

Client-side

Browser, HTML, Javascript

Browser

1. Handles requests for HTML (and plain text) files
2. Interprets links correctly
3. Deals with embedded images
 - If image display disabled, show alternative string data (should be present in all requests for embedded images)
 - otherwise
 - make another network request to fetch image data,
 - decode image (conversion of compressed image to Luminance/ Chrominance or 24-bit RGB is platform independent, final display requires platform specific code)

Browser

4. Helper applications
5. Image maps
6. Frames
7. Forms
 - actions
 - methods (GET, POST)

Browser

8. Scripting
9. Plug-ins
10. Embedded executables

Browser basics

- Accept input of URL (note pathname of file for future reference)
- Generate correct HTTP request to GET file:
 - GET, fully qualified file name, protocol level
 - Details of browser (User agent)
 - Specification of file types acceptable
 - ...

Browser basics

- Read header of response
 - interpret integer response code
 - decent browser should automatically interpret any "redirect" response codes (contain updated URL if resource has been moved)
 - If html/text data retrieved, read all data
 - (response *should* include a Content.Length field, but can't totally rely on this)
 - allocate working data buffer, expand if necessary

Browser basics

- **Interpret HTML**
 - Omit any unrecognized directives (or show as text) --- this relies on correct use of matching open and close tags, e.g. `<script>...</script>`, and commenting conventions
 - Deal with all hrefs (Unix style decoding)
 - if starts with `'/'` it's a new absolute filename
 - if starts with letter or `./`, then put current pathname in front of filename
 - if starts with `../`, remove one directory from current pathname before use

Interpreting HTML documents

- Documents have two sections – head, body
- **Head**
 - Mainly “meta-data”
- **Body**
 - HTML markup tags
 - Content text

Browser basics

- **Display document info (from `<head>` section) separately**
- **Display text of body**
 - Structure information like `<H1>` tag selects font and size
 - For most text, simply fill line as completely as possible, going to new line at convenient word break
 - Ideally, any image request tag will have specified display size, if not then may have to delay text display until size information arrives with image

Most data in `<head>` section is for use by other programs - web search engines etc - and is never displayed to user

Browser basics

- **Fetch any embedded images**
 - May start multiple concurrent network connections
 - **Interleaved gifs**
 - modified coding scheme, crude blocky form of picture followed by successive refinements (gives user chance to stop fetch)
 - **Animated gifs**
 - essentially several gifs in same file, decoder understands scheme and shows them one at a time

Helper applications

- Reasonable for a browser to include code for popular image formats like gif or jpeg
- **But what about**
 - pdf and postscript
 - audio files - including Microsoft (wav), Apple (aiff, snd), Sun's au, real audio,
 - movies - including quicktime, mpeg, avi,
 - virtual reality
 - compressed text, file archives, ...
 - ...

Helper applications & MIME types

- A browser that tried to support all types of data would become bloated.
- **Alternative -**
 - Browser fetches data file and copies it to local disk
 - When file complete, browser launches another “helper” application, this will read and process data file (“fork” and “exec” on Unix)

MIME types

- Multipurpose Internet Mail Extensions
- MIME types invented bit earlier to allow email to have attached files of different types
- MIME types adopted by growing WWW

Helper applications & MIME types

- Informal standards:
 - Suffix in file name indicates data type
 - Server sends details of "MIME" type of data as part of header
 - Browser has table
 - file suffix MIME type helper application
- MIME types used include official ones as proposed for email, plus others (type name begins with x-) for use just with Web

Helper applications & MIME types

.txt	text/plain	Simple text
.html, .htm	text/html	HTML
.gif	image/gif	Most common graphics
.jpeg, .jpg	image/jpeg	Superior graphics

These types should be handled by browser.

Helper applications & MIME types

.au	audio/basic	Sun (low quality) audio format
.wav	audio/x-wav	Microsoft
.aiff	audio/x-aiff	Mac
.qt, .mov	video/quicktime	Mac animation
.mpeg	video/mpeg	MPEG video

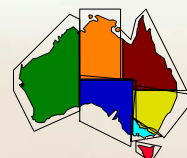
Browser is "configured" by identifying helper programs

Helper applications & MIME types

.gz	application/x-gzip	Gnu ZIP compression
.Z	application/x-compress	Unix compress
.tar	application/x-tar	Unix directory Packer
.sit	application/x-stuffit	Mac compression

Image maps

- Picture divided into regions,
- Each region acts as link to another web resource.



Have gif image
Have table with 7 entries
each consists of polygon
and a URL of another
page with data on specific
state

Image maps

- Original version fairly clumsy
 - Image displayed
 - Mouse click by user on image converted into image coordinates
 - Image coordinates sent to server with request that a (image specific) program - data file combination be run to process coordinates
 - This program worked out region containing coordinates, and hence the next html file to be delivered as response

Image maps

- “Client side” image maps (implemented in browser) came later.

```
<img src=“...” usemap=“#map1”>
```

...

```
<map name=“map1”>
```

```
  <area shape=“...” coords=“...” href=“...”>
```

```
  <area shape=“...” coords=“...” href=“...”>
```

...

```
</map>
```

Image maps

- Image used in map can be any gif (or jpeg)
- Map defined in same HTML file in terms of “area” and hrefs that they link to.
- Shape can be specified
 - circular (x,y,r)
 - rectangular (two corners’ x,y coords)
 - polygon (coords of perimeter points)
 - default (any part of image not matching defined area)

Image maps

- Image map is a simple example of a form of “scripting” in HTML file;
 - browser has built in code for interpreting map ...

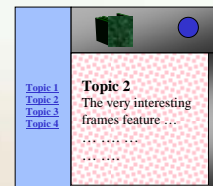
```
if(mouseclick in image with associated map) {  
  for area in map do  
    if (mouseclick in area ) use specified link  
}
```

Frames

- Original browsers based on idea of “displaying a structured document in a window”.
- Can now use “frames” to divide a window into separate parts
 - separated by rules
 - individually scrollable
 - content representing different parts of document (or different documents)

Frames

- Three frames
 - document index
 - logo
 - content area



Useful for company catalogues etc; can constantly display company details (and links to pages where can submit order) while showing details of different products.

Frames

- “Master” HTML document doesn’t have normal <body> section, instead has <frameset>
 - specifies <frame> entries
 - % space given to different rows and columns
 - what document should be displayed (at least initially)
 - “names” for frames whose content can be changed
- An “index” frame will contain links that include the name of frame where fetched document should be displayed (“target”)

Forms

- Browsing?
 - Too limited, want more capabilities for input.
- Input -
 - help select material that is to be displayed (example of a Web site for a Real Estate agent, want customer to see only worthwhile properties)
 - perform searches
 - place orders

Forms

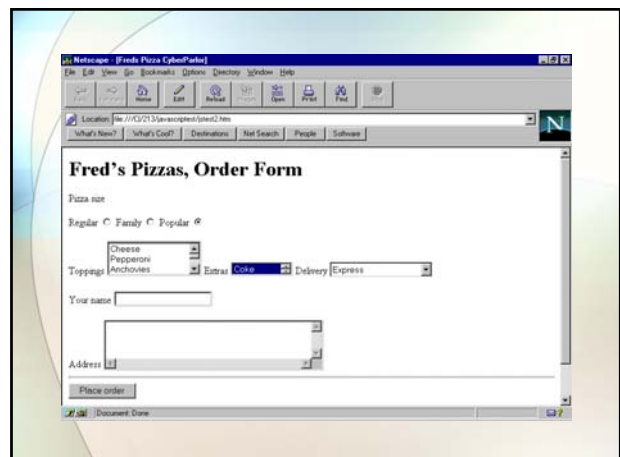
- Level 2 HTML (which introduced “forms”) was a decisive contribution to the commercialization and popularization of the Web.
- Forms go together with server-side “Common Gateway Interface” (CGI) technology

Forms

- User fills in fields of form (enter text, select from predefined choices etc)
- User “submits” form
- Web client packages data entered by user and sends it to web server asking it to run a program identified by a URL in form description.
- Web server program starts requested program and passes it the data.
- Requested program generates text of an HTML page which is returned.
- Web client displays received page.

Forms

- Forms allow creation Web pages that resemble dialog boxes as used in Macs and PC personal productivity software:
 - labels (static text)
 - text entry fields
 - checkboxes
 - radio button clusters
 - select (individual or multiple items) from pop-up menu or list



Forms

- **Forms have an associated “action”**
 - this will (usually) specify the URL of a (CGI) program that will process data entered
 - there will be a “submit” action button, when activated the data from other input fields are packaged and sent in an http message sent to server identified by action’s URL

Forms

- **Forms extension to HTML was immediately recognized as a way of breaking away from proprietary systems and special purpose ad hoc programs.**
- **Further, allow escape from requirement that develop clients compatible with GUIs for each possible platform.**
- **Avoid Microsoft domination of “desktop”.**

Forms

- **Actions:**
 - `<form action=“http://www.serve_u.com/cgi-bin/fredcook`
 - `<form action=“mailto:bl123@uow.edu.au`
- **action is normally the URL of a program that will process any data entered by the user (cgi-bin/fredcook)**
- **alternative uses the “mailto” protocol (useful for people who want a form on their personal web page but don’t have the privilege of getting programs run on the web server hosting their page)**

Forms

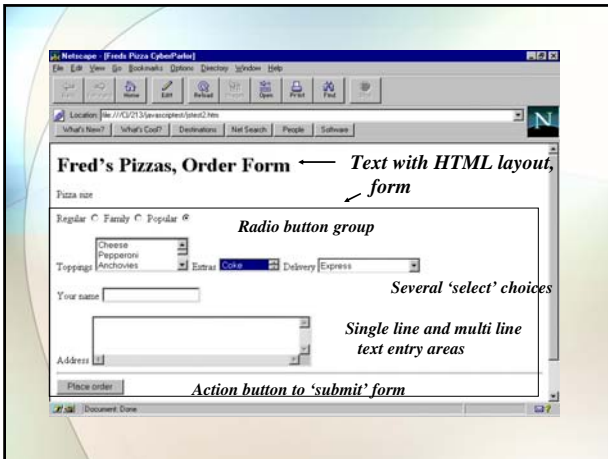
- **Enctype (encoding data from form)**
 - `<form action=“...” enctype=...`
- **enctype is optional (get standard “application/x-www-form-urlencoded)**
 - standard reformats any text (spaces become + signs, non-alphanumeric characters become hex escape sequences)
- **alternatives**
 - text/plain (use with a “mailto” action)
 - multipart/form-data (specialist, up-load a file)
 - encrypted ? Maybe in future

Forms

- **Method (GET or POST)**
 - `<form action=“...” method=...`
- **the method attribute determines how data entered in the form will be sent to the web server (and how the web server will then pass the data on to the program that does the work)**
- **you wrote the data processing program (or picked up a standard documented one) so you choose the appropriate method**

Forms

- **Method and program go together.**
- **Advice**
 - **GET**
 - small forms, short input fields, (preferably little or no user text input) and a slightly easier to write processing program
 - **POST**
 - larger amounts of text entered in form, desire for at least some primitive security,
- **Alternative advice: GET for simple query, POST for database update**



```

<html>
<head>
<title>Fred's Pizza CyberParlor</title>
</head>
<body>
<h1>Fred's Pizzas, Order Form</h1>
Pizza size
<p>
<form action="http://www.serve_u.com/cgi-bin/fredcook"
method=POST >
Regular <input type=radio name=p_size value=reg>
Family <input type=radio name=p_size value=fam>
Popular <input type=radio checked name=p_size value=pop>
</p>

```

```

Toppings
<select name=tops size=3 multiple>
<option>Cheese
<option>Pepperoni
...
<option>Sun dried tomatoes
</select>
Extras
<select name=xtra size=1 multiple>
<option selected>Coke
...
</select>

```

```

<p>
Your name
<input type=text size=20 maxlength=50
name=customer>
<p>
Address
<textarea name=address cols=40 rows =3>
</textarea>
<hr>
<input type=submit value="Place Order">
</form>
<hr>
Have a good day.
</body>
</html>

```

- ### Scripting
- **Scripts**
 - Program snippets written in an interpreted language
 - **Browser contains an interpreter**
 - Each time script is run, interpreter parses next statement of script and executes required actions
 - Interpreters are slow, mechanism suits only trivial computations

- ### Scripting
- Code fragments scattered through HTML page (can become maintenance problem)
 - Some fragments executed before page is displayed.
 - Other fragments associated with elements of a form

Scripting

```
<script language=javascript>
<!-- 'open HTML comment',

javascript code, function and variable
declarations

// -->
</script>
```

Comment brackets are for 'script challenged' browsers

Scripting

- Can have executable Javascript statements
 - in header
 - these executed as page is loaded, before data display
 - in body
 - these executed (once) as HTML interpreter reaches that part of body

Scripting

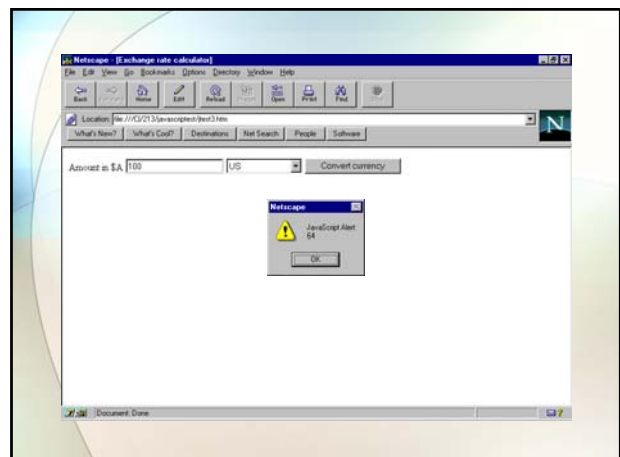
- More common, Javascript code associated with components in forms (sometimes also with frames, links, and setting up of embedded executables)
- Code:
 - function declarations (in <head> section) and a function call associated with component
 - very small fragments directly associated with component

Scripting

- Typical code actions
 - validate data in field of form (and put up an alert box if data entered are inappropriate)
 - put an explanatory comment in status bar of browser when mouse pointer clicked (or, in some cases, simply passes over) an active field

Simple scripting example

- Travel agency wants to provide a calculator that lets you convert \$A into other currencies.
 - HTML page
 - newly generated each morning as it has to contain current exchange rates
 - has simplified FORM,
 - this form never submits anything, simply provides a set of input fields and action buttons
 - has Javascript code associated with action button



Javascripting exchange rate

- **Elements of form**
 - an input field (for amount in \$A)
 - pop-up selection for picking target currency
 - action button with call to Javascript function
- **function causes an alert box to be displayed with converted amount (could have used another 'input' text field on form)**

```
<html>
<head>
<title>Exchange rate calculator</title>
<script language=javascript>
<!--
...
// -->
</script>
</head>
<body>
<form name=Xchng>
...
</form>
</body>
</html>
```

```
<body>
<form name=Xchng>
Amount in $A
<input name=Adollar type=text size=20
maxlength=15 >
<select name=currencies size=1>
<option selected>US
<option>Pound Sterling
<option>Yen
</select>
<input type=button value="Convert currency"
onClick="doCalculate(Xchng)" >
</form>
</body>
```

*Note lack of action, method
etc in form; this form does
not submit data to server*

*HTML form code with
call to Javascript function*

```
var usrate = 0.87;
var sterlrate = 0.44;
var yenrate = 89.0;
function getRate(selectObj){
  crate = 1.0;
  if(selectObj[0].selected) crate = usrate;
  if(selectObj[1].selected) crate = sterlrate;
  if(selectObj[2].selected) crate = yenrate;
  return crate;
}
function doCalculate(formObj){
  rate = getRate(formObj.currencies);
  ausdollars = formObj.Adollar.value;
  other = ausdollars * rate;
  alert(other);
}
```

Javascript code

Javascript code

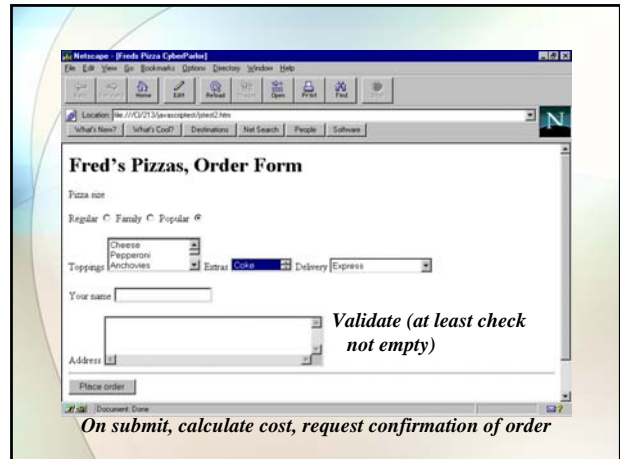
- **“Objects” (data structures)**
 - Forms, input fields, select fields all represented as data structures
 - Javascript can access members of object
- **Typeless language**
 - note switches between text and numerics (and back again) in sample code
- **A little sloppy about variable declarations, ...**

Javascript solutions

- **Not many real applications that require such trivial computations.**
- **Those that do are better done in Javascript than in Java**

Typical scripting example

- Most Javascript associated with validating forms.
- Example (code later in section on HTML and Javascript coding) is Fred's pizza form.
- Javascript extensions
 - check name and address fields have data
 - calculate price of pizza, display to user and ask for order to be confirmed before actual submit
 - provide info in status bar while selections made



Plug ins

- A browser can exploit **helper applications** to deal with all kinds of strange data, but the scheme is a bit clumsy
 - wait until data file completely downloaded and stored on local disk
 - fork() & exec() (or equivalent) to launch another application
 - other application runs,
 - usually, user must explicitly terminate other application and return control to browser

Plug ins

- Individual users don't want to handle ALL types of data; each user may want a particular selection of extended data types they would like to view.
- Maybe
 - users can pick the extensions that they want
 - extensions somehow combined into browser program so that user can view data directly (possibly in main browser window)

Plug ins

- Browser writers can't write code for all types of data (they don't have specialist knowledge of proprietary data structures and algorithms)
- Maybe browser writers can provide a standard interface
 - specify functions that browser expects to use when telling an extension to handle code
 - providing functions in browser that can be used by extension code to get configuration info.

Plug ins "dynamic linking"

- If the OS is sufficiently sophisticated, it will support some form of '**dynamic linking**'.
 - After a program has started running, it can load in additional code
 - Function entry points and calls joined so that new code can be called.

Plug ins “dynamic linking”

- With “dynamic linking”, extension code doesn’t have to be part of browser.
- Browser has
 - standard directory where it can store extension code to deal with special data types;
 - a configuration table where it lists the MIME types of special data and associated extension code (similar to table for “helper applications”)
 - when browser encounters special data, it links in required extension code.

Plug ins

- This arrangement for extensions was introduced in Netscape 2.0; the extensions were referred to as Plug ins.
- Note that Plug ins must be
 - native code (machine code)
 - compiled to run on particular platform (hardware/OS combination)
 - may require special compilation options to allow dynamic linking

Plug ins

- **Unlike Javascript and Java, Plug ins are NOT platform independent.**
- Plug in supplier must provide versions for
 - WindowsNT, Windows 2000, Windows XP
 - MacOS
 - Unix (solaris, linux, irix, ...)
- User must acquire plug in appropriate for their system

Plug ins

- Use of plug ins reduces universality of Web page - becomes targeted at subset of users.
 - Links to pages that hold special data aren’t too much trouble (other users don’t follow the link)
 - “Embedding” other data in page (i.e. trying for something like having a gif image surrounded by text) is more problematic
 - if appropriate plug in not available, browser must negotiate with user (“Do you want to load the plug in? ...”)

Plug ins

- Examples
 - Adobe Acrobat reader (PDF files),
 - Flash
 - RealAudio (radio broadcasts via Internet)
 - QuickView (handles many popular word processor files from PCs)
 - a PowerPoint slide viewer
 - ...

Plug ins

- **Linked document:**
`Report (as Word5.0 document)`
- **Embedded:**
`<H1>London tourism</H1>`
`<hr>`
Virtual bus tour of London
`<embed src="clip.mov" height = 200 width =300`
`autostart=true>`

Embedded executables

- Finally, we have Java applets embedded in Web pages.
- The browser now must incorporate a Java byte code interpreter (Java virtual machine)

*Java virtual machine + Javascript interpreter + image map handling + image display + frames + forms + basic HTML interpreter + support for HTTP, FTP, MAILTO, news, ... + ...
The days of "simple browser is about 1000 lines of C" are long gone!*

Java applets

- HTML page contains an APPLET tag
`<applet code=myapplet.class height=100 width=100>
</applet>`
- When reached by HTML interpreter, another TCP/IP connection is made to server to ask for byte codes of (compiled) Java program.
- When code loaded, the byte array is passed to Java virtual machine for execution.

Java applets

- Typically, the HTML code also includes parameters that are to be passed to the applet
 - the scheme is similar to environment variables for C/C++ programs on Unix

```
<applet code=myapplet.class height=100 width=100>  
  <param name=picture1 value=images/pic1.gif>  
  ...  
</applet>
```

Applets

- Not that much used
 - BBC news page
 - Applet (which doesn't have visual display) is doing something about measuring how long pages take to download – and sending this info back to BBC (for optimization of page layout)
 - UOW
 - SMP package (used to enter student marks) employs applets for entering marks in tables and resubmitting

ActiveX "controls"

- Microsoft technology
 - Real executable code downloaded to IE running on Windows
 - Code has full access to your machine and can do anything
- Security?
 - Well, you get asked if you really want to download and run the code, and do have "certification" mechanisms that identify source of code
 - So, you are responsible for security, you only download and run the control if you trust the source

HTML

"programming"

More detailed example material, useful for assignment 1, at <http://www.uow.edu.au/~nabg/399/Basics/HTMLStuff.html>

Using HTML

- Most web pages are created by secretaries, sales representatives, graphics designers etc.
- There are domain specific web page editors intended to help such people when laying out pages.
- Many word processors have “Save as HTML” options that produce (sometimes poor quality) HTML from word processing documents.
- *No knowledge of HTML needed for such usage.*

“Programming” HTML

- You will have to write programs that generate HTML.
- You may need to compose HTML pages by hand.
- So, you need to understand the HTML “language” in order to produce good pages.

HTML

- Head
- Body
- Frames, forms

HTML tags: generally they have to be paired (<head> ... </head>), but there are exceptions (<meta>), and many browsers will disregard omission of certain closing tags (, </option>, ...)

```
<html> <head>
<meta ...
<style ...
<title ...
<link ...
</head>
<body>
<h1>
...
```

<head>

- Title
- Info for search systems <meta ... “keyword”
- Info inter-relating documents
 - not widely used
- Info for special effects
- Define “styles”
- Define functions and ‘global’ variables for Javascript

<head>...</head>

<meta name=... content=...>

- meta name/content tags should appear in documents that are permanent and that are to be indexed by search engines.

```
<meta name=“keywords” content=“for-loop, array, ...”>
<meta name=“Authors” content=“Ian Piper”>
```

There is no </meta>; all data for meta go in bracketed block

<link>, <base>

- <link> and <base> have limited uses
- Defined in the HTML language and intended to allow for definition of relationships among a group of documents.
- Original idea was for browsers to be able to take advantage of them and extract data that would allow creation of a diagram showing structure of a cluster of documents.

<base>

- <base> tag changes how browser will interpret links in this document (those that are just relative filenames etc)
- Normally, effective base is the directory which contained the current document; but it can be altered.

<link>

- <link> will contain href of a related page.
- Attributes in <link ...> meant to convey semantics (rel tag describes relation from this page to referenced page, e.g. next; rev tag defines reverse relationship from referenced page back to this page)

<meta http-equiv=... >

- These meta tags are used to achieve various special effects.
- An example use is to provide an “expires” date (for use with caching):

```
<meta http-equiv="expires" content="1 Apr 1998">
```

<meta http-equiv="Refresh" ...>

- One moderately important use is for “*client pull*” documents.
 - The meta directive here specifies a display time and a URL for another document.
 - When display time has elapsed, your browser will automatically fetch the document with the URL specified in the tag.

Client pull is considered somewhat *antisocial* for documents accessed off the Web; too much network traffic.

<meta http-equiv="Refresh" ...>

```
<meta http-equiv="Refresh" content="45";  
URL="...">
```

- Leaves current document displayed for 45 seconds, then moves to next.
- URL must be “absolute”[†]:
 - <http://www.xyx.com/sales/demo23.html>
 - <file:///exhibition/kiosk/sales/demo23.html>

[†]That is what it said in manual. Was very inconvenient. Now browsers appear to handle relative URLs. Don't know whether specification updated, or browser implementors simply being helpful.

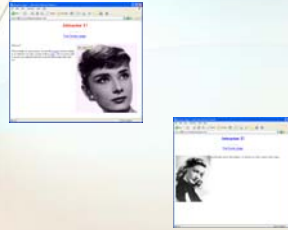
<meta http-equiv="Refresh" ...>

- Appropriate use:
 - your organisation has a display stand at an exhibition
 - you have computers on stand running a browser that is working with files with company data
 - you want an “*attractor*” to get people to stop
- use a sequence of attention grabbing pages with auto refresh to cycle amongst them

<meta http-equiv="Refresh" ...>

- **Attractor pages:**
 - form cycle, *demo1.html* references *demo2.html* which references *demo3.html*, ..., ...references *demo50.html* which references *demo1.html*
 - have relatively short hold times,
 - simple image or “headline” content
 - one or two links to main pages from a normal web of information documents
- **Main pages:** also have refresh links (back into attractor sequence, but with long timeout).

See examples in ~nabg/399/Basics



<style ...

- **Tags (like <h1> header) have some default interpretation in browser - such as *put in 18pt font*.**
- **Can add qualifiers to an individual layout tag -**
`<h1 style="color: red">LOOK HERE</h1>`
- **Suppose you want to say that <h1> headers should *all* be in red?**
- Enter “**styles**”

<style ...

- **Styles**
 - applied to individual directives (“inline styles”)
 - **defined in head section of page and applying to all parts of page**
 - defined in a separate “style sheet” file that is linked into head of page
 - preferred mechanism for establishing consistency at a Web site
 - `<link rel=stylesheet type="text/css" href=mystyle.css>`

“cascade”

- **“Cascading style sheets”**
 - **Styles**
 - Local (inline)
 - Page specific (defined in head section)
 - Defined by external file
 - **More local definitions take priority over more general definitions**

<style ...

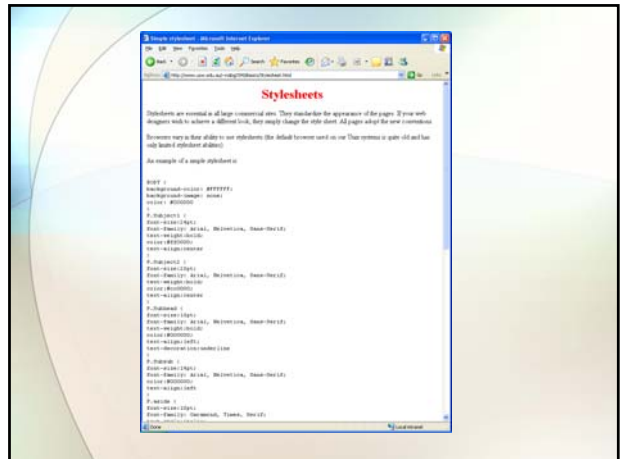
- **What can you do with styles?**
 - **change properties** (there are more than 50 properties!) of standard tags (like <h1> as in earlier example)
 - **define new “style classes”**
 - a new style class for paragraphs specially for mathematical formulae
 - specifying that it uses Symbol font, is center justified etc
 - can then start a paragraph with a <p> tag that specifies a “maths class” paragraph and you will get its text in required font and justification

```

<head>
...
<style>
<!--
H1 {color: red; text-align: center; font-weight: 900}
H2 {font-variant: small-caps}
P.note { font-size: xx-small; font-weight: 100
left-margin: 0.5cm; right-margin: 1.0cm;
color: rgb(177, 202, 89)
}
-->
</style>

```

Defining a "class" for a new style of paragraph



- ### Where to find out about attributes?
- What is the web for?
 - Use a search engine.
 - <http://www.wdvl.com>
 - Probably has links to tables listing all the attributes that you can set via styles

- ### Javascript stuff
- Also in head section, have most of **Javascript** associated with page:
 - global variable declarations
 - function declarations
 - any Javascript code that is to be executed after page fetched but before any content displayed.

<body>

- **Layout**
- **Content!**

```

<html><head> ... </head>
<body>
content, content, content, ...

...content
<hr>
<--
some page and authorship id!
-->
...
</body></html>

```

- ### backgrounds, fillers etc
- Be careful with background colours etc
 - can reduce readability of page
 - **red** on black for drama
 - **yellow** on grey for svelte style
 - **<body ...>**
 - background colour, text colour, colours of links
 - colours defined as hex triplets for r, g, b intensities (or names for standard colours)
- Maybe my machine don't have enough colour contrast, both are unreadable!

backgrounds, fillers etc

```
<body bgcolor=#0a0a0a text=linen alink=darkkhaki  
vlink=mintcream link=palegoldenrod>
```

- `link` (``)
- `vlink` a link that has been followed
- `alink` select a link, it changes to this colour

linen, darkkhaki etc - names for standard colours, same as X- colour table

backgrounds, fillers etc

```
<body background="images/bkgd1.gif">
```

- Browser has to fetch image specified by URL, (here its bkgd1.gif from subdirectory images in same directory as current page).
- Window (or frame) used for page is "tiled" by repeating this image as necessary.



on ...

- You can also specify a variety of "on" conditions with `<body>` tag; these relate to events that can trigger execution of Javascript code.

- `onload`, `onunload`
- `onblur`, `onfocus`

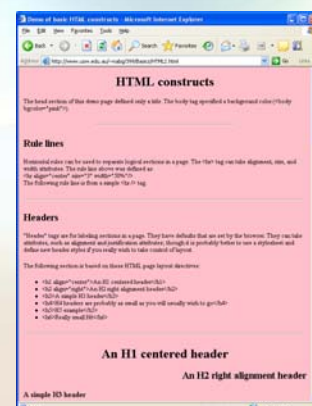
```
<body onload="javascript:alert('You must be over 18 to  
view this page!')">
```

standard layout tags

- Sometime at school you must have composed a HTML page using standard tags!
 - `
`, `<p>`, `<pre></pre>`, `<h1></h1>`(etc), `...` (``, list options etc), definition lists (`<dl>`, `<dt>`, `<dd>`), ...
 - "semantic" tags (`<cite>`, `<pre>`, `<quote>`, ...)
 - "physical" tags (``, ``, `<blink>`, ...)
 - images

Important tags for basic layout

- `<p>...</p>` : paragraph
- lists
 - `...` numbered list
 - `...` bullet list
 - `...` bracket each item in list
- `
`
- `<hr />`
- `<h1>...</h1>` ... `<h4>...</h4>`



tables

- Obvious use is for tables of numbers, such as output from spreadsheet.
- Controls provided by table can make them useful for creating neater looking pages

Captions etc.

Each section is bit of text, or image (& probably a link)

tables

- Table
 - can have caption
 - is composed of rows
 - th (header)
 - td (data)
 - th, td really differ just in respect to default formats (text style, alignment etc)
 - can contain text or image

See a HTML programming book, tables too complex to cover fully here!

```

<table border> <caption>Results for 121</caption>
<tr>
  <td colspan=2 rowspan=2></td>
  <th colspan=2 align=center>Examination</th>
</tr>
<tr>
  <th>Fail</th><th>Pass</th> </tr>
<tr align=center>
  <th rowspan=2>Practical</th><th>Fail</
  <td>35</td> <td>3</td>
</tr>
<tr align=center>
  <th>Pass</th> <td>29</td> <td>163</td>
</tr>
</table>
    
```

Results for 121		
Practical	Examination	
	Fail	Pass
35	3	
	29	163

Tables and forms

- A second very important use of tables is the layout of forms
 - Flowlayout forms (default) are ugly
 - Really have to use tables to get realistic looking forms
- Some HTML generator programs that format word-processor documents make excessive use of tables for layout – resulting in huge HTML documents for only a little content text!

Form with/without table

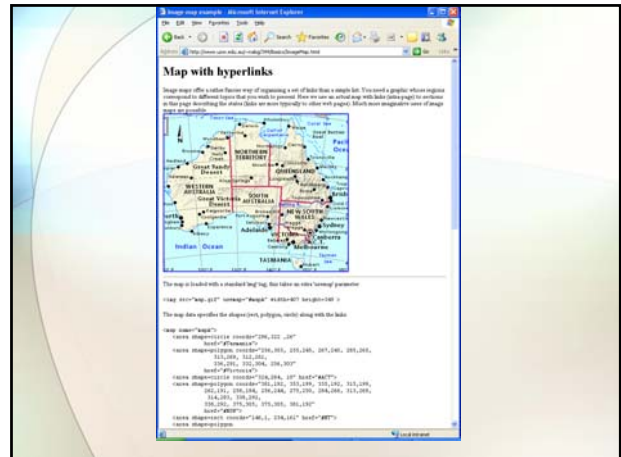
client side image maps

- Use `` to include picture; specify `usemap` attribute with named map.
- Define map with areas
 - several small programs accessible over net that (on Mac or PC) let you sketch out areas over picture and generate the coords that you need
- Support for server side maps probably not needed (put image inside a `<a...>...` link, and specify `ismap` attribute on image)

client side image maps

```

<map name="mapA">
  <area shape=rect coords="30,50, 80,110"
    href="kidsU.html"
    onMouseOver="self.status='Kids Uni';return true">
  <area shape=poly coords="20,120,70,120,..."
    href="Union"
    onMouseOver="self.status='Union; food hall; bookstore';
    return true">
  ..
</map>
```



Links <a ... >

• Two roles

- main one is to tag a link to another point in same page or to a different page

```
<a href="#Subjects">...</a>
```

```
<a href="admissions.html">...</a>
```

- secondary use is to place a label within current page that can be referenced in a real link

```
<a name="Subjects">Subject Information</a>
```

Links <a ... >

• <a> tag has following attributes

- name or href (most of other attributes only meaningful with href)
- title (if referenced page doesn't have a title?)
- rel,rev description of relationship to referenced page
- on... **onclick**, **onmouseout**, **onmouseover** ... associate with events that can trigger Javascript code
- target where to display fetched item

target

- Used mainly (but not exclusively) with frames
- Identifies frame or window where fetched item will appear
 - _blank always make new window
 - _parent, _top in a frameset page, use area of enclosing frameset (or entire window)
 - window_name create window with this name or use it if it already exists

URLs

- http
 - `http://server[:port]/pathname/resource[#tag]`
 - port number defaults to 80, http-daemon
 - #tag only needed if want to link to specific named point in a HTML page
- ftp
 - `ftp://joe:A78pffz@alpha.com[:port]/path/resource[;t type=typecode]`
 - `ftp://ftpserver.das.edu/pub/graphics/jpeg/decoder.c`

Don't put password authenticated links in pages!

URLs

- **file**
 - **file:///path/resource**
 - there is a version that allows you to specify another machines file, but not clear how browser would respond
- **mailto**
 - **mailto:Joe_Cool@xyx.com.au**

URLs

- **telnet**
 - **telnet:// joe:A78pffz@alpha.com[:port]**
- **gopher**
 - **gopher://server[:port]/path**
- **javascript**
 - **bit specialized; another way to trigger Javascript code**

Don't put password authenticated links in pages!

URLs

- **href paths**
 - **absolute URLs have server [port] path etc**
 - these used when linking from one cluster of documents to another cluster (usually on a different server)
 - **relative URLs are used within a group of related pages on same Web site**
 - start with / --- begin at root of http daemon's file hierarchy
 - starts with ../ (repeated), up relative to current directory
 - starts with name (in current directory)

<frameset>

- **Partition browser window into separately scrollable regions that can display contents of different pages.**
- **Control page defines a frameset:**

```
<html><head><title>XYX Corporation</title></head>
<frameset rows="40%,*" cols="50%,*">
  <frame src="images/XYXlogo.gif">
  <frame src="images/XYXboss.gif">
  ...
<noframes>If your browser doesn't handler frames, <a
href="noframe.html">alternative source</a> is
available.</noframes>
</frameset> </html>
```

<frameset>

- **frameset**
 - defines a rectangular grid of frames,
 - specification partitions space available by column and by row
- **nested framesets are allowed (and usually necessary to get useful layouts)**

frameset specifying two columns, one row;
one is just a frame; other is a frameset with
one column and two rows



<frameset>

- **frameset has similar "on" conditions as body (onload etc)**
- **frames can specify**
 - **border**, margin height and width, **scrolling**,
 - **name** (used in "target" specifications for links)
 - **src** initial source for content of frame

100level
200level
300level

`<frame name="main">`

`300level`

Caution on links in pages displayed in frames ...

<form>

- **<form>**
 - **action** url of processing program (or mailto); omit if simply serving as means to get input for Javascript code
 - **method** **get, post**
 - **name** needed for reference from Javascript scripts
 - **on...** **onreset, onsubmit**; for events that trigger Javascript processing
 - **target** results can be sent to named frame (or named window)

input elements

- **button** (trigger Javascript processing)
`<input type=button name=... value="XXX" onClick=...>`
- **standard buttons** (standard form handling)
`<input type=submit value="Place order">`
`<input type=reset>`
- **checkbox** (non-exclusive choices)
Where have you seen details of our special offer?
`<input type=checkbox name=source value="Newspaper">`
`<input type=checkbox name=source value="TV">`
`<input type=checkbox name=source value="WWW">`

input elements

- **radio buttons** (exclusive choices)
`<input type=radio name=size value="Regular">`
`<input type=radio name=size value="Family">`
- **text** (short textual inputs)
`<input type=text name=email size=30 maxlength=128>`
 - size is approximate width of field in characters
- **password**
`<input type=password name=passwd size=10 maxlength=8>`
 - simply doesn't echo characters as they are typed, no encryption, if GET method on form the password appears in ? part of URL

input elements

`<textarea cols=... rows=... name=...>`
text that appears as initial editable content (it shouldn't contain any HTML tags)
`</textarea>`

- **Several on** conditions (onblur, onchange, onfocus, onselect)
- **No maxlength** attribute (so data may overrun your CGI server!)

input elements

- **<select>...</select>**
 - **name** for reference in Javascript
 - **multiple** single (default) or multiple items?
 - **size** how many choices shown in form
 - **on...** onchange, onclick, onfocus
- **<option>**
 - **selected** predefined (default) selection(s)
 - **value** (alternative to using select string)

input elements

```
<select name=toppings size=3 multiple>
  <option>Extra cheese
  <option selected>Pineapple
  <option selected>Ham
  <option>Pepperoni
  ...
</select>
```

The screenshot shows a web form titled "Add a show" with the following fields and sections:

- Theatre/Wel show identifier:** Text input with value "TWOLF-2".
- Venue:** Dropdown menu with value "Hall".
- Type of show:** Dropdown menu with values "Drama", "Opera", and "Film".
- From date:** Date input with value "2007-02-11".
- To date:** Date input with value "2007-02-11".
- Title:** Text input with value "V for Vendetta".
- Company:** Text input with value "Leftover movies".
- Performances:** A table with two rows:

Performances	Enter performance details
2007-02-11, Mac110002	Type of show: Evening
2007-02-11, Evening2	Date: 2007-02-11
- Buttons:** "Add performance" and "Add show".

Page design

- **Book**
 - "Don't make me think" ISBN: 0789723107
 - Library 025.04/135

Javascript programming

Scripts

- **Script languages (shell, JavaScript,...)**
 - (limited) syntax check when script file loaded
 - run via interpreter
 - each time a statement executed repeat
 - full syntax check of statement
 - create parse tree representing operations required
 - execute parse tree (typically a post order traversal of a tree with operations defined at nodes, operands or subexpressions in subnodes)
 - costly and slow at run-time

Scripts

- **run time system** has the code for
 - syntax checking
 - generation of parse tree for expression mechanism for evaluating a parse tree expression
 - model of machine
 - "stack" representing function call sequence
 - memory
- run time system quite elaborate and large

LiveScript

- Netscape was developing this scripting language (“LiveScript”) prior to Java
- Name JavaScript adopted as marketing ploy when Java started to become popular
- One dialect has an official standard defined by some EU standards authority; Microsoft prefers the name ECMAScript (the EU standard)

JavaScript

- JavaScript is
 - an object based language!
 - an event handling system!
 - a typeless language!
 - ...

JavaScript

- JavaScript is “object based” ! (?)
 - well, to a *limited extent*
 - can define data types (classes) by specifying a set of members (identifying data fields and functions)
 - can define functions that relate to manipulation of data elements of specific types
 - can tell a data object to perform one of functions appropriate to its type
 - the syntax used to define data types (classes) is however very messy
 - also, term “object” commonly used to mean both class & instance

JavaScript

- JavaScript is “object based” ! (?)
 - programmer defined object types are possible
 - but mainly use objects provided by system (browser)
 - object that provides information about browser
 - object that represents browser window associated with code
 - object that represents current page
 - objects that represent individual forms, and input elements of forms etc
 - “date”, “array”, “string” (also, “math”)

JavaScript

- JavaScript is an event handling system! ?
 - well, *no*
 - but most of your JavaScript code will specify functions that are to be called when particular events occur (mostly by user interaction with elements of forms)
 - many HTML tags can now have “on...” attributes that identify the function to be called when specific event occurs (e.g. onClick=“...”)

JavaScript

- JavaScript is a “typeless language”! ?
 - well, *yes*; this is actually true for most interpreted script style languages
 - you define a variable
 - get a data structure that corresponds to variable name
 - data structure has some “value” field
 - can at different times hold integer, character, string, “reference” to an object, ...
 - run time system checks whether requested operation on a variable actually appropriate to data type of thing in “value” field

JavaScript in a page

```
<HTML><HEAD>
...
<SCRIPT LANGUAGE=
"JavaScript">
<!--
...
--> </SCRIPT>
...
</HEAD>

<BODY>
...
<... onClick="doX"
<SCRIPT...

```

"Head" portion of page

Function definitions, global variables, ... code to be executed as page loaded

"Body" portion of page

calls to functions associated with events code to be executed as page loaded

JavaScript in a page

- Major use
 - function declarations
 - event related calls
- Code executed when loading
 - "document.write(string)", "document.writeln(string)" display of generated text
 - statements executed in order that they occur on page
 - ?
 - alternative is onLoad event associated with <body ...> tag

Language

- Included between `<script ...>` and `</script>` tags
- Convention is to use HTML comment brackets along with code in case page loaded by browser that does not handle script
- Calls to functions in "onX" parts of tags
 - `<BODY ... onLoad="doWelcome()"`
- Can have sequence of JavaScript statements in such an attribute

Language

- Usual stuff
 - identifiers `letter[letter|digit]*`
 - keywords *function, var, if, else, for, true, false, ...*
 - operators
 - assignment `= += -= *= /= %=`
 - arithmetic `+ - * / % ++ --`
 - bits `& | ^ << >>> >>`
 - logical `&& || !`
 - comparison `== != > < >= <=`

Language

- Usual stuff
 - operators
 - can't define own operators for other data types; as in Java there is an exception for strings where + means concatenation
 - precedence
 - function call, array access
 - negation, increment
 - multiply, divide
 - add, subtract
 - shift
 - relation, ...

standard precedence relations

Language

- Usual stuff!
 - control constructs
 - if (condition) ...
 - if(condition) ... else ...
 - for(initialize;termination test; loop updates)
 - while(condition) ...
 - break, continue


```

<html><head>
<title>Binary Number</title>
<script language = "JavaScript">
<!--
function getBinary(aForm) {...}
// -->
</script>
</head><body>
<H1>Convert Number to Binary</H1>
<body>
<form name=bnum>
..
</form></body>
</html>

```

```

<form name=bnum>
Number
<input name=anInteger type=text size=15
maxlength=10 value=0>
<br>
Binary pattern
<input name=aResult type=text size=32
value=0 >
<br>
<input type=button value="Convert to Binary"
onClick="getBinary(bnum) ">
</form>

```

```

function getBinary(aForm) {
  var number = aForm.anInteger.value;
  var result = ""
  for(var i=1; i <= 32; i++) {
    if(number & 1 == 1) {
      result = "1" + result
    } else {
      result = "0" + result
    }
    number = number >> 1
  }
  aForm.aResult.value = result
}

```

Declaration and scope of variables

- **“var” used to declare variables**

```
var number = aForm.anInteger.value;
var result = ""
```
- **If declared outside a function, a variable is global, otherwise local.**
- **Initializers not required.**
- **Can simply use an undeclared variable; interpreter declares a global for you.**

“Defining a class”

- **Syntax is very sloppy (and likely to confuse those who start learning JavaScript before a proper language like Java, C, Pascal, or C++)**
 - no explicit “class”
 - no obvious association of member function definitions with class
 - no explicit list of data members
 - ...

“Defining a class”

- **“Constructor” function serves as class definition as far as there is one.**
- **Example**
 - a “class Pt { public: Pt(int x, int y); void Move(int dx, int dy); void Scale(int mult); ...; private: int fX; int fY; ... }
 - becomes something like:

```

function Pt(x, y)
{ this.X = x; this.Y = y; this.Move = Move; this.Scale = Scale; ... }
function Move(dx, dy) { this.X += dx; this.y += dy; }

```

Creating and Using an instance of class

- Use “new” operator as is standard

```
var aPt
aPt = new Pt(17,32)
```

- then normal “object, perform your X function”:

```
aPt.Move(10, 4)
```

“System provided objects”

- Classes really, but some will be singletons (exactly one instance of class); containership relations indicated by indentation

- window
 - history, location, frame
- document
 - anchor, link
 - form
 - button, submit, reset
 - text, textarea, password,
 - select
 - checkbox
 - radio

also:
navigator
date
“math”
array
...

Identifying objects in containership hierarchy

- Objects identified by specifying qualified name that reflects containership relations
- Example
 - window, displays two frames (frame1 & frame2),
 - frame2 holds a document (HTML page)
 - document has a form named dataentry
 - form dataentry has a radiobutton named choice2

```
window.frame2.document.dataentry.choice2
```

Identifying objects in containership hierarchy

- Can use incomplete names, provided unambiguous
- Example:
 - alert(message) a method of window
 - call should be something like `window.alert(“Invalid credit card number”)`
 - but normally “window.” can be omitted, following is sufficient `alert(“Invalid credit card number”)`

Window

- Properties
 - frames (array of enclosed frames if this is a frameset)
 - name (name of window/frameset; aka ‘self’, ‘window’)
 - status (String displayed in status bar)
 - ...
- Methods
 - alert(message) displays message in dialog box
 - confirm(message) displays message, have OK,Cancel buttons
 - open(url, name, features) opens another window (or uses named frame) to display content of URL, properties defines things for new window like whether need scrollbars
 - prompt(message, response) dialog to get string from user
- Events
 - onLoad, onUnload
 - events specified in <body ...> or <frameset ...> tags

Document

- Properties
 - lots of things like colour of background, colour of links, ...
 - location its URL
 - forms an array of form objects
 - links an array of <a href...> links
 - ...
- Methods
 - clear()
 - open(mimetype)
 - write()
- Events? No. Documents don’t deal directly with any event

Link

- **Properties**
 - target [optional attribute from HTML tag]
- **Events**
 - **onClick** execute code (before chasing link) if hyperlink selected
 - **onMouseOver** executed when mouse crosses link (usually, it is a couple of embedded JavaScript statements that put info in the status bar)

```
<a href=outline.html
onMouseOver= 'window.status="get course outline"; return true' ... />
```

- **Note:** it is possible to have "URL" of a link to be some JavaScript code --- `<a href="javascript: ..."`

Form

- **Properties**
 - action, method attributes from HTML form tag
 - elements array with parts of form (inputs, text areas, etc)
 - ...
- **Methods**
 - submit()
- **Events**
 - **onSubmit()** goes in `<FORM ... >` HTML tag, invokes function to be executed *before* form data actually submitted

```
<form action=... method=...
onSubmit= "return confirmPizza(pizzaform)"
```

- **this is where should invoke code that does main checking of form data**

Form Elements

- **Elements**
 - **button, reset, submit**
 - **properties** name, value
 - **method** click() { a JavaScript function can fake a click in a button, checkbox, ...}
 - **event** onClick
 - **checkbox**
 - **properties** name, value, checked, defaultchecked
 - **method** click()
 - **event** onClick
 - **radio**
 - same as checkbox, with additional length (size of radio set)

```
<input type=button value="Convert to Binary"
onClick="getBinary(bnum) ">
```

Form Elements

- **Elements**
 - **text, textarea**
 - **properties** name, value, defaultValue
 - **methods**
 - focus(), blur(), select()
 - programmed faking of user actions on a text area
 - **events**
 - **onFocus**
 - do something when field "acquires focus" (user clicks on field and starts to type)
 - **onBlur**
 - do something when this field "loses focus" (something else acquired focus)
 - **onSelect**
 - **onChange**

Form Elements

- **Elements**
 - **select**
 - **properties**
 - length (number of options), name,
 - options (array of options objects)
 - selectedIndex (if select doesn't allow multiples)
 - ...
 - **methods** none
 - **events**
 - onFocus
 - onBlur
 - onChange

HTML page objects

- **Most of Javascript functions -**
 - called directly (or indirectly) in association with an event for an element involving something on HTML page
 - will get passed a reference to object (as an argument in call)

Other system provided "objects"

- **Math**
 - bit like Java's math class, just a collection of functions like `sin()`, ...
- **Date and string objects**
 - created by code in Java functions

Date

- `x = new Date(dateinfo)`
 - `dateinfo` is a string like "month day, year hours:minutes:seconds"
 - if just use `Date()` get current date
- **methods**
 - `getDate()`, `getDay()`, `getMinutes()`, ...
 - `setYear(...)`, `setTime(...)`

String

- **methods**
 - changing style `big()`, `blink()`, `bold()`, ...
 - `indexOf(string, startingindex)`
 - `charAt(index)`
 - `substring(first, last)`

A few free functions in library

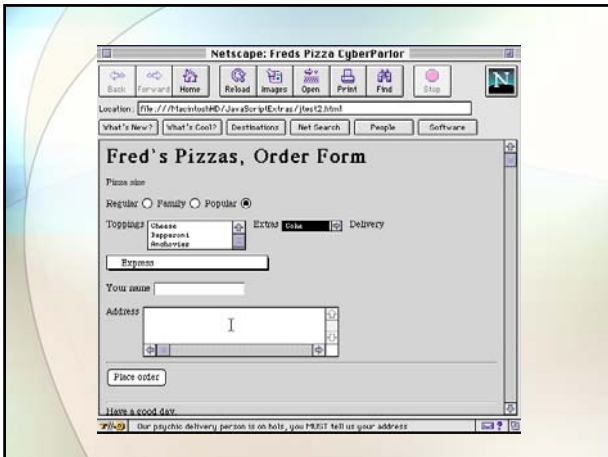
- `parseInt(string, base)`
- `parseFloat(string, base)`
- `escape(string)`
- `unescape(string)`
- ...

Events

- **onClick**
 - `button`, `reset`, `submit`, `radio`, `checkbox`, `link`
- **onMouseOver**
 - `link`
- **onSubmit**
 - `form`
- **onChange, onFocus, onBlur**
 - `text`, `textarea`, `select`
- **onSelect**
 - `text`, `textarea`
- **onLoad, onUnload**
 - `window`

Events

- **onX** attributes simply added to appropriate HTML tags
- Associated value strings are in Javascript, usually simply calls to functions
- `onSubmit`'s code should return a true/false result (confirm whether really to submit)



Examples

- **Checking forms (and providing feedback during process of forms entry) is most useful application of JavaScript.**
- **Fred's Pizza CyberParlor**
 - Javascript to -
 - Make sure Name and Address provided
 - Work out cost of pizza, and seek confirmation before order submitted
 - Feedback on entry

```

<html><head><title>Fred's Pizza CyberParlor</title>
<script language="javascript"><!--
function emptyField(textObj) { ... }
function basecost(radioObj) { ... }
function toppingscost(selectObj) { ... }
function extrascost(selectObj) { ... }
function deliverycost(selectObj) { ... }
function confirmPizza(formObj) { ... }
// --></script></head><body>
<h1>Fred's Pizzas, Order Form</h1>
Pizza size<p>
<form ...> ...
</form><hr>Have a good day.</body></html>

```

```

<form action=...
onSubmit= "return confirmPizza(pizzaform)"
method=POST >
Regular <input type=radio name=p_size value=reg>
...
<p>Toppings <select name=tops size=3 multiple>
<option>Cheese
...
<option>Sun dried tomatoes
</select>
Extras <select name=xtra size=1 multiple>
<option selected>Coke
...
</select>

```

```

...
Delivery
<select name=deliv size=1
onfocus="{ window.status='Choose among delivery
options!;}'" >
<option selected>Express
<option>Fat lady who sings
<option>Female stripper
...
<option>Bozo the programmer
<option>Fred the cook
</select>
<p>Your name

```

```

<p>Your name
<input type=text size=20 maxlength=50 name=customer
onfocus="{ window.status='remember to tell us your
name;}'">
<p>
Address
<textarea name=address cols=40 maxlength = 200 rows =3
onfocus="{ window.status='Our psychic delivery person
is on hols, you MUST tell us your address;}'">
</textarea>
<hr>
<input type=submit Value="Place order">
</form>

```

```

function confirmPizza(formObj) {
    if(emptyField(formObj.customer) ) {
        alert("You forgot to enter your name"); return false; }
    if(emptyField(formObj.address) ) {
        alert("We really do need a delivery address");
        return false;
    }
    total = basecost(formObj.p_size);
    total +=toppingcost(formObj.tops);
    total +=extracost(formObj.xtra);
    total +=deliverycost(formObj.deliv);
    return confirm("The cost is " + total + ", place
    order?");
}

```

```

function emptyField(textObj) {
    if(textObj.value.length == 0) return true;
    for(var i=0; i<textObj.value.length;i++) {
        ch = textObj.value.charAt(i);
        if(ch != ' ' && ch != '\t') return false;
    }
    return true;
}

function basecost(radioObj) {
    if(radioObj[0].checked) cost = 7.00; // regular
    if(radioObj[1].checked) cost = 11.00; // family
    if(radioObj[2].checked) cost = 15.00; // popular
    return cost;
}

```

```

function toppingcost(selectObj) {
    tcost = 0.0;
    for(var i=0;i<selectObj.length;i++)
        if(selectObj[i].selected) tcost += 0.50;
    return tcost;
}

function extracost(selectObj){
    ecost = 0;
    if(selectObj[0].selected) ecost += 1.70; // Coke
    if(selectObj[1].selected) ecost += 1.70; // Lemonade
    if(selectObj[2].selected) ecost += 3.50; // Ice cream
    if(selectObj[3].selected) ecost += 4.50; // Salad
    return ecost;
}

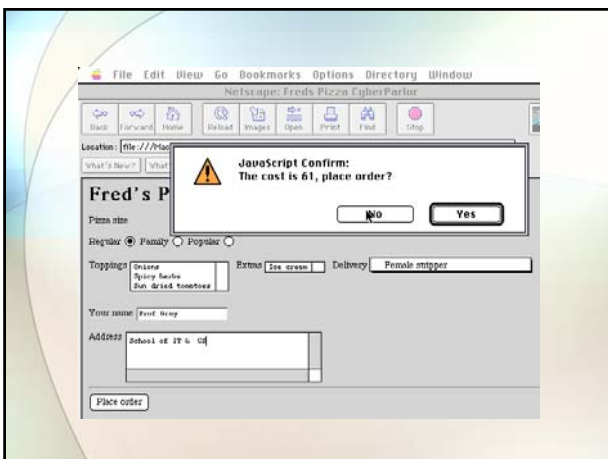
```

```

function deliverycost(selectObj)
{
    if(selectObj[0].selected) Dcost = 1.50; // Express
    else if(selectObj[1].selected) Dcost = 15.0; // Gorilla
    else if(selectObj[2].selected) Dcost = 20.0; // Fat lady
    else if(selectObj[3].selected) Dcost = 50.0; // Female stripper
    else if(selectObj[4].selected) Dcost = 40.0; // Male stripper
    else if(selectObj[5].selected) Dcost = 25.0; // Coco the clown
    else if(selectObj[6].selected) Dcost = 1.65; // Bozo the programmer
    else if(selectObj[7].selected) Dcost = 101.50; // Fred the cook

    return Dcost;
}

```



Javascript for "page pretties"

- Javascript originally introduced primarily for checking of form data prior to submission.
- Later, a second use
 - Animations
 - Pop-up menus
 - Fancier navigation
 - Roll-over features
- Since Javascript is there as source code in downloaded page, if you see a page that has some clever animation you can immediately read the code – and adapt for your pages.

Example: Javascript "Roll-over"



Roll-over

- **HTML code**
 - A table with images as its elements
 - OnMouseOver, OnMouseOut events control animations
- **Javascript code:**
 - Load all images used
 - Handle on-events by changing the image referenced by the table

Roll-over

- **HTML code**

```
<table border=0 cellpadding=0 cellspacing=0 width=600>
<tr>
  <td valign=bottom align=center>
    <a href="#Sci" ONMOUSEOUT="resetText()"
      ONMOUSEOVER="chooseText(1)">
      <img src=images/science.gif border="0" width="180"
        height="140" Alt="Science">
    </a>
    ...
  </td>
</tr>
```

Roll-over

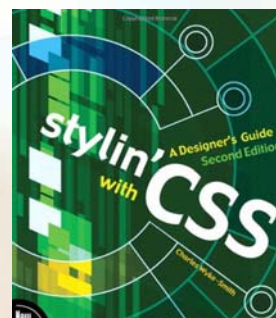
- **Javascript**

```
function chooseText(num) {
  document.images.text.src = choices[num].src
}
function resetText() {
  document.images.text.src =
    "./images/home.gif"
}
```

A little more on CSS

Source ...

- These examples are highly simplified versions of examples from this book



... and they don't work Internet Explorer

- Wyke-Smith grumbles a lot about IE6 and includes "work arounds" to make his examples work.
- My simplified versions mostly don't work with IE8 let alone older versions of IE.
 - Try using code by Wyke-Smith from the website associated with text.

Prettying up tables

- Tables are essential for presenting all those reports on contents of databases
- but by default they do look a bit tacky ...

Cell1	Cell2	Cell3	Cell4
Row1-data1	Row1-data2	Row1-data3	Row1-data4
Row1-data1	Row1-data2	Row1-data3	Row1-data4
Row1-data1	Row1-data2	Row1-data3	Row1-data4

This table example does work with IE8 (and maybe older siblings)

Table markup

```
<table border="1">
  <caption>A table</caption>
  <thead>
    <tr>
      <th>Cell1</th>
      <th>Cell2</th>
      <th>Cell3</th>
      <th>Cell4</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Row1-data1</td>
      <td>Row1-data2</td>
      <td>Row1-data3</td>
      <td>Row1-data4</td>
    </tr>
    <tr>
      <td>Row1-data1</td>
      <td>Row1-data2</td>
      <td>Row1-data3</td>
      <td>Row1-data4</td>
    </tr>
  </tbody>
</table>
```

Does this look better?

Cell1	Cell2	Cell3	Cell4
Row1-data1	Row1-data2	Row1-data3	Row1-data4
Row1-data1	Row1-data2	Row1-data3	Row1-data4
Row1-data1	Row1-data2	Row1-data3	Row1-data4

The actual table markup hasn't changed – it is going to have to be generated by a program and we wouldn't want to have to output lots more markup. But the table has been placed in a <div ...> with a specified identifier, and the page imports a stylesheet that sets format properties.



As well as illustrating actual prettying up of a table; this example illustrates "context dependent" styling. The two tables have the same HTML code; the second is in a named <div> for which the CSS stylesheet file has a set of changes to <table>, <th>, <td> formats.

Table 2 is in a named div

<p>Here is the table that uses some stylesheet features.</p>

<div id="Table2Div" >

<table border="1">

...

...

</table>

</div>

Changes

- **First**
 - Table gets a border around it, so do the cells.
 - This results in the double outline shown

A table			
Col1	Col2	Col3	Col4
Row1-data1	Row1-data2	Row1-data3	Row1-data4
Row1-data1	Row1-data2	Row1-data3	Row1-data4
Row1-data1	Row1-data2	Row1-data3	Row1-data4

- Remove the border from the table itself
- **Second**
 - That table caption is a bit wishy-washy; make it more emphatic

Style sheet

```
div#Table2Div table {
  border-top-style: none;
  border-bottom-style: none;
  border-left-style: none;
  border-right-style: none;
}
```

In the context of a div named Table2div, redefine some formatting for table – remove border

```
div#Table2Div table caption {
  font-family: Arial,Helvetica,sans-serif;
  font-size: 18px;
  background-color: #606060;
}
```

In the context of the caption section of a table inside a div named Table2div, redefine default font

A table			

More prettying

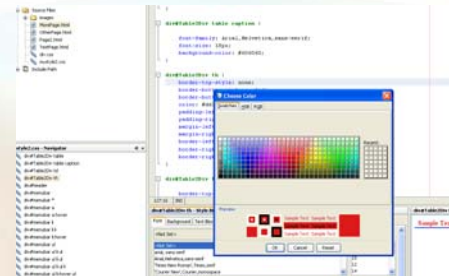
- Change the colour for <th> headers and modify the way borders are added to headers

```
div#Table2Div th {
  border-top-style: none;
  border-bottom-style: solid;
  border-bottom-color: #0b27e6;
  color: #dd1919;
  padding-left: 8px;
  padding-right: 8px;
  margin-left: 10px;
  margin-right: 10px;
  border-left-style: none;
  border-right-style: dotted;
  border-right-color: #e51e1e;
}
```

Col1	Col2	Col3	Col4

How?

- It is easiest to edit stylesheets in NetBeans or some similar wizard style editing environment.



CSS

- **Context sensitive rules ...**
 - Almost any degree of elaboration
 - **div table caption { }**
 - Would affect all captions of tables inside divs; but would not affect a table that wasn't in a div.
 - **div.myclass table caption { }**
 - Would affect caption in tables that were inside divs with class=myclass
 - `<div class='myclass' >`

```
<table>
  <caption>Something</caption>
  ...
</table>
</div>
```

CSS

- **Define a class**
 - .name
 - Qualifier to some other tag
 - Can have many instances
 - Class='...' as attribute in tag
- **Define an individual**
 - #name
 - Qualifier on another tag
 - Only single instance
 - Id='...' as attribute in tag

How about this?

Left/right text alignment; text appearance, and on some browsers minimum width controls

Doesn't work properly on IE!

This one involves setting widths for paragraphs etc

```
<body>
  <div id="header">
    <p id="headline">
      The second styled page.
    </p>
    <p id="byline">
      nabg@uow.edu.au
    </p>
  </div>
```

CSS

```
div#header {
  font-family: Georgia,'Times New Roman',times,serif;
  font-size: 24px;
  background-color: #000000;
  color: #ececce;
}

p#headline {
  text-align: left;
  min-width:600px;
}

p#byline {
  text-align: right;
  font-size:18px;
  min-width:500px;
}
```

Doesn't work properly with IE

IE appears not to honour these width settings.

(It should clip the content as the page is resized smaller)

CSS – info popup

Move mouse over active area and text pops up (well, on CSS compliant browsers it does, but not on ...)

The HTML ...

```
<body>
  <p>
    Following "stylin with CSS"
  </p>
  <div id="Darcydiv" >
    
    <p id="Darcytxt" >Darcey Bussell dancing
    the role Nikia in a Royal Ballet production of
    Makarova's version of
    Bayadere.</p>
  </div>
```

CSS

1. Div defines a reserved area (slightly larger than the image that is active)
2. Paragraph defines presentation style for text and sets it so that by default it is not displayed.
3. ":hover" property associated with div, will affect all contained paragraphs and set them so that they are displayed
 - (IE doesn't understand the :hover property, but there is a hack that makes it work – see the Stylin' with CSS textbook)

CSS

```
div#Darcydiv {
  background-color: #ffbfff;
  border-top-width: 2px;
  border-bottom-width: 2px;
  border-left-width: 2px;
  border-right-width: 2px;
  border-top-color: #000000;
  border-bottom-color: #000000;
  border-left-color: #000000;
  border-right-color: #000000;
  width: 300px;
  height: 500px;
  padding: 5px;
  position: relative;
}
```

Basically, claim an area 300x500, and colour it

Lots of subtleties relating to "position" – this enclosing div must be "relative"

CSS

```
p#Darcytxt {
  display:none;
  width: 150px;
  border-top-style: solid;
  border-bottom-style: solid;
  border-left-style: solid;
  border-right-style: solid;
  border-top-width: 1px;
  border-bottom-width: 1px;
  border-left-width: 1px;
  border-right-width: 1px;
  padding: .3em;
  background-color: #ffcfff;
  position: absolute;
  left: 180px;
  top: 20px;
}

div#Darcydiv:hover p { display:block; }
```

Define an area for the paragraph – position is absolute relative to enclosing div. Given a slightly different fill colour, and contained text indented slightly from the edges

If mouse is hovering within the div named Darcydiv, all contained paragraphs should be displayed.

Popup menu with interpage links



HTML

```
<div id="menubar">
  <ul>
    <li><a href="#">People</a>
    <ul>
      <li><a href="/TomPage.html">Tom</a></li>
      <li><a href="/DickPage.html">Dick</a></li>
      <li><a href="/HarryPage.html">Harry</a></li>
      <li><a href="/SuePage.html">Sue</a></li>
    </ul>
    <li><a href="#">Places</a>
    <ul>
      <li><a href="/North.html">North</a></li>
      <li><a href="/East.html">East</a></li>
      <li><a href="/South.html">South</a></li>
      <li><a href="/West.html">West</a></li>
    </ul>
    <li><a href="#">Colours</a>
    <ul>
      <li><a href="/RedPage.html">Red</a></li>
      <li><a href="/GreenPage.html">Green</a></li>
    </ul>
  </ul>
</div>
<p>Some content text here maybe.</p>
</div>
```

CSS

```
div#menubar {
  float:left;
  min-width:200px;
  width:100%;
  background-color: #303030;
  font-family: arial, sans-serif;
  font-size:.8em; /* size of menu's type relative to parent element */
  background-color:transparent;
  margin-top: 10px;
}

div#menubar * {
  margin:0; padding:0;
}
```

Menubar div set up, font defined; top margin separates it a little from whatever is above.

Everything defined inside the div has its margin and padding defaulted to zero.

CSS

```
div#menubar ul {  
  border-left:1px solid #CCB;  
  display:inline;  
  float:left;  
  font-size:1em;  
}  
  
div#menubar li {  
  background-color:#E0E7C9;  
  border-right:1px solid #CCB;  
  float:left;  
  list-style-type:none;  
  position:relative;  
}  
  
div#menubar li:hover {  
  background-color:#F0F7D9;  
}
```

*An unordered-list displayed in menubar is inline – across the bar, moved left as far as possible.
List items don't have any bullets or numbering, given a beige colour; if the mouse is over them, the colour fades a bit.*

CSS

```
div#menubar a {  
  color:#776;  
  padding:.2em 5px;  
  display:block;  
  text-decoration:none;  
}  
  
div#menubar a:hover {  
  color:#443;  
}
```

<a> href links don't get underlined; coloured greyish, they darken if mouse hovers above them.

CSS

```
div#menubar ul li ul {  
  width:12em;  
  position:absolute;  
  border:0;  
  margin-left:0px;  
  display: none;  
}  
  
div#menubar li li {  
  background-color:#E0E7C9;  
  border-bottom:1px solid #CCB;  
}
```

*Sublists (a ul inside an li within a ul) are given a fixed width (well it depends on font used) and are positioned "absolute" relative to their parent.
A list item descendant of a list item has a colour – actually same as menubar colour*

CSS

```
div#menubar ul li ul li {  
  width:100%;  
  padding:0;  
  border-left:0;  
  border-right:0;  
}  
  
div#menubar ul li ul (display:none;)  
  
div#menubar ul li:hover ul {  
  display:block;  
}
```

*Sublists (ul li ul) within the menubar are not normally displayed.
If mouse hovers over list item in menu bar, any sublist is displayed.*

CSS?!

- Those context rules do get pretty awful.
- But they are standard; once someone has got them to work – everyone else just copies!