository

The benefit of cloning repos is that the commit history is maintained

- Changes can be sent back between the original and the clone

Cloning is achieved with the clone command:

- Or through the GUI

```
$git clone source destination_url
```

The GUI branch visualiser gives us a very useful way to see the origins of branches

QA

Cloning a repository

Cloning copies the entire repository to your hard drive

- The full commit history is maintained

To clone a remote repository:

```
$ git clone [repository]

$ git clone https://bitbucket.org/username/repositoryname
```

To clone a specific branch:

```
$ git clone –b branchName repositoryAddress
```