Università di Pisa
Dipartimento di Ingegneria dell'informazione

Ingegneria Informatica

Progettazione WEB

A.A 2020-2021 - Primo semestre

Diapositive di Francesco Marcelloni

Indice

1 HTML **2**

2 CSS **75**

3 JavaScript **129**

4 PHP **213**

5 HTTP **257**

Avviso Attenzione a come stampate le pagine. In copisteria sanno sicuramente come effettuare correttamente la stampa.

Designing Web Applications

HTML 5

Francesco Marcelloni

Department of Information Engineering
University of Pisa
ITALY

Outline

- The HTML 5.0 language.
- The cascade style sheets (CSS).
- Client-side Programming: JavaScript.
- Server-Side Programming: PHP.
- The HTTP protocol
- Laboratories: development of client-side and server-side applications by using HTML, CSS, JavaScript and PHP.

Suggested references

- Standards (http://info.iet.unipi.it/~france/p_docs/pweb/home_specifiche.html)
- Any book on HTML, CSS, JavaScript, PHP and HTTP.
- Slides provided by the teacher

- **M. Avvenuti, G. Cecchetti, M.G.C.A. Cimino, "Lezioni di programmazione web", edito da Esculapio, Bologna, Ottobre 2010.**
- **M. Avvenuti, M.G.C.A. Cimino, "Laboratori di programmazione web", edito da McGraw-Hill, 2010.**

Useful Information

- The slides can be downloaded at the following address: http://info.iet.unipi.it/~france/p_docs/pweb/home.html

user: sistemiInf
password: abcd4321

To Pass Examination

- Project
  - Pre-requisite to take the exam is the development of a Web application which performs some specific service. You can find all the relevant information at the following address: http://info.iet.unipi.it/~france/p_docs/pweb/home.html
- Exam
  - Practical exam performed in laboratory (approximately 1.5 hours)
- Final Mark and Project Evaluation
  - Only if the exam is passed (mark >=18), then the project will be discussed and a final mark will be assigned based on the marks obtained both in the exam and in the project

Università di Pisa Francesco Marcelloni

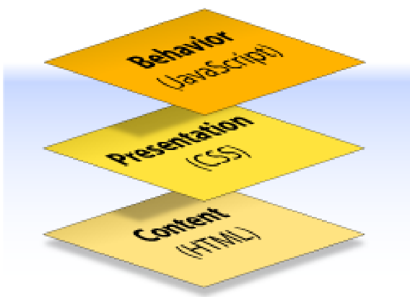
World Wide Web

- World Wide Web (WWW): network of information resources.
- Information resources: documents or pages which can contain link to other documents or pages with correlated information
- Documents with only text: Hypertext
- Documents with also images, voice and so on: Hypermedia
- Users access documents, independently of their location, through interactive programs, called browsers, or other multimedia programs.
- Such programs allow transferring these documents, showing the contents, navigating through the hypertext and hypermedia network.

Università di Pisa Francesco Marcelloni

Web documents

- A web document can consist of up to three layers
  - Content
  - Presentation
  - Behaviour



Università di Pisa Francesco Marcelloni

Content Layer

- The *content layer* is always present.
- It comprises the information the author wishes to convey to his or her audience, and is embedded within HTML markup that defines its structure and semantics.
- Most of the content on the Web today is text, but content can also be provided through images, animations, sound, video, and whatever else an author wants to publish.

Presentation Layer

- The *presentation layer* defines how the content will appear to a human being who accesses the document in one way or another.
- The conventional way to view a web page is with a regular web browser, of course, but that is only one of many possible access methods.
  - For example, content can also be converted to synthetic speech for users who have impaired vision or reading difficulties.

Behavior Layer

- The *behavior layer* involves real-time user interaction with the document.
- This task is normally handled by JavaScript.
- The interaction can be anything from a trivial validation that ensures a required field is filled in before an order form can be submitted, to sophisticated web applications that work much like ordinary desktop programs.

Basic concept

- HTML -> structure and content
- CSS -> style and appearance
- Javascript -> behavior

Example: <h2>Via Diotisalvi 2</h2>
!!!
<h2>Address</h2>
<p>Via Diotisalvi 2</p>

This is some text!
Use CSS to define the font face, font size, and font color of text.

WWW – Three basic mechanisms

- WWW relies on three basic mechanisms:
  - A uniform naming scheme for locating resources on the Web (e.g., URIs)
  - Protocols, for access to named resources over the Web (e.g., HTTP), followed by "://"
  - Hypertext, for easy navigation among the resources (e.g., HTML)

Uniform Resource Identifier (URI)

- Each information resource on the Web has an address which can be codified by a URI.
- A URI is defined as

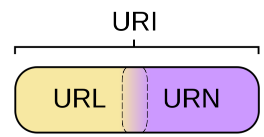
  <scheme>:<scheme-specific-part>
- The <scheme> is the name of the mechanism used to access the resource (for instance, http protocol)
  - A colon character (:)
  - A scheme-specific part.

    For instance
    - The name of the machine hosting the resource (for instance, www.w3.org)
    - The name of the resource itself, given as path

Universal Resource Identifier (URI)

- A **URI** may be a locator (URL) or a name (URN), or both.
- A Uniform Resource Locator (URL) is a URI that, in addition to identifying a resource, specifies the means of acting upon and obtaining the representation.

  For example, the URL http: //www.w3.org/TR



Universal Resource Identifier (URI)

- A Uniform Resource Name (URN) is a URI that identifies a resource by name, in a particular namespace. The resource does not need to necessarily be network homed.

  For example, the URN *urn:isbn:0-395-36341-1* is a URI that specifies the identifier system, i.e. International Standard Book Number (ISBN), as well as the unique reference within that system and allows one to talk about a book, but does not suggest where and how to obtain an actual copy of it.
- URN - person's name
- URL – person`s street address.

Universal Resource Identifier (URI)

- Fragment Identifier

  Some URIs refer to a location within a resource. The URI ends with "#" followed by an anchor identifier (called the *fragment identifier*).

  http://somesite.com/html/top.html#section_2

- Relative URI

  A *relative URI* contains no naming scheme information. Relative URIs are resolved to fill URIs using a base URI.

  <img src="../icons/logo.gif" alt="logo">

  http://somesite.com/icons/logo.gif

Universal Resource Identifier (URI)

In HTML, URIs are used to:

- Link to another document or resource,
- Link to an external style sheet or script,
- Include an image, object, or applet in a page,
- Create an image map,
- Submit a form,
- Create an iframe document,
- Cite an external reference,
- Refer to metadata conventions describing a document

Protocols

- http - Hypertext transfer Protocol
- ftp - File Transfer protocol
- mailto - Electronic mail address
- news - USENET news
- file - Host-specific file names

Protocol HTTP

- HTTP is a request-response standard typical of client-server computing: web browsers typically act as clients, while an application running on the computer hosting the web site acts as a server.
  - The client establishes a connection with the server and sends a request for a document
  - The server replies to the request by using the connection opened by the client
- Since the protocol is stateless (the server does not recognise the client and does not record the requests), if the connection falls, the request has to be repeated.

HTML

- HyperText Markup Language (HTML) is the language used to publish information on the Web
- HTML 4.01 is an Standard Generalized Markup Language (SGML) application conforming to International Standard ISO 8879.
  - Publish online documents with headings, text, tables, lists, photos, etc.
  - Retrieve online information via hypertext links, at the click of a button.
  - Design forms for conducting transactions with remote services.
  - Searching for information, making reservations, ordering products, etc.
  - Include spread-sheets, video clips, sound clips, and other applications directly in their documents.

HTML

- HTML 5
  - Compatibility
  - Utility
    - The user is king
  - Interoperability simplification
    - Native browser ability instead of complex JavaScript code
    - Powerful yet simple HTML5 APIs
    - HTML5 specification is also more detailed than previous ones to prevent misinterpretation: the specification is over 900 pages long!
    - HTML5 is also designed to handle errors well, with a variety of improved and ambitious error handling plans. It prefers graceful error recovery to hard failure, again giving top priority to the interest of the end user.
  - Universal Access
    - Accessibility: support users with disabilities
    - Media Independence
    - Support for all world languages

HTML Syntax

HTML 5.0 Document Structure

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8">
   <title> My first HTML document </title>
   <!– A simple html document -->
 </head>
 <body>
   <p> Hello world! </p>
 </body>
</html>
```


Constructs used in HTML

- Elements
  - Everything from the start tag to the end tag
    `<h1>Sample page</h1>`
- Attributes
  - Provide additional information about HTML elements
    `<p>This is a <a href="demo.html">simple</a> sample.</p>`
- Text
  - Text is allowed inside elements, attribute values and comments
- Character references
  - Examples: é (é) è (è)
- Comments
  - Examples: `<!– comment -->`

HTML Elements

<start_tag attribute_list>
element_content
<end_tag>

- The attribute list may contain the name and the identifier of the element
- The element names and identifiers are always case-insensitive.
- Some HTML elements have empty content. Empty elements are closed in the start
  -
 is an empty element without a closing tag (it defines a line break).
- Most HTML elements can have attributes
- Most HTML elements can be nested.

HTML Elements

- Note: Most browsers will display HTML correctly even if you forget the end tag:

  <p>This is a paragraph

- The example above will work in most browsers, but don't rely on it. Forgetting the end tag can produce unexpected results or errors.
- Elements are not tags. Some people refer to elements as tags (e.g., "the p tag"). For instance, the head element is always present, even though both start and end head tags may be missing in the markup.

HTML Elements

- HTML tags are not case sensitive: <P> means the same as <p>.
- Plenty of web sites use uppercase HTML tags in their pages.
- The World Wide Web Consortium (W3C) recommends lowercase in HTML 4, and demands lowercase tags in future versions of (X)HTML

Element definitions An example

4.4.1 The p element

Categories:
Flow content.
Palpable content.

Contexts in which this element can be used:
Where flow content is expected.

Content model:
Phrasing content.

Content attributes:
Global attributes

Tag omission in text/html:
A p element's end tag may be omitted if the p element is immediately followed by an address, article, aside, blockquote, div, dl, fieldset, footer, form, h1, h2, h3, h4, h5, h6, header, hgroup, hr, main, nav, ol, p, pre, section, table, or ul, element, or if there is no more content in the parent element and the parent element is not an a element.

Allowed ARIA role attribute values:
Any role value.

Allowed ARIA state and property attributes:
Global aria-* attributes
Any aria-* attributes applicable to the allowed roles.

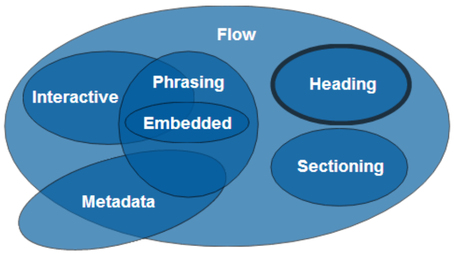
DOM interface:

```
IDL  interface HTMLParagraphElement : HTMLElement {};
```

Element definitions
Category

- Categories
  - Each element in HTML falls into zero or more categories that group elements with similar characteristics together.

Categories:
Flow content.
Palpable content.



Element definitions
Category

Content Type	Description
Embedded	Content that imports other resources into the document, for example `audio`, `video`, `canvas`, and `iframe`
Flow	Elements used in the body of documents and applications, for example `form`, `h1`, and `small`
Heading	Section headers, for example `h1`, `h2`, and `hgroup`
Interactive	Content that users interact with, for example `audio` or `video controls`, `button`, and `textarea`
Metadata	Elements—commonly found in the `head` section— that set up the presentation or behavior of the rest of the document, for example `script`, `style`, and `title`
Phrasing	Text and text markup elements, for example `mark`, `kbd`, `sub`, and `sup`
Sectioning	Elements that define sections in the document, for example `article`, `aside`, and `title`

Element definitions
Context

- Context in which the element can be used
  - A non-normative description of where the element can be used
    - For simplicity, only the most specific expectations are listed. For example, an element that is both flow content and phrasing content can be used anywhere that either flow content or phrasing content is expected, but since anywhere that flow content is expected, phrasing content is also expected (since all phrasing content is flow content), only "where phrasing content is expected" will be listed.

Contexts in which this element can be used:
Where flow content is expected.

Element definitions
Content

- Content model
  - A normative description of what content must be included as children and descendants of the element.

Content model:
Phrasing content.

Element definitions
Attributes

- Content attributes
  - A normative list of attributes that may be specified on the element (except where otherwise disallowed), along with non-normative descriptions of those attributes.

Content attributes:
Global attributes

Element definitions
Tag omission

- Tag omission in text/html
  - A non-normative description of whether, in the text/html syntax, the start and end tags can be omitted.

Tag omission in text/html:

A `p` element's end tag may be omitted if the `p` element is immediately followed by an `address`, `article`, `aside`, `blockquote`, `div`, `dl`, `fieldset`, `footer`, `form`, `h1`, `h2`, `h3`, `h4`, `h5`, `h6`, `header`, `hgroup`, `hr`, `main`, `nav`, `ol`, `p`, `pre`, `section`, `table`, or `ul`, element, or if there is no more content in the parent element and the parent element is not an `a` element.

Element definitions
DOM Interface

- DOM (Document Object Model) interface
  - A normative definition of a DOM interface that such elements must implement.
  - What is DOM?
    - Let us consider the following example

Element definitions
DOM Interface

```
<!DOCTYPE html>
<html>
 <head>
  <title>Sample page</title>
 </head>
 <body>
  <h1>Sample page</h1>
  <p>This is a <a href="demo.html">simple</a> sample.</p>
  <!-- this is a comment -->
 </body>
</html>
```

Element definitions
DOM Interface

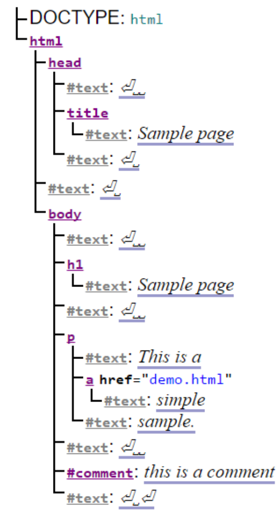
- HTML user agents (e.g. Web browsers) parse this markup, turning it into a DOM (Document Object Model) tree.
- A DOM tree is an in-memory representation of a document.
- DOM trees contain several kinds of nodes, in particular a DocumentType node, Element nodes, Text nodes, Comment nodes, and in some cases ProcessingInstruction nodes.

DOM interface:

```
IDL  interface HTMLParagraphElement : HTMLElement {};
```


Element definitions
DOM Interface

<!DOCTYPE html>
<html>
<head>
<title>Sample page</title>
</head>
<body>
<h1>Sample page</h1>
<p>This is a simple sample.</p>
<!-- this is a comment -->
</body>
</html>



Element definitions

- WAI-ARIA (Web Accessibility Initiative -Accessible Rich Internet Applications specification)
- Client-side scripting causes accessibility problems when they do not convey the information necessary to Assistive Technologies (AT) – the devices and user systems typically used by people with disabilities. In simple terms assistive technology needs to know:
  1. What things are: The Role. (E.g., I am a checkbox)
  2. What they are doing: The state. (E.g., I am now checked)
  3. What the relationships are. (E.g., I am labeled by that text, I am part of this group, etc.)
  4. Also the focus needs to be accessible from the keyboard and the interaction needs to be predictable. (E.g., you can tab to this checkbox and press enter, then I am checked)

Element definitions

- ARIA role attributes
  - The attribute describes the role(s) the current element plays in the context of the document.
    - This could allow a user to make informed decisions on which actions may be taken on an element and activate the selected action in a device independent way.
  - The document "Using WAI-ARIA" is a practical guide for developers on how to add accessibility information to HTML elements, which defines a way to make Web content and Web applications more accessible to people with disabilities.
  - This document demonstrates how to use WAI-ARIA, which especially helps with dynamic content and advanced user interface controls developed with Ajax, HTML, JavaScript, and related technologies

Element definitions

- ARIA State and Properties attributes
  - Every HTML element may have ARIA state and property attributes specified. These attributes are defined by [ARIA] in Section 6.6, Definitions of States and Properties (all aria-* attributes).
  - ARIA State and Property attributes can be used on any element.

Allowed ARIA role attribute values:

Any role value.

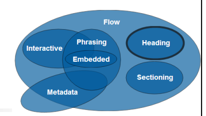
Allowed ARIA state and property attributes:

Global aria-* attributes

Any `aria-*` attributes applicable to the allowed roles.

Università di Pisa
Francesco Marcelloni

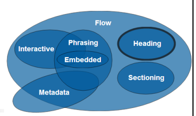
Metadata Content



- Metadata content is content that sets up the presentation or behavior of the rest of the content, or that sets up the relationship of the document with other documents, or that conveys other "out of band" information.
  - base link meta noscript script style template title

Università di Pisa
Francesco Marcelloni

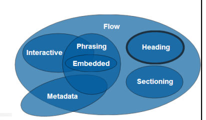
Flow Content



- Most elements that are used in the body of documents and applications are categorized as flow content.
- a abbr address area (if it is a descendant of a map element) article aside audio b bdi bdo blockquote br button canvas cite code data datalist del dfn div dl em embed fieldset figure footer form h1 h2 h3 h4 h5 h6 header hr i iframe img input ins kbd keygen label main map mark math meter nav noscript object ol output p pre progress q ruby s samp script section select small span strong sub sup svg table template textarea time u ul var video wbr

Università di Pisa
Francesco Marcelloni

Sectioning Content



- Sectioning content is content that defines the scope of headings and footers
- article aside nav section

Heading Content

- Heading content defines the header of a section (whether explicitly marked up using sectioning content elements, or implied by the heading content itself)
- h1 h2 h3 h4 h5 h6

Phrasing Content

- Phrasing content is the text of the document, as well as elements that mark up that text at the intra-paragraph level. Runs of phrasing content form paragraphs
- a abbr area (if it is a descendant of a map element) audio b bdi bdo br button canvas cite code data datalist del dfn em embed i iframe img input ins kbd keygen label map mark math meter noscript object output progress q ruby s samp script select small span strong sub sup svg template textarea time u var video wbr text
- Text, in the context of content models, means either nothing, or Text nodes. Text is sometimes used as a content model on its own, but is also phrasing content, and can be inter-element whitespace (if the Text nodes are empty or contain just space characters).

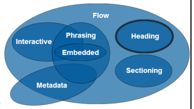
Embedded Content

- Embedded content is content that imports another resource into the document, or content from another vocabulary that is inserted into the document.
- audio canvas embed iframe img math object svg video

Interactive Content

- Interactive content is content that is specifically intended for user interaction.
- a audio (if the controls attribute is present) button embed iframe img (if the usemap attribute is present) input (if the type attribute is not in the hidden state) keygen label object (if the usemap attribute is present) select textarea video (if the controls attribute is present)

Palpable Content



- As a general rule, elements whose content model allows any flow content or phrasing content should have at least one node in its contents that is palpable content and that does not have the hidden attribute specified.

- a abbr address article aside audio (if the controls attribute is present) b bdi bdo blockquote button canvas cite code data dfn div dl (if the element's children include at least one name-value group) em embed fieldset figure footer form h1 h2 h3 h4 h5 h6 header i iframe img input (if the type attribute is not in the hidden state) ins kbd keygen label main map mark math meter nav object ol (if the element's children include at least one li element) output p pre progress q ruby s samp section select small span strong sub sup svg table textarea time u ul (if the element's children include at least one li element) var video text that is not inter-element whitespace

Università di Pisa
Francesco Marcelloni

Script-supporting Elements

- Script-supporting elements are those that do not represent anything themselves (i.e. they are not rendered), but are used to support scripts, e.g. to provide functionality for the user.
- script template

Università di Pisa
Francesco Marcelloni

Transparent content models

- Some elements are described as transparent. The content model of a transparent element is derived from the content model of its parent element: the elements required in the part of the content model that is "transparent" are the same elements as required in the part of the content model of the parent of the transparent element in which the transparent element finds itself.

- For instance, an ins element inside a ruby element cannot contain an rt element, because the part of the ruby element's content model that allows ins elements is the part that allows phrasing content, and the rt element is not phrasing content.

Università di Pisa
Francesco Marcelloni

HTML Attributes

- Attribute/value pairs appear before the final ">" of an element’s start tag.
- Any number of (legal) attribute value pairs, separated by spaces, may appear in an element’s start tag.
- Attributes may appear in any order.
- The attribute value can remain unquoted if it does not contain space characters or any of " ' ` = < or >. Otherwise, it must be delimited using either double quotation marks (ASCII decimal 34) or single quotation marks (ASCII decimal 39)

Attributes

- Example

This is a link

HTML links are defined with the <a> tag.
The link address is provided as an attribute

- Attribute names are always case-insensitive.
- Attribute values are generally case-insensitive.

Attributes

- Attribute values may only contain:
  - letters (a-z and A-Z),
  - digits (0-9),
  - hyphens (ASCII decimal 45),
  - periods (ASCII decimal 46),
  - underscores (ASCII decimal 95),
  - colons (ASCII decimal 58).

- The W3C recommends lowercase attributes/attribute values

Global Attributes

- Global Attributes

  May be specificed on all the HTML elements
  - core attributes
  - event-handler attributes
  - xml attributes

Global Attributes

- Core Attributes
  - accesskey: specifies a shortcut key to activate/focus an element
  - class: Specifies one or more classnames for an element (refers to a class in a style sheet)
  - contenteditable: Specifies whether the content of an element is editable or not
  - dir: Specifies the text direction for the content in an element
  - draggable: Specifies whether an element is draggable or not
  - dropzone: Specifies whether the dragged data is copied, moved, or linked, when dropped (not implemented in the browsers)

Global Attributes

- Core Attributes
  - hidden: Specifies that an element is not yet, or is no longer, relevant
  - id: Specifies a unique id for an element
  - lang: Specifies the language of the element's content
  - spellcheck: Specifies whether the element represents an element whose contents are subject to spell checking and grammar checking.
  - style: Specifies an inline CSS style for an element
  - tabindex: Specifies the tabbing order of an element
  - title: Specifies extra information about an element
  - translate: specifies whether the content of an element should be translated or not (not properly supported in any of the major browsers)

Università di Pisa Francesco Marcelloni

Global Attributes

- Event-handler Attributes

  An event handler content attribute is a content attribute for a specific event handler. The name of the content attribute is the same as the name of the event handler.

  Some common events attributes are
  - onabort - Load of element was aborted by the user.
  - onblur - Element lost focus.
  - onchange - User committed a change to the value of element (form control).
  - onclick - User pressed pointer button down and released pointer button over element
  - ondblclick - User clicked pointer button twice over element
  - ondrag - User is continuing to drag element.

Università di Pisa Francesco Marcelloni

Global Attributes

- Event-handler Attributes (continued)
  - ondragend - User ended dragging element.
  - ondragenter - User’s drag operation entered element.
  - ondragleave - User’s drag operation left element.
  - ondragover - User is continuing drag operation over element.
  - ondragstart - User started dragging element.
  - onerror - element failed to load properly.
  - onfocus - element received focus.
  - oninput - user changed the value of element (form control).
  - oninvalid - element (form control) did not meet validity constraints.
  - onkeydown - user pressed down a key.

Università di Pisa Francesco Marcelloni

Global Attributes

- Event-handler Attributes (continued)
  - onkeypress - User pressed down a key that is associated with a character value.
  - onkeyup - User released a key.
  - onLoad - Element finished loading.
  - onmousedown - User pressed down pointer button over element.
  - onmousemove - User moved mouse.
  - onmouseout - User moved pointer off boundaries of element.
  - onmouseover - User moved pointer into boundaries of element or one of its descendant elements.
  - onmouseup - User released pointer button over element.

Global Attributes

- Event-handler Attributes (continued)
  - onmousewheel - User rotated wheel of mouse or other device in a manner that emulates such an action.
  - onreset - the form element was reset.
  - onscroll - Element or document view was scrolled.
  - onselect – User selected some text.
  - onsubmit – The form element was submitted.

- Other event-handler attributes can be found in the HTML5 specification.

Character references

- *Character references* are a form of markup for representing single individual characters. There are three types of character references:
  - named character references
  - decimal numeric character references
  - hexadecimal numeric character references

Character references

- Named character references
  - an "&" character.
  - one of the names listed in the "Named character references" section of the HTML5 specification [HTML5], using the same case.
  - a ";" character.

- Example:
  - †   for the character †;

Character references

- Decimal Numeric Character Reference
  - an "&" character.
  - a "#" character.
  - one or more digits in the range 0–9, representing a base-ten integer that itself is a Unicode code point that is not U+0000, U+000D, in the range U+0080–U+009F, or in the range 0xD8000–0xDFFF (surrogates).
  - a ";" character.
- Example:
  - †   for the character †;

Character references

- Hexadecimal Numeric Character Reference
  - an "&" character.
  - a "#" character.
  - either a "x" character or a "X" character.
  - one or more digits in the range 0–9, a–f, and A–F, representing a base-sixteen integer that itself is a Unicode code point that is not U+0000, U+000D, in the range U+0080–U+009F, or in the range 0xD800–0xDFFF (surrogates).
  - a ";" character.
- Example:
  - † for the character +;

Università di Pisa
Francesco Marcelloni

Comments

- HTML comments have the following syntax:

  <!-- this is a comment -->

  <!-- and so is this one,
  which occupies more than one line -->

- The text part of comments has the following restrictions:
  - must not start with a ">" character
  - must not start with the string "->"
  - must not contain the string "--"
  - must not end with a "-" character

Università di Pisa
Francesco Marcelloni

Authoring HTML documents

- Separate structure and presentation
  - reduces the cost of serving a wide range of platforms, media, etc., and facilitates document revisions.
- Consider universal accessibility to the Web
  - authors should consider how their documents may be rendered on a variety of platforms
  - in order for documents to be interpreted correctly, authors should include in their documents information about the natural language and direction of the text, how the document is encoded, and other issues related to internationalization.
- Help browsers with incremental rendering
  - Allows browsers to render documents more quickly. For instance, to design tables for incremental rendering

Università di Pisa
Francesco Marcelloni

The Elements of HTML

HTML 5.0 Document Structure

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8">
   <title> My first HTML document </title>
 </head>
 <body>
   <p> Hello world! </p>
 </body>
</html>
```


Università di Pisa Francesco Marcelloni

Root element
HTML Element

4.1.1 The `html` element

Categories:
None.

Contexts in which this element can be used:
As the root element of a document.
Wherever a subdocument fragment is allowed in a compound document.

Content model:
A `head` element followed by a `body` element.

Content attributes:
Global attributes
`manifest` — Application cache manifest

Tag omission in text/html:
An `html` element's start tag can be omitted if the first thing inside the `html` element is not a comment.
An `html` element's end tag can be omitted if the `html` element is not immediately followed by a comment.

Allowed ARIA role attribute values:
none

Allowed ARIA state and property attributes:
Global aria-* attributes

DOM interface:

```
IDL  interface HTMLHtmlElement : HTMLElement {};
```

Università di Pisa Francesco Marcelloni

Root element
HTML Element

- Cache Manifest basic:
  - HTML5 introduces application cache, which means that a web application is cached, and accessible without an internet connection.

    <html manifest="demo.appcache">
  - The manifest file is a simple text file, which tells the browser what to cache (and what to never cache). The manifest file has three sections:
  - CACHE MANIFEST - Files listed under this header will be cached after they are downloaded for the first time
  - NETWORK - Files listed under this header require a connection to the server, and will never be cached
  - FALLBACK - Files listed under this header specifies fallback pages if a page is inaccessible

Università di Pisa Francesco Marcelloni

Root element
HTML Element

- Example of manifest file

```
CACHE MANIFEST
# 2012-02-21 v1.0.0
/theme.css
/logo.gif
/main.js

NETWORK:
login.asp

FALLBACK:
/html/ /offline.html
```

Document Metadata
Head Element

- The head element contains information about the current document, such as its title, keywords that may be useful to search engines, and other data that is not considered document content.
- Every HTML document **must** have a title element in the head section.

```
<head>
<title>HTML course</title>
</head>
```


Document Metadata
Title

- Use the title element to identify the contents of a document.
- Since users often consult documents out of context, authors should provide context-rich titles. Thus, instead of a title such as “A study", authors should supply a title such as “A study of population dynamics" instead.
- User agents must always make the content of the title element available to users
- Titles may contain character entities (for accented characters, special characters, etc.), but may not contain other markup (including comments).

Document Metadata
Meta Data

- Meta Data -- information about a document

```
<head>
<meta charset="utf-8">
<meta name="author" content="Hege Refsnes">
<meta name="description" content="Free Web tutorials">
<meta name="generator" content="Notepad++">
<meta name="keywords" content="HTML,CSS,XML,JavaScript">
<title>HTML5 Example</title>
</head>
```

- Possible values for name: application-name, author, description, generator, keywords

Document Metadata
Meta Data

- The http-equiv attribute integrates an HTTP header with the information in the content attribute.
- The value of the http-equiv attribute depends on the value of the content attribute.

```
<meta http-equiv="Refresh" content="10">
```

Defines a time interval for the document to refresh itself

Document Metadata
Meta Data

- When several meta elements provide language-dependent information, search engines may filter on the lang attribute to display search results using the language preferences of the user.

```
<!-- For speakers of US English -->
<meta name="keywords" lang="en-us"
content="vacation, Greece, sunshine">
<!-- For speakers of British English -->
<meta name="keywords" lang="en"
content="holiday, Greece, sunshine">
<!-- For speakers of French -->
<meta name="keywords" lang="fr"
content="vacances, Gr&egrave;ce, soleil">
```


Document Metadata
Base element

- The base element allows authors to specify the document base URL for the purposes of resolving relative URLs, and the name of the default browsing context for the purposes of following hyperlinks. The element does not represent any content beyond this information.

```
<!DOCTYPE html>
<html>   <head>
    <meta charset="utf-8">
     <title>This is an example for the &lt;base&gt; element</title>
     <base href="http://www.example.com/news/index.html">
   </head>
   <body>
     <p>Visit the <a href="archives.html">archives</a>.</p>
   </body>
</html>
```

The link in the above example would be a link to "http://www.example.com/news/archives.html".

Document Metadata
Link element

- The link element allows authors to link their document to other resources.
- A link element must have a rel attribute. Possible values for rel are “alternate”, “author”, “help”, …, “licence”, “stylesheet”,…

```
<link rel="author license" href="/about">
<link rel=alternate href="/en/html" hreflang=en type=text/html
title="English HTML">
<link rel=alternate href="/fr/html" hreflang=fr type=text/html
title="French HTML">
<link rel=alternate href="/en/html/print" hreflang=en type=text/html
media=print title="English HTML (for printing)">
<link rel=alternate href="/fr/html/print" hreflang=fr type=text/html
media=print title="French HTML (for printing)">
<link rel=alternate href="/en/pdf" hreflang=en type=application/pdf
title="English PDF">
<link rel=alternate href="/fr/pdf" hreflang=fr type=application/pdf
title="French PDF">
```


Document Metadata
Style element

- The style element allows authors to embed style information in their documents. The style element is one of several inputs to the styling processing model. The element does not represent content for the user.
- The type attribute gives the styling language.
- The media attribute says which media the styles apply to. If the media attribute is omitted, is "all", meaning that by default styles apply to all media.

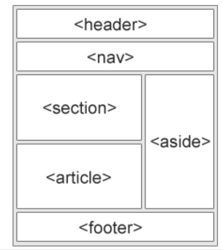
```
<html lang="en-US">
 <head>
  <title>My favorite book</title>
  <style>
   body { color: black; background: white; }
   em { font-style: normal; color: red; }
  </style>
 </head>
```

Sections
Body Element

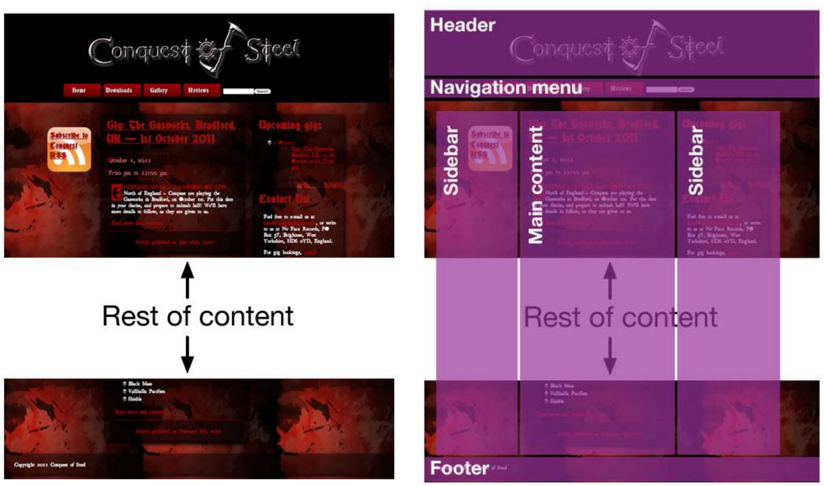
- The body of a document contains the document's content.
- There is only one body element
- The onblur, onerror, onfocus, onload, onresize, and onscroll event handlers of the Window object, exposed on the body element, replace the generic event handlers with the same names normally supported by HTML elements

Sections
Semantic and non-semantic elements

- A semantic element clearly describes its meaning to both the browser and the developer.
- Examples of non-semantic elements: <div> and <span> - Tells nothing about its content.
- Examples of semantic elements: <form>, <table>, and <img> - Clearly defines its content.

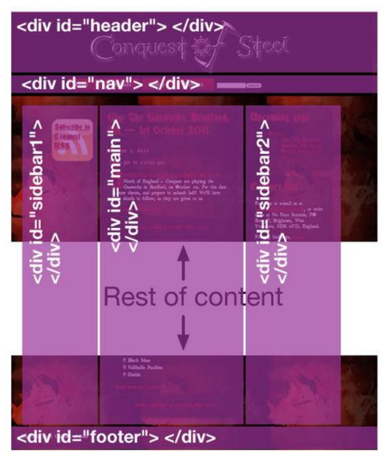


Sections
Document Organization

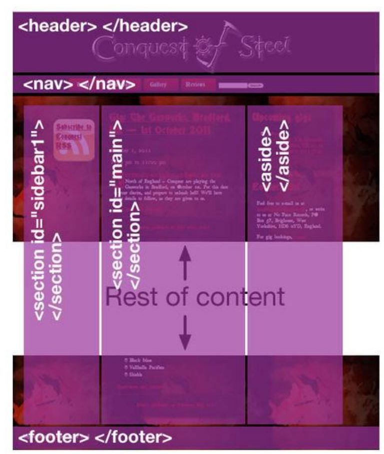


Sections
Document Organization

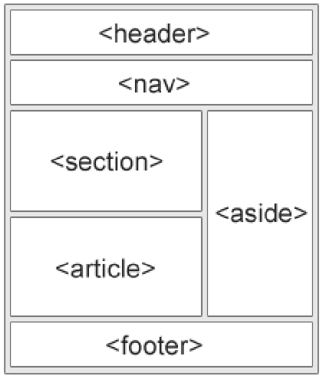
HTML 4



HTML 5



Sections
Document Organization

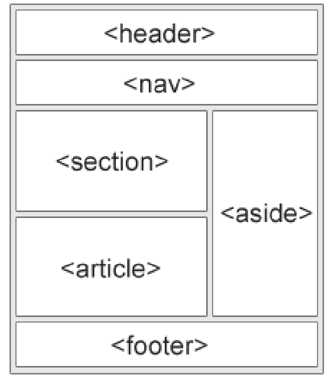


Header Element

- The header element represents introductory content for its nearest ancestor sectioning content or sectioning root element. A header typically contains a group of introductory or navigational aids.
- When the nearest ancestor sectioning content or sectioning root element is the body element, then it applies to the whole page.
- The header element is not sectioning content; it doesn't introduce a new section.

```
<header>
  <!-- header content goes in here -->
</header>
```


Sections
Document Organization

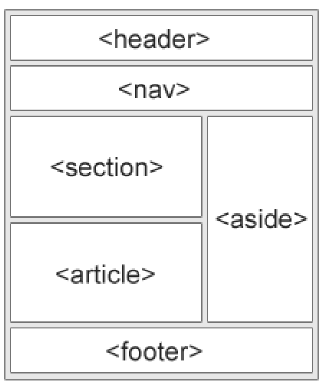


Nav Element

- The nav element represents a section of a page that links to other pages or to parts within the page: a section with navigation links.
- When the nearest ancestor sectioning content or sectioning root element is the body element, then it applies to the whole page.
- The header element is not sectioning content; it doesn't introduce a new section.

```
<nav>
  <!-- navigation menu goes in here -->
</nav>
```


Sections
Document Organization

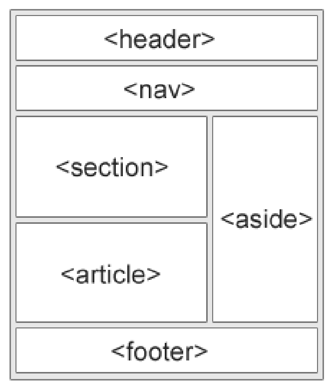


Section Element

- The section element represents a generic section of a document or application.
- A section, in this context, is a thematic grouping of content. The theme of each section should be identified, typically by including a heading (h1-h6 element) as a child of the section element.
- Examples of sections would be chapters, the various tabbed pages in a tabbed dialog box, or the numbered sections of a thesis.

```
<section id="sidebar1">
  <!-- sidebar content goes in here -->
</section>
```


Sections
Document Organization

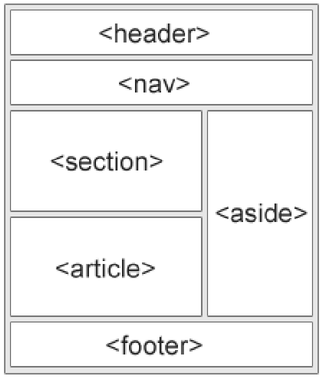


Article Element

- The article element represents a complete, or self-contained, composition in a document, page, application, or site and that is, in principle, independently distributable or reusable, e.g. in syndication.
  - This could be a forum post, a magazine or newspaper article, a blog entry, a user-submitted comment, an interactive widget or gadget, or any other independent item of content.
- When article elements are nested, the inner article elements represent articles that are related to the contents of the outer article.

```
<article>
  <!-- article content goes in here -->
</article>
```

Sections
Document Organization

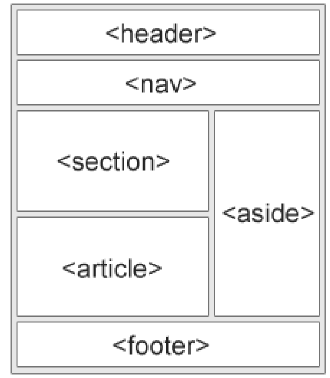


Aside Element

- The aside element represents a section of a page that consists of content that is tangentially related to the content around the aside element, and which could be considered separate from that content. Such sections are often represented as sidebars in printed typography.
- The element can be used for typographical effects like pull quotes or sidebars, for advertising, for groups of nav elements, and for other content that is considered separate from the main content of the page.

```
<aside>
  <!-- article content goes in here -->
</aside>
```


Sections
Document Organization



Footer Element

- Represents a footer for its nearest ancestor sectioning content or sectioning root element.
  - 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.
  - Contact information for the author or editor of a section belongs in an address element, possibly itself inside a footer.

```
<footer>
  <!-- article content goes in here -->
</footer>
```


Sections
Document Organization

Heading Elements

- A heading element briefly describes the topic of the section it introduces. Heading information may be used by user agents, for example, to construct a table of contents for a document automatically.
- There are six levels of headings in HTML with h1 as the most important and h6 as the least.
- Use HTML headings for headings only. Don't use headings to make text BIG or bold.
- Search engines use your headings to index the structure and content of your web pages. Since users may skim your pages by its headings, it is important to use headings to show the document structure.
- h1 headings should be used as main headings, followed by h2 headings, then less important h3 headings, and so on.

```
<h1>This is a heading 1</h1>
```


Sections
Document Organization

Address Element

- The address element represents the contact information for its nearest article or body element ancestor. If that is the body element, then the contact information applies to the document as a whole.
- The address element must not be used to represent arbitrary addresses (e.g. postal addresses), unless those addresses are in fact the relevant contact information.
- Typically, the address element would be included along with other information in a footer element.

```
<address>
 For more details, contact
 <a href="mailto:js@example.com">John Smith</a>.
</address>
```

Sections Example

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>HTML5</title>
 </head>
<body>
  <header>
    <h1>Header</h1>
    <h2>Subtitle</h2>
    <h4>HTML5 Rocks!</h4>
  </header>
```


Sections Example

```
<div id="container">
    <nav>
     <h3>Nav</h3>
     <a href="">Link 1</a>
     <a href="">Link 2</a>
     <a href="">Link 3</a>
    </nav>

    <section>
      <article>
        <header>
          <h1>Article Header</h1>
        </header>
```


Sections Example

```
        <p>HTML5: "Lorem ipsum dolor nunc aut nunquam sit amet,
        consectetur adipiscing elit. Vivamus at est eros, vel fringilla urna.
        Pellentesque odio</p>
        <footer>
          <h2>Article Footer</h2>
        </footer>
      </article>
    </section>

    <aside>
      <h3>Aside</h3>
      <p>HTML5: "Lorem ipsum dolor nunc aut nunquam sit amet,
      consectetur adipiscing elit. Vivamus at est eros, vel fringilla urna.
      Pellentesque odio rhoncus</p>
    </aside>
```


Sections Example

```
    <footer>
      <h2>Footer</h2>
        <address>
        For more details, contact  <a id="mail"
                href="mailto:js@example.com">John Smith</a>.
        </address>
        <p><small>© copyright 2014 Example Corp.</small></p>
    </footer>
  </div>
</body>

</html>
```

Sections
Example with Styles

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8" />
 <title>HTML5</title>
<link rel="stylesheet" href="html5.css">

 </head>
  <body>
  <header>
   <h1>Header</h1>
   <h2>Subtitle</h2>
   <h4>HTML5 Rocks!</h4>
  </header>
```


Grouping content elements

Grouping Content
Element p

- The p element represents a paragraph.
- The p element should not be used when a more specific element is more appropriate.

```
<section>
 <!-- ... -->
 <p>Last modified: 2001-04-23</p>
 <p>Author: fred@example.com</p>
</section>
```

Technically correct, but not appropriate

```
<section>
 <!-- ... -->
 <footer>Last modified: 2001-04-23</footer>
 <address>Author: fred@example.com</address>
</section>
```

Better

Grouping Content
Element hr

- The hr element represents a paragraph-level thematic break, e.g. a scene change in a story, or a transition to another topic within a section of a reference book
- There is no need for an hr element between the sections themselves, since the section elements and the h1 elements imply thematic changes themselves.

Grouping Content
Element pre

- The pre element represents a block of preformatted text, in which structure is represented by typographic conventions rather than by elements.
- Some examples of cases where the pre element could be used:
  - Including an e-mail, with paragraphs indicated by blank lines, lists indicated by lines prefixed with a bullet, and so on.
  - Including fragments of computer code, with structure indicated according to the conventions of that language.
  - Displaying ASCII art.

```
<p>This is the <code>Panel</code> constructor:</p>
<pre><code>function Panel(element, canClose, closeHandler) {
 this.element = element;
 this.canClose = canClose;
 this.closeHandler = function () { if (closeHandler) closeHandler() };
}</code></pre>
```


Grouping Content
Element blockquote

- The blockquote element represents content that is quoted from another source, optionally with a citation which must be within a footer or cite element, and optionally with in-line changes such as annotations and abbreviations.

```
<blockquote>
 The people recognize themselves in their commodities; they find their
 soul in their automobile, hi-fi set, split-level home, kitchen equipment.
 — <cite><a href="http://en.wikipedia.org/wiki/Herbert_Marcuse">Herbert
Marcuse</a></cite>
 </blockquote>
```


Grouping Content
Element figure

- The figure element represents some flow content, optionally with a caption, that is self-contained (like a complete sentence) and is typically referenced as a single unit from the main flow of the document.
- The element can thus be used to annotate illustrations, diagrams, photos, code listings, etc.
- The first figcaption element child of the element, if any, represents the caption of the figure element's contents. If there is no child figcaption element, then there is no caption.
- A figure element's contents are part of the surrounding flow. If the purpose of the page is to display the figure, for example a photograph on an image sharing site, the figure and figcaption elements can be used to explicitly provide a caption for that figure.

Grouping Content
Element figure

Some uses

```
<figure>
 <p>'Twas brillig, and the slithy toves<br>
 Did gyre and gimble in the wabe;<br>
 All mimsy were the borogoves,<br>
 And the mome raths outgrabe.</p>
 <figcaption><cite>Jabberwocky</cite> (first verse). Lewis Carroll, 1832-
98</figcaption>
</figure>

<figure>
  <figcaption>Oil-based paint on canvas. Maria Towle, 1858.</figcaption>
  <img src="castle1858.jpeg" alt="The castle now has two towers and two
walls.">
 </figure>
```

Grouping Content
Element div

- The div element has no special meaning at all. It represents its children. It can be used with the class, lang, and title attributes to mark up semantics common to a group of consecutive elements.
- Example: div used to set the language of two paragraphs at once

```
<div lang="en-GB">
 <p>My other cat, coloured black and white, is a sweetie. He followed
 us to the pool today, walking down the pavement with us. Yesterday
 he apparently visited our neighbours. I wonder if he recognises that
 their flat is a mirror image of ours.</p>
 <p>Hm, I just noticed that in the last paragraph I used British
 English. But I'm supposed to write in American English. So I
 shouldn't say "pavement" or "flat" or "colour"...</p>
</div>
```


Grouping Content
Element main

- The main element represents the main content of the body of a document or application. The main content area consists of content that is directly related to or expands upon the central topic of a document or central functionality of an application.
- The main content area of a document includes content that is unique to that document and excludes content that is repeated across a set of documents such as site navigation links, copyright information, site logos and banners and search forms (unless the document or applications main function is that of a search form).
- Authors must not include more than one main element in a document.

```
<main>
  <h1>Skateboards</h1>
  <p>The skateboard is the way cool kids get around</p>
  <article> </article>
  <article> </article>
</main>
```


Grouping Content
Unordered Lists

- HTML supports ordered, unordered and definition lists.
- **Unordered lists**
  - An unordered list starts with the <ul> tag.
  - Each list item starts with the <li> tag.
  - The list items are marked with bullets (typically small black circles).

```
<p>I have lived in the following countries:</p>
<ul>
 <li>Norway
 <li>Switzerland
 <li>United Kingdom
 <li>United States
</ul>
```


Grouping Content
Ordered Lists

- **Ordered lists**
  - An ordered list starts with the <ol> tag.
  - Each list item starts with the <li> tag.
  - The list items are marked with numbers.

The li element has an ordinal value.

The value attribute, if present, must be a valid integer giving the ordinal value of the list item. If the attribute's value cannot be converted to a number, the attribute must be treated as if it was absent. The attribute has no default value.

Grouping Content
Ordered Lists

- **Ordered lists**
  - The ol element has three specific attributes
    - reversed – number the list backwards
    - start – ordinal value of the first item
    - type – kind of list marker

  The type attribute represents the state given in the cell in the second column of the row whose first cell matches the attribute's value; if none of the cells match, or if the attribute is omitted, then the attribute represents the decimal state

Keywords	State	Description
1 (U+0031)	decimal	Decimal numbers
a (U+0061)	lower-alpha	Lowercase latin alphabet
A (U+0041)	upper-alpha	uppercase latin alphabet
i (U+0069)	lower-roman	lowercase roman numerals
I (U+0049)	upper-roman	uppercase roman numerals

Grouping Content
Ordered Lists

- **Ordered lists** (example 1)

```
<figure>  <figcaption>The top 10 movies of all time</figcaption>
 <ol>
  <li value="10"><cite>Josie and the Pussycats</cite>, 2001</li>
  <li value="9"><cite lang="sh">Црна мачка, бели мачор</cite>,
1998</li>
  <li value="8"><cite>A Bug's Life</cite>, 1998</li>
  <li value="7"><cite>Toy Story</cite>, 1995</li>
  <li value="6"><cite>Monsters, Inc</cite>, 2001</li>
  <li value="5"><cite>Cars</cite>, 2006</li>
  <li value="4"><cite>Toy Story 2</cite>, 1999</li>
  <li value="3"><cite>Finding Nemo</cite>, 2003</li>
  <li value="2"><cite>The Incredibles</cite>, 2004</li>
  <li value="1"><cite>Ratatouille</cite>, 2007</li>
 </ol> </figure>
```


Grouping Content
Ordered Lists

- **Ordered lists** (example 2)

```
<figure>  <figcaption>The top 10 movies of all time</figcaption>
 <ol reversed type="a">
  <li><cite>Josie and the Pussycats</cite>, 2001</li>
  <li><cite lang="sh">Црна мачка, бели мачор</cite>, 1998</li>
  <li><cite>A Bug's Life</cite>, 1998</li>
  <li><cite>Toy Story</cite>, 1995</li>
  <li><cite>Monsters, Inc</cite>, 2001</li>
  <li><cite>Cars</cite>, 2006</li>
  <li><cite>Toy Story 2</cite>, 1999</li>
  <li><cite>Finding Nemo</cite>, 2003</li>
  <li><cite>The Incredibles</cite>, 2004</li>
  <li><cite>Ratatouille</cite>, 2007</li>
</ol></figure>
```


Grouping Content
Ordered Lists

- **Ordered lists** (example 2)

```
<figure>  <figcaption>The top 10 movies of all time</figcaption>
 <ol reversed type="a" start="10">
  <li><cite>Josie and the Pussycats</cite>, 2001</li>
  <li><cite lang="sh">Црна мачка, бели мачор</cite>, 1998</li>
  <li><cite>A Bug's Life</cite>, 1998</li>
  <li><cite>Toy Story</cite>, 1995</li>
  <li><cite>Monsters, Inc</cite>, 2001</li>
  <li><cite>Cars</cite>, 2006</li>
  <li><cite>Toy Story 2</cite>, 1999</li>
  <li><cite>Finding Nemo</cite>, 2003</li>
  <li><cite>The Incredibles</cite>, 2004</li>
  <li><cite>Ratatouille</cite>, 2007</li>
</ol></figure>
```

Grouping Content
Definition Lists

- **Definition lists**
  - Lists of items (terms), with a description of each item (term).
  - A definition list starts with a <dl> tag (definition list).
  - Each term starts with a <dt> tag (definition term).
  - Each description starts with a <dd> tag (definition description).

Name-value groups may be terms and definitions, metadata topics and values, questions and answers, or any other groups of name-value data.

The values within a group are alternatives; multiple paragraphs forming part of the same value must all be given within the same dd element.

Grouping Content
Definition Lists

- **Definition lists** (Example)

```
<dl>
 <dt lang="en-US"> <dfn>color</dfn> </dt>
 <dt lang="en-GB"> <dfn>colour</dfn> </dt>
 <dd> A sensation which (in humans) derives from the ability
of the fine structure of the eye to distinguish three differently
 filtered analyses of a view. </dd>
</dl>
```


Text-level semantics

Text-level semantics
Element a

- If the a element has an href attribute, then it represents a hyperlink (a hypertext anchor) labeled by its contents.
- If the a element has no href attribute, then the element represents a placeholder for where a link might otherwise have been placed, if it had been relevant, consisting of just the element's contents.
- Attributes
  - href - Address of the hyperlink
  - target - Default browsing context for hyperlink navigation and form submission
  - download - Whether to download the resource instead of navigating to it, and its file name if so
  - rel - Relationship between the document containing the hyperlink and the destination resource
  - hreflang - Language of the linked resource
  - type - Hint for the type of the referenced resource

Text-level semantics (Target)

Keyword	Ordinary effect	Effect in an `iframe` with...	
		`sandbox=""`	`sandbox="allow-top-navigation"`
none specified, for links and form submissions	current	current	current
empty string	current	current	current
`_blank`	new	maybe new	maybe new
`_self`	current	current	current
`_parent` if there isn't a parent	current	current	current
`_parent` if parent is also top	parent/top	none	parent/top
`_parent` if there is one and it's not top	parent	none	none
`_top` if top is current	current	current	current
`_top` if top is not current	top	none	top
name that doesn't exist	new	maybe new	maybe new
name that exists and is a descendant	specified descendant	specified descendant	specified descendant
name that exists and is current	current	current	current
name that exists and is an ancestor that is top	specified ancestor	none	specified ancestor/top
name that exists and is an ancestor that is not top	specified ancestor	none	none
other name that exists with common top	specified	none	none
name that exists with different top, if familiar and one permitted sandboxed navigator	specified	specified	specified
name that exists with different top, if familiar but not one permitted sandboxed navigator	specified	none	none
name that exists with different top, not familiar	new	maybe new	maybe new

Text-level semantics
Element a

Link type	Effect on...		Brief description
	`link`	`a` and `area`	
`alternate`	Hyperlink	Hyperlink	Gives alternate representations of the current document.
`author`	Hyperlink	Hyperlink	Gives a link to the author of the current document or artic Remove developer-view styles
`bookmark`	*not allowed*	Hyperlink	Gives the permalink for the nearest ancestor section.
`help`	Hyperlink	Hyperlink	Provides a link to context-sensitive help.
`icon`	External Resource	*not allowed*	Imports an icon to represent the current document.
`license`	Hyperlink	Hyperlink	Indicates that the main content of the current document is covered by the copyright license described by the referenced document.
`next`	Hyperlink	Hyperlink	Indicates that the current document is a part of a series, and that the next document in the series is the referenced document.
`nofollow`	*not allowed*	Annotation	Indicates that the current document's original author or publisher does not endorse the referenced document.
`noreferrer`	*not allowed*	Annotation	Requires that the user agent not send an HTTP `Referer` (sic) header if the user follows the hyperlink.
`prefetch`	External Resource	External Resource	Specifies that the target resource should be preemptively cached.
`prev`	Hyperlink	Hyperlink	Indicates that the current document is a part of a series, and that the previous document in the series is the referenced document.
`search`	Hyperlink	Hyperlink	Gives a link to a resource that can be used to search through the current document and its related pages.
`stylesheet`	External Resource	*not allowed*	Imports a stylesheet.
`tag`	*not allowed*	Hyperlink	Gives a tag (identified by the given address) that applies to the current document.

Text-level semantics
Element a

- Example

```
<nav>
 <ul>
  <li> <a href="http://www.unipi.it" target="_blank">New</a> </li>
  <li> <a href="http://www.unipi.it" target="_self">Self</a> </li>
  <li> <a href="http://www.unipi.it" target="top" rel="tag">Top</a> </li>
 </ul>
</nav>
```


Text-level semantics
Elements em, strong, small

- The em element represents stress emphasis of its contents.
- The strong element represents strong importance, seriousness, or urgency (contents that the user needs to see sooner than other parts of the document) for its contents.
- The small element represents side comments such as small print.

Example

```
<p>Example Corp today announced <em>record profits</em> for the second quarter <small>(Full Disclosure: Foo News is a subsidiary of Example Corp)</small>, <strong>leading to speculation about a third quarter merger with Demo Group </strong>.</p>
```

Text-level semantics
Elements s and cite

- The s element represents contents that are no longer accurate or no longer relevant.

  <p>Buy our Iced Tea and Lemonade!</p>

  <p><s>Recommended retail price: $3.99 per bottle</s></p>

  <p>Now selling for just $2.99 a bottle!</p>

- The cite element represents a reference to a creative work. It must include the title of the work or the name of the author(person, people or organization) or an URL reference, which may be in an abbreviated form as per the conventions used for the addition of citation metadata.

  <p>In the words of <cite>Charles Bukowski</cite> -

Text-level semantics
Element q

- The q element represents some phrasing content quoted from another source.

<p>The W3C page <cite>About W3C</cite> says the W3C's mission is <q cite="http://www.w3.org/Consortium/">To lead the World Wide Web to its full potential by developing protocols and guidelines that ensure long-term growth for the Web</q>. I disagree with this mission.</p>

Text-level semantics
Elements dfn and abbr

- The dfn element represents the defining instance of a term. The paragraph, description list group, or section that is the nearest ancestor of the dfn element must also contain the definition(s) for the term given by the dfn element.
- If the dfn element has a *title* attribute, then the exact value of that attribute is the term being defined.
- The abbr element represents an abbreviation or acronym, optionally with its expansion. The *title* attribute may be used to provide an expansion of the abbreviation. The attribute, if specified, must contain an expansion of the abbreviation, and nothing else.

Text-level semantics
Elements dfn and abbr

- Example

  <p>The <dfn id=gdo><abbr title="Garage Door Opener">GDO</abbr></dfn>

  is a device that allows off-world teams to open the iris.</p>

  <!-- ... later in the document: -->

  <p>Teal'c activated his <abbr title="Garage Door Opener">GDO</abbr>

  and so Hammond ordered the iris to be opened.</p>

Text-level semantics
Elem code, var, samp, kbd, sup, sub

- The code element represents a fragment of computer code.
- The var element represents a variable.
- The samp element represents (sample) output from a program or computing system.
- The kbd element represents user input (typically keyboard input, although it may also be used to represent other input, such as voice commands).
- The sup element represents a superscript and the sub element represents a subscript.

```
<p>The coordinate of the <var>i</var>th point is
<code>(<var>x<sub><var>i</var></sub></var>,
<var>y<sub><var>i</var></sub></var>)</code>.
For example, the 10th point has coordinate
<code>(<var>x<sub>10</sub></var>,
<var>y<sub>10</sub></var>)</code>.</p>
```


Text-level semantics
Elements i, b and mark

- The i element represents a span of text in an alternate voice or mood, or otherwise offset from the normal prose in a manner indicating a different quality of text, such as a taxonomic designation, a technical term, an idiomatic phrase from another language, transliteration, a thought, or a ship name in Western texts.
- The b element represents a span of text to which attention is being drawn for utilitarian purposes without conveying any extra importance and with no implication of an alternate voice or mood, such as key words in a document abstract, product names in a review, actionable words in interactive text-driven software, or an article lede.
- The mark element represents a run of text in one document marked or highlighted for reference purposes, due to its relevance in another context.

Text-level semantics
Elements ruby, rt, rp

- The ruby element allows one or more spans of phrasing content to be marked with ruby annotations. Ruby annotations are short runs of text presented alongside base text, primarily used in East Asian typography as a guide for pronunciation or to include other annotations.
- The rt element marks the ruby text component of a ruby annotation.
- The rp element is used to provide fallback text to be shown by user agents that don't support ruby annotations.

```
<body>
 <ruby> ♥ <rp>: </rp><rt>Heart</rt><rp>.</rp></ruby>
 <ruby> ☘ <rp>: </rp><rt>Shamrock</rt><rp>.</rp></ruby>
 <ruby> ✶ <rp>: </rp><rt>Star</rt><rp>.</rp></ruby>
</body>
```


Text-level semantics
Elements bdi, bdo

- The bdi element represents a span of text that is to be isolated from its surroundings for the purposes of bidirectional text formatting.
- The bdo element represents explicit text directionality formatting control for its children.

```
<ul>
 <li>User <bdi>jcranmer</bdi>: 12 posts.
 <li>User <bdi>hober</bdi>: 5 posts.
 <li>User <bdi>إيان</bdi>: 3 posts.
</ul>
```

Text-level semantics
Elements span, br, wbr

- The span element doesn't mean anything on its own, but can be useful when used together with the global attributes, e.g. class, lang, or dir. It represents its children.
- The br element represents a line break.
- The wbr element represents a line break opportunity.

```
<p>So then he pointed at the tiger and screamed
"there<wbr>is<wbr>no<wbr>way<wbr>you<wbr>are<wbr>ever
<wbr>going<wbr>to<wbr>catch<wbr>me"!</p>
```


Università di Pisa
Francesco Marcelloni

Edits

Università di Pisa
Francesco Marcelloni

Edits
Elements ins, del

- The ins element represents an addition to the document.
- The del element represents a removal from the document.
  - The cite attribute may be used to specify the address of a document that explains the change.
  - The datetime attribute may be used to specify the time and date of the change.

```
<h1>List of <del>fruits</del><ins>colors</ins></h1>
<ul>
<li><del datetime="2009-10-10T23:38-07:00">Apple</del></li>
 <li>Orange</li>
 <li><del>Pear</del></li>
 <li><ins>Teal</ins></li>
 <li><del>Lemon</del><ins>Yellow</ins></li>
 <li><ins>Olive</ins></li>
</ul>
```


Università di Pisa
Francesco Marcelloni

Embedded content

Embedded Content
The img element

- An img element represents an image.
- The <img> element is empty, which means that it contains attributes only and it has no closing tag.
- To display an image on a page, you need to use the src attribute. The value of the src attribute is the URL of the image you want to display on your page.
- The syntax of defining an image:

  <img src="url" alt="sample.txt">
- The browser puts the image where the image tag occurs in the document.
- Except where otherwise specified, the alt attribute must be specified and its value must not be empty; the value must be an appropriate functional replacement for the image.

Embedded Content
The img element

```
<p> <img src= "tower.jpg" width="40" height="40" alt="Tower of Pisa"> </p>
<p> <img src= "tower.jpg" width="90"  height="90" alt="Tower of Pisa"> </p>
<p>
  You can make an image smaller or larger by changing the
  values of the "height" and "width" attributes.
</p>
```


Embedded Content
The img element

- An img is always in one of the following states:
  - Unavailable
    - The user agent has not obtained any image data.
  - Partially available
    - The user agent has obtained some of the image data.
  - Completely available
    - The user agent has obtained all of the image data and at least the image dimensions are available.
  - Broken
    - The user agent has obtained all of the image data that it can, but it cannot even decode the image enough to get the image dimensions (e.g. the image is corrupted, or the format is not supported, or no data could be obtained).

Embedded Content
The image map

- Image maps allow authors to specify regions of an image and assign a specific action to each region (e.g., retrieve a document, run a program, etc.). When the region is activated by the user, the action is executed.
- An image map is created by associating an image with a specification of sensitive geometric areas on the image.
- The map element, in conjunction with an img element and any area element descendants, defines an image map. The element represents its children.
- The name attribute gives the map a name so that it can be referenced. The attribute must be present and must have a non-empty value with no space characters.

Embedded Content
The image map

```
<p>Click on the sun or on one of the planets to get more
information:</p>
<div>
<img src="planets.gif" width="145" height="126" alt="Planets"
usemap="#planetmap">
<map name="planetmap">
  <area shape="rect" coords="0,0,82,126" alt="Sun"
href="sun.html">
  <area shape="circle" coords="90,58,3" alt="Mercury"
href="mercur.html">
  <area shape="circle" coords="124,58,8" alt="Venus"
href="venus.html">
</map>
</div>
```


Embedded Content
The image map

- The <area> tag defines an area inside an image-map (an image-map is an image with clickable areas).
- The area element is always nested inside a <map> tag
- Note: The usemap attribute in the <img> tag is associated with the map element's name attribute, and creates a relationship between the image and the map.

Embedded Content
The image map

- Area attributes

Attribute	Value	Description
alt	*text*	Specifies an alternate text for an area
coords	*coordinates*	Specifies the coordinates of an area
href	*URL*	Specifies the destination of a link in an area
download	download	hyperlink is used for downloading a resource
shape	default rect circle poly	Specifies the shape of an area
target	_blank _parent _self _top	Specifies where to open the linked page specified in the href attribute

Embedded Content
The image map

- Coords attribute

  This attribute specifies the position and shape on the screen. The number and order of values depends on the shape being defined. Possible combinations:

  - rect: left-x, top-y, right-x, bottom-y.
  - circle: center-x, center-y, radius.
  - poly: x1, y1, x2, y2, ..., xN, yN.

    The first x and y coordinate pair and the last should be the same to close the polygon. When these coordinate values are not the same, user agents should infer an additional coordinate pair to close the polygon.

Embedded Content
The iframe element

- The iframe element represents a nested browsing context.
- The src attribute gives the address of a document that the nested browsing context is to contain.
- The srcdoc attribute gives the HTML content that the nested browsing context is to contain. The value of the attribute is the source of an iframe srcdoc document.

```
<iframe srcdoc="<p>Hello world!</p>"
src="demo_iframe_srcdoc.htm"></iframe>
```

- The name attribute, if present, must be a valid browsing context name.
  - The given value is used to name the nested browsing context. When the browsing context is created, if the attribute is present, the browsing context name must be set to the value of this attribute; otherwise, the browsing context name must be set to the empty string.

Embedded Content
The iframe element

```
<body>
<iframe src="html25.html" name="iframe_a">
<p>Your browser does not support iframes.</p>
</iframe>
<a href= "html26.html" target="iframe_a">Planets</a>
<p><b>Note:</b> Because the target of the link matches the name
of the iframe, the link will open in the iframe.</p>
</body>
</html>
```


Embedded Content
The iframe element

- The sandbox attribute, when specified, enables a set of extra restrictions on any content hosted by the iframe.
- The width and height attributes determine the horizontal and vertical dimensions, respectively.

```
<p>Example</p>
<iframe sandbox="allow-same-origin allow-forms allow-scripts" src=
"html26.html" width="400" height="500">
[Your user agent does not support frames or is currently configured
not to display frames.
However, you may visit <a href="foo.html">the related
document.</a>]
```


Embedded Content
The embed element

- The embed element provides an integration point for an external (typically non-HTML) application or interactive content.
- The src attribute gives the address of the resource being embedded. The attribute, if present, must contain a valid non-empty URL potentially surrounded by spaces.
- The type attribute, if present, gives the MIME type by which the plugin to instantiate is selected.

```
<embed src="vowels.swf" width="720" height="500">
```

Embedded Content
The object element

- The object element can represent an external resource, which, depending on the type of the resource, will either be treated as an image, as a nested browsing context, or as an external resource to be processed by a plugin.
- The data attribute, if present, specifies the address of the resource.
- The type attribute, if present, specifies the type of the resource. If present, the attribute must be a valid MIME type.
- The typemustmatch attribute is a boolean attribute whose presence indicates that the resource specified by the data attribute is only to be used if the value of the type attribute and the Content-Type of the aforementioned resource match.

```
<object id="vowels" type="application/x-shockwave-flash"
      data="./wowels.swf" width="720" height="500">
      <param name="movie" value="./wowels.swf">
      </object>
```


Embedded Content
The param element

- The param element defines parameters for plugins invoked by object elements. It does not represent anything on its own.
- The name attribute gives the name of the parameter.
- The value attribute gives the value of the parameter.
- Both attributes must be present. They may have any value.

Embedded Content
The video element

- A video element is used for playing videos or movies, and audio files with captions.
- The video element is a media element whose media data is ostensibly video data, possibly with associated audio data.

```
<video width="320" height="240" src="movie.mp4"
type="video/mp4" controls>
Your browser does not support the video tag.
</video>
```


Embedded Content
The video element

Attribute	Value	Description
autoplay	autoplay	Specifies that the video will start playing as soon as it is ready
controls	controls	Specifies that video controls should be displayed (such as a play/pause button etc).
height	pixels	Sets the height of the video player
loop	loop	Specifies that the video will start over again, every time it is finished
muted	muted	Specifies that the audio output of the video should be muted
poster	URL	Specifies an image to be shown while the video is downloading, or until the user hits the play button
preload	auto metadata none	Specifies if and how the author thinks the video should be loaded when the page loads. Metadata: only metadata. None: the browser should not load the video.
src	URL	Specifies the URL of the video file
width	pixels	Sets the width of the video player

Embedded Content
The source element

- The source element allows authors to specify multiple alternative media resources for media elements. It does not represent anything on its own.
- The src attribute gives the address of the media resource. The value must be a valid non-empty URL potentially surrounded by spaces. This attribute must be present.

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
  Your browser does not support the video tag.
</video>
```


Embedded Content
The audio element

- An audio element represents a sound or audio stream.
- Content may be provided inside the audio element. User agents should not show this content to the user; it is intended for older Web browsers which do not support audio, so that legacy audio plugins can be tried, or to show text to the users of these older browsers informing them of how to access the audio contents.

```
<audio controls>
  <source src= "applause.ogg" type="audio/ogg">
  <source src= "applause.mp3" type="audio/mpeg">
  Your browser does not support the audio element.
</audio>
```


HTML Links

Links
Introduction

- Links are a conceptual construct, created by a, area, and link elements, that represent a connection between two resources, one of which is the current Document. There are two kinds of links in HTML:
- Links to external resources
  - These are links to resources that are to be used to augment the current document, generally automatically processed by the user agent.
- Hyperlinks
  - These are links to other resources that are generally exposed to the user by the user agent so that the user can cause the user agent to navigate to those resources, e.g. to visit them in a browser or download them.

Links
Introduction

- For link elements with an href attribute and a rel attribute, links must be created for the keywords of the rel attribute, as defined for those keywords in the link types section.
- Similarly, for a and area elements with an href attribute and a rel attribute, links must be created for the keywords of the rel attribute as defined for those keywords in the link types section.

Links
Elements <a> and <area>

Attribute	Value	Description
download	filename	Specifies that the target will be downloaded when a user clicks on the hyperlink
href	URL	Specifies the URL of the page the link goes to
hreflang	language_code	Specifies the language of the linked document
rel	alternate author bookmark help license next nofollow noreferrer prefetch prev search tag	Specifies the relationship between the current document and the linked document
target	_blank _parent _self _top framename	Specifies where to open the linked document
type	Mime_type	Specifies the Mime type of the linked document

Links
Link Types (attribute rel)

Rel type	Effect on... link	a and area	Brief description
alternate	Hyperlink	Hyperlink	Gives alternate representations of the current document.
author	Hyperlink	Hyperlink	Gives a link to the author of the current document or article.
bookmark	*not allowed*	Hyperlink	Gives the permalink for the nearest ancestor section.
help	Hyperlink	Hyperlink	Provides a link to context-sensitive help.
icon	External Resource	*not allowed*	Imports an icon to represent the current document.
license	Hyperlink	Hyperlink	Indicates that the main content of the current document is covered by the copyright license described by the referenced document.
next	Hyperlink	Hyperlink	Indicates that the current document is a part of a series, and that the next document in the series is the referenced document.

Links
Link Types (attribute rel)

Rel type	Effect on... link	a and area	Brief description
nofollow	*not allowed*	Annotation	Indicates that the current document's original author or publisher does not endorse the referenced document.
noreferrer	*not allowed*	Annotation	Requires that the user agent not send an HTTP Referer (sic) header if the user follows the hyperlink.
prefetch	External Resource	External Resource	Specifies that the target resource should be preemptively cached.
prev	Hyperlink	Hyperlink	Indicates that the current document is a part of a series, and that the previous document in the series is the referenced document.
search	Hyperlink	Hyperlink	Gives a link to a resource that can be used to search through the current document and its related pages.
stylesheet	External Resource	*not allowed*	Imports a stylesheet.
tag	*not allowed*	Hyperlink	Gives a tag (identified by the given address) that applies to the current document.

Links
Example 1

```
<h1>Table of Contents</h1>
<p><a href="#section1">Introduction</a><br>
<a href="#section2">Some background</a><br>
<a href="#section2.1">On a more personal note</a><br>
...the rest of the table of contents... ...the document body...</p>
<h2><a id="section1">Introduction</a></h2>
<p>...section 1...</p>
<h2><a id="section2">Some background</a></h2>
<p>...section 2...</p>
<h3><a id="section2.1">On a more personal
note</a></h3>
<p>...section 2.1...</p>
```


Links
Example 2

We may achieve the same effect by making the header elements themselves the anchors:

```
<h1> Table of Contents</h1>
  <p>  <a href="#section1">Introduction</a><br>
    <a href="#section2">Some background</a><br>
    <a href="#section2.1">On a more personal note</a><br>
    ...the rest of the table of contents... ...the document  body...
  </p>
  <h2 id="section1"> Introduction</h2>
     <p>...section 1...</p>
  <h2 id="section2">Some background</h2>
     <p>...section 2...</p>
  <h3 id="section2.1"> On a more personal note </h3>
     <p>...section 2.1...</p>
```


Links
Element <link>

Attribute	Value	Description
href	URL	Specifies the URL of the page the link goes to
hreflang	language_code	Specifies the language of the linked document
media	media_query	Specifies what media/device the linked document is optimized for
rel	alternate author bookmark help license next nofollow noreferrer prefetch prev search tag	Specifies the relationship between the current document and the linked document
sizes	value	Sizes of the icons (for rel="icon")
type	Mime_type	Specifies the Mime type of the linked document

Links
Example 3

```
<head>
<link rel="stylesheet" type="text/css" href="theme.css">
<link rel="stylesheet" type="text/css" href="print.css"
media="print">
</head>
```

HTML Tables

HTML Table

- Tables are defined with the <table> tag
  - A table is divided into rows (with the <tr> tag), and each row is divided into data cells (with the <td> tag).
  - The <caption> element defines a table caption.
  - The <caption> element must be inserted immediately after the <table> tag.
    - You can specify only one caption per table.
    - Usually the caption will be centered above the table.
  - Headings in a table are defined with the <th> tag
  - The letters td stands for "table data" which is the content of a data cell.

HTML Table

- A data cell can contain text, images, lists, paragraphs, forms, tables, etc.
- If you do not specify a border attribute the table will be displayed without any borders
  - To display a table with borders, you will have to use the border attribute

Content Model:

Optionally a caption element, followed by zero or more colgroup elements, followed optionally by a thead element, followed optionally by a tfoot element, followed by either zero or more tbody elements or one or more tr elements, followed optionally by a tfoot element (but there can only be one tfoot element child in total), optionally intermixed with one or more script-supporting elements.

HTML Table

```
<head>
 <meta charset="utf-8" />
 <title>The a element</title>
 <style>
 table { border-collapse: collapse; border: solid thick; }
 colgroup, tbody { border: solid medium; }
 td { border: solid thin; height: 1.4em; width: 1.4em; text-align:
  center; padding: 0; }
 </style>
</head>
```

HTML Table

```
<body>
 <h1>Today's Sudoku</h1>
 <table>
  <colgroup><col><col><col>
  <colgroup><col><col><col>
  <colgroup><col><col><col>
  <tbody>
   <tr> <td> 1 <td>   <td> 3 <td> 6 <td>   <td> 4 <td> 7 <td>   <td> 9
   <tr> <td>   <td> 2 <td>   <td>   <td> 9 <td>   <td>   <td> 1 <td>
   <tr> <td> 7 <td>   <td>   <td>   <td>   <td>   <td>   <td>   <td> 6
  <tbody>
   <tr> <td> 2 <td>   <td> 4 <td>   <td> 3 <td>   <td> 9 <td>   <td> 8
   <tr> <td>   <td>   <td>   <td>   <td>   <td>   <td>   <td>   <td>
   <tr> <td> 5 <td>   <td>   <td> 9 <td>   <td> 7 <td>   <td>   <td> 1
```


HTML Table

```
<tbody>
 <tr> <td> 6 <td>   <td>   <td> 5 <td>   <td>   <td>   <td>   <td> 2
 <tr> <td>   <td>   <td>   <td>   <td> 7 <td>   <td>   <td>   <td>
 <tr> <td> 9 <td>   <td>   <td> 8 <td>   <td> 2 <td>   <td>   <td> 5
</table>
</body>
```


HTML Table
The colgroup element

- The colgroup element represents a group of one or more columns in the table that is its parent.
- If the colgroup element contains no col elements, then the element may have a span content attribute specified, whose value must be a valid non-negative integer greater than zero.
- The colgroup element and its span attribute take part in the table model.

HTML Table
The colgroup element

```
<table>
 <colgroup>
  <col span="2" style="background-color:red">
  <col style="background-color:yellow">
 </colgroup>
 <tr> <th>ISBN</th> <th>Title</th> <th>Price</th> </tr>
 <tr> <td>3476896</td> <td>My first HTML</td>
  <td>$53</td> </tr>
 <tr> <td>5869207</td> <td>My first CSS</td>
  <td>$49</td> </tr>
</table>
```

HTML Table (thead, tbody, tfoot)

- The thead, tbody and tfoot elements are used to group the header content, the body content and the footer content in an HTML table.
- Note: <tfoot> must appear before <tbody> within a table, so that a browser can render the foot before receiving all the rows of data.
- This division enables user agents to support scrolling of table bodies independently of the table head and foot.
  - When long tables are printed, the table head and foot information may be repeated on each page that contains table data.

HTML Table (thead, tbody, tfoot)

```
<body>
<table>
 <thead>
  <tr>
   <th>Month</th>
   <th>Savings</th>
  </tr>
 </thead>
 <tfoot>
  <tr>
   <td>Sum</td>
   <td>$180</td>
  </tr>
 </tfoot>
```


HTML Table (thead, tbody, tfoot)

```
<tbody>
  <tr>
   <td>January</td>
   <td>$100</td>
  </tr>
  <tr>
   <td>February</td>
   <td>$80</td>
  </tr>
 </tbody>
</table>
</body>
```

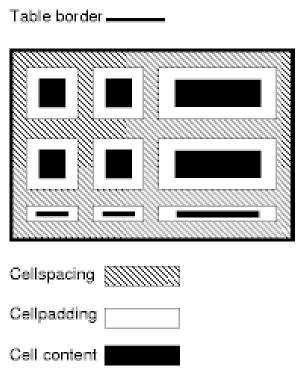

HTML Table (rowspan and colspan)

- The td and th elements may have
  - rowspan
    defines the number of rows a cell should span
  - colspan
    defines the number of columns a cell should span

```
<table width="100%" border="1">
 <tr> <th>Month</th>  <th>Amount</th> <th>Percentage</th></tr>
 <tr> <td> January </td> <td colspan="2">-</td>  </tr>
 <tr> <td> February </td> <td>100</td> <td>10</td> </tr>
</table>
```

HTML Table

- Defining the style of a table



HTML Forms

HTML Forms

- A form is a section of a document containing normal content, markup, special elements called controls and labels on those controls.
- Controls:
  - Buttons
  - Checkboxes
  - Radio buttons
  - Menus
  - Text input
  - File select
  - Hidden control
  - Object controls

HTML Forms

- Users generally "complete" a form by modifying its controls (entering text, selecting menu items, etc.), before submitting the form to an agent for processing (e.g., to a Web server, to a mail server, etc.)
- A control's "control name" is given by its name attribute.
  - The scope of the name attribute for a control within a form element is the form element.

HTML Forms

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
   <title>Pizza Ordering Form</title>
  </head>
<form>
  <p><label>Customer name: <input></label></p>
  <p><label>Telephone: <input type=tel></label></p>
  <p><label>E-mail address: <input type=email></label></p>
```


HTML Forms

```
<fieldset>
 <legend> Pizza Size </legend>
 <p><label> <input type=radio name=size> Small </label></p>
 <p><label> <input type=radio name=size> Medium
 </label></p>
 <p><label> <input type=radio name=size> Large </label></p>
</fieldset>
<fieldset>
 <legend> Pizza Toppings </legend>
 <p><label> <input type=checkbox> Bacon </label></p>
 <p><label> <input type=checkbox> Extra Cheese </label></p>
 <p><label> <input type=checkbox> Onion </label></p>
 <p><label> <input type=checkbox> Mushroom </label></p>
</fieldset>
```


HTML Forms

```
<p><label>Preferred delivery time: <input type=time
 min="11:00" max="21:00" step="900"></label></p>
<p><label>Delivery instructions:
 <textarea></textarea></label></p>
<p><button>Submit order</button></p>
</form>
</html>
```


HTML Forms
Server Communication

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
   <title>Pizza Ordering Form</title>
  </head>
<form method="post"  enctype="application/x-www-form-urlencoded"
    action="https://pizza.example.com/order.php">
 <p><label>Customer name: <input name="custname"></label></p>
 <p><label>Telephone: <input type=tel name="custtel"></label></p>
 <p><label>E-mail address: <input type=email
  name="custemail"></label></p>
```

HTML Forms
Server Communication

```
<fieldset>
 <legend> Pizza Size </legend>
 <p><label> <input type=radio name=size value="small"> Small
 </label></p>
 <p><label> <input type=radio name=size value="medium"> Medium
 </label></p>
 <p><label> <input type=radio name=size value="large"> Large
 </label></p>
</fieldset>
```


HTML Forms
Server Communication

```
<fieldset>
 <legend> Pizza Toppings </legend>
 <p><label> <input type=checkbox name="topping" value="bacon">
 Bacon </label></p>
 <p><label> <input type=checkbox name="topping" value="cheese">
 Extra Cheese </label></p>
 <p><label> <input type=checkbox name="topping" value="onion">
 Onion </label></p>
 <p><label> <input type=checkbox name="topping"
 value="mushroom"> Mushroom </label></p>
</fieldset>
```


HTML Forms
Server Communication

```
<p><label>Preferred delivery time: <input type=time min="11:00"
 max="21:00" step="900" name="delivery"></label></p>
<p><label>Delivery instructions: <textarea
 name="comments"></textarea></label></p>
<p><button>Submit order</button></p>
</form>
```

For example, if the customer entered "Denise Lawrence" as their name, "555-321-8642" as their telephone number, did not specify an e-mail address, asked for a medium-sized pizza, selected the Extra Cheese and Mushroom toppings, entered a delivery time of 7pm, and left the delivery instructions text field blank, the user agent would submit the following to the online Web service:

```
custname=Denise+Lawrence&custtel=555-321-
8624&custemail=&size=medium&topping=cheese&topping=mushroom
&delivery=19%3A00&comments=
```


HTML Forms
Client-side form validation

- Forms can be annotated in such a way that the user agent will check the user's input before the form is submitted.
- The simplest annotation is the required attribute, which can be specified on input elements to indicate that the form is not to be submitted until a value is given.

```
<p><label>Customer name: <input name="custname" required></label></p>
....
<legend> Pizza Size </legend>
 <p><label> <input type=radio name=size required value="small"> Small
</label></p>
 <p><label> <input type=radio name=size required value="medium"> Medium
</label></p>
 <p><label> <input type=radio name=size required value="large"> Large
</label></p>
 </fieldset>
..
<p><label>Preferred delivery time: <input type=time min="11:00" max="21:00"
step="900" name="delivery" required></label></p>
```

HTML Forms
Client-side form validation

- It is also possible to limit the length of the input, using the maxlength attribute.

<p><label>Delivery instructions: <textarea name="comments" maxlength=1000></textarea></label></p>

HTML Forms

- The form element acts as a container for controls, by specifying:
  - The layout of the form (given by the contents of the element).
  - The program that will handle the completed and submitted form (the action attribute). The receiving program must be able to parse name/value pairs in order to make use of them.
  - The method by which user data will be sent to the server (the method attribute).
  - A character encoding that must be accepted by the server in order to handle this form (the accept-charset attribute). User agents may advise the user of the value of the accept-charset attribute and/or restrict the user's ability to enter unrecognized characters.

HTML Forms

- In general, a control's "initial value" may be specified with the control element's value attribute.
- However, the initial value of a textarea element is given by its contents
- A control's initial value does not change. Thus, when a form is reset, each control's current value is reset to its initial value

HTML Forms

- Attributes of the form element

Attribute	**Value**	**Description**
action	*URL*	Specifies where to send the form-data when a form is submitted
accept-charset	*charset*	Specifies the character-sets the server can handle for form-data
autocomplete	*on,off*	Specifies if the autofill is on or off
enctype	application/x-www-form-urlencoded multipart/form-data text/plain	Specifies how form-data should be encoded before sending it to a server (multipart/form-data text/plain is only for POST)
method	get post	Specifies how to send form-data
name	*name*	Specifies the name for a form
novalidate	*on,off*	Specifies whether the form has to be validated or not
target	_*blank* new window _*self* same frame _*top* full body of the window *framename* in a named iframe	The target attribute specifies a name or a keyword that indicates where to display the response that is received after submitting the form.

HTML Forms

- If the method is "get", the user agent takes the value of action, appends a ? to it, then appends the form data set, encoded using the application/x-www-form-urlencoded content type. The user agent then traverses the link to this URI. In this scenario, form data are restricted to ASCII codes.
- If the method is "post", the user agent conducts an HTTP post transaction using the value of the action attribute and a message created according to the content type specified by the enctype attribute.

HTML Forms: Control Types

- Buttons types
  - submit: When activated, a submit button submits a form. A form may contain more than one submit button.
  - reset: When activated, a reset button resets all controls to their initial values.
  - buttons: Buttons have no default behavior.
    - Each button may have client-side scripts associated with the element's event attributes.

HTML Forms: Control Types

- Checkboxes
  - Checkboxes (and radio buttons) are on/off switches that may be toggled by the user. A switch is "on" when the control element's checked attribute is set.
  - When a form is submitted, only "on" checkbox controls can become successful.
  - Several checkboxes in a form may share the same control name. Thus, for example, checkboxes allow users to select several values for the same property.
  - The input element is used to create a checkbox control.

HTML Forms: Control Types

- Radio Buttons
  - Radio buttons are like checkboxes except that when several share the same control name, they are mutually exclusive: when one is switched "on", all others with the same name are switched "off".
  - The input element is used to create a radio button control.
- Menus
  - Menus offer users options from which to choose.
  - The select element creates a menu, in combination with the optgroup and option elements.

HTML Forms: Control Types

- Text input
  - Authors may create two types of controls that allow users to input text.
  - The input element creates a single-line input control and the textarea element creates a multi-line input control.
  - In both cases, the input text becomes the control's current value
- File Select
  - This control type allows the user to select files so that their contents may be submitted with a form.
  - The input element is used to create a file select control.

HTML Forms: Control Types

- How are the control types implemented?
  - <input> element
  - <button> element
  - <select> element
  - <datalist> element
  - <optgroup> element
  - <option> element
  - <textarea> element
  - <keygen> element
  - <output> element
  - <progress> element
  - <meter> element
  - <fieldset> element
  - <legend> element

HTML Forms
Input Element

The type attribute of the Input Element

Keyword	State	Data type	Control type
hidden	Hidden	An arbitrary string	n/a
text	Text	Text with no line breaks	A text field
search	Search	Text with no line breaks	Search field
tel	Telephone	Text with no line breaks	A text field
url	URL	An absolute URL	A text field

HTML Forms
Input Element

The type attribute

Keyword	State	Data type	Control type
email	E-mail	An e-mail address or list of e-mail addresses	A text field
password	Password	Text with no line breaks (sensitive information)	A text field that obscures data entry
date	Date	A date (year, month, day) with no time zone	A date control
time	Time	A time (hour, minute, seconds, fractional seconds) with no time zone	A time control

HTML Forms
Input Element

The type attribute

Keyword	State	Data type	Control type
number	Number	A numerical value	A text field or spinner control
range	Range	A numerical value, with the extra semantic that the exact value is not important	A slider control or similar
color	Color	An sRGB color with 8-bit red, green, and blue components	A color well
checkbox	Checkbox	A set of zero or more values from a predefined list	A checkbox

HTML Forms
Input Element

The type attribute

Keyword	State	Data type	Control type
radio	Radio Button	An enumerated value	A radio button
file	File Upload	Zero or more files each with a MIME type and optionally a file name	A label and a button
submit	Submit Button	An enumerated value, with the extra semantic that it must be the last value selected and initiates form submission	A button
image	Image Button	A coordinate, relative to a particular image's size, with the extra semantic that it must be the last value selected and initiates form submission	Either a clickable image, or a button

HTML Forms
Input Element

The type attribute

Keyword	State	Data type	Control type
reset	Reset Button	n/a	A button
button	Button	n/a	A button

HTML Forms
Input Element

Input element attributes.

Different attributes depending on the specific value of attribute type.

List of attributes and their use can be found in the HTML5 specification

http://www.w3.org/TR/html5/forms.html#the-input-element

HTML Forms
Input Element

Some examples of specific attributes

- The list attribute is used to identify an element that lists predefined options suggested to the user.
- If present, its value must be the ID of a datalist element in the same document.

```
<label>Homepage: <input name=hp type=url list=hpurls></label>
<datalist id=hpurls>
 <option value="http://www.google.com/" label="Google">
 <option value="http://www.reddit.com/" label="Reddit">
</datalist>
```


HTML Forms
Input Element

Some examples of specific attributes

- The placeholder attribute represents a short hint (a word or short phrase) intended to aid the user with data entry when the control has no value.

```
<fieldset>
 <legend>Mail Account</legend>
 <p><label>Name: <input type="text" name="fullname"
placeholder="John Ratzenberger"></label></p>
 <p><label>Address: <input type="email" name="address"
placeholder="john@example.net"></label></p>
 <p><label>Password: <input type="password"
name="password"></label></p>
 <p><label>Description: <input type="text" name="desc"
placeholder="My Email Account"></label></p>
</fieldset>
```


HTML Forms
Input Element

Some examples of specific attributes

- The readonly attribute is a boolean attribute that controls whether or not the user can edit the form control.
- The required attribute is a boolean attribute. When specified, the element is required.
- The pattern attribute specifies a regular expression against which the control's value, or, when the multiple attribute applies and is set, the control's values, are to be checked.
- The min and max attributes indicate the allowed range of values for the element.
- The step attribute indicates the granularity that is expected (and required) of the value, by limiting the allowed values.

HTML Forms
Input Element

Example:

```
<form action="products.php" method="post" enctype="multipart/form-data">
 <table>
  <tr> <th> Product ID <th> Product name <th> Price <th> Action
  <tr>
   <td> <input readonly="readonly" name="1.pid" value="H412">
   <td> <input required="required" name="1.pname" value="Floor lamp
  Ulke">
   <td> $<input required="required" type="number" min="0" step="0.01"
  name="1.pprice" value="49.99">
   <td> <button formnovalidate="formnovalidate" name="action"
  value="delete:1">Delete</button>
  <tr>
```

HTML Forms
Input Element

```
<td> <input readonly="readonly" name="2.pid" value="FG28">
  <td> <input required="required" name="2.pname" value="Table lamp
  Ulke">
  <td> $<input required="required" type="number" min="0" step="0.01"
  name="2.pprice" value="24.99">
  <td> <button formnovalidate="formnovalidate" name="action"
  value="delete:2">Delete</button>
```


HTML Forms
Input Element

```
<tr>
  <td> <input required="required" name="3.pid" value="" pattern="[A-Z0-
  9]+">
  <td> <input required="required" name="3.pname" value="">
  <td> $<input required="required" type="number" min="0" step="0.01"
  name="3.pprice" value="">
  <td> <button formnovalidate="formnovalidate" name="action"
  value="delete:3">Delete</button>
 </table>
<p> <button formnovalidate="formnovalidate" name="action"
  value="add">Add</button> </p>
 <p> <button name="action" value="update">Save</button> </p>
</form>
```


HTML Forms
Input Element

Some examples of specific attributes

- The multiple attribute is a boolean attribute that indicates whether the user is to be allowed to specify more than one value.
  - The multiple attribute works with the following input types: email, and file.
  - For <input type="file">: to select multiple files, hold down the CTRL or SHIFT key while selecting.
  - For <input type="email">: separate each email with a comma, like: mail@example.com, mail2@example.com, mail3@example.com in the email field.

```
<form action="demo_form.php">
  Select images: <input type="file" name="img" multiple>
  <input type="submit">
</form>
```


HTML Forms
Button Element

- Buttons created with the button element function just like buttons created with the input element, but they offer richer rendering possibilities:
  - the button element may have content. For example, a button element that contains an image functions like and may resemble an input element whose type is set to "image", but the button element type allows content.
- Always specify the type attribute for the button: submit, reset, button (does nothing).

Università di Pisa — Francesco Marcelloni

HTML Forms
Button Element

Main attributes of the button element

- autofocus: allows the author to indicate that a control is to be focused as soon as the page is loaded, allowing the user to just start typing without having to manually focus the main control.
- disabled
- form: to explicitly associate the button element with its form owner (the value is the ID of the form)
- formnovalidate: indicates that the form is not to be validated during submission
- value: The value attribute gives the element's value for the purposes of form submission. The element's value is the value of the element's value attribute, if there is one, or the empty string otherwise.

209

Università di Pisa — Francesco Marcelloni

HTML Forms
Button Element

```
<form action="http://somesite.com/prog/adduser"  method="post">
   <p>
     First name: <input type="text"  name="firstname"><br>
     Last name: <input type="text"   name="lastname"><br>
     email: <input type="text" name="email"><br>
    <input type="radio" name="sex" value="Male"> Male<br>
     <input type="radio" name="sex" value="Female"> Female<br>
     <button name="submit"  value="submit" type="submit"> Send
      <img src="ok.png"  width="40" heigh="40" alt="OK"></button>
      <button name="reset"  type="reset">Reset
      <img src="reset.jpg"  width="40" heigh="40"
 alt="oops"></button>
   </p>
  </form>
```

210

Università di Pisa — Francesco Marcelloni

HTML Forms
select, optgroup and option elements

- The select element creates a menu.
  - Each choice offered by the menu is represented by an option element.
  - A select element must contain at least one option element.
- The optgroup element allows authors to group choices logically. The label value gives the name of the group.
  - Helpful when the user must choose from a long list of options; groups of related choices are easier to grasp and remember than a single long list of options.
- The option element represents an option in a select element or as part of a list of suggestions in a datalist element. The selected attribute represents the default selectedness of the element

211

Università di Pisa — Francesco Marcelloni

HTML Forms
select, optgroup and option elements

<select> attributes

- disabled – specifies that a drop-down list should be disabled
- multiple – specifies that multiple options can be selected
- size – gives the number of options to show to the user
- required – the user is required to select a value

212

HTML Forms
select, optgroup and option elements

- When rendering a menu choice, user agents should use the value of the label attribute of the option element as the choice.
  - If this attribute is not specified, user agents should use the contents of the option element.
- The label attribute of the optgroup element specifies the label for a group of choices
- Only selected options will be successful (using the control name "c" in the following example).
- When no options are selected, the control is not successful and neither the name nor any values are submitted to the server when the form is submitted.

HTML Forms
select, optgroup and option elements

- Example of the <optgroup> element

```
<form action="http://www.watch.com/watch.php" method="get">
 <p>Which course would you like to watch today?
 <p><label>Course:
  <select name="c">
   <optgroup label="8.01 Physics I: Classical Mechanics">
    <option value="8.01.1">Lecture 01: Powers of Ten
    <option disabled value="8.01.2">Lecture 02: 1D Kinematics
    <option value="8.01.3">Lecture 03: Vectors
   <optgroup disabled label="8.02 Electricity and Magnestism">
```


HTML Forms
select, optgroup and option elements

```
<option value="8.02.1">Lecture 01: What holds our world together?
    <option value="8.02.2">Lecture 02: Electric Field
    <option value="8.02.3">Lecture 03: Electric Flux
   <optgroup label="8.03 Physics III: Vibrations and Waves">
    <option value="8.03.1">Lecture 01: Periodic Phenomenon
    <option value="8.03.2">Lecture 02: Beats
    <option value="8.03.3">Lecture 03: Forced Oscillations with
   Damping
  </select>
 </label>
 <p><input type="submit" value="Play">
</form>
```


HTML Forms
Textarea element

- The textarea element creates a multi-line text input control. User agents should use the contents of this element as the initial value of the control and should render this text initially.
- The cols and rows attribute specify the expected maximum number of characters per line and the number of lines to show, respectively.
- The maxlength attribute specifies the allowed maximum length.

HTML Forms
Textarea element

<form action="http://www.elaborate.com/elab.php" method="get">
<p>
<textarea name="thetext" rows="10" cols="80" maxlength="20">
</textarea>
<input type="submit" value="Send">
<input type="reset" value="Reset"></p>
</form>

HTML Forms
Label element

- Some form controls automatically have labels associated with them (press buttons) while most do not (text fields, checkboxes and radio buttons, and menus).
- For those controls that have implicit labels, user agents should use the value of the “value” attribute as the label string.
- The label element is used to specify labels for controls that do not have implicit labels.

HTML Forms
Label element

- The label element represents a caption in a user interface.
- The caption can be associated with a specific form control, known as the label element's labeled control, either using the for attribute, or by putting the form control inside the label element itself.
- The for attribute explicitly associates the label being defined with another control. When present, the value of this attribute must be the same as the value of the id attribute of some other control in the same document. When absent, the label being defined is associated with the element’s contents.
- More than one label may be associated with the same control by creating multiple references via the for attribute.
- When a LABEL element receives focus, it passes the focus on to its associated control.

HTML Forms
Label element

<form action="http://somesite.com/prog/adduser" method="post">
<p><label for="firstname">First name:</label> <input type="text" id="firstname">
<label for="lastname">Last name:</label> <input type="text" id= "lastname">
<label for="email">email:</label> <input type="text" id= "email">
<input type="radio" name="sex" value="Male"> Male
<input type="radio" name="sex" value="Female"> Female
<input type="submit" value="Send"> <input type="reset"></p>
</form>

HTML Forms
Label element

- To associate a label with another control implicitly, the control element must be within the contents of the label element.
  - In this case, the label may only contain one control element.
  - The label itself may be positioned before or after the associated control.

```
<form action="..." method="post">
<p><label>First Name <input type="text"
  name="firstname"></label>
<label><input type="text" name="lastname"> Last
  Name</label></p>
</form>
```


HTML Forms
Focus to an element

- There are several ways to give focus to an element:
  - Designate the element with a pointing device.
  - Navigate from one element to the next with the keyboard.
    - The document's author may define a *tabbing order* that specifies the order in which elements will receive focus if the user navigates the document with the keyboard. Once selected, an element may be activated by some other key sequence.
  - Select an element through an *access key* (sometimes called "keyboard shortcut" or "keyboard accelerator").

HTML Forms
Tabbing navigation

tabindex = *number*

- The tabindex attribute specifies the position of the current element in the tabbing order for the current document.
  - Number is between 0 and 32767.
  - Values need not be sequential nor must they begin with any particular value.
  - The tabbing order may include elements nested within other elements.
  - The following elements support the tabindex attribute: a, area, button, input, object, select, and textarea.

HTML Forms
Tabbing navigation

- Elements should be navigated by user agents according to the following rules:

1. First, elements that support the tabindex attribute and assign a positive value to it are navigated:
   - from the element with the lowest tabindex value to the element with the highest value in the order they appear in the character stream if have identical tabindex values
2. Then, elements that do not support the tabindex attribute or support it and assign it a value of "0" are navigated next. These elements are navigated in the the order they appear in the character stream.
3. Elements that are disabled do not participate in the tabbing order.

HTML Forms
Tabbing navigation

<p>Go to the W3C Web site. ...some more... <button type="button" name=

"get-database" tabindex="18" onclick="get-database">Get the

current database.</button> ...some more...</p>

<form action="..." method="post">

<p><input tabindex="1" type="text" name="field1">

<input tabindex="1" type="text" name="field2"> <input tabindex=

"2" type="submit" name="submit"></p>

</form>

HTML Forms
Access key

accesskey = *character*

- This attribute assigns an access key to an element. An access key is a single character from the document character set.
- Pressing an access key assigned to an element gives focus to the element. The action that occurs when an element receives focus depends on the element.
- The following elements support the accesskey attribute: a, area, button, input, label, legend, and textarea

HTML Forms
Access key

<form action="..." method="post">

<p><label for="fuser" accesskey="u">User Name</label>

<input type="text" name="user" id="fuser"></p>

</form>

- The invocation of access keys depends on the underlying system. For instance, on machines running MS Windows, one generally has to press the "alt" key in addition to the access key. On Apple systems, one generally has to press the "cmd" key in addition to the access key.

HTML Forms
Disabled controls

Disabled

- When set for a form control, this boolean attribute disables the control for user input.
  - Disabled controls do not receive focus
  - Disabled controls are skipped in tabbing navigation
  - Disabled controls cannot be successful.
- The following elements support the disabled attribute: button, input, optgroup, option, select and textarea.

<input disabled name="fred" value="stone">

HTML Forms readonly controls

Readonly

- When set for a form control, this boolean attribute prohibits changes to the control.
  - Read-only elements receive focus but cannot be modified by the user.
  - Read-only elements are included in tabbing navigation
  - Read-only elements may be successful
- The following elements support the readonly attribute: input and textarea.

  <input readonly name="fred" value="stone">

HTML Forms Form submission

- A *successful control* is "valid" for submission. Every successful control has its control name paired with its current value as part of the submitted form data set.
- Note that

  Controls that are disabled cannot be successful.
  - Submit button: If a form contains more than one submit button, only the activated submit button is successful.
  - Checkboxes: All "on" checkboxes may be successful.
  - Radio Buttons: For radio buttons that share the same value of the name attribute, only the "on" radio button may be successful.

HTML Forms Form submission

- Menus: the control name is provided by a select element and values are provided by option elements. Only selected options may be successful. When no options are selected, the control is not successful and neither the name nor any values are submitted to the server when the form is submitted.
- File select: The current value of a file select is a list of one or more file names. Upon submission of the form, the *contents* of each file are submitted with the rest of the form data. The file contents are packaged according to the form's content type.
- Object control: The current value of an object control is determined by the object's implementation

HTML Forms Form submission

- If a control does not have a current value when the form is submitted, user agents are not required to treat it as a successful control.
- Furthermore, user agents should not consider the following controls successful:
  - Reset buttons.
  - object elements whose declare attribute has been set.
- Hidden controls and controls that are not rendered because of style sheet settings may still be successful. For example

HTML Forms
Form submission

```
<form action="..." method="post">
<p><input type="password" style="display:none"
  name=
"invisible-password" value="mypassword"></p>
</form>
```

will still cause a value to be paired with the name "invisible-password" and submitted with the form

- Note: this mechanism affords only light security protection. Although the password is masked by user agents from casual observers, it is transmitted to the server in clear text.

HTML Forms
Processing form data

The user agent processes a form as follows:

1. **Identify the successful controls**
2. **Build a form data set**

   A *form data set* is a sequence of control-name /current-value pairs constructed from successful controls
3. **Encode the form data set**

   The form data set is then encoded according to the content type specified by the enctype attribute of the FORM element.
4. **Submit the encoded form data set**
5. **Send to the Processing agent**

   The encoded data is sent to the processing agent designated by the action attribute using the protocol specified by the method attribute.

HTML Forms
Processing form data

HTML 5 user agents must support the established conventions in the following cases:

- Method = "get" and action = HTTP URI
  - The user agent
    - takes the value of action,
    - appends a '?' to it,
    - appends the form data set, encoded using the "application/x-www-form-urlencoded" content type.
  - The user agent then traverses the link to the specified URI.
    - In this scenario, form data are restricted to ASCII codes.

HTML Forms
Processing form data

Example method "get"

```
<form action="/find.php" method=get>
 <input type=text name=t>
 <input type=search name=q>
 <input type=submit>
</form>
```

- The user agent will load

  /find.php?t=cats&q=fur.

HTML Forms
Processing form data

- Method = "post" and action = HTTP URI
  - The user agent conducts an HTTP "post" transaction using the value of the action attribute and a message created according to the content type specified by the enctype attribute.
- For any other value of action or method, behavior is unspecified.

HTML Forms
Form content type

- The enctype attribute of the FORM element specifies the content type used to encode the form data set for submission to the server.
- **application/x-www-form-urlencoded**

  This is the default content type. Forms submitted with this content type must be encoded as follows:
  - Control names and values are escaped. Space characters are replaced by '+', and then reserved characters are escaped: Non-alphanumeric characters are replaced by '%HH', a percent sign and two hexadecimal digits representing the ASCII code of the character. Line breaks are represented as "CR LF" pairs (i.e., '%0D%0A').
  - The control names/values are listed in the order they appear in the document. The name is separated from the value by '=' and name/value pairs are separated from each other by '&'.

HTML Forms
Form content type

- **multipart/form-data**

  This content type should be used for submitting forms that contain files, non-ASCII data, and binary data.

  A "multipart/form-data" message contains a series of parts, each representing a successful control. The parts are sent to the processing agent in the same order the corresponding controls appear in the document stream.

  Each part has an optional "Content-Type" header that defaults to "text/plain". User agents should supply the "Content-Type" header, accompanied by a "charset" parameter.

HTML Forms
Form content type

- **multipart/form-data** (continued)

  Each part is expected to contain:
  - a "Content-Disposition" header whose value is "form-data".
  - a name attribute specifying the control name of the corresponding control.

  Thus, for example, for a control named "mycontrol", the corresponding part would be specified:

  Content-Disposition: form-data; name="mycontrol"

  As with all MIME transmissions, "CR LF" (i.e., '%0D%0A') is used to separate lines of data.

HTML Forms
Form content type

Suppose we have the following form:

<form action="http://server.com/cgi/handle" enctype=

"multipart/form-data" method="post">

<p>What is your name? <input type="text" name="submit-name">

What files are you sending? <input type="file" name="files">

<input type="submit" value="Send"> <input type="reset"></p>

</form>

HTML Forms
Form content type

The user agent might send back:

Content-Type: multipart/form-data; boundary=AaB03x

--AaB03x

Content-Disposition: form-data; name="submit-name"

Larry

--AaB03x

Content-Disposition: form-data; name="files"; filename="file1.txt"

Content-Type: text/plain

... contents of file1.txt ...

--AaB03x--

HTML Forms
Form content type

If the user selected a second (image) file "file2.gif":

Content-Type: multipart/form-data; boundary=AaB03x

--AaB03x

Content-Disposition: form-data; name="submit-name"

Larry

--AaB03x

Content-Disposition: form-data; name="files"

Content-Type: multipart/mixed; boundary=BbC04y

HTML Forms
Form content type

--BbC04y

Content-Disposition: file; filename="file1.txt"

Content-Type: text/plain

... contents of file1.txt ...

--BbC04y

Content-Disposition: file; filename="file2.gif"

Content-Type: image/gif

Content-Transfer-Encoding: binary

...contents of file2.gif...

--BbC04y--

--AaB03x--

Checking Forms with validation

- Form validation is an optimization because it alone is not sufficient to guarantee that forms submitted to the server are correct and valid.
- It is an optimization because it is designed to help a web application fail fast. In other words, it is better to notify a user that a page contains invalid form controls right inside the page, using the browser's built-in processing.

Checking Forms with validation

- HTML5 does introduce eight handy ways to enforce correctness on form control entry.
- Object ValidityState allows accessing their status

  var valCheck = document.myForm.myInput.validity;

  valCheck contains a reference to the ValidityState object of the form element named myInput.

  valCheck.valid

  returns a Boolean value which informs us whether or not all validity constraints are currently met on this particular form control.

  If all eight constraints are passing, the valid flag will be true. Otherwise, if any of the validity constraints fail, the valid attribute will be false.

Checking Forms Validity constraints (1)

The valCheck.valueMissing Constraint

- Purpose: Ensure that some value is set on this form control
- Usage: Set the required attribute on the form control to true
- Usage example: <input type="text" name="myText" required>
- Details: If the required attribute is set on a form control, the control will be in an invalid state unless the user or a programmatic call sets some value to the field. For example, a blank text field will fail a required check, but will pass as soon as any text is entered. When blank, the valueMissing will return true.

Checking Forms Validity constraints (2)

The valCheck.typeMismatch Constraint

- Purpose: Guarantee that the type of the value matches expectations (number, email, URL, and so on)
- Usage: Specify one of the appropriate type attributes on the form control
- Usage example: <input type="email" name="myEmail">
- Details: Special form control types aren't just for customized phone keyboards! If your browser can determine that the value entered into a form control doesn't conform to the rules for that type—for example, an email address without an @ symbol—the browser can flag this control as having a type mismatch. Another example would be a number field that cannot parse to a valid number. In either case, the typeMismatch will return true.

Checking Forms
Validity constraints (3)

The valCheck.patternMismatch Constraint

- Purpose: Enforce any pattern rule set on a form control which details specific valid formats
- Usage: Set the pattern attribute on the form control with the appropriate pattern
- Usage example: <input type="text" name="creditcardnumber" pattern="[0-9]{16}" title="A credit card number is 16 digits with no spaces or dashes">
- Details: The pattern attribute gives developers a powerful and flexible way of enforcing a regular expression pattern on the value of a form control. When a pattern is set on a control, the patternMismatch will return true whenever the value does not conform to the rules of the pattern. To assist users and assistive technology, you should set the title on any pattern-controlled field to describe the rules of the format.

Checking Forms
Validity constraints (4)

The valCheck.tooLong Constraint

- Purpose: Make sure that a value does not contain too many characters
- Usage: Put a maxLength attribute on the form control
- Usage example: <input type="text" name="limitedText" maxLength="140">
- Details: This humorously-named constraint will return true if the value length exceeds the maxLength. While form controls will generally try to enforce the maximum length during user entry, certain situations including programmatic settings can cause the value to exceed the maximum.

Checking Forms
Validity constraints (5)

The valCheck.rangeUnderflow Constraint

- Purpose: Enforce the minimum value of a numeric control
- Usage: Set a min attribute with the minimum allowed value
- Usage example: <input type="range" name="ageCheck" min="18">
- Details: In any form controls that do numeric-range checking, it is possible for the value to get temporarily set below the allowable range. In these cases, the ValidityState will return true for the rangeUnderflow field.

Checking Forms
Validity constraints (6)

The valCheck.rangeOverflow Constraint

- Purpose: Enforce the maximum value of a numeric control
- Usage: Set a max attribute with the maximum allowed value
- Usage example: <input type="range" name="kidAgeCheck" max="12">
- Details: Similar to its counterpart rangeUnderflow, this validity constraint will return true if the value of a form control becomes greater than the max attribute.

Checking Forms Validity constraints (7)

The valCheck.stepMismatch Constraint

- Purpose: Guarantee that a value conforms to the combination of min, max, and step
- Usage: Set a step attribute to specify the granular steps of a numeric value
- Usage example: <input type="range" name="confidenceLevel" min="0" max="100" step="5">
- Details: This constraint enforces the sanity of the combinations of min, max, and step. Specifically, the current value must be a multiple of the step added to the minimum value. For example, a range from 0 to 100 with steps at every 5 would not allow a value of 17 without stepMismatch returning true.

Checking Forms Validity constraints (8)

The valCheck.customError Constraint

- Purpose: Handle errors explicitly calculated and set by the application code
- Usage: Call setCustomValidity(message) to put a form control into the customError state
- Usage example: passwordConfirmationField.setCustomValidity("Password values do not match.");
- Details: For those cases where the built-in validity checks don't apply, the custom validity errors can suffice. Application code should set a custom validity message whenever a field does not conform to semantic rules.

Checking Forms Validity fields and functions

The willValidate Attribute of the HTMLInputElement

- Indicates whether validation will be checked on this form control at all. If any of the above constraints—e.g. the required attribute, pattern attribute, etc.—are set on the control, the willValidate field will let you know that validation checking is going to be enforced.

The checkValidity Function of the HTMLInputElement

- The checkValidity function allows you to check validation on the form without any explicit user input. Normally, a form's validation is checked whenever the user or script code submits the form. This function allows validation to be done at any time.

Checking Forms Validity fields and functions

The validationMessage Attribute of the HTMLInputElement

- lets you query programmatically a localized error message that the browser would display based on the current state of validation. For example, if a required field has no value, the browser might present an error message to the user that "This field requires a value."

Checking Forms Validity feedback

- The specification does not dictate the terms of how the user interface is updated to present an error message, and existing implementations differ fairly significantly.
- Any form in an invalid state will deliver an invalid event.
- This event can be ignored, observed, or even cancelled.
- To add an event handler to a field which will receive this notification

Checking Forms Event Handler for Invalid Events

```
function invalidHandler(evt) {
  var validity = evt.target.validity;
      // check the validity to see if a particular constraint failed
  if (validity.valueMissing) {
      // present a UI to the user indicating that the field is missing a value
}
    // perhaps check additional constraints here…
    // If you do not want the browser to provide default validation feedback,
    // cancel the event as shown here
  evt.preventDefault();
}

      // register an event listener for "invalid" events
myField.addEventListener("invalid", invalidHandler, false);
```


Checking Forms Event Handler for Invalid Events (Ex.)

```
function invalidHandler(evt) {
      // find the label for this form control
  var label =
  evt.target.parentElement.getElementsByTagName("label")[0];
      // set the label's text color to red
  label.style.color = 'red';
      // stop the event from propagating higher
  evt.stopPropagation();
      // stop the browser's default handling of the validation error
  evt.preventDefault();
}
      // register an event listener for "invalid" events
myField.addEventListener("invalid", invalidHandler, false);
```


CANVAS

Canvas

- The canvas element provides scripts with a resolution-dependent bitmap canvas, which can be used for rendering graphs, game graphics, art, or other visual images on the fly.
- Authors should not use the canvas element in a document when a more suitable element is available.
- Two attributes to control the size of the element's bitmap: width and height.
- The HTML5 Canvas API supports the same two-dimensional drawing operations that most modern operating systems and frameworks support.
  - To programmatically use a canvas, you have to first get its context. You can then perform actions on the context and finally apply those actions to the context.

Canvas

- Canvas definition

```
<canvas>
  Update your browser to enjoy canvas!
</canvas>
```

Alternative text if canvas not supported

CSS can be applied to the canvas element itself to add borders, padding, margins, etc.

Additionally, some CSS values are inherited by the contents of the canvas; fonts are a good example, as fonts drawn into a canvas default to the settings of the canvas element itself.

Furthermore, properties set on the context used in canvas operations follow the CSS syntax. Colors and fonts, for example, use the same notation on the context that they use throughout any HTML or CSS document.

Canvas
Checking for support

```
try {
        document.createElement("canvas").getContext("2d");
        } catch (e) {
        alert("HTML5 Canvas is not supported in your browser.“);
}
```


Canvas
Draw a Diagonal line

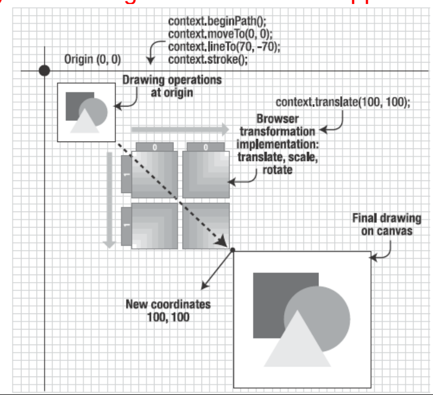
```
<!DOCTYPE html>
<html>
 <title>Diagonal line example</title>
 <canvas id="diagonal" style="border: 1px solid;"  width="200"
height="200"> </canvas>
 <script>
    function drawDiagonal() {
      // Get the canvas element and its drawing context
      var canvas = document.getElementById('diagonal');
      var context = canvas.getContext('2d');
```

Canvas
Draw a Diagonal line

```
// Create a path in absolute coordinates
        context.beginPath();       //start a new path
        context.moveTo(70, 140);  //move the context point
        context.lineTo(140, 70);  //draw straight lines
        context.lineWidth = 15;
        context.strokeStyle = '#ff0000'; //set the color
        context.lineCap = 'round';  //butt, round, square
        // Stroke the line onto the canvas (makes the line visible)
        context.stroke();
     }
     window.addEventListener("load", drawDiagonal, true);
 </script>
</html>
```


Canvas
Applying Transformations

- A key recommendation for reusable code is that you usually want to draw at the origin (coordinate 0,0) and apply transformations—scale, translate, rotate, and so forth—to modify your drawing code into its final appearance



Canvas
Draw a Diagonal line

```
function drawDiagonal() {
var canvas = document.getElementById('diagonal');
var context = canvas.getContext('2d');
// Save a copy of the current drawing state
context.save();
// Move the drawing context to the right, and down
context.translate(70, 140);
// Draw the same line as before, but using the origin as a start
context.beginPath();
context.moveTo(0, 0);
context.lineTo(70, -70);
context.stroke();
// Restore the old drawing state
context.restore();
}
```


Canvas
Draw a Diagonal line

Why context.save()?

- If you do not save the state, the modifications you're making during the operation (translate, scale, and so on) will continue to be applied to the context in future operations, and that might not be desirable.
- At the end, you restore the context to its clean original state, so that future canvas operations are performed without the translation that was applied in this operation.

Canvas Working with Paths

```
function createCanopyPath(context) {
// Draw the tree canopy
context.beginPath();
context.moveTo(-25, -50); context.lineTo(-10, -80); context.lineTo(-20, -80);
context.lineTo(-5, -110); context.lineTo(-15, -110);
// Top of the tree
context.lineTo(0, -140); context.lineTo(15, -110);
context.lineTo(5, -110); context.lineTo(20, -80);
context.lineTo(10, -80); context.lineTo(25, -50);
// Close the path back to its start point
context.closePath();
}
```

closePath -> similar to lineTo but the destination is assumed to be the origination of the path

Canvas Working with Paths

```
function drawTrails() {
        var canvas = document.getElementById('trails');
        var context = canvas.getContext('2d');
        context.save();
        context.translate(130, 250);
        // Create the shape for our canopy path
        createCanopyPath(context);
        // Stroke the current path
        context.stroke();
        context.restore();
}
    window.addEventListener("load", drawTrails, true);
```


Canvas Working with Styles

```
// Increase the line width
context.lineWidth = 4;

// Round the corners at path joints
context.lineJoin = 'round';

// Change the color to brown
context.strokeStyle = '#663300';

// Set the fill color to green and fill the canopy
context.fillStyle = '#339900';
context.fill();
```


Canvas Working with Styles

```
// Change fill color to brown
context.fillStyle = '#663300';

// Fill a rectangle for the tree trunk
context.fillRect(-5, -50, 10, 50);

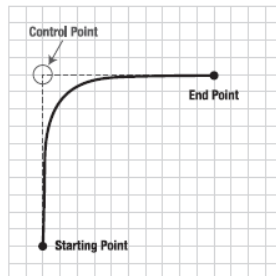
// takes the x and y location as well as the width and the height
```

Canvas
Drawing Curves

context.quadraticCurveTo(xC, yC, xS, yS);

(xC,yC) – coordinates of the control point. The control point sits to the side of the curve (not on it) and acts almost as a gravitational pull for the points along the curve path. By adjusting the location of the control point, you can adjust the curvature of the path you are drawing.

(xS,yS) – final stop in the curve



Canvas
Drawing Curves

```
// Save the canvas state and draw the path
context.save();
context.translate(-10, 350);
context.beginPath();
// The first curve bends up and right
context.moveTo(0, 0);
context.quadraticCurveTo(170, -50, 260, -190);
// The second curve continues down and right
context.quadraticCurveTo(310, -250, 410,-250);
// Draw the path in a wide brown stroke
context.strokeStyle = '#663300';
context.lineWidth = 20;
context.stroke();
// Restore the previous canvas state
context.restore();
```


Canvas
Inserting Images into a Canvas

- Images can be stamped, stretched, modified with transformations, and often be the focus of the entire canvas.
- But images also add a complication to the canvas operations: you must wait for them to load.
- The image has to be loaded completely before you attempt to render it.

Solution

```
// Load the bark image
var bark = new Image();
bark.src = "bark.jpg";
// Once the image is loaded, draw on the canvas
bark.onload = function () {
drawTrails();
}
```


Canvas
Inserting Images into a Canvas

Drawing an image on a canvas

```
// Draw the bark pattern image where
// the filled rectangle was before
context.drawImage(bark, -5, -50,     10,        50);
                         x    y     width     height
```

This option will scale the image to fit into the 10 × 50 pixel space that we have allocated for our trunk.

Canvas
Using Gradients

- Gradients allow you to apply a gradual algorithmic sampling of colors as either a stroke or fill style
- Creating gradients requires a three-step process:
  1. Create the gradient object itself.
  2. Apply color stops to the gradient object, signaling changes in color along the transition.
  3. Set the gradient as either a fillStyle or a strokeStyle on the context.
- If you supply points A and B as the arguments to the creation of a gradient, the color will be transitioned for any stroke or fill that moves in the direction of point A to point B.

Canvas
Using Gradients

```
// Create a 3 stop gradient horizontally across the trunk
var trunkGradient = context.createLinearGradient(-5, -50, 5, -50);
// The beginning of the trunk is medium brown
trunkGradient.addColorStop(0, '#663300');
// The middle-left of the trunk is lighter in color
trunkGradient.addColorStop(0.4, '#996600');
// The right edge of the trunk is darkest
trunkGradient.addColorStop(1, '#552200');
// Apply the gradient as the fill style, and draw the trunk
context.fillStyle = trunkGradient;
context.fillRect(-5, -50, 10, 50);
```


Canvas
Using Gradients

```
// A second, vertical gradient creates a shadow from the
// canopy on the trunk
var canopyShadow = context.createLinearGradient(0, -50, 0, 0);
// The beginning of the shadow gradient is black, but with
// a 50% alpha value
canopyShadow.addColorStop(0, 'rgba(0, 0, 0, 0.5)');
// Slightly further down, the gradient completely fades to
// fully transparent. The rest of the trunk gets no shadow.
canopyShadow.addColorStop(0.2, 'rgba(0, 0, 0, 0.0)');
// Draw the shadow gradient on top of the trunk gradient
context.fillStyle = canopyShadow;
context.fillRect(-5, -50, 10, 50);
```


Canvas
Using Background Patterns

- The HTML5 Canvas API also includes an option to set an image as a repeatable pattern for either a path stroke or fill.

```
// Replace the bark image with
// a trail gravel image
var gravel = new Image();
gravel.src = "gravel.jpg";
gravel.onload = function () {
drawTrails();
}
// Replace the solid stroke with a repeated
// background pattern
context.strokeStyle = context.createPattern(gravel, 'repeat');
context.lineWidth = 20;
context.stroke();
```

Canvas
Using Background Patterns

Repetition Patterns

Repeat	Value
repeat	(Default) The image is repeated in both directions
repeat-x	The image is repeated only in the X dimension
repeat-y	The image is repeated only in the Y dimension
no-repeat	The image is displayed once and not repeated

Canvas
Scaling Canvas Objects

```
// Move tree drawing into its own function for reuse
function drawTree(context) {
var trunkGradient = context.createLinearGradient(-5, -50, 5, -50);
…
}

// Draw the second tree at X=260, Y=500
context.save();
context.translate(260, 500);
// Scale this tree twice normal in both dimensions
context.scale(2, 2);
drawTree(context);
context.restore();
```


Canvas
Scaling Canvas Objects

Note: transforms such as scale and rotate operate from the origin.

- If you perform a rotate transform to a shape drawn off origin, a rotate transform will rotate the shape around the origin rather than rotating in place.
- Similarly, if you performed a scale operation to shapes before translating them to their proper position, all locations for path coordinates would also be multiplied by the scaling factor.
- Depending on the scale factor applied, this new location could even be off the canvas altogether, leaving you wondering why your scale operation just ‘deleted’ the image.

Canvas
Scaling Canvas Objects

```
context.save();
// rotation angle is specified in radians
context.rotate(1.57);
context.drawImage(myImage, 0, 0, 100, 100);
context.restore();
```

Canvas
Scaling Canvas Objects

Example (continued)

```
// Create a slanted tree as the shadow by applying
// a shear transform, changing X values to increase
// as Y values increase
context.transform(1, 0,-0.5, 1, 0, 0);
// Shrink the shadow down to 60% height in the Y dimension
context.scale(1, 0.6);
// Set the tree fill to be black, but at only 20% alpha
context.fillStyle = 'rgba(0, 0, 0, 0.2)';
context.fillRect(-5, -50, 10, 50);
// Redraw the tree with the shadow effects applied
createCanopyPath(context);
context.fill();
// Restore the canvas state
context.restore();
```


Canvas
Scaling Canvas Objects

transform(a, b, c, d, e, f);

a: Horizontal scaling.
b: Horizontal skewing.
c: Vertical skewing.
d: Vertical scaling.
e: Horizontal moving.
f: Vertical moving.

Canvas
Using Canvas Text

The text drawing routines consist of two functions on the context object:

- fillText (text, x, y, maxwidth)
- strokeText (text, x, y, maxwidth)

Both functions take the text as well as the location at which it should be drawn.

Optionally, a maxwidth argument can be provided to constrain the size of the text by automatically shrinking the font to fit the given size.

In addition, a measureText function is available to return a metrics object containing the width of the given text should it be rendered using the current context settings.

Canvas
Using Canvas Text

Property	**Values**	**Note**
font	CSS font string	Example: italic Arial, sans-serif
textAlign	start, end, left, right, center	Defaults to start
textBaseline	top, hanging, middle, alphabetic , ideographic, bottom	Defaults to alphabetic

```
// Draw title text on our canvas
context.save();
// The font will be 60 pixel, Impact face
context.font = "60px impact";
// Use a brown fill for our text
context.fillStyle = '#996600';
// Text can be aligned when displayed
context.textAlign = 'center';
// Draw the text in the middle of the canvas with a max
// width set to center properly
context.fillText('Happy Trails!', 200, 60, 400);
context.restore();
```

Canvas
Applying shadows

Property	Values	Note
shadowColor	Any	CSS color Can include an alpha component
shadowOffsetX	Pixel count	Positive values move shadow to the right, negative left
shadowOffsetY	Pixel count	Positive values move shadow down, negative up
shadowBlur	Gaussian blur	Higher values cause blurrier shadow edges

```
// Set some shadow on our text, black with 20% alpha
context.shadowColor = 'rgba(0, 0, 0, 0.2)';
// Move the shadow to the right 15 pixels, up 10
context.shadowOffsetX = 15;
context.shadowOffsetY = -10;
// Blur the shadow slightly
context.shadowBlur = 2;
```


Canvas
Events

- There are many different application possibilities for using the Canvas API:
  - graphs, charts, image editing, and so on.
- However, one of the most intriguing uses for the canvas is to modify or overlay existing content.
  - Areas on the map with high levels of activity are colored as hot (for example, red, yellow, or white).
  - Areas with less activity show no color change at all, or minimal blacks and grays.
- We are able to perform these uses because the Canvas object supports all the standard events
- See the example linked to this slide

Designing Web Applications

CSS

Francesco Marcelloni

Department of Information Engineering
University of Pisa
ITALY

Cascading Style Sheets

- Cascading Style Sheets (CSS) is the recommended way to control the presentation layer in a web document. The main advantage of CSS over presentational HTML markup is that the styling can be kept entirely separate from the content.
- For example, it is possible to store all the presentational styles for a 10,000-page web site in a single CSS file. CSS also provides far better control over presentation than do presentational element types in HTML.

CSS Benefits

- By externalizing the presentation layer, CSS offers a number of significant benefits:
  - All styling is kept in a limited number of style sheets. Editing one style sheet is obviously more efficient than editing 10,000 HTML files!
  - The overall saving in bandwidth is measurable.
    - The style sheet is cached after the first request
    - Removing all presentational markup from your web pages reduces their size and bandwidth usage — by more than 50% in many documented cases
      - Benefits for the site owner, through lower bandwidth and storage costs,
      - Benefits for the site's visitors, for whom the web pages load faster.

CSS Benefits

- The separation of content from presentation makes it easier for site owners to reuse the content for other purposes, such as RSS feeds or text-to-speech conversion.
- Separate styling rules can be used for different output media. We no longer need to create a special version of each page for printing
  - we can simply create a single style sheet that controls how every page on the site will be printed.
- Although CSS is designed to be independent of the markup language of the documents to which it is applied, in reality, it is used mainly with HTML and XML (including XHTML).

CSS Versions

- The first CSS specification, CSS1, became a World Wide Web Consortium (W3C) recommendation in December 1996.
  - It included properties for controlling typography, such as fonts, text alignment, spacing, margins, and list formatting.
- CSS2 came out in 1998, and contained a lot of the features that designers had been longing for.
  - Boxes could be made to behave like HTML table cells, or they could be positioned in different ways;
  - more powerful selectors were available;
  - style sheets could be imported into other style sheets;
  - style rules could be specific to certain output media; and so on.
  - Vast improvements in the areas of paged media (printing), and the generation of content from the style sheet.

Università di Pisa
Francesco Marcelloni

CSS Versions

- As it turned out, some parts of CSS2 were very difficult to implement, so the W3C decided to revise the specification and adapt it to real-world situations.
- The name of the revised version was "Cascading Style Sheets, Level 2 Revision 1"—CSS2.1 for short.
- In the last years, updates of different sections, called modules – CSS3

Università di Pisa
Francesco Marcelloni

Including Style Sheets in HTML Documents

Università di Pisa
Francesco Marcelloni

To insert CSS into an HTML document

There are three ways of inserting a style sheet into an HTML document:

- Inline style
- Internal style sheet
- External style sheet

Inline Styles

- Place the rules in inline CSS specified in a style attribute of a markup tag
- The style attribute can contain any CSS property.

```
<p style="color:red; margin-left: 10%">This is a paragraph.</p>
```

- Use this method sparingly!
  - Using style attributes creates a strong coupling between the content and the presentation, which is usually undesirable.
  - Inline CSS is more limited than internal or external style sheets. It is not possible to specify a media type, so style attributes will apply for all media types.

Internal Style Sheet

- Place the rules within a separate, internal style sheet in the head section of an HTML page, by using the <style> tag
- An internal style sheet should be used when a single document has a unique style.

```
<head>
<style type="text/css"; media="screen">
<!--
hr {color:red;}
p {margin-left:20px;}
body {background:red;}
-->
</style></head>
```


Internal Style Sheet

- NOTE: Do not leave spaces between the property value and the units!

  "margin-left:20 px" (instead of "margin-left:20px") will work in IE, but not in Firefox or Opera.

External Style Sheet

- Place the rules in a separate, external style sheet that is referenced by a link element or an @import rule in a style element
- Ideal when the style is applied to several pages
  - You can change the look of an entire Web site by changing one file.
  - An external style sheet can be written in any text editor.
  - The file should not contain any html tags.
  - Your style sheet should be saved with a .css extension.

External Style Sheet

```
<head>
 <title>  CSS Example </title>
 <link rel="stylesheet" type="text/css" href="ex1.css">
</head>
<body>
 <h1>
  This header is 36 pt
 </h1>
 <h2>
  This header is blue
 </h2>
 <p>
  This paragraph has a left margin of 50 pixels
 </p>
</body>
```


External Style Sheet

```
/*file ex1.css
body
{ background-color:yellow;}
h1
{ font-size:36pt; }
h2
{ color:blue;}
p
{ margin-left:50px; }
```


Including External Style Sheet The Link Tag

First Method: the Link Tag

- The <link> tag is the most used method to link to style sheets
- Note: The link element must be embedded in the head section, and it can appear any number of times.
- The attribute rel specifies the relationship between the current document and the linked document. When used in the style sheet inclusion rel="stylesheet".
- The attribute type specifies the Multipurpose Internet Mail Extensions (MIME) type of the linked document

Including External Style Sheet The @import at-rule

Second Method: the directive @import

- The directive is used in the content of a style tag

```
<head>
<style type="text/css">
 @import url(/style.css);
</style>
</head>
```

Media Type

- Through style sheets you can specify how a document is to be presented on different media: on the screen, on paper, with a speech synthesizer, with a braille device, etc.
- Certain CSS properties are only designed for certain media (e.g., the 'page-break-before' property only applies to paged media). Style sheets for different media types may share a property, but require different values for that property.

How can you specify the Media Type?

- There are currently two ways to specify media dependencies for style sheets:

1. Specify the target medium from a style sheet with the @import at-rule:

```
@import url("fancyfonts.css") screen;
```

or with the @media at-rules.

```
@media print {
    /* style sheet for print goes here */
  }
```


Media Type

2. Specify the target medium within the document language.

```
<head>
  <title>  Link to a target medium   </title>
  <link rel="stylesheet"
       type="text/css"
       media="print, handheld"
       href="foo.css">
 </head>
 <body>
  <p>  The body...  </p>
 </body>
```


Media Type

Media Type	Description
all	Used for all media type devices
braille	Used for braille tactile feedback devices
embossed	Used for paged braille printers
handheld	Used for small or handheld devices
print	Used for printers
projection	Used for projected presentations, like slides
screen	Used for computer screens
speech	Used for speech synthesizers
tty	Used for media using a fixed-pitch character grid, like teletypes and terminals
tv	Used for television-type devices

Media Type

```
<head>
<style type="text/css" media="screen">
 h1 { background:black; color:white; text-align:center }
 h2 { color: red; font-style:italic}   h3 { display: none }
</style>
<style type="text/css" media="print">
 h1 { color: black; text-align: left } h2 { display: none }
</style>
<title>Example Media Type</title>
</head>
<body>
<h1>Example</h1>
<h2>This is not printed</h2>
<h3>This is not displayed</h3>
</body>
```


Cascading Style Sheets: Syntax

CSS style sheet

- A CSS style sheet consists of a list of statements. There are two kinds of statements:
  - at-rules
  - rule sets.
- There may be white space around the statements.
- Example

```
@import "subs.css";
@media print {
  @import "print-main.css";
  body { font-size: 10pt }
}
h1 {color: blue }
```


CSS style sheet At-rule

- At-rules
  - At-rules start with an at-keyword, an '@' character followed immediately by an identifier (for example, '@import', '@page').
  - An at-rule consists of everything up to and including the next semicolon (;) or the next block, whichever comes first.
  - A block starts with a left curly brace ({) and ends with the matching right curly brace (}).

CSS style sheet
Rule sets

- Rule sets
  - A rule set (also called "rule") consists of a selector followed by a declaration block.
  - A declaration block starts with a left curly brace ({) and ends with the matching right curly brace (}). In between there must be a list of zero or more semicolon-separated (;) declarations.
  - The selector consists of everything up to (but not including) the first left curly brace ({). A selector always goes together with a declaration block. When a user agent cannot parse the selector (i.e., it is not valid CSS 3.0), it must ignore the selector and the following declaration block (if any) as well.

CSS style sheet
Rule set

- Rule sets (example)

```
h1, h2 {color: green }
h3, h4 & h5 {color: red }
h6 {color: black }
```

Since the "&" is not a valid token in a CSS 3.0 selector, a CSS 3.0 user agent must ignore the whole second line

CSS style sheet
Declarations

- Declarations
  - A declaration is either empty or consists of a property name, followed by a colon (:), followed by a value. Around each of these there may be white space.

```
h1 { font-weight: bold }
h1 { font-size: 12px }
h1 { line-height: 14px }
h1 { font-family: Helvetica }
h1 { font-variant: normal }
h1 { font-style: normal }
```

=

```
h1 {
  font-weight: bold;
  font-size: 12px;
  line-height: 14px;
  font-family: Helvetica;
  font-variant: normal;
  font-style: normal
}
```


CSS style sheet
Declarations

- Declarations
  - A user agent must ignore a declaration with an invalid property name or an invalid value.

```
h1 { color: red; font-style: 12pt }  /* Invalid value: 12pt */
p { color: blue;  font-vendor: any;  /* Invalid prop.: font-vendor
  */
   font-variant: small-caps }
em { font-style: normal }
```

The CSS parser will reduce the style sheet to

```
h1 { color: red; }
p { color: blue;  font-variant: small-caps }
em em { font-style: normal }
```

CSS style sheet
Comments

- Comments
  - Begin with the characters "/*" and end with the characters "*/".
  - May occur anywhere between tokens, and their contents have no influence on the rendering.
  - May not be nested.

CSS style sheet
Selectors

- Pattern matching rules determine which style rules apply to elements in the document tree.
  - These patterns, called selectors, may range from simple element names to rich contextual patterns.
  - If all conditions in the pattern are true for a certain element, the selector matches the element.
- A selector is a chain of one or more simple selectors separated by combinators.
- A simple selector is either a type selector or universal selector followed immediately by zero or more attribute selectors, ID selectors, or pseudo-classes, in any order. The simple selector matches if all of its components match.
- Combinators are: white space, ">", and "+". White space may appear between a combinator and the simple selectors around it.

Selectors
Type Selector

- Type selector
  - A type selector matches the name of a document language element type.

```
h1 { font-family: sans-serif }
p { text-align: justify }
div { position: absolute; left: 10px; top: 50px }
table { width: 80%; height: 50% }
```


Selectors
Universal Selector

- Universal selector
  - The universal selector "*" matches the name of any element type.

```
<head>
<style type="text/css">
 body { font-size: large; }
 h1 {color: green;}
 * { color: red; }
</style>
   <title>Universal Selector</title>
</head>
 <body>
   <h1> The heading is green </h1>
   <p>      This text is red    </p>
 </body>
```

Selectors
Grouping

- Grouping
  - When several selectors share the same declarations, they may be grouped into a comma-separated list.
  - In this example, we condense three rules with identical declarations into one. Thus,

```
h1 { font-family: sans-serif }
h2 { font-family: sans-serif }
h3 { font-family: sans-serif }
```

is equivalent to:

```
h1, h2, h3 { font-family: sans-serif }
```


Selectors
Class Selectors

- Class Selectors
  - If only type selectors were available, we would have that all the element instances would have the same format.
    For instance:
    p { font-family: Arial }
  - To have different fonts for different paragraphs, we can exploit the class attribute.
  - We can assign style information to all elements with class= "myclass" as follows:

```
*.myclass { color: green }  /* all elements with class=myclass */
   or just
.myclass { color: green }  /* all elements with class=myclass */
```


Selectors
Class Selectors

- The following example assigns different font families to different paragraphs

```
<style type="text/css">
 .arial { font-family: Arial }
 .serif { font-family: serif }
</style>
<title> Font  </title>
</head>
<body>
<p>Paragraph</p>
<p class="arial">This paragraph is in Arial font</p>
<p class="serif">This paragraph is in Serif font</p>
</body>
```


Selectors
Multiple Classes

- Multiple classes

  p.green.bold { color: green; font-weight: bold }

  The rule matches any p element whose "class" attribute has been assigned a list of space-separated values that includes "green" and "bold"

  <p class="green serif bold"> (OK)

  <p class="serif bold"> (NO)

Selectors
id selector

- id selector
  - Document languages may contain attributes that are declared to be of type id.
    - Important: no two such attributes can have the same value;
    - whatever the document language, an id attribute can be used to uniquely identify its element.
  - A CSS id selector contains a "#" immediately followed by the id value, which must be an identifier

Selectors
id selector

- id selector (Example)

```
<head>
<style type="text/css">
 .red { color: red }
 #black { color: black }
</style>
</head>
<body>
<p>...</p>
<p class="red">This paragraph is red</p>
<p class="red" id=black>This paragraph is black</p>
</body>
```


Selectors
Pseudo-Elements

- Pseudo-elements create abstractions about the document tree beyond those specified by the document language.
  - For instance, document languages do not offer mechanisms to access the first letter or first line of an element's content.
- Pseudo-elements may also provide style sheet designers a way to assign style to content that does not exist in the source document (e.g., the :before and :after pseudo-elements give access to generated content).

Selectors
Pseudo-Elements

- The :first-line pseudo-elements
  - applies special styles to the contents of the first formatted line of a paragraph. For instance:

    p:first-line { text-transform: uppercase }
- The :first-letter pseudo-elements
  - select the first letter of the first line of a block, if it is not preceded by any other content (such as images or inline tables) on its line.
    - The :first-letter pseudo-element may be used for "initial caps" and "drop caps", which are common typographical effects.

Selectors Pseudo-Elements

- The :before and :after pseudo-elements
  - can be used to insert generated content before or after an element's content. For instance:

```
<style type="text/css">
 p.red { color: red }
 p.red:before{content: "Note that: "}
 #black { color: black }
 #black:before { content: "Observe that: "}
</style> <title>Color</title></head>
<body>
<p>Normal</p>
<p class="red">This paragraph is red</p>
<p class="red" id=black>This paragraph is black</p>
</body>
```


Selectors Pseudo-Elements

- Pseudo-Element (example)

```
<head>
<title>Drop cap initial letter</title>
<style type="text/css">
  p            { font-size: 12pt; line-height: 1.2 }
  p:first-letter { font-size: 200%; font-style: italic;
                  font-weight: bold; float: left }
  span          { text-transform: uppercase }
</style>
</head>
<body>
<p><span>The first</span> few words of an article in The
Economist.</p>
</body>
```


Selectors Pseudo-Classes

- Pseudo-classes classify elements on characteristics other than their name, attributes or content; in principle characteristics that cannot be deduced from the document tree.
- Pseudo-classes may be dynamic, in the sense that an element may acquire or lose a pseudo-class while a user interacts with the document.
  - The exceptions are ':first-child', which can be deduced from the document tree, and ':lang()', which can be deduced from the document tree in some cases

Selectors Pseudo-Classes

- The :first-child pseudo-class matches an element that is the first child element of some other element.

```
<style type="text/css">
   div > p:first-child { text-indent: 10pt } /*add indentation */
        </style>          /*for the first paragraph of a div */
 </head>
 <body>
  <div class="note">
  <p>The first P inside the note.</p>
  <p>The second P inside the note.</p>
   </div>
 </body>
```

Selectors Pseudo-Classes

- The :link and :visited pseudo-classes
  - The :link pseudo-class applies for links that have not yet been visited.
  - The :visited pseudo-class applies once the link has been visited by the user.

```
a:link { color: blue; }
a:visited { color: fuchsia; text-decoration: none }
...
<a href ="pagina.html">Prova link</a>
```


Selectors Pseudo-Classes

- The :hover, :active, and :focus pseudo-classes
  - The :hover pseudo-class applies while the user designates an element (with some pointing device), but does not activate it.
    - A visual user agent could apply this pseudo-class when the cursor (mouse pointer) hovers over a box generated by the element (must be placed after the A:link and A:visited rules).
  - The :active pseudo-class applies while an element is being activated by the user.
    - For example, between the times the user presses the mouse button and releases it.
  - The :focus pseudo-class applies while an element has the focus (accepts keyboard events or other forms of text input).

```
a:hover { text-transform: uppercase }
a:active { color: red }
```

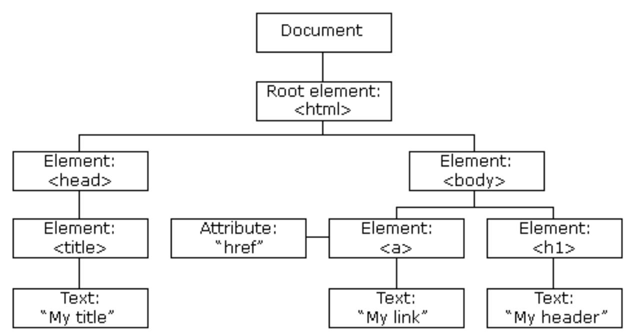

Selectors Pseudo-Classes

- The :target pseudo-class
  - URLs with an # followed by an anchor name link to a certain element within a document. The element being linked to is the target element.
  - The :target selector can be used to style the current active target element.

```
<style>
:target {
   border: 2px solid #D4D4D4;
   background-color: #e5eecc;
}
</style>
```


Inheritance

- DOM tree (example)



Summary of the selector syntax

Pattern	Meaning	Example
E	Matches any E element (i.e., an element of type E).	h1 { font-family: sans-serif }
E F	Matches any F element that is a descendant of an E element.	h1 em { color: blue } This headline is very important
E > F	Matches any F element that is a child of an element E.	div ol>li p { color: blue }
E + F	Matches any F element immediately preceded by a sibling element E.	h1 + h2 { margin-top: -5mm}
E[attrib]	Matches any E element with the "attrib" attribute set (whatever the value).	h1[title] { color: blue; }

Summary of the selector syntax

Pattern	Meaning	Example
E[attrib="value"]	Matches any E element whose "attrib" attribute value is exactly equal to "value".	span[class="example"] { color: blue; }
E[attrib~="value"]	Matches any E element whose "attrib" attribute value is a list of space-separated values, one of which is exactly equal to "value".	a[rel~="copyright"] match the value "copyright copyleft copyeditor"
E[attrib|="value"]	Matches any E element whose "attrib" attribute has a hyphen-separated list of values beginning (from the left) with "value".	*[lang|="en"] { color : red } matches "en", "en-US", and "en-cockney"
E#myid	Matches any E element with ID equal to "myid".	h1#chapter1 { text-align: center }

Summary of the selector syntax

Pattern	Meaning	Described in section	First defined in CSS level
*	any element	Universal selector	2
E	an element of type E	Type selector	1
E[foo]	an E element with a "foo" attribute	Attribute selectors	2
E[foo="bar"]	an E element whose "foo" attribute value is exactly equal to "bar"	Attribute selectors	2
E[foo~="bar"]	an E element whose "foo" attribute value is a list of whitespace-separated values, one of which is exactly equal to "bar"	Attribute selectors	2
E[foo^="bar"]	an E element whose "foo" attribute value begins exactly with the string "bar"	Attribute selectors	3
E[foo$="bar"]	an E element whose "foo" attribute value ends exactly with the string "bar"	Attribute selectors	3
E[foo*="bar"]	an E element whose "foo" attribute value contains the substring "bar"	Attribute selectors	3
E[foo|="en"]	an E element whose "foo" attribute has a hyphen-separated list of values beginning (from the left) with "en"	Attribute selectors	2

Summary of the selector syntax

Pattern	Meaning	Described in section	First defined in CSS level
E:root	an E element, root of the document	Structural pseudo-classes	3
E:nth-child(n)	an E element, the n-th child of its parent	Structural pseudo-classes	3
E:nth-last-child(n)	an E element, the n-th child of its parent, counting from the last one	Structural pseudo-classes	3
E:nth-of-type(n)	an E element, the n-th sibling of its type	Structural pseudo-classes	3
E:nth-last-of-type(n)	an E element, the n-th sibling of its type, counting from the last one	Structural pseudo-classes	3
E:first-child	an E element, first child of its parent	Structural pseudo-classes	2

Summary of the selector syntax

Pattern	Meaning	Described in section	First defined in CSS level
E:last-child	an E element, last child of its parent	Structural pseudo-classes	3
E:first-of-type	an E element, first sibling of its type	Structural pseudo-classes	3
E:last-of-type	an E element, last sibling of its type	Structural pseudo-classes	3
E:only-child	an E element, only child of its parent	Structural pseudo-classes	3
E:only-of-type	an E element, only sibling of its type	Structural pseudo-classes	3

Summary of the selector syntax

Pattern	Meaning	Described in section	First defined in CSS level
E:empty	an E element that has no children (including text nodes)	Structural pseudo-classes	3
E:link E:visited	an E element being the source anchor of a hyperlink of which the target is not yet visited (:link) or already visited (:visited)	The link pseudo-classes	1
E:active E:hover E:focus	an E element during certain user actions	The user action pseudo-classes	1 and 2
E:target	an E element being the target of the referring URI	The target pseudo-class	3
E:lang(fr)	an element of type E in language "fr" (the document language specifies how language is determined)	The :lang() pseudo-class	2
E:enabled E:disabled	a user interface element E which is enabled or disabled	The UI element states pseudo-classes	3

Summary of the selector syntax

Pattern	Meaning	Described in section	First defined in CSS level
E:checked	a user interface element E which is checked (for instance a radio-button or checkbox)	The UI element states pseudo-classes	3
E::first-line	the first formatted line of an E element	The ::first-line pseudo-element	1
E::first-letter	the first formatted letter of an E element	The ::first-letter pseudo-element	1
E::before	generated content before an E element	The ::before pseudo-element	2
E::after	generated content after an E element	The ::after pseudo-element	2
E.warning	an E element whose class is "warning" (the document language specifies how class is determined).	Class selectors	1

Summary of the selector syntax

Pattern	Meaning	Described in section	First defined in CSS level
E#myid	an E element with ID equal to "myid".	ID selectors	1
E:not(s)	an E element that does not match simple selector s	Negation pseudo-class	3
E F	an F element descendant of an E element	Descendant combinator	1
E > F	an F element child of an E element	Child combinator	2
E + F	an F element immediately preceded by an E element	Adjacent sibling combinator	2
E ~ F	an F element preceded by an E element	General sibling combinator	3

Cascade, Specificity and Inheritance

Cascade, specificity and inheritance

- The Cascading term indicates that style declarations cascade down to elements from many origins.
- The cascade computes a weight for each applicable declaration based on:
  - source order
  - importance
  - origin
  - specificity.
- This weight is used to determine exactly—and without conflict—which declaration should be applied to any given element.

The Cascade
Process of resolution

1. For a given property, the user agent finds all declarations that apply to a specific element for the target media type
   - by analysing three sources
     - the user agent
     - the author
     - the user style sheets (customized set of styles to use by default which some user agent allows a user to create).

The Cascade
Process of resolution

- Declarations are applied to the element, if
  - the associated selector matches the element **and**
  - the target medium matches the media list on all @media rules containing the declaration and on all links on the path through which the style sheet was reached.
- If there is more than one applicable declaration that sets a specific property, the cascade proceeds to step two.

The Cascade
Process of resolution

2. Sort the declarations according to their levels of importance, and origins.
   - Declarations that are appended with the !important statement are called important declarations;
   - Otherwise they are called normal declarations.

     p { font-size: 1em !important;}

The Cascade
Process of resolution

2. Sort the declarations according to their levels of importance, and origins.
   - Declarations are sorted in the following ascending order:
     1. Transition declarations
     2. Important user agent declarations
     3. Important user declarations
     4. Important override declarations
     5. Important author declarations
     6. Animation declarations
     7. Normal override declarations
     8. Normal author declarations
     9. Normal user declarations
     10. Normal user agent declarations

The Cascade
Process of resolution

3. Sort declarations with the same level of importance and origin by selector specificity.
   - The specificity of a selector is represented by four comma-separated values *a*,*b*,*c*,*d*, where the values in "a" are the most important and those in "d" are least important.

The Cascade
Process of resolution

- A selector's specificity is calculated as follows:
  - *a* = 1 if the declaration is from a 'style' attribute rather than a rule with a selector (an inline style), *a* = *0* otherwise.
  - *b* = number of ID attributes in the selector.
  - *c* = number of other attributes and pseudo-classes in the selector.
  - *d* = number of element names and pseudo-elements in the selector.

The Cascade
Process of resolution

4. Finally, if declarations have the same level of importance, origin, and specificity, sort them by the order in which they are specified; the last declaration wins.

- Note: If the link to the external style sheet is placed after the internal style sheet in HTML <head>, the external style sheet will override the internal style sheet!

The Cascade
Process of resolution

Some examples

```
* {}                /* a=0 b=0 c=0 d=0
li {}               /* a=0 b=0 c=0 d=1
li:first-line {}    /* a=0 b=0 c=0 d=2
ul li {}            /* a=0 b=0 c=0 d=2
ul ol+li {}         /* a=0 b=0 c=0 d=3
h1 + *[rel=up]{}    /* a=0 b=0 c=1 d=1
ul ol li.red {}     /* a=0 b=0 c=1 d=3
li.red.level {}     /* a=0 b=0 c=2 d=1
#x34y {}            /* a=0 b=1 c=0 d=0
style=""            /* a=1 b=0 c=0 d=0
```


The Cascade
Process of resolution

```
<style type="text/css">
    #redP { color: red }
    p.bluStyle {color:blue}
</style>
<title>Resolution Process</title>
</head>
<body>
<p id="redP" class="bluStyle" style="color:
  green"> Example 1 </p>
<p class="bluStyle"> Example 2 </p>
</body>
</html>
```


The Cascade
Process of resolution

```
<head>
  <style type="text/css">
   body {
     color: #000;
     background-color: #fff;
   }
   #wrap {
     font-size: 2em;
     color: #333;
   }
   div {
     font-size: 1em;
   }
```

The Cascade
Process of resolution

```
em {
  color: #666;
}
p.item {
  color: #fff;
  background-color: #ccc;
  border-style: dashed;
}
p {
  border: 1px solid black;
  padding: 0.5em;
}
</style>
</head>
```


The Cascade
Process of resolution

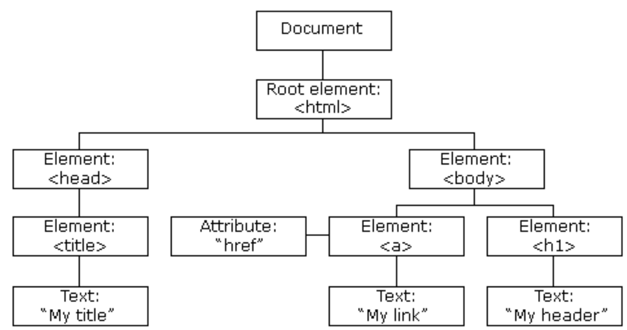
```
<body>
  <div id="wrap">
    <p>Normal Paragraph</p>
    <p class="item">
      This is the <em>cascade</em> in
      <a href="#">action</a>
    </p>
  </div>
</body>
```


Inheritance

- Inheritance is the process by which properties are passed from parent to child elements even though those properties have not been explicitly defined by other means.
- Inheritance is a mechanism that is separate from the cascade: inheritance applies to the DOM (Document Object Model) tree, while the cascade deals with the style sheet rules.
- However, CSS properties set on an element via the cascade can be inherited by that element's child elements.

Inheritance

- DOM tree (example)



Inheritance

```
<head>
<style type="text/css">
  body { color: red }   h2 { color: black }
  </style>
    <title>  Example   </title>
</head>
 <body>
   <h1>Heading (red)</h1>
   <p>
     Text of the <b>paragraph</b> (the colour is inherited)
   </p>
   <h2>  The style of heading 2 is redefined  </h2>
 </body>
```


Università di Pisa
Francesco Marcelloni

Cascading Style Sheets: Properties

Università di Pisa
Francesco Marcelloni

Color
Foreground color

- Foreground color: the 'color' property

'color'

Value: <color> | inherit
Initial: depends on user agent
Applies to: all elements
Inherited: yes
Percentages: N/A
Media: visual
Computed value: as specified

```
em { color: red }            /* predefined color name */
em { color: rgb(255,0,0) }     /* RGB range 0-255   */
```


Università di Pisa
Francesco Marcelloni

Color
Foreground color

- A <color> is either a keyword or a numerical RGB specification.
- The list of color keywords is: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, orange, purple, red, silver, teal, white, and yellow. These 17 colors have the following values

maroon #800000	**red** #ff0000	**orange** #ffA500	**yellow** #ffff00	**olive** #808000
purple #800080	**fuchsia** #ff00ff	**white** #ffffff	**lime** #00ff00	**green** #008000
navy #000080	**blue** #0000ff	**aqua** #00ffff	**teal** #008080	
black #000000	**silver** #c0c0c0	**gray** #808080		

Color
Foreground color

- Hexadecimal notation: a '#' immediately followed by either three or six hexadecimal characters.
  - The three-digit RGB notation (#rgb) is converted into six-digit form (#rrggbb) by replicating digits, not by adding zeros. This format allows defining 16777216 colors
- Functional notation: 'rgb(' followed by a comma-separated list of three numerical values (either three integer values or three percentage values) followed by ')'.
  - The integer value 255 corresponds to 100%, and to F or FF in the hexadecimal notation: rgb(255,255,255) = rgb(100%,100%,100%) = #FFF.
  - White space characters are allowed around the numerical values.

Color
Foreground color

- Examples

```
body {color: black; background: white }
h1 { color: maroon }
h2 { color: olive }
em { color: #f00 }          /* #rgb */
em { color: #ff0000 }       /* #rrggbb */
em { color: rgb(255,0,0) }
em { color: rgb(100%, 0%, 0%) }
```


Color
Opacity

- The opacity property sets the opacity level for an element.
- The opacity-level describes the transparency-level, where 1 is not transparent at all, 0.5 is 50% see-through, and 0 is completely transparent.

```
<head>
<style>
div {
   background-color: red;
   opacity: 0.2;
}
</style>
</head>
```


Color
Background color

- 'background-color'

  Value: <color> | transparent | inherit
  Initial: transparent
  Applies to: all elements
  Inherited: no
  Percentages: N/A
  Media: visual

- This property sets the background color of an element.

```
h1 { background-color: #F00 }
```

Color Background color

- 'background-color' (example)

```
<head>
   <title>  Background Color </title>
<style type="text/css">
  body { background-color: #ccc }
   #father { background-color: teal }
   #son { background-color: red}
</style>
 </head>
 <body>
 <div id="father">      Father element
     <div id="son">       Son Element       </div>
   </div>
 </body>
```


Background Image

- 'background-image'
  - **Value**: <uri> | none | inherit
  - **Initial**: none
  - **Applies to**: all elements
  - **Inherited**: no
  - **Percentages**: N/A
  - **Media**: visual
  - **Computed value**: absolute URI or none
- When setting a background image, authors should also specify a background color that will be used when the image is unavailable. When the image is available, it is rendered on top of the background color.
- By default, the image is repeated so it covers the entire element.

Background Image

- 'background-image' (example)

```
<head>
   <title>  Background Image  </title>
<style type="text/css">
  body { background-color: #ccc; background-image: url("tower.jpg")}
   #father { background-color: teal; }
   #son { background-color: red; }
</style>
 </head>
 <body>
   <div id="father">      Father element
     <div id="son">       Son Element      </div>
   </div>
 </body>
```


Background Repeat

- 'background-repeat'
  - **Value:** repeat | repeat-x | repeat-y | no-repeat | space | round | inherit
  - **Initial**: repeat
  - **Applies to:** all elements
  - **Inherited**: no
  - **Percentages**: N/A
  - **Media**: visual
  - **Computed value**: as specified
- If a background image is specified, this property specifies whether the image is repeated (tiled), and how.
- By default, the background-image property repeats an image both horizontally and vertically

Background Repeat

- 'background-repeat'
  - repeat

    The image is repeated both horizontally and vertically.
  - repeat-x

    The image is repeated horizontally only.
  - repeat-y

    The image is repeated vertically only.
  - no-repeat

    The image is not repeated: only one copy of the image is drawn.

Background Repeat

- 'background-repeat'
  - space

    The image is repeated as often as will fit within the background positioning area without being clipped and then the images are spaced out to fill the area.
  - round

    The image is repeated as often as will fit within the background positioning area. If it doesn't fit a whole number of times, it is rescaled so that it does.

Background Repeat

- 'background-repeat' (example)

```
<style type="text/css">
  body {
        background-color: #ccc; background-image: url("tower.jpg");
        background-repeat: space;
        }
 </style>

<style type="text/css">
 body {
        background-color: #ccc; background-image: url("tower.jpg");
        background-repeat: round;
        }
 </style>
```


Background Attachment

- 'background-attachment'

  Value: scroll | fixed | inherit
  Initial: scroll
  Applies to: all elements
  Inherited: no
  Percentages: N/A
  Media: visual
  Computed value: as specified
- If a background image is specified, this property specifies whether it is fixed with regard to the viewport ('fixed') or scrolls along with the containing block ('scroll').

Background Attachment

- 'background-attachment' (example)

```
<head>
   <title>     Background attachment     </title>
<style type="text/css">
  body { background-color: #ccc; background-image:
  url("tower.jpg");
      background-repeat: repeat-x;
      background-position: center;
      background-attachment: fixed;}
</style>
 </head>
 <body>
<p>Text <br> <br><br><br><br><br> Text </p>
</body>
```

Fixed Scroll

Background Position

- 'background-position'

  Value: [[<percentage> | <length> | left | center | right] [<percentage> | <length> | top | center | bottom]?] | [[left | center | right] || [top | center | bottom]] | inherit

  Initial: 0% 0%

  Applies to: all elements

  Inherited: no

  Percentages: refer to the size of the box itself

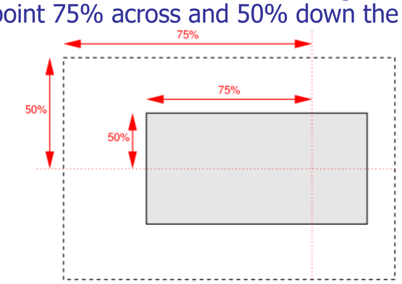
  Media: visual

  Computed value: for <length> the absolute value, otherwise a percentage

- If a background image has been specified, this property specifies its initial position. If only one value is specified, the second is assumed to be 'center'.

Background Position

- 'background-position'
  - <percentage> A percentage X aligns the point X% across (for horizontal) or down (for vertical) the image with the point X% across (for horizontal) or down (for vertical) the element's padding box.
    - For example, with a value pair of '75% 50%', the point 75% across and 50% down the image is to be placed at the point 75% across and 50% down the padding box.



Background Position

- 'background-position'
  - <length> A length L aligns the top left corner of the image a distance L to the right of (for horizontal) or below (for vertical) the top left corner of the element's padding box.
    - For example, with a value pair of '2cm 1cm', the upper left corner of the image is placed 2cm to the right and 1cm below the upper left corner of the padding box.

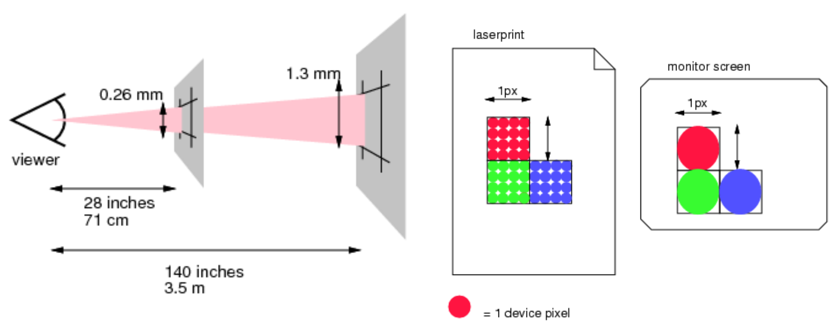
Measurement units

- Relative units
  - em: the 'font-size' of the relevant font (preferred)
  - ex: the 'x-height' of the relevant font
- Absolute units
  - in: inches — 1 inch is equal to 2.54 centimeters.
  - cm: centimeters
  - mm: millimeters
  - pt: points — the points used by CSS 2.1 are equal to 1/72nd of an inch.
  - pc: picas — 1 pica is equal to 12 points.
  - px: pixels – 1/96 inch

Relative Length

- It is recommended that the *reference pixel* be the visual angle of one pixel on a device with a pixel density of 96dpi and a distance from the reader of an arm's length.
- It is recommended that the pixel unit refer to the whole number of device pixels that best approximates the reference pixel.
- Relative units are:
  - For reading at arm's length, 1px thus corresponds to about 0.26 mm (1/96 inch).
  - When printed on a laser printer, meant for reading at a little less than arm's length (55 cm, 21 inches), 1px is about 0.20 mm.
  - On a 300 dots-per-inch (dpi) printer, that may be rounded up to 3 dots (0.25 mm); on a 600 dpi printer, it can be rounded to 5 dots.

Relative Length



Background Position

- 'background-position'
  - top
    Equivalent to '0%' for the vertical position.
  - right
    Equivalent to '100%' for the horizontal position.
  - bottom
    Equivalent to '100%' for the vertical position.
  - left
    Equivalent to '0%' for the horizontal position.
  - center
    Equivalent to '50%' for the horizontal position if it is not otherwise given, or '50%' for the vertical position if it is.

Background Position

- 'background-position' (examples)

```
body { background-image: url("tower.jpg");
       background-repeat:no-repeat;
       background-position: right top;   /* 100%   0% */ }

body { background-image: url("tower.jpg");
       background-repeat:no-repeat;
       background-position: 0% 0%;   /* 0%   0% */ }

body { background-image: url("tower.jpg");
       background-repeat:no-repeat;
       background-position: 100pt 50pt;   /* 100   50 */ }
```


Background Clip

- 'background-clip'

Determines the background painting area, which determines the area within which the background is painted.

Values have the following meanings:

'border-box'
The background is painted within (clipped to) the border box.
'padding-box'
The background is painted within (clipped to) the padding box.
'content-box'
The background is painted within (clipped to) the content box.

Background Clip

- 'background-clip'

```
<style>
div {
   width: 300px;
   height: 300px;
   padding: 50px;
   background-color: yellow;
   background-clip: content-box;
   border: 2px solid #92b901;
}
</style>
```

content-box padding-box

Background Origin

- 'background-origin'

The background-origin property specifies what the background-position property should be relative to.

Values have the following meanings:

'padding-box'
The position is relative to the padding box.
'border-box'
The position is relative to the border box.
'content-box'
The position is relative to the content box.

Background Size

- 'background-size'

The background-size property specifies the size of the background images. Size can be expressed in length and percentages. Further, other values are:

'cover'

Scale the background image to be as large as possible so that the background area is completely covered by the background image.

'contain'

Scale the image to the largest size such that both its width and its height can fit inside the content area

Background

- 'background'

  Value: <bg-image> || <position> [/ <bg-size>]? || <repeat-style> || <attachment> || <box> || <box>

  Initial: see individual properties

  Applies to: all elements

  Inherited: no

  Percentages: allowed on 'background-position'

  Media: visual

  Computed value: see individual properties

- The 'background' property is a shorthand property for setting the individual background properties

Text format Alignment

- 'text-align'

  Value: left | right | center | justify | inherit

  Initial: a nameless value that acts as 'left' if 'direction' is 'ltr', 'right' if 'direction' is 'rtl'

  Applies to: block-level elements, table cells and inline blocks

  Inherited: yes

  Percentages: N/A

  Media: visual

  Computed value: the initial value or as specified

- This property describes how inline content of a block is aligned.

Text format Alignment

- 'text-align' (example)

```
<head>
  <title>     text format     </title>
<style type="text/css">
  .centered { text-align: center }
  p { text-align: justify }
</style>
 </head>
 <body>
  <h1 class="centered"> Centered text </h1>
    <p>justified text justified text justified text justified text
    justified text justified text justified text justified text </p>
 </body>
```

Text format
Indentation

- 'text-indent'

  Value: <length> | <percentage> | inherit
  Initial: 0
  Applies to: block-level elements, table cells and inline blocks
  Inherited: yes
  Percentages: refer to width of containing block
  Media: visual
  Computed value: the percentage as specified or the absolute length

- This property specifies the indentation of the first line of text in a block.

Text format
'text-indent'

- 'text-indent' (example)

```
<head>
  <title>  Text format </title>
<style type="text/css">
  p { text-indent: 10px }
  div {text-indent: 10%; width: 200px; text-align: justify}
</style>
 </head>
 <body>
   <p>Paragraph </p>
     <div>Div </div>
 </body>
```


Text format
Decoration

- 'text-decoration'

  Value: none | [underline || overline || line-through || blink] | inherit
  Initial: none
  Applies to: all elements
  Inherited: no (see prose)
  Percentages: N/A
  Media: visual
  Computed value: as specified

- This property describes decorations that are added to the text of an element using the element's color.

Text format
Decoration

- 'text-decoration' (example)

```
<style type="text/css">
   .under { text-decoration: underline }
   .over { text-decoration: overline }
   .through { text-decoration: line-through }

</style>
...
<p>Normal</p>
<div class="under">Underlined
<p>underlined</p>
</div>
<p class="over">Overlined</p>
<p class="through">Line through</p>
```

Text format
Transformation

- 'text-transform'

  Value: capitalize | uppercase | lowercase | none | inherit
  Initial: none
  Applies to: all elements
  Inherited: yes
  Percentages: N/A
  Media: visual
  Computed value: as specified

- This property controls capitalization effects of an element's text.

  h1 { text-transform: uppercase }

Text format
Letter Spacing

- 'letter-spacing'

  Value: normal | <length> | inherit
  Initial: normal
  Applies to: all elements
  Inherited: yes
  Percentages: N/A
  Media: visual
  Computed value: 'normal' or absolute length

- This property specifies spacing behavior between text characters.

Text format
Letter Spacing

- 'letter-spacing'
  - normal

    The spacing is the normal spacing for the current font. This value allows the user agent to alter the space between characters in order to justify text.
  - <length>

    This value indicates inter-character space in addition to the default space between characters. Values may be negative, but there may be implementation-specific limits. User agents may not further increase or decrease the inter-character space in order to justify text.

Text format
Letter Spacing

- 'letter-spacing' (example)

```
<head>
<title>Text format</title>
<style type="text/css">
        .wide { letter-spacing: 10px }
        .narrow { letter-spacing: -2px }
</style>
</head>
<body>
<p>Normal</p>
<p class="wide">Text with large space</p>
<p class="narrow">Text with narrow space</p>
</body>
```

Text format
Word Spacing

- 'word-spacing'
  Value: normal | <length> | inherit
  Initial: normal
  Applies to: all elements
  Inherited: yes
  Percentages: N/A
  Media: visual
  Computed value: for 'normal' the value '0'; otherwise the absolute length
- This property specifies spacing behavior between words.

Text format
Word Spacing

- 'word-spacing' (example)

```
<head>
<title>Text format</title>
<style type="text/css">
    .wide { word-spacing: 10px }
    .narrow { word-spacing: -2px }
</style>
</head>
<body>
<p>Normal text</p>
<p class="wide">Text with separate words</p>
<p class="narrow">Text with close words</p>
</body>
```


Text format
White Space

- 'white-space'
  Value: normal | pre | nowrap | pre-wrap | pre-line | inherit
  Initial: normal
  Applies to: all elements
  Inherited: yes
  Percentages: N/A
  Media: visual
  Computed value: as specified
- This property declares how white space inside the element is handled.

Text format
White Space

- 'white-space'
  - normal - This value directs user agents to collapse sequences of white space, and break lines as necessary to fill line boxes.
  - pre - This value prevents user agents from collapsing sequences of white space. Lines are only broken at newlines in the source.
  - nowrap - This value collapses white space as for 'normal', but suppresses line breaks within text.
  - pre-wrap - This value prevents user agents from collapsing sequences of white space. Lines are broken at newlines in the source as necessary to fill line boxes (not in Explorer).
  - pre-line - This value directs user agents to collapse sequences of white space. Lines are broken at newlines in the source as necessary to fill line boxes.

Text format
White Space

- 'white-space' (example)

```
<style type="text/css">
  #first { white-space: pre; font-size:xx-large; }
  #second { white-space: nowrap; font-size:xx-large; }
  #third { white-space: pre-wrap; font-size:xx-large; }
  #fourth { white-space: pre-line; font-size:xx-large; }
</style>
..
<p>Here, the spaces      are collapsed</p>
<p id="first">Here, the spaces      are not collapsed</p>
<p id="second">Here, the spaces…      </p>
<p id="third">Here, the spaces  …      </p>
<p id="fourth">Here, the spaces    …    </p>
</body>
```


Character Style
Font Family

- 'font-family'

  Value: [[<family-name> | <generic-family>] [, <family-name>| <generic-family>]*] | inherit

  Initial: depends on user agent

  Applies to: all elements

  Inherited: yes

  Percentages: N/A

  Media: visual

  Computed value: as specified

- This property allows choosing the family font.

Character Style
Font Family

- 'font-family'
  - <generic family> - a group of font families with a similar look (like "Serif" or "Monospace")
  - <font family> - a specific font family (like "Times New Roman" or "Arial")

Generic family	Font family	Description
Serif	Times New Roman Georgia	Serif fonts have small lines at the ends on some characters
Sans-serif	Arial Verdana	"Sans" means without - these fonts do not have the lines at the ends of characters
Monospace	Courier New Lucida Console	All monospace characters has the same width

Character Style
Font Family

- 'font-family'
  - The font-family property should hold several font names as a "fallback" system. If the browser does not support the first font, it tries the next font
    - Start with the font you want, and end with a generic family, to let the browser pick a similar font in the generic family, if no other fonts are available.
  - Serif fonts (stroke endings have ornamentations)

    p {font-family:"Times New Roman", Times, serif;}
  - Sans-Serif fonts (stroke endings are plain)

    p {font-family:Arial, Helvetica, sans-serif;}
  - Monospace Fonts (characters have the same fixed width)

    p {font-family:"Courier New", Courier, monospace;}

Character Style
Font Size

- 'font-size'

  Value: <absolute-size> | <relative-size> | <length> | <percentage> | inherit

  Initial: medium

  Applies to: all elements

  Inherited: yes

  Percentages: refer to parent element's font size

  Media: visual

  Computed value: absolute length
- The font-size property sets the size of the text.
- If you do not specify a font size, the default size for normal text, like paragraphs, is 16px (16px=1em).

Character Style
Font Size

- 'font-size'
  - <absolute-size> - is an index to a table of font sizes computed and kept by the user agent. Possible values are:

    [xx-small | x-small | small | medium | large | x-large | xx-large]
  - <relative-size> - is interpreted relative to the table of font sizes and the font size of the parent element. Possible values are: [larger | smaller].

    For example, if the parent element has a font size of 'medium', a value of 'larger' will make the font size of the current element be 'large'.

Character Style
Font Size

- 'font-size' (example)

```
<head>
   <title>  Text format  </title>
<style type="text/css">
h1 {font-size:300%;}   /* best solution */
h2 {font-size:xx-large;}
p {font-size:2em;}    /* em=16px */
em {font-size:larger}
</style>
 </head>
 <body>
   <h1>   This is <em>heading 1</em> </h1>
   <h2>   This is heading 2  </h2>
   <p>    This is a <em>paragraph</em>.   </p>
 </body>
</html>
```


Character Style
Font Style

- 'font-style'

  Value: normal | italic | oblique | inherit

  Initial: normal

  Applies to: all elements

  Inherited: yes

  Percentages: N/A

  Media: visual

  Computed value: as specified
- Differences between italic and oblique are not often perceptible.

Character Style
Font Weight

- 'font-weight'

  Value: normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit

  Initial: normal

  Applies to: all elements

  Inherited: yes

  Percentages: N/A

  Media: visual

  Computed value: see text

- The 'font-weight' property selects the weight of the font. The values '100' to '900' form an ordered sequence, where each number indicates a weight that is at least as dark as its predecessor.

Character Style
Font Weight

- 'font-weight'
  - 'normal' is synonymous with '400'
  - 'bold' is synonymous with '700'
  - 'bolder' selects the next weight that is assigned to a font that is darker than the inherited one.
  - 'lighter' selects the next lighter keyword with a different font from the inherited one.

- There is no guarantee that there will be a darker face for each of the 'font-weight' values; for example, some fonts may have only a normal and a bold face, while others may have eight face weights. There is no guarantee on how a UA will map font faces within a family to weight values.

Character Style
Font Weight

- 'font-weight' (example)

```
<head>
  <title> Text format </title>
<style type="text/css">
 div { font-weight: normal; border: thin solid red}
 div * {font-weight: bolder}
</style>
 </head>
 <body>
   <p>  Normal </p>
   <div>  Normal
     <p>  slightly bolder   </p>
   </div>
 </body>
```


Character Style
Font Variant

- 'font-variant'

  Value: normal | small-caps | inherit

  Initial: normal

  Applies to: all elements

  Inherited: yes

  Percentages: N/A

  Media: visual

  Computed value: as specified

- In a small-caps font the lower case letters look similar to the uppercase ones, but in a smaller size and with slightly different proportions.

Character Style
Line Height

- 'line-height'

  Value: normal | <number> | <length> | <percentage> | inherit

  Initial: normal

  Applies to: all elements

  Inherited: yes

  Percentages: refer to the font size of the element itself

  Media: visual

  Computed value: for <length> and <percentage> the absolute value; otherwise as specified

- Specifies the minimal height between two text lines (in particular, between the baselines, without considering the letters such ‘g’ and ‘q’).

Character Style
Line Height

- 'line-height'
  - <length> The specified length is used in the calculation of the line box height. Negative values are illegal.
  - <number> The used value of the property is this number multiplied by the element's font size. Negative values are illegal. The computed value is the same as the specified value.
  - <percentage> The computed value of the property is this percentage multiplied by the element's computed font size. Negative values are illegal.

Character Style
Line Height

- 'line-height' (example)

```
…
<style type="text/css">
 div { width: 200px; border: thin solid; margin: 30px}
.par1 { font-size: 12pt; line-height: 250% }
.par2 { font-size: 12pt; line-height: 0.6 }
</style>
</head>
<body>
<div>Normal flkmgnlkg sgksklgnszng lsgdknsdklgnsdgsg</div>
<div class="par1">New interline fddgdfgdhjkjhkhjk
thgfhfghfghfhfghhgfhr  </div>
<div class="par2"> New interline gdggfdgggdfg ddfgdfghseuheh
hherhhhtrhdf  </div> </body>
```


Character Style
Font

- 'font'

  Value: [[<'font-style'> || <'font-variant'> || <'font-weight'>]?<'font-size'> [/ <'line-height'>]? <'font-family'>] | caption | icon | menu | message-box | small-caption | status-bar | inherit

  Initial: see individual properties

  Applies to: all elements

  Inherited: yes

  Percentages: see individual properties

  Media: visual

  Computed value: see individual properties

  p { font: bold italic large Palatino, serif }

Character Style
Font

- 'font'
  - caption - The font used for captioned controls (e.g., buttons, drop-downs, etc.).
  - icon - The font used to label icons.
  - menu - The font used in menus (e.g., dropdown menus and menu lists).
  - message-box - The font used in dialog boxes.
  - small-caption - The font used for labeling small controls.
  - status-bar - The font used in window status bars.

```
button p { font: menu }
```


Università di Pisa
Francesco Marcelloni

Box Model

Università di Pisa
Francesco Marcelloni

Block and Inline Elements

- A block element is an element that takes up the full width available, and has a line break before and after it.
- Examples of block elements:
  <h1>
  <p>
  <div>

- An inline element only takes up as much width as necessary, and does not force line breaks.

- Examples of inline elements:
  <span>
  <a>

Università di Pisa
Francesco Marcelloni

Box Model

- All HTML elements can be considered as boxes. In CSS, the term "box model" is used when talking about design and layout.
- The CSS box model is essentially a box that wraps around HTML elements, and it consists of: margins, borders, padding, and the actual content.

Box Model

- Margin - does not have a background color, and it is completely transparent
- Border – is affected by the background color of the box
- Padding – is affected by the background color of the box
- Content - The content of the box, where text and images appear

Note: When you specify the width and height properties of an element with CSS, you are just setting the width and height of the content area.

Total element width = width + left padding + right padding + left border + right border + left margin + right margin

Total element height = height + top padding + bottom padding + top border + bottom border + top margin + bottom margin

Box Model Example

```
<head>
  <title> Text format </title>
<style type="text/css">
 div
 { width:220px; padding:10px; border:5px solid gray; margin:30px
 10px;}
</style>
</head>
<body>
  <div>This is a box. Total width = 270px </div>
  <div>This is a box. Total width = 270px </div>
</body>
```


Box Model Padding

- 'padding-top', 'padding-right', 'padding-bottom', 'padding-left'
  - **Value**: <padding-width> | inherit
  - **Initial**: 0
  - **Applies to**: all elements except table-row-group, table-header-group, table-footer-group, table-row, table-column-group and table-column
  - **Inherited**: no
  - **Percentages**: refer to width of containing block
  - **Media**: visual
  - **Computed value**: the percentage as specified or the absolute length
- Values for padding cannot be negative

Box Model Padding

- 'padding-top', 'padding-right', 'padding-bottom', 'padding-left'

  <padding-width>
  - <length>

    Specifies a fixed width.
  - <percentage>

    The percentage is calculated with respect to the width of the generated box's containing block, even for 'padding-top' and 'padding-bottom'.

Box Model Padding

- 'padding'

  Value: <padding-width>{1,4} | inherit

  Initial: see individual properties

  Applies to: all elements except table-row-group, table-header-group, table-footer-group, table-row, table-column-group and table-column

  Inherited: no

  Percentages: refer to width of containing block

  Media: visual

  Computed value: see individual properties

- This property is a shorthand property for setting 'padding-top', 'padding-right', 'padding-bottom', and 'padding-left' at the same place in the style sheet.

Box Model Padding

- 'padding'
  - If there is only one value, it applies to all sides.
  - If there are two values, the top and bottom paddings are set to the first value and the right and left paddings are set to the second.
  - If there are three values, the top is set to the first value, the left and right are set to the second, and the bottom is set to the third.
  - If there are four values, they apply to the top, right, bottom, and left, respectively.

```
h1 {
 background: white;
 padding: 1em 2em;
}
```


Box Model Padding

```
<style type="text/css">
    ul { padding: 3px 3px 3px 3px }
    li.pad30 {padding: 30px 30px 30px 30px}
            li.pad60 {padding: 60px 60px 60px 60px}
</style></head>
<body><ul>
<li>First element of list</li>
<li class="pad30">Second element with padding
 30</li>
<li class="pad60">Second element with padding
 60</li>
</ul></body>
```


Box Model Border Width

- 'border-top-width', 'border-right-width', 'border-bottom-width', 'border-left-width'

  Value: <border-width> | inherit

  Initial: medium

  Applies to: all elements

  Inherited: no

  Percentages: N/A

  Media: visual

  Computed value: absolute length; '0' if the border style is 'none' or 'hidden'

- These properties set the width of the top, right, bottom, and left border of a box.

Box Model
Border Width

- 'border-top-width', 'border-right-width', 'border-bottom-width', 'border-left-width'

  <border-width>
  - thin - a thin border.
  - medium – a medium border.
  - thick - a thick border.
  - <length> - the border's thickness has an explicit value. Explicit border widths cannot be negative.

The interpretation of the first three values depends on the user agent. The following relationships must hold, however:
'thin' <='medium' <= 'thick'.

Box Model
Border Width

- 'border-width'

  Value: <border-width>{1,4} | inherit
  Initial: see individual properties
  Applies to: all elements
  Inherited: no
  Percentages: N/A
  Media: visual
  Computed value: see individual properties

- This property is a shorthand property for setting 'border-top-width', 'border-right-width', 'border-bottom-width', and 'border-left-width' at the same place in the style sheet.

Box Model
Border Width

- 'border-width'
  - If there is only one value, it applies to all sides.
  - If there are two values, the top and bottom borders are set to the first value and the right and left are set to the second.
  - If there are three values, the top is set to the first value, the left and right are set to the second, and the bottom is set to the third.
  - If there are four values, they apply to the top, right, bottom, and left, respectively.

```
h1 { border-width: thin }         /* thin thin thin thin */
h1 { border-width: thin thick }   /* thin thick thin thick */
```


Box Model
Border Color

- 'border-top-color', 'border-right-color', 'border-bottom-color', 'border-left-color'

  Value: <color> | transparent | inherit
  Initial: the value of the 'color' property
  Applies to: all elements
  Inherited: no
  Percentages: N/A
  Media: visual
  Computed value: when taken from the 'color' property, the computed value of 'color'; otherwise, as specified

  - <color> - Specifies a color value.
  - transparent - The border is transparent (though it may have width).

Box Model
Border Color

- 'border-color'

  Value: [<color> | transparent]{1,4} | inherit
  Initial: see individual properties
  Applies to: all elements
  Inherited: no
  Percentages: N/A
  Media: visual
  Computed value: see individual properties

- The 'border-color' property can have from one to four values, and the values are set on the different sides as for 'border-width'.

  div {border-color: red yellow blue green}

Box Model
Border Style

- 'border-top-style', 'border-right-style', 'border-bottom-style', 'border-left-style'

  Value: <border-style> | inherit
  Initial: none
  Applies to: all elements
  Inherited: no
  Percentages: N/A
  Media: visual
  Computed value: as specified

- The border style properties specify the line style of a box's border (solid, double, dashed, etc.).

Box Model
Border Style

- 'border-top-style', 'border-right-style', 'border-bottom-style', 'border-left-style'

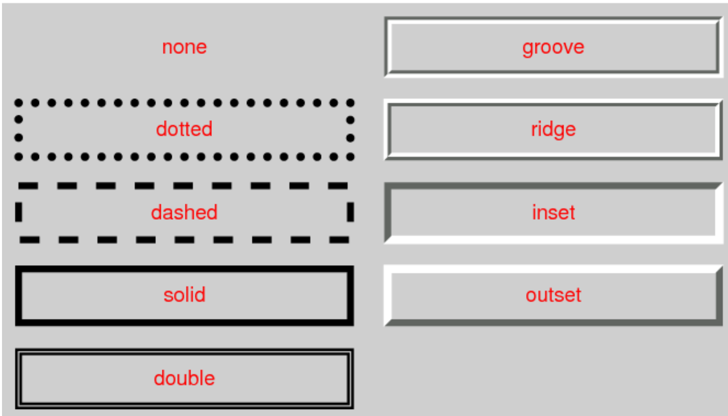
  <border-style>
  - none - no border; the computed border width is zero.
  - hidden - same as 'none', except in terms of border conflict resolution for table elements.
  - dotted - the border is a series of dots.
  - dashed - the border is a series of short line segments.
  - solid - the border is a single line segment.
  - double - the border is two solid lines. The sum of the two lines and the space between them equals the value of 'border-width'.
  - groove - the border looks as though it were carved into the canvas.

Box Model
Border Style

- 'border-top-style', 'border-right-style', 'border-bottom-style', 'border-left-style'

  <border-style> (continued)
  - ridge - the opposite of 'groove': the border looks as though it were coming out of the canvas.
  - inset - the border makes the box look as though it were embedded in the canvas.
  - outset - the opposite of 'inset': the border makes the box look as though it were coming out of the canvas.

Box Model
Border Style



Box Model
Border Style

- 'border-style'
  Value: <border-style>{1,4} | inherit
  Initial: see individual properties
  Applies to: all elements
  Inherited: no
  Percentages: N/A
  Media: visual
  Computed value: see individual properties

- The 'border-style' property sets the style of the four borders. It can have from one to four values, and the values are set on the different sides as for 'border-width' above.

```
#xy34 { border-style: solid dotted }
```


Box Model
Border Style

```
<style type="text/css">
    div { border-style: inset; border-width: thick;
        border-color: red; }
    li {border-color: blue}
    li.bor1 {border-style: groove; border-width:
        medium medium medium medium}
    li.bor2 {border-style: ridge; border-width:
        thin thin thin thin}
</style>
</head>
```


Box Model
Border Style

```
<body>
<div>
<ul>
<li>Element without specific border</li>
<li class="bor1">Element with groove border</li>
<li>Element without specific border</li>
<li class="bor2">Element with ridge border</li>
</ul>
</div>
</body>
```

Box Model
Border

- 'border-top', 'border-right', 'border-bottom', 'border-left'

  Value: [<border-width> || <border-style> || <'border-top-color'>] | inherit
  Initial: see individual properties
  Applies to: all elements
  Inherited: no
  Percentages: N/A
  Media: visual
  Computed value: see individual properties

- This is a shorthand property for setting the width, style, and color of the top, right, bottom, and left border of a box.

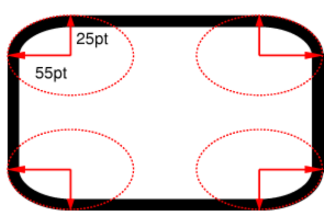
  h1 { border-bottom: thick solid red }

Box Model
Border radius

- border-top-left-radius, border-top-right-radius, border-bottom-right-radius, border-bottom-left-radius
- **Value**: [<length> || <percentage>] | inherit
  Initial: 0
  Applies to: all elements
  Inherited: no
  Percentages: Refer to corresponding dim. of the border box
  Media: visual
  Computed value: two absolute <length> or percentages

Box Model
Border radius

- The two length or percentage values of the 'border-*-radius' properties define the radii of a quarter ellipse that defines the shape of the corner of the outer border edge (see the diagram below). The first value is the horizontal radius, the second the vertical radius.



Box Model
Border radius

Example:

```
div {
   border: 2px solid #a1a1a1;
   padding: 10px 40px;
   background: #dddddd;
   width: 300px;
   border-radius: 25px;
}
```

Box Model
Border image

- border-image
- **Value**: <‘border-image-source’> || <‘border-image-slice’> [/ <‘border-image-width’> | / <‘border-image-width’>? / <‘border-image-outset’>]? || <‘border-image-repeat’>

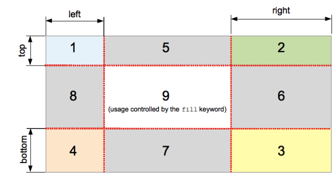
  Initial: See individual properties
  Applies to: See individual properties
  Inherited: no
  Percentages: N/A
  Media: visual
  Computed value: See individual properties

  Allows specifying that an image can be used in place of the border styles.

Box Model
Border image

Example:

```
<style>
div {
   background-color: lightgrey;
   border: 30px solid transparent;
   border-image: url('border.png');
   border-image-slice: 30;
   border-image-repeat: repeat;
}
</style>
```



Box Model
Box shadow

- Box-shadow

  Value: none | <shadow> [, <shadow>]*
  Initial: none
  Applies to: all elements
  Inherited: no
  Percentages: N/A
  Media: visual
  Computed value: any <length> made absolute; any specified color computed; otherwise as specified

  <shadow> = inset? && <length>{2,4} && <color>?

  The ‘box-shadow’ property attaches one or more drop-shadows to the box.

Box Model
Box shadow

- 1st <length>

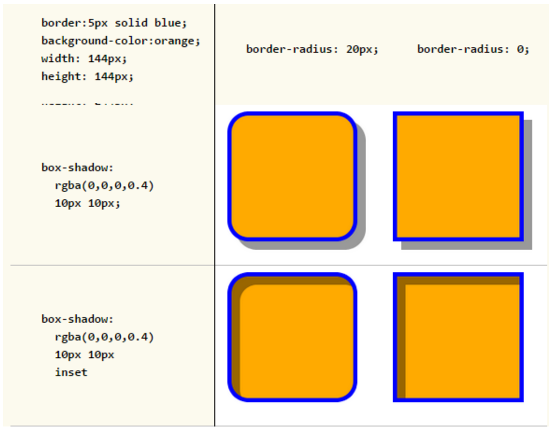
  Specifies the horizontal offset of the shadow. A positive value draws a shadow that is offset to the right of the box, a negative length to the left.
- 2nd <length>

  Specifies the vertical offset of the shadow. A positive value offsets the shadow down, a negative one up.
- 3rd <length>

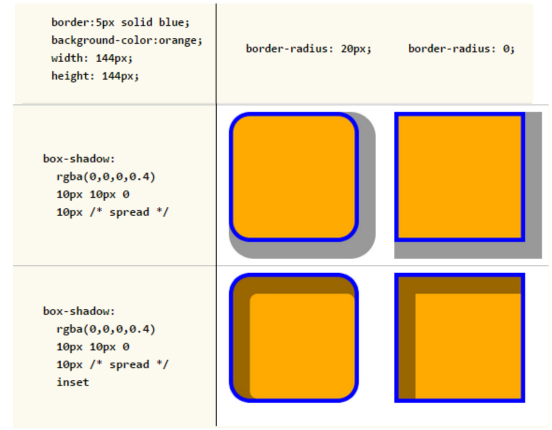
  Specifies the blur radius. Negative values are not allowed. If the blur value is zero, the shadow’s edge is sharp. Otherwise, the larger the value, the more the shadow’s edge is blurred.
- 4th <length>

  Specifies the spread distance. Positive values cause the shadow to expand in all directions by the specified radius. Negative values cause the shadow to contract.

Box Model
Box shadow



Box Model
Box shadow



Box Model
Margin

- 'margin-top', 'margin-bottom', 'margin-right', 'margin-left'
  - **Value**: <margin-width> | inherit
  - **Initial**: 0
  - **Applies to**: all elements except elements with table display types other than table-caption, table and inline-table
  - **Inherited**: no
  - **Percentages**: refer to width of containing block
  - **Media**: visual
  - **Computed value**: the percentage as specified or the absolute length
- These properties set the top, right, bottom, and left margin of a box.
  h1 { margin-top: 2em }

Box Model
Margin

- 'margin-top', 'margin-bottom', 'margin-right', 'margin-left'

<margin-width>
  - <length> - specifies a fixed width.
  - <percentage> - the percentage is calculated with respect to the width of the generated box's containing block.
  - auto – the margins are automatically computed based on the values of width, height, padding and borders in such a way that all the spaces in the box is occupied.

- Note: adjoining margins (no non-empty content, padding or border areas) of two or more boxes (which may be next to one another or nested) combine to form a single margin corresponding to the maximum value (collapsing margins).

Box Model
Margin

- 'margin'

  Value: <margin-width>{1,4} | inherit

  Initial: see individual properties

  Applies to: all elements except elements with table display types other than table-caption, table and inline-table

  Inherited: no

  Percentages: refer to width of containing block

  Media: visual

  Computed value: see individual properties
- The 'margin' property is a shorthand property for setting 'margin-top', 'margin-right', 'margin-bottom', and 'margin-left' at the same place in the style sheet.

Box Model
Margin

- 'margin'
  - If there is only one value, it applies to all sides.
  - If there are two values, the top and bottom margins are set to the first value and the right and left margins are set to the second.
  - If there are three values, the top is set to the first value, the left and right are set to the second, and the bottom is set to the third.
  - If there are four values, they apply to the top, right, bottom, and left, respectively.

Box Model
Margin (Example)

- 'margin' (example)

```
<style type="text/css">
   ul {
     background: yellow;
     margin: 12px 12px 12px 12px;
     padding: 3px 3px 3px 3px;
     }
   li {
     color: white;              /* text color is white */
     background: blue;          /* Content, padding will be blue */
     margin: 12px 12px 12px 12px;
     padding: 12px 0px 12px 12px; /* Note 0px padding right*/
     list-style: none           /* no glyphs before a list item */
}
```


Box Model
Margin (Example)

- 'margin' (example) (continued)

```
li.withborder {
       border-style: dashed;
       margin: 36px 36px 36px 36px;
       border-width: medium;  /* sets border width on all sides*/
       border-color: lime;
   }
</style>
…
<ul>
<li>First element of list</li>
<li class="withborder">Second element of list is a bit longer to
illustrate wrapping.</li>
</ul>
```

Box Model
Width

- 'width'

  Value: <length> | <percentage> | auto | inherit

  Initial: auto

  Applies to: all elements but non-replaced inline elements, table rows, and row groups

  Inherited: no

  Percentages: refer to width of containing block

  Media: visual

  Computed value: the percentage or 'auto' as specified or the absolute length; 'auto' if the property does not apply

- This property specifies the content width of boxes

Box Model
Height

- 'height'

  Value: <length> | <percentage> | auto | inherit

  Initial: auto

  Applies to: all elements but non-replaced inline elements, table columns, and column groups

  Inherited: no

  Percentages: refer to the height of the generated box's containing block

  Media: visual

  Computed value: the percentage or 'auto' or the absolute length; 'auto' if the property does not apply

- This property specifies the content height

Box Model
Width and Height (Example)

- 'width' and 'height' (example)

```
li {
      …
      height: 200px
    }

li.withborder {
      ..
    height: 100px
    }
```


Box Model
Positioning

- A box may be laid out according to three positioning schemes:
  - Normal flow. Normal flow includes block formatting of block boxes, inline formatting of inline boxes, relative positioning of block or inline boxes, and positioning of run-in boxes.
  - Floats. In the float model, a box is first laid out according to the normal flow, then taken out of the flow and shifted to the left or right as far as possible. Content may flow along the side of a float.
  - Absolute positioning. In the absolute positioning model, a box is removed from the normal flow entirely (it has no impact on later siblings) and assigned a position with respect to a containing block.

Box Model Positioning

- The 'position' and 'float' properties determine which of the positioning algorithms is used to calculate the position of a box.
- 'position'

  Value: static | relative | absolute | fixed | inherit
  Initial: static
  Applies to: all elements
  Inherited: no
  Percentages: N/A
  Media: visual
  Computed value: as specified

Box Model Positioning

- 'position'
  - static

    The box is laid out according to the normal flow. The 'top', 'right', 'bottom', and 'left' properties do not apply.

```
<head>
<title>Examples of positioning </title>
<style type="text/css">
img {position: static;}
</style>
</head>
<body>
<p>Example of image positioning </p>
<p> <img src="tower.jpg" alt="Tower"> </p>
<p>Example of image positioning</p>
</body>
```


Box Model Positioning

- 'position'
  - relative

    The box's position is calculated according to the normal flow (this is called the position in normal flow). Then the box is offset relative to its normal position.

    When a box B is relatively positioned, the position of the following box is calculated as though B were not offset.

```
<style type="text/css">
img {position: relative; left: 400px; top: 200px}
</style>
```


Box Model Positioning

- 'position'
  - absolute

    The box's position (and possibly size) is specified with the 'top', 'right', 'bottom', and 'left' properties. These properties specify offsets with respect to the first block containing the box in which a positioning is defined. Absolutely positioned boxes are taken out of the normal flow. This means they have no impact on the layout of later siblings. Also, though absolutely positioned boxes have margins, they do not collapse with any other margins.

```
<style type="text/css">
img {position: absolute; left: 400px; top: 200px}
</style>
```

Box Model
Positioning

- 'position'
  - fixed

    The position is calculated according to the 'absolute' model, but the box is fixed with respect to some reference.
    - The margins do not collapse with any other margins.
    - In the case of handheld, projection, screen, tty, and tv media types, the box is fixed with respect to the viewport and does not move when scrolled.
    - In the case of the print media type, the box is rendered on every page, and is fixed with respect to the page box, even if the page is seen through a viewport (in the case of a print-preview, for example).
    - For other media types, the presentation is undefined.

    Note: fixed does not work in Explorer

Box Model
Positioning

- 'position'
  - fixed (example)

```
<style type="text/css">
img {position:static; margin:30px}
img.fix {position: fixed; left: 400px; top: 200px}
</style>
</head>
<body>
<p><img src="tower.jpg" alt="Tower"></p>
<p><img src="tower.jpg" alt="Tower"></p>
<p><img src="tower.jpg" alt="Tower"></p>
<p><img class="fix" src="tower.jpg" alt="Tower"></p>
<p>Example of image positioning</p>
</body>
```


Box Model
Positioning - offset

- An element is said to be positioned if its 'position' property has a value other than 'static'. Positioned elements generate positioned boxes, laid out according to four properties:
- 'top', 'right', 'bottom', 'left'

  Value: <length> | <percentage> | auto | inherit
  Initial: auto
  Applies to: positioned elements
  Inherited: no
  Percentages: refer to edges of the containing block
  Media: visual

Box Model
Positioning – z-index

- 'z-index'

  Value: auto | <integer> | inherit
  Initial: auto
  Applies to: positioned elements
  Inherited: no
  Percentages: N/A
  Media: visual

- For a positioned box, the 'z-index' property specifies:
  - The stack level of the box in the current stacking context.
  - Whether the box establishes a local stacking context.

Box Model
Positioning – z-index

- 'z-index'
  - <integer> - This integer is the stack level of the generated box in the current stacking context. The box also establishes a local stacking context in which its stack level is '0'.
  - Auto - The stack level is the same as its parent's box. The box does not establish a new local stacking context.

  Each box has an integer stack level, which is its position on the z-axis relative to other boxes in the same stacking context.
  - Boxes with greater stack levels are always formatted in front of boxes with lower stack levels.
  - Boxes may have negative stack levels.
  - Boxes with the same stack level in a stacking context are stacked back-to-front according to document tree order.

Box Model
Positioning – z-index

- 'z-index' (example)

```
<head>
<style type="text/css">
#father1 {  position: absolute;  left: 10px; top: 10px;  width: 300px; height: 200px;  background-color: yellow;  z-index: 3;}
#father2 {  position: absolute;  left: 50px; top: 50px;  width: 300px; height: 200px;  background-color: red;  z-index: 2;}
#son1 { position: absolute; z-index: 1}
#son2 { position: absolute; z-index: 4}
</style>
</head>
```


Box Model
Positioning – z-index

- 'z-index' (example) (continued)

```
<body>
<div id="father1">    Father 1
<p id="son1"> Son 1 </p>
</div>
<div id="father2">    Father 2
<p id="son2"> Son 2 </p>
</div>
</body>
```


Box Model
Float

- 'float'

  Value: left | right | none | inherit
  Initial: none
  Applies to: all, but positioned elements
  Inherited: no
  Percentages: N/A
  Media: visual
- This property specifies whether a box should float to the left, right, or not at all. It may be set for any element, but only applies to elements that generate boxes that are not absolutely positioned.
- **Elements are floated horizontally**, this means that an element can only be floated left or right, not up or down.

Box Model
Float

- 'float'
  - left

    The element generates a block box that is floated to the left. Content flows on the right side of the box, starting at the top (subject to the 'clear' property).
  - right

    Similar to 'left', except the box is floated to the right, and content flows on the left side of the box, starting at the top.
  - none

    The box is not floated.

The element has to have an explicit width

Box Model
Float

- 'float' (example)

```
<style type="text/css">
img { float: none }     /* could be also omitted*/
</style>
</head>
<body>
  <p> float: none </p>
 <p><img src="tower.jpg" alt="Tower"></p>
  <p>  The text follows the normal flow   </p>
</body>

img { float: left }          img { float: right }
```


Box Model
Clear

- 'clear'

  Value: none | left | right | both | inherit
  Initial: none
  Applies to: block-level elements
  Inherited: no
  Percentages: N/A
  Media: visual

This property indicates which sides of an element's box(es) may not be adjacent to an earlier floating box. The 'clear' property does not consider floats inside the element itself or in other block formatting contexts.

Box Model
Clear

- 'clear'

  left - place the top border edge below the bottom outer edge of any left-floating boxes that resulted from elements earlier in the source document.

  right - place the top edge below the bottom outer edge of any right-floating boxes that resulted from elements earlier in the source document.

  both - place the top border edge below the bottom outer edge of any right-floating and left-floating boxes that resulted from elements earlier in the source document.

  none - no constraint on the box's position with respect to floats.

Box Model
Clear

- 'clear' (example)

```
<head>
  <title>  Examples of positioning  </title>
<style type="text/css">
 img { float: right }
 p {clear: right }
</style>
 </head>
 <body>
  <p> float: none </p>
    <img src="tower.jpg">
  <p>  The text follows the normal flow   </p>
 </body>
```


Visual Effects
Visibility

- 'visibility'

  Value: visible | hidden | collapse | inherit
  Initial: visible
  Applies to: all elements
  Inherited: yes
  Percentages: N/A
  Media: visual
- The 'visibility' property specifies whether the boxes generated by an element are rendered.
  - Invisible boxes still affect layout (set the 'display' property to 'none' to suppress box generation altogether).
- This property may be used in conjunction with scripts to create dynamic effects.

Visual Effects
Visibility

- 'visibility'
  - visible - the generated box is visible.
  - hidden - the generated box is invisible (fully transparent, nothing is drawn), but still affects layout.
    - Descendants of the element will be visible if they have 'visibility: visible'.
  - collapse –
    - In a table, it causes the entire row or column to be removed from the display, and the space normally taken up by the row or column to be made available for other content (It does not work in IE)
    - In elements other than rows, row groups, columns, or column groups, 'collapse' has the same meaning as 'hidden'.

Visual Effects
Visibility

- 'visibility' (example)

```
<style type="text/css">
#im1 { position: static; top: 2in; left: 2in; width: 2in;
       visibility: hidden; }
 #im2 { position: static;  top: 2in; left: 2in; width: 2in;}
 </style>
 ….
 <div id="im1">
 <img alt="Image 1" width="100" height="100"  src="tower.jpg">
   <p>Tower 1</p>
 </div>
 <div id="im2">
   <img alt="Image 2" width="100" height="100" src="tower.jpg">
   <p>Tower 2</p>
 </div>
```

Visual Effects
Overflow

- 'overflow'
  - **Value**: visible | hidden | scroll | auto | inherit
  - **Initial**: visible
  - **Applies to:** non-replaced block-level elements, table cells, and inline-block elements
  - **Inherited**: no
  - **Percentages**: N/A
  - **Media**: visual
- This property specifies whether content of a block-level element is clipped when it overflows the element's box. It affects the clipping of all of the element's content except any descendant elements whose containing block is the viewport or an ancestor of the element.

Visual Effects
Overflow

- 'overflow'
  - visible - the content is not clipped, i.e., it may be rendered outside the block box.
  - hidden - the content is clipped and that no scrolling user interface should be provided to view the content outside the clipping region.
  - scroll – indicates that the content is clipped and that if the user agent uses a scrolling mechanism that is visible on the screen (such as a scroll bar or a panner), that mechanism should be displayed for a box whether or not any of its content is clipped.
  - auto - the behavior of the 'auto' value is user agent-dependent, but should cause a scrolling mechanism to be provided for overflowing boxes

Visual Effects
Overflow

- 'overflow' (example)

```
<head>
 <title>
  Examples of visual effects
 </title>
<style type="text/css">
div {  overflow: hidden;  width: 100px; height: 45px;}
</style>
</head>
<body>
        <div> The text too large results to be hidden </div>
</body>
```


Visual Effects
Cursor

- 'cursor'
  - Value: [[<uri> ,]* [auto | crosshair | default | pointer | move | e-resize | ne-resize | nw-resize | n-resize | se-resize | sw-resize | s-resize | w-resize | text | wait | help | progress]] | inherit
  - **Initial**: auto
  - **Applies to**: all elements
  - **Inherited**: yes
  - **Percentages**: N/A
  - **Media**: visual, interactive
- This property specifies the type of cursor to be displayed for the pointing device when the cursor is upon the element

Visual Effects
Cursor

- 'cursor'
  - auto - the user agent determines the cursor to display based on the current context.
  - crosshair - a simple crosshair (e.g., short line segments resembling a "+" sign).
  - default - the platform-dependent default cursor. Often rendered as an arrow.
  - pointer - the cursor is a pointer that indicates a link.
  - move - indicates something is to be moved.
  - e-resize, ne-resize, nw-resize, n-resize, se-resize, sw-resize, s-resize, w-resize - indicate that some edge is to be moved. For example, the 'se-resize' cursor is used when the movement starts from the south-east corner of the box.

Visual Effects
Cursor

- 'cursor' (continued)
  - text - indicates text that may be selected. Often rendered as an I-beam.
  - wait - indicates that the program is busy and the user should wait. Often rendered as a watch or hourglass.
  - progress - a progress indicator. The program is performing some processing, but is different from 'wait' in that the user may still interact with the program. Often rendered as a spinning beach ball, or an arrow with a watch or hourglass.
  - help - help is available for the object under the cursor. Often rendered as a question mark or a balloon.

Visual Effects
Cursor

- 'cursor' (example)

```
<head>
 <title>
  Examples of visual effects
 </title>
<style type="text/css">
div {  cursor:text;  width: 100px; height: 45px;}
</style>
</head>
<body>
        <div> The text too large results to be hidden </div>
</body>
```


Visual Effects
Display

- 'display'

  Value: inline | block | list-item | run-in | inline-block | table | inline-table | table-row-group | table-header-group | table-footer-group | table-row | table-column-group | table-column | table-cell | table-caption | none | inherit
  Initial: inline
  Applies to: all elements
  Inherited: no
  Percentages: N/A
  Media: all

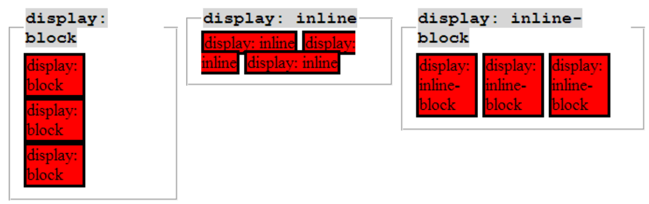
- This property is used to change the behavior of the elements: for instance, it can be used to transform a paragraph into inline, that is, does not form a new block of content.

Visual Effects
Display

- 'display'
  - block - this value causes an element to generate a block box.
  - inline - causes an element to generate one or more inline boxes.
  - inline-block - causes an element to generate a block box, which itself is flowed as a single inline box, similar to a replaced element. The inside of an inline-block is formatted as a block box, and the element itself is formatted as an inline replaced element.
  - list-item - causes an element (e.g., li in HTML) to generate a principal block box and a list-item inline box.

Visual Effects
Display

- <div style="width: 50px" ...> with different display



Visual Effects
Display

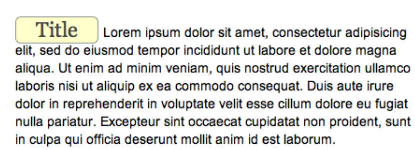
- 'display' (continued)
  - none - causes an element to not appear in the formatting structure (i.e., in visual media the element generates no boxes and has no effect on layout).
    - Descendant elements do not generate any boxes either; the element and its content are removed from the formatting structure entirely.
    - This behavior cannot be overridden by setting the 'display' property on the descendants.
  - run-in - this value creates either block or inline boxes, depending on context. Properties apply to run-in boxes based on their final status (inline-level or block-level).

Visual Effects
Display

- display: run-in example

```
h2 {
 font: normal 1.5em Georgia, serif;
 color: #454545;
 margin: 0 5px 5px 0;
 display: run-in;
 padding: 1px 1em;
 background: #ffc;
 border: 1px solid #999;
 border-radius: 6px;
}

<h2>...</h2>
<p>...</p>
```



Visual Effects Display

- 'display' (continued)
  - table, inline-table, table-row-group, table-column, table-column-group, table-header-group, table-footer-group, table-row, table-cell, and table-caption - cause an element to behave like a table element.
- Changing an inline element to a block element, or vice versa, can be useful for making the page look a specific way, and still follow web standards

```
li {display:inline;}
span {display:block;}
```


Navigation Bars

- Navigation bar from a standard HTML list

```
<head>
<style>
ul { list-style-type:none; margin:0; padding:0; }
</style>
</head>
```

- list-style-type:none - Removes the bullets. A navigation bar does not need list markers
- Setting margins and padding to 0 to remove browser default settings

Navigation Bars

```
<style type="text/css">
 ul { list-style-type:none; margin:0; padding:0; }
</style>
</head>
<body>
<ul>
<li><a href="#home">Home</a></li>
<li><a href="#news">News</a></li>
<li><a href="#contact">Contact</a></li>
<li><a href="#about">About</a></li>
</ul>
</body>
</html>
```


Navigation Bars

```
<style type="text/css">
 ul { list-style-type:none; margin:0; padding:0; }
 li {display:inline}
</style>
</head>
<body>
<ul>
<li><a href="#home">Home</a></li>
<li><a href="#news">News</a></li>
<li><a href="#contact">Contact</a></li>
<li><a href="#about">About</a></li>
</ul>
</body>
</html>
```

Horizontal Navigation Bars

- Alternative solution

```
<style type="text/css">
a {display:inline}
</style>
</head>
<body>
<p><a href="#home">Home</a>
<a href="#news">News</a>
<a href="#contact">Contact</a>
<a href="#about">About</a></p>
</body>
```


Vertical Navigation Bars

- To build a vertical navigation bar we only need to style the <a> elements, in addition to the code above:

```
a { display:block;}
```

Tecnologie Web

JavaScript

Francesco Marcelloni

Dipartimento di Ingegneria dell'Informazione
Università di Pisa
ITALY

What is JavaScript?

- JavaScript is a **scripting language**.
- JavaScript **works on the client side**, which means that it runs on the user's computer and not on the Web server.
- JavaScript **is designed for use on Web pages** and is closely integrated with HTML.
- **JavaScript statements** embedded in an HTML page **can recognize and respond to User Events** such as Mouse Clicks, Form Input, and Page Navigation.

Scripting Languages

- Scripts offer authors a means to extend HTML documents in highly active and interactive ways. For example:
  - Scripts may be evaluated as a document loads to modify the contents of the document dynamically.
  - Scripts may accompany a form to process input as it is entered.
  - Scripts may be triggered by events that affect the document, such as loading, unloading, element focus, mouse movement, etc.
  - Scripts may be linked to form controls (e.g., buttons) to produce graphical user interface elements.

Main Features

Like all the scripting languages, JavaScript:

- is interpreted (easy to test program code, look at the results, make changes, and test it)
- contains a limited and easy-to-learn command set and syntax
- is designed for performing a well-defined set of tasks
- is suited to producing small programs
- is especially well designed for repetitive, event-invoked tasks
- is integrated into the browser and can interact directly with the HTML pages
- JavaScript makes it possible to program responses to user events such as mouse clicks and data entry in forms

Main Features

Strengths	Weakness
Quick Development	Limited Range of Built-in Methods
Easy to Learn	Generally, only client-side script
Platform Independence	No Code Hiding
Small Overhead	

JavaScript Objectives

- To make HTML pages dynamic, interactive and effective through graphics and animations.
- To develop client-side applications with particular functions on the HTML object management
- To validate the data input from the users
- To manage search tables on the client
- To store and retrieve the cookies so as to obtain information on the client-server interaction

JavaScript Standard

- JavaScript was invented by Netscape and was first used in Netscape browsers.
- Now, all the browsers support JavaScript
- ECMAScript, standardized version of JavaScript, is documented in the ECMA-262 specification (5.1 Edition).
- ECMA (European Computer Manufacturers Association). ECMA is an international standards association for information and communication systems.
- The ECMA-262 standard is also approved by the ISO (International Organization for Standards) as ISO-16262.

What cannot you do with JavaScript

- You cannot access or write to server files
- You do, however, have access to documents that are loaded into other browser windows and frames, provided that the documents are from the same server
- You also cannot touch the local hard drive, send a print job, or edit the user's bookmarks or browser preferences. NOTE: The exception are cookies – which JavaScript can read from the hard drive and write to the hard drive.

JavaScript versus Object-Oriented Languages

JavaScript	Java-C++
Object-based. Code uses built-in, extensible objects, but no classes or inheritance.	Object-oriented. Applets consist of object classes with inheritance.
Variable data types not declared (loose typing).	Variable data types must be declared (strong typing).
You can write fully-functional JavaScript using a simple text editor	To write fully-functional Java/C++, you need the Java/C++ Developer's Kit.

How to insert a JavaScript into an HTML page?

- Use the <script> tag. Inside the <script> tag use the type attribute to define the scripting language.

```
<html>
<body>
<script type="text/javascript">
...
</script>
</body>
</html>
```

- The script element may appear any number of times in the head or body of an HTML document.

How to insert a JavaScript into an HTML page?

- <script> tag
  - src = uri

    This attribute specifies the location of an external script.
  - type = content-type

    This attribute specifies the scripting language of the element's contents and overrides the default scripting language.
    - The scripting language is specified as a content type (e.g., "text/javascript").
    - Authors must supply a value for this attribute. There is no default value for this attribute.

How to insert a JavaScript into an HTML page?

- Example
  - The document.write command is a standard JavaScript command for writing output to a page.
  - By entering the document.write command between the <script> and </script> tags, the browser will recognize it as a JavaScript command and execute the code line.

```
<html>
<body>
<script type="text/javascript">
  document.write("Hello World!");
</script>
</body>
</html>
```

How to handle simple browsers?

- Browsers that do not support JavaScript will display JavaScript as page content.
- To prevent them from doing this, and as a part of the JavaScript standard, the HTML comment tag should be used to "hide" the JavaScript.
- Just add an HTML comment tag <!-- before the first JavaScript statement, and a --> (end of comment) after the last JavaScript statement.

```
<html>
<body>
<script type="text/javascript">
<!--
document.write("Hello World!");
-->
</script>
</body>
</html>
```


Università di Pisa Francesco Marcelloni

Where to insert the JavaScript Code

Università di Pisa Francesco Marcelloni

Where to Put the JavaScript? Head solution

- When in the head, JavaScripts in a page will be executed immediately while the page loads into the browser.
- To prevent the automatic execution, the script has to be inserted into a function.
- Functions can be put in the head section, thus they do not interfere with page content

```
<html>
<head>
<script type="text/javascript">
function message()
{
alert("This alert box was called
with the onload event");
}
</script>
</head>
<body onload="message()">
</body>
</html>
```


Università di Pisa Francesco Marcelloni

Where to Put the JavaScript? Body Solution (not recommended)

- If you don't want your script to be placed inside a function, or if your script should write page content, it should be placed in the body section
- You can place an unlimited number of scripts in your document, so you can have scripts in both the body and the head section.

```
<html>
<head>
</head>

<body>
<script type="text/javascript">
document.write("This message
is written by JavaScript");
</script>
</body>
</html>
```

Note: we could have problems whether different scripts depend on each other

Where to Put the JavaScript? External File

- If you want to run the same JavaScript on several pages, without having to write the same script on every page, you can write a JavaScript in an external file.
- Save the external JavaScript file with a .js file extension.
- Note: The external script cannot contain any HTML tags (in particular, <script></script> tags!)
- To use the external script, point to the .js file in the "src" attribute of the <script> tag:

```
<html>
<head>
<script type="text/javascript"
        src="xxx.js"></script>
</head>
<body>
</body>
</html>
```


The default scripting language

- Authors should specify the default scripting language for all scripts in a document by including the following meta declaration in the head:

  <meta http-equiv="Content-Script-Type" content="type">

  where "type" is a content type naming the scripting language. Examples of values include "text/tcl", "text/javascript", "text/vbscript".

- The tag script allows using scripting languages different from the default.

The noscript element

- The noscript element allows authors to provide alternate content when a script is not executed.
- Its content should only be rendered if:
  - The user agent is configured not to evaluate scripts.
  - The user agent does not support a scripting language invoked by a script element earlier in the document.
- Example:

  <noscript> <meta http-equiv="Refresh" content="2"; url="paginasenzaJavaScript.html"></noscript>
- If the noscript element is rendered, the user is re-addressed after 2 seconds to the url specified in the noscript element
- Use both comments and the noscript element

When is a script executed?

- Global code (not in the body of functions):
  - is executed when it is met during the rendering of the page. The global code can be in the HTML page or in an external file;
- Code in functions
  - is executed only if the function is called
- Event onLoad
  - The code corresponding to the event is executed when the page is loaded on the browser and after the execution of the code external to each function

When is a script executed?

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
<title>Execution order</title>
<script type="text/javascript">
<!-- HTML comment
```


When is a script executed?

```
  window.alert("Not in function");
  document.write("<h1> This is a heading. </h1>");
  document.write("<p>This is a paragraph.</p>");
  document.write("<p>This is another paragraph.</p>");
  function loading() {  window.alert("onLoad event");} -->
</script>
</head>
<body onLoad="loading()" >
<script type="text/javascript">
  window.alert("Code in Body");
</script>
</body>
</html>
```


JavaScript Statements

JavaScript Code

- JavaScript code is a sequence of Javascript statements
- A JavaScript statement is a command to a browser.
  document.write("Hello Dolly");
- The semicolon is optional (according to the JavaScript standard), and the browser is supposed to interpret the end of the line as the end of the statement.
  - Note: Using semicolons makes it possible to write multiple statements on one line.
- JavaScript is case sensitive

JavaScript Statements

- Statements are executed by the browser in the sequence they are written.

Example

```
<script type="text/javascript">
<!-- document.write("<h1>Introduction</h1>");
  document.write("<p>In this course we will introduce <a href=\"http://devedge-temp.mozilla.org/central/javascript/index_en.html\">
        Javascript</a></p>");
  document.write("<h2>Javascript Statements</h2>");
  document.write("<p>A Javascript statement is ... </p>"); -->
</script>
</head>
<body><p>Example</p></body>
```


JavaScript Blocks

- JavaScript statements can be grouped together in blocks.
  - Blocks start with a left curly bracket { and ends with a right curly bracket }.
  - The purpose of a block is to execute the sequence of statements together.

Example

```
<script type="text/javascript">
{ document.write("<h1>This is a heading</h1>");
  document.write("<p>This is a paragraph.</p>");
  document.write("<p>This is another paragraph.</p>");
}
</script>
```


JavaScript Comments

- Comments can be added to explain the JavaScript, or to make the code more readable.
  - Single line comments start with //.
  - Multi line comments start with /* and end with */.

```
<script type="text/javascript">
/*
The code below will write one heading and two paragraphs
*/
document.write("<h1>This is a heading</h1>");
// Write a heading
document.write("<p>This is a paragraph.</p>"); //paragraph
document.write("<p>This is another paragraph.</p>");
</script>
```


JavaScript Variables

- Rules for JavaScript variable names:
  - Variable names are case sensitive (y and Y are two different variables)
  - The first character in the name must be a letter (a-z or A-Z) or an underscore (_).
  - The rest of the name can be made up of letters (a-z or A-Z), numbers (0-9), or underscores (_).
  - Names should describe what variables are.
- The type of the variable is not specified

JavaScript Variables

- JavaScript allows declaring variables by simply using them
- Anyway, declaring variables helps to ensure that programs are well organized and helps to keep track of the scope of variables
- To declare JavaScript variables you can use the var statement

```
var x;
var carname;
```

After the declaration, the variables are empty (they have no value yet)

JavaScript Variables

- Variables can be initialized

```
var x=5;
var carname="Volvo";
```

- If you assign values to variables that have not been declared yet, the variables will automatically be declared.
- If you redeclare a JavaScript variable, it will not lose its original value.

```
var x=5;
var x;
```

- NOTE: the variable x will still have the value of 5. The value of x is not reset when you redeclare it.

Loosely typed

- JavaScript is what is called a loosely typed programming language:
  - the type of a variable is not defined when a variable is created and can, at times, change based on the context.

```
var text1 = "19";
var num1 = 96;
num1 = text1 + num1;
```

The variable num1 contains "1996".

Types of Values

- JavaScript recognizes the following types of values:
  - Numeric
  - String
  - Boolean
  - Null – a special keyword denoting a null value; null is also a primitive value
  - Undefined - a top-level property whose value is undefined;
    - undefined is also a primitive value (NaN in numeric contexts).

Numeric Values

- Integer

NUMBER SYSTEM	NOTATION
Decimal (base 10)	A normal integer without a leading 0 (zero) (ie, 752)
Octal (base 8)	An integer with a leading 0 (zero) (ie, 056)
Hexadecimal (base 16)	An integer with a leading 0x or 0X (ie, 0x5F or 0XC72)

- Floating Point Values
  - 2.3e-3
  - 2.3E-3

String Values

- String

contains zero or more characters enclosed in single or double quotes

- NOTE: the empty string is distinct from the null value
- NOTE: Strings are different from other data types. Strings are actually Objects.
- The backslash (\) is used to insert apostrophes, new lines, quotes, and other special characters into a text string

Code	Outputs
\'	single quote
\"	double quote
\&	ampersand
\\	backslash

String type

Escape characters

Character	Description
\n	new line
\t	tab
\r	carriage return
\f	form feed
\b	backspace

NOTE: the escape characters only work in the following situations:

- within <pre> tags
- alert(), confirm() and prompt()
- within <textarea> tags

Boolean and null Values

- boolean
  - Note: Values of 1 and 0 are not considered Boolean values in JavaScript
- null Value
  - Represents Nothing
- NaN – Not a Number (returned by some functions like parseInt() and parseFloat())

Variable Scope

```
<script type="text/javascript">
<!-- HTML comment
var cc = 0 ;          //global
var dd = scr();    // global
document.writeln("global: " + cc); // print value of cc
document.writeln("local: " + dd);  // print value of dd
function scr() {
  var cc = 3;    //local variable hides the global variable cc
// without var, it would be an assignment to global variable cc
  return cc;
}
-->
</script>
</head>
```


JavaScript Operators
Arithmetic Operators

- Let us assume that y=5

Operator	Description	Example	Result
+	Addition	x=y+2	x=7
-	Subtraction	x=y-2	x=3
*	Multiplication	x=y*2	x=10
/	Division	x=y/2	x=2.5
%	Modulus (division remainder)	x=y%2	x=1
++	Increment	x=++y x=y++	x=6 x=5
--	Decrement	x=--y x=y--	x=4 x=5

JavaScript Operators
Assignment Operators

Operator	Example	Same As	Result
=	x=y		x=5
+=	x+=y	x=x+y	x=10
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=25
/=	x/=y	x=x/y	x=5
%=	x%=y	x=x%y	x=0

JavaScript Operators
The + Operator used on Strings

- The + operator can also be used to add string variables or text values together

```
txt1="This is a very";
txt2="nice day";
txt3=txt1+" "+txt2;
```

- Note: if you add a number and a string, the result will be a string

```
<script type="text/javascript">
x="5"+"5";
document.write(x);
document.write("<br>");            //55
x=5+"5";
document.write(x);
document.write("<br>");            //55
</script>
```

JavaScript Operators
Comparison Operators

- Given x=5, the table below explains the comparison operators

Operator	Description	Example
==	is equal to	x==8 is false x=='5' is true
===	is exactly equal to (value and type)	x===5 is true x==='5' is false
!=	is not equal (it attempts conversion)	x!=8 is true x!=5 is false
!==	is not equal and/or not of the same type	x!=='5' is true
>	is greater than	x>8 is false
<	is less than	x<8 is true
>=	is greater than or equal to	x>=8 is false
<=	is less than or equal to	x<=8 is true

JavaScript Operators
Comparison Operators

- Comparison operators
  - If either or both values are NaN, then they are not equal.
  - Objects, arrays, and functions are compared by reference. This means that two variables are equal only if they refer to the same object.
  - If both are null, or both undefined, they are equal.
  - If one value is null and one undefined, they are equal.

- Two separate arrays are never equal by the definition of the == operator, even if they contain identical elements.

JavaScript Operators
Logical Operators

- Given x=6 and y=3, the table below explains the logical operators

Operator	Description	Example
&&	and	(x < 10 && y > 1) is true
||	or	(x==5 || y==5) is false
!	not	!(x==y) is true

JavaScript Operators
Bitwise Operators

Operator	Description	Example
&	and	a & b
|	or	a | b
!	xor	a ^ b
~	not	~a
<<	Left shift	a<<b
>>	Sign-propagating right shift	a>>b
>>>	Zero-fill right shift	a>>>b

JavaScript Operators

- If the types of the two values differ, attempt to convert them into the same type so they can be compared:
  - If one value is a number and the other is a string
    - convert the string to a number and try the comparison again, using the converted value.
  - If either value is true
    - convert it to 1 and try the comparison again.
  - If either value is false
    - convert it to 0 and try the comparison again.
  - If one value is an object and the other is a number or string
    - convert the object to a primitive value by either its toString() method or its valueOf() method. Native JavaScript classes attempt valueOf() conversions before toString() conversion.
  - Any other combinations of types are not equal.

JavaScript Operators

- Conditional Operator

  (condition) ? **val1** : **val2**

  Example:

```
var username = prompt("Please enter your name", "");
var greeting = "Hello ";
greeting += ((username != null) ? username : "guy");
```


JavaScript Operators
typeof operator

- typeof operator

  Two ways:
  1. typeof operand
  2. typeof (operand)
- The typeof operator returns a string indicating the type of the operand. The parentheses are optional.

```
var num1 = 3; var num2 = 3.0;
var bool=true; var shape="round";
typeof num1;     // returns number
typeof num2;     // returns number
typeof bool;     // returns boolean
typeof shape;    // returns string
```


JavaScript Operators
void operator

- void operator

  Two ways:
  1. void (expression)
  2. void expression
- The void operator specifies a JavaScript expression to be evaluated without returning a value. The parentheses surrounding the expression are optional, but it is good style to use them.
- The following code creates a hypertext link that submits a form when the user clicks it.

```
<a href="javascript:void(document.form.submit())">
Click here to submit</a>
```

Conditional Statements
if statement

```
if (condition)
 {
 code to be executed if condition is true
 }
```

NOTE: "if" is written in lowercase letters. Using uppercase letters (IF) will generate a JavaScript error!

Conditional Statements
if … else statement

```
if (condition)
 {
 code to be executed if condition is true
 }
else
 {
 code to be executed if condition is not true
 }
```


Conditional Statements
if … else if … else statement

```
if (condition1)
 {
 code to be executed if condition1 is true
 }
else if (condition2)
 {
 code to be executed if condition2 is true
 }
...
else
 {
 code to be executed if condition1 and condition2 are not true
 }
```


Conditional Statements
if … else if … else statement

```
<script type="text/javascript">
var d = new Date();
var time = d.getHours();
if (time<12)
{
document.write("<em>Good
  morning</em>");
}
else if (time>=12 && time<17)
{
document.write("<em>Good
  afternoon</em>");
}
else if (time>=17 && time<20)
{
document.write("<em>Good
  evening</em>");
}
else
{
document.write("<em>Good
  night!</em>");
}
</script>
```

Conditional Statements switch statement

```
switch(n)
{
case 1:
  execute code block 1
  break;
case 2:
  execute code block 2
  break;
default:
  code to be executed if n is different from case 1 and 2
}
```


Conditional Statements switch statement

```
<head>
<meta charset="utf-8">
<title>Example</title>
<style type="text/css">
p {color: white; background-color: grey; }
p.sat {color: red; background-color: black; }
p.sun {color: green; background-color: red; }
</style>
<script type="text/javascript">
<!-- HTML comment
var d=new Date();
theDay=d.getDay();
```


Conditional Statements switch statement

```
switch (theDay)
{ case 6:
    document.write("<p class='sat'>Super Saturday</p>");
    break;
  case 0:
   document.write("<p class='sun'>Sleepy Sunday</p>");
   break;
default:
  document.write("<p>I'm looking forward to this weekend!</p>");
}
-->
</script>
</head>
```


Loop statements for

```
for (var=startvalue;var OP endvalue;var=var+increment)
{
code to be executed
}
```

where OP is any comparison operator.

Example

```
<script type="text/javascript">
var i=0;
for (i=0;i<=5;i++)
        { document.write("<p>The number is " + i + "</p>");}
</script>
```

Loop statements
while

```
while (var OP endvalue)
 {
 code to be executed
 }
```

where OP is any comparison operator.

Example

```
<script type="text/javascript">
var i=0;
while (i<=5)
        {document.write("<p>The number is " + i++ + "</p>");}
</script>
```


Loop statements
do...while

```
do
 {
code to be executed
}
while (var OP endvalue);
```

where OP is any comparison operator.

Example

```
<script type="text/javascript">
var i=0;
do
 {document.write("<p>The number is " + i++ + "</p>");}
 while (i<=5)
</script>
```


Break statement

- The break statement will break the loop and continue executing the code that follows after the loop (if any).

```
<script type="text/javascript">
var i=0;
while (i<=5)
 { document.write("<p>The number is " + i++ + "</p>");
   if (i==4) break;
 }
</script>
</head>
```


Continue statement

- The continue statement will break the current loop and continue with the next value

```
<script type="text/javascript">
var i=0;
for (i=0;i<=10;i++)
        { if (i%4) continue;
        document.write("<p>The number is " + i + "</p>");
        }
</script>
```

Label statement

- A label provides a statement with an identifier that lets you refer to it elsewhere in your program.

  label : statement
- The value of label may be any JavaScript identifier that is not a reserved word.
- On using label with break and continue
  - break [*label*] - terminates the specified enclosing label statement
  - continue [*label*] - restarts a label statement or continues execution of a labelled loop with the next iteration

continue statement (example)

```
var i=0, j=8;
checki : while (i<8) {
        document.write("<p>i = " + i + "</p>");
        checkj : while (j>0) {
            j--;i++;
            if ((j%3)==0) continue checki;
            if ((j%2)==0) continue checkj;
             document.write("<p>j = " + j + " is odd.</p>");
        }
}
```


Loop statements for .. in

- The for...in statement loops through the elements of an array or through the properties of an object

```
for (variable in object)
 {
 code to be executed
 }
```

- Note: The code in the body of the for...in loop is executed once for each element/property.
- Note: The variable argument can be a named variable, an array element (not an index), or a property (not the value of the property, but the name) of an object.

Loop statements for .. in

```
<script type="text/javascript">
<!-- HTML comment
var x;
var mycars = new Array();
mycars[0] = "Saab";
mycars[1] = "Volvo";
mycars[2] = "BMW";

for (x in mycars)
  document.write("<p>" + x + ": " + mycars[x] + "</p>");
-->
</script>
```

with statement

- The with statement establishes the default object for a set of statements.
- JavaScript looks up any unqualified names within the set of statements to determine if the names are properties of the default object. If an unqualified name matches a property, then the property is used in the statement; otherwise, a local or global variable is used.
- A with statement looks as follows:

```
with (object){
statements
}
```

```
var a, x, y;
var r=10;
with (Math) {
a = PI * r * r;
x = r * cos(PI);
y = r * sin(PI/2);
}
```


Functions

```
function name(var1,var2,...,varX)
{ var x;
  some code
  return x;
}
```

- You may call a function from anywhere within a page (or even from other pages if the function is embedded in an external .js file).
- Functions can be defined both in the <head> and in the <body> section of a document. However, to assure that a function is read/loaded by the browser before it is called, it could be wise to put functions in the <head> section.

Functions

- All parameters are passed to functions by value; the value is passed to the function, but if the function changes the value of the parameter, this change is not reflected globally or in the calling function.
- Objects are passed by reference: if the function changes the object's properties, that change is visible outside the function.
- A function can even be recursive, that is, it can call itself.

Functions

```
<script type="text/javascript">
function fun(s) {      // function definition
  var a = 10;
  var c = 10 * s;
  document.write("<p>This is the result: " + c + "</p>");
  }
</script>
</head>

<body onLoad="fun(10)">
<!-- call to the function fun -->
<script type="text/javascript">
  window.alert("Code in Body");
</script>
```

Functions

- The arguments of a function are maintained in the array "arguments".
- Within a function, you can address the parameters passed to it by:
  arguments[i]
  functionName.arguments[i]
  where i is the ordinal number of the argument, starting at zero.
  arguments[0] -> the first argument passed to a function.
- The total number of arguments is indicated by arguments.length.

Functions

- Using the arguments array, you can call a function with more arguments than it is formally declared to accept.
  - This is often useful if you do not know in advance how many arguments will be passed to the function.
  - You can use arguments.length to determine the number of arguments actually passed to the function, and then treat each argument using the arguments array.
- No check on the order of the parameters

Functions

```
<script type="text/javascript">
<!-- HTML comment
function myConcat(separator) {
result=""; // initialize list
// iterate through arguments
for (var i=1; i<arguments.length; i++) {
result += arguments[i] + separator;
}
result += "<br>";
return result;
}
-->
</script>
</head>
```


Functions

```
<body>
<script type="text/javascript">
<!-- HTML comment
// returns "red, orange, blue, "
document.write(myConcat(", ","red","orange","blue"));
// returns "elephant; giraffe; lion; cheetah;"
document.write(myConcat(";","elephant","giraffe","lion","cheetah"));
// returns "sage. basil. oregano. pepper. parsley. "
document.write(myConcat(".","sage","basil","oregano","pepper",
  "parsley"));
-->
</script>
```

The return statement

- The return statement is used to specify the value that is returned from the function.

```
<head>
<script type="text/javascript">
function product(a,b)
{ return a*b; }
</script>
</head>
<body>
<script type="text/javascript">
document.write(product(4,3));
</script>
```


Predefined functions

- JavaScript has several top-level predefined functions:
  - eval(string) - Evaluates a string and executes it as if it was script code
    eval("x=10;y=20;document.write(x*y)"); //200
  - isFinite - Determines whether a value is a finite, legal number
    document.write(isFinite(123)+ "<br>"); //true
    document.write(isFinite("2005/12/12")+ "<br>"); //false
  - isNaN -The isNaN() function determines whether a value is an illegal number (Not-a-Number).
    This function returns true if the value is NaN, and false if not.
    - document.write(isNaN(123)+ "<br>"); //false
    - document.write(isNaN("2005/12/12")+ "<br>"); //true

Predefined functions

- parseInt(string,radix)
  Parses a string and returns an integer of the specified radix (base).
  radix - a number (from 2 to 36) that represents the numeral system to be used
- parseFloat(string)
  Parse a string and returns a float number.

  If the first character cannot be converted to a number, the two functions return NaN.
- number(object) and string(object)
  Converts the object argument to a number or to a string that represent the object's value.
  If the value cannot be converted to a legal number, NaN is returned.

Predefined functions

```
<body>
<script type="text/javascript">
<!-- HTML comment
eval("x=10;y=20;document.write(x*y)");
document.write("<br>" + isFinite(123)+ "<br>");
document.write(isFinite("2005/12/12")+ "<br>");
document.write(isNaN(123)+ "<br>");
document.write(isNaN("2005/12/12")+ "<br>");
document.write(parseInt("His age is 40 years")+ "<br>");
document.write(parseInt("40 years")+ "<br>");
-->
</script>
</body>
```

Predefined functions

- escape(string) and unescape(string)

  The escape() function encodes a string. This function makes a string portable, so it can be transmitted across any network to any computer that supports ASCII characters. This function encodes special characters, with the exception of: * @ - _ + . /

  The unescape() function decodes an encoded string.

```
<script type="text/javascript">
<!-- HTML comment
str="Need tips? Visit W3Schools!";
str_esc=escape(str);
document.write(str_esc + "<br>");
document.write(unescape(str_esc));
-->
</script>
```


document.write() and document.writeln() methods

- document.write(exp1,exp2,exp3,...)

  The write() method writes HTML expressions or JavaScript code to a document.

  Multiple arguments can be listed and they will be appended to the document in order of occurrence

- document.writeln(exp1,exp2,exp3,...)

  The writeln() method is identical to the write() method, with the addition of writing a newline character after each statement.

Since HTML ignores the newline characters, the effects of the methods on the HTML pages are equal except when used within <pre> tags

The document.write() and document.writeln() methods

```
<body>
<pre>
<script type="text/javascript">
  document.write("Hello World!");
  document.write("Have a nice day!");
</script>
</pre>
<pre>
<script type="text/javascript">
  document.writeln("Hello World!");
  document.writeln("Have a nice day!");
</script>
</pre>
</body>
```


Popup Boxes
Alert Box

- An alert box is often used if you want to make sure information comes through to the user.
- When an alert box pops up, the user will have to click "OK" to proceed.

```
alert("sometext");
```

Popup Boxes
Alert Box

```
<script type="text/javascript">
function show_alert()
{ alert("I am an alert box!");}
</script>
</head>
<body>
<p><input type="button" onclick="show_alert()" value="Show alert
  box">
</p>
</body>
```


Popup Boxes
Confirm Box

- A confirm box is often used if you want the user to verify or accept something.
- When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
- If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

```
confirm("sometext");
```


Popup Boxes
Confirm Box

```
<script type="text/javascript">
function show_confirm()
{ var r=confirm("Press a button!");
      if (r==true)
      { alert("You pressed OK!");  }
      else
      {  alert("You pressed Cancel!");  }
}
</script>
```


Popup Boxes
Prompt Box

- A prompt box is often used if you want the user to input a value before entering a page.
- When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
- If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

```
prompt("sometext","defaultvalue");
```

Popup Boxes
Prompt Box

```
<script type="text/javascript">
function show_prompt()
{  var name=prompt("Please enter your name","Harry Potter");
   if (name!=null && name!="")
      {  document.write("Hello " + name + "! How are you today?");  }
}
</script>
</head>
<body>
<input    type="button"    onclick="show_prompt()"    value="Show
  prompt box">
</body>
```


Catching Errors
try...catch statement

```
try
  {
  //Run some code here
  }
catch(err if expression)
  {
  //Handle errors here
  }
```

- The try...catch statement allows you to test a block of code for errors.
- The try block contains the code to be run, and the catch block contains the code to be executed if an error occurs.
- *err* is initialized with the exception object
- *expression* is a test expression

Catching Errors
try...catch statement

```
try {    myroutine();        // may throw three exceptions
}
catch (e if e instanceof TypeError) {
   // statements to handle TypeError exceptions
}
catch (e if e instanceof RangeError) {
   // statements to handle RangeError exceptions
}
catch (e if e instanceof EvalError) {
   // statements to handle EvalError exceptions
}
catch (e){
   // statements to handle any unspecified exceptions
   logMyErrors(e) // pass exception object to error handler
}
```


Catching Errors
try...catch statement

```
<script type="text/javascript">
function message()
{ try  {   addlert("Welcome guest!");  }
  catch(err)
  {   var txt="There was an error on this page.\n\n";
          txt+="Click OK to continue viewing this page,\n";
          txt+="or Cancel to return to the home page.\n\n";
          if(!confirm(txt))
          {     document.location.href=
"http://devedge-temp.mozilla.org/central/javascript/index_en.html";
          }
  }
}
</script></head>
```

Catching Errors try...catch statement

```
<body>
<div>
<input type="button" value="View message"
  onclick="message()">
</div>
</body>
```


Catching Errors Throw statement

- The throw statement allows you to create an exception.

  throw(exception)

- The exception can be a string, integer, Boolean or an object.

```
<body>
<script type="text/javascript">
var x=prompt("Enter a number between 0 and 10:","");
try
 {   if(x>10)
     {   throw "Err1";   }
     else if(x<0)
     {   throw "Err2"; }
     else if(isNaN(x))
     {   throw "Err3";   }
 }
```


Catching Errors Throw statement

```
catch(er)
 { if(er=="Err1")
   {   alert("Error! The value is too high"); }
   if(er=="Err2")
   { alert("Error! The value is too low");   }
   if(er=="Err3")
   {   alert("Error! The value is not a number");   }
 }
</script>
</body>
```


JavaScript Objects

JavaScript Objects

- No inheritance
- No private properties and methods
- Both the object name and property name are case sensitive.
- You define a property by assigning it a value.
  - For example, suppose there is an object named myCar (for now, just assume the object already exists). You can give it properties named make, model, and year as follows:

```
myCar.make = "Ford";
myCar.model = "Mustang";
myCar.year = 1969;
```


JavaScript Objects

- Two ways for accessing the properties of an object:
  - objectName.propertyName
  - objectName["propertyName"]
- The second method is inherited from the associative arrays.
- Actually, associative arrays and objects in JavaScript are different interfaces of the same data structure
- Associative arrays: each index element is associated with a string value

JavaScript Objects

- Associative array notation (example)

```
function show_props(obj, obj_name) {
  var result = "";
  for (var i in obj) // i assumes as values all the property names
  result += obj_name + "." + i + " = " + obj[i] + "\n";
  return result
}
```

The function call show_props(myCar, "myCar") returns

```
myCar.make = Ford
myCar.model = Mustang
myCar.year = 1967
```


JavaScript Objects
Creating new objects

- Two ways for creating objects
  - use an object initializer
  - create a constructor function and then instantiate an object using that function and the new operator.
- The first way is useful when you need to create a unique instance of the object; otherwise use the second way.

JavaScript Objects
Object Initializer

- Object initializer

objectName = {property1:value1,..., propertyN:valueN}

objectName: name of the new object

propertyI: identifier (either a name, a number, or a string literal), *valueI*: expression whose value is assigned to the propertyI.

The property of an object can be an object in its turn.

The objectName and assignment is optional. If you do not need to refer to this object elsewhere, you do not need to assign it to a variable.

Note: new properties can be added dynamically

objectName.newproperty=value

JavaScript Objects
Object Initializer

- Object initializer (example)

```
guy = { name: "Paul",
        age: 40,
        country: "Italy",
        auto: { trademark: "Ferrari", colour: "rosso" }}
...
<pre>
<script type="text/javascript">
for (var a in guy)
        if (typeof(guy[a])=="object")
         for (var i in guy[a])
                document.writeln(a + '.' + i + ':' + guy[a][i]);
        else document.writeln(a + ':' + guy[a]);
</script>
</pre>
```


JavaScript Objects
Constructor Function

- Contructor Function

1. Define the object type by writing a constructor function.
2. Create an instance of the object with new.

To define an object type, create a function for the object type that specifies its name, properties, and methods.

```
function Car(make, model, year) {
this.make = make
this.model = model
this.year = year
}

mycar = new Car("Eagle", "Talon TSi", 1993)
kenscar = new Car("Nissan", "300ZX", 1992)
vpgscar = new Car("Mazda", "Miata", 1990)
```

this keyword!

JavaScript Objects
Constructor Function

- Contructor Function

An object can have a property that is itself another object.

```
function Person(name, age, sex) {
this.name = name
this.age = age
this.sex = sex
}
function Car(make, model, year, owner) {
this.make = make
this.model = model
this.year = year
this.owner = owner
}
owner = new Person("Frankie Black",45,"M")
mycar = new Car("Eagle", "Talon TSi", 1993, owner)
```

JavaScript Operators
new operator

- new operator

  You can use the new operator to create an instance of a user-defined object type or of one of the predefined object types Array, Boolean, Date, Function, Image, Number, Object, Option, RegExp, or String.

```
objectName = new objectType (param1 [,param2] ...[,paramN] );
```


JavaScript Objects
Adding a new property

- Properties can be added to an object at run time.
- To add a property to a specific object you have to assign a value to the object

```
car1.enginepower = 100
```

- Note: only the car1 object will have the property enginepower

JavaScript Object Types
Adding a new property

- You can add a property to a previously defined object type by using the prototype property.
- This defines a property that is shared by all objects of the specified type, rather than by just one instance of the object.
- The following code adds a color property to all objects of type car, and then assigns a value to the color property of the object car1.

```
Car.prototype.color=null
car1.color="black"
```


JavaScript Object Types
Adding a new property

```
<head>
<meta http-equiv="Content-type" content="text/html; charset=ISO-8859-1">
<meta http-equiv="Content-Script-Type" content="text/javascript">
<title>Example</title>
<script type="text/javascript"><!--
function displayCar(car)
{ for (var a in car)
        if (typeof(car[a])=="object")
         for (var i in car[a])
                document.writeln(a + '.' + i + ':' + car[a][i]);
        else document.writeln(a+':'+car[a]);
        document.writeln();
  }
...
</script></head>
```

JavaScript Object Types
Adding a new property

```
<script type="text/javascript"><!-- HTML comment
owner = new Person("Frankie Black",45,"M")
mycar1 = new Car("Eagle", "Talon TSi", 1993, owner)
displayCar(mycar1)
Car.prototype.color="red"
mycar2 = new Car("Nissan", "300ZX", 1992,owner)
mycar3 = new Car("Mazda", "Miata", 1990,owner)
mycar3.color = "black"
displayCar(mycar1)
displayCar(mycar2)
displayCar(mycar3) -->
```


JavaScript Objects
Defining methods

- The following syntax associates a function with an existing object:

  object.methodname = function_name

- You can then call the method in the context of the object as follows:

  object.methodname(params);

- You can define methods for an object type by including a method definition in the object constructor function.

JavaScript Objects
Defining methods

```
function displayCar(){
for (var a in this)
  if (typeof(this[a])=="object")
    for (var i in this[a]) document.writeln(a + '.' + i + ':' + this[a][i]);
  else if (typeof(this[a])!="function") document.writeln(a+':'+this[a]);
  document.writeln();
  }

function Car(make, model, year, owner) {
     this.make = make;
     this.model = model;
     this.year = year;
     this.owner = owner;
     this.displayCar = displayCar;
     }
```


JavaScript Objects
Defining methods

```
<script type="text/javascript">
<!-- HTML comment
     owner = new Person("Frankie Black",45,"M")
     mycar1 = new Car("Eagle", "Talon TSi", 1993, owner)
     mycar1.displayCar();
     Car.prototype.color="red"
     mycar2 = new Car("Nissan", "300ZX", 1992,owner)
     mycar3 = new Car("Mazda", "Miata", 1990,owner)
     mycar3.color = "black"
     mycar1.displayCar();
     mycar2.displayCar();
     mycar3.displayCar();
-->
</script>
```

JavaScript Objects
Defining methods

Alternatively

```
function Car(make, model, year, owner) {
      this.make = make;
      this.model = model;
      this.year = year;
      this.owner = owner;
      }

Car.prototype.displayCar = function () {
for (var a in this)
  if (typeof(this[a])=="object")
    for (var i in this[a]) document.writeln(a + '.' + i + ':' + this[a][i]);
  else if (typeof(this[a])!="function") document.writeln(a+':'+this[a]);
  document.writeln();
  }
```


JavaScript Operators
delete operator

```
delete objectName
delete objectName.property
delete objectName[index]
delete property // legal only within a "with" statement
```

where objectName is the name of an object, property is an existing property, and index is an integer representing the location of an element in an array.

- If the delete operator succeeds, it sets the property or element to undefined.
- The delete operator returns true if the operation is possible; it returns false if the operation is not possible.
- You can use the delete operator to delete variables declared implicitly but not those declared with the var statement.

JavaScript Operators
delete operator

- delete operator (example)

```
x=42;
var y= 43;
myobj=new Number();
myobj.h=4; // create property h
delete x; // returns true (can delete if declared implicitly)
delete y; // returns false (cannot delete if declared with var)
delete Math.PI; // returns false (cannot delete predefined
                          properties)
delete myobj.h; // returns true (can delete user-defined properties)
delete myobj; // returns true (delete user-defined object)
var myobj1=new Number();
window.alert(delete myobj1); //returns false (cannot delete if
                                     declared with var)
```


JavaScript Operators
delete operator

- delete operator (array elements)

When you delete an array element, the array length is not affected. For example, if you delete a[3], a[4] is still a[4] and a[3] is undefined.

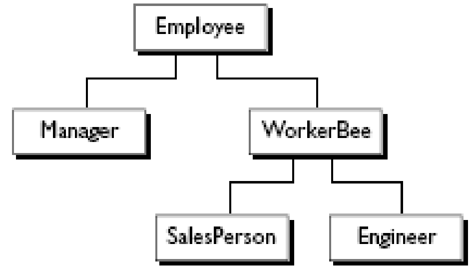
When the delete operator removes an array element, that element is no longer in the array.

```
trees=new Array("redwood","bay","cedar","oak","maple");
delete trees[3];
if (3 in trees) {
// this does not get executed
}
trees[3] = "oak";
if (3 in trees) {
// this does get executed
}
```

Inheritance

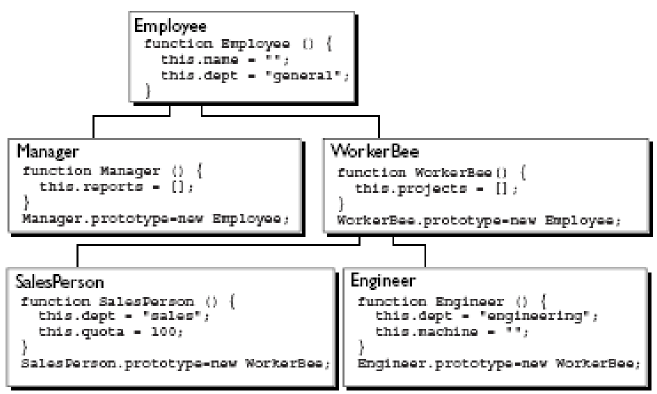
JavaScript Objects
Inheritance

- JavaScript implements inheritance by allowing you to associate a prototypical object with any constructor function.



JavaScript Objects
Inheritance

- When you create a new Manager, it inherits the name and dept properties from the Employee object.



JavaScript Objects
Inheritance

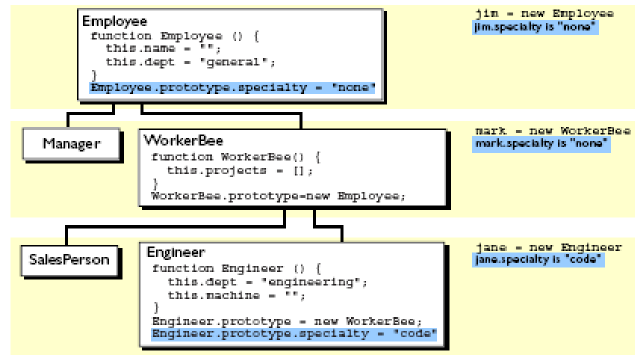
- The new operator creates a new generic object and passes this new object as the value of the this keyword to the constructor function.
- When you ask for the value of a property, JavaScript first checks to see if the value exists in that object.
  - If it does, that value is returned.
  - If the value is not there locally, JavaScript checks the prototype chain (using the __proto__ property).
    - If an object in the prototype chain has a value for the property, that value is returned.
    - If no such property is found, JavaScript says the object does not have the property.

JavaScript Objects Inheritance

```
function display(obj)
{  for (var a in obj)
   {       document.write(a);
           if (typeof(obj[a])=="object" && obj[a].length!=0)
           {          document.write("."); display(obj[a]);}
           else       document.writeln(' = ' + obj[a]);
   }
}
salesP = new SalesPerson;
display(salesP);
document.writeln();
eng = new Engineer;
display(eng);
```


JavaScript Objects Inheritance – adding properties

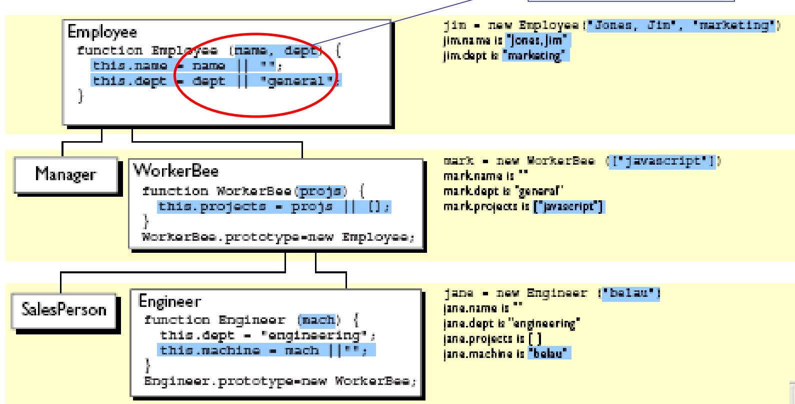
- If you add a new property to an object that is being used as the prototype for a constructor function, you add that property to all objects that inherit properties from the prototype.



JavaScript Objects Inheritance – adding properties

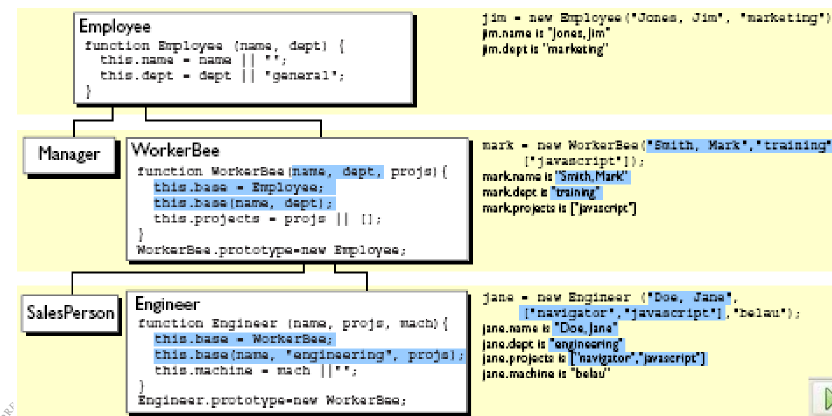
- Constructors with arguments

Default values



JavaScript Objects Inheritance – adding properties

- Constructors with arguments



JavaScript Objects Inheritance – adding properties

- Note: The name of the base property is not special. You can use any legal property name; base is simply evocative of its purpose.
- Note: call the WorkerBee constructor from inside the Engineer constructor does not set up inheritance appropriately for Engineer objects.
  - Calling the WorkerBee constructor ensures that an Engineer object starts out with the properties specified in all constructor functions that are called. However, if you later add properties to the Employee or WorkerBee prototypes, those properties are not inherited by the Engineer object.

JavaScript Objects Inheritance – adding properties

- For example, assume you have the following statements:

```
function Engineer (name, projs, mach) {
this.base = WorkerBee; this.base(name, "engineering", projs);
this.machine = mach || "";}

jane = new Engineer("Doe, Jane", ["navigator", "javascript"], "belau");
Employee.prototype.specialty = "none";
```

- The jane object does not inherit the specialty property.

```
Engineer.prototype = new WorkerBee;
jane = new Engineer("Doe, Jane", ["navigator", "javascript"], "belau");
Employee.prototype.specialty = "none";
```

- Now, the value is inherited

Built-in Objects

Objects

- Predefined built-in objects

  Array, Boolean, Date, Function, Math, Number, RegExp, and String
- Predefined client-side objects
  - When you load a document in Navigator, it creates a number of JavaScript objects with property values based on the HTML in the document and other pertinent information.
  - These objects exist in a hierarchy that reflects the structure of the HTML page itself.
  - Each page has always the objects: Navigator, Window, Document, Location, History.

Built-in Objects
Array

- An array is an ordered set of values that you refer to with a name and an index.
- Array elements may belong to different types
- The Array object has methods for manipulating arrays in various ways, such as joining, reversing, and sorting them.
  - It has a property for determining the array length and other properties for use with regular expressions.

Built-in Objects
Array

- Create an Array

  Three forms
  - var myCars=new Array(); // (add an optional integer
    myCars[0]="Saab"; // argument to control array's size)
    myCars[1]="Volvo";
    myCars[2]="BMW";
  - var myCars=new Array("Saab","Volvo","BMW"); //condensed
  - var myCars=["Saab","Volvo","BMW"]; //literal array
    - It is also possible to avoid to specify all the values of the elements of the array
      var myCars=["Saab", ,"BMW"];

Built-in Objects
Array

- Access an Array
  document.write(myCars[0]);
- Modify values in an Array
  myCars[0]="Opel";
- An array can dynamically be extended
  myCars[6]="Fiat";

The size of the array is extended to 7. The elements with indexes 3, 4 and 5 are undefined.

Built-in Objects
Array

- Array properties

Property	Description
constructor	Returns the function that created the Array object's prototype
length	Sets or returns the number of elements in an array
prototype	Allows you to add properties and methods to an object

Built-in Objects
Array

Method	Description
concat()	Joins two or more arrays, and returns a copy of the joined arrays
	array.concat(array2, array3, ..., arrayX)
join()	Joins all elements of an array into a string
	array.join(separator)
	If the separator is omitted, the elements are separated with a comma
pop()	Removes the last element of an array, and returns that element
	array.pop()
push()	Adds new elements to the end of an array, and returns the new length
	array.push(element1, element2, ..., elementX)

Built-in Objects
Array

Method	Description
reverse()	Reverses the order of the elements in an array
	array.reverse()
shift()	Removes the first element of an array, and returns that element
	array.shift()
slice()	Selects a part of an array, and returns the new array
	array.slice(start, end)
	start Required. An integer that specifies where to start the selection. You can also use negative numbers to select from the end of an array
	end Optional. An integer that specifies where to end the selection. If omitted, slice() selects all elements from the start position and to the end of the array

Built-in Objects
Array

Method	Description
sort()	Sorts the elements of an array
	array.sort(sortfunc)
	sortfunc Optional. A function that defines the sort order Default: sorts the elements alphabetically and ascending. However, numbers will not be sorted correctly (40 comes before 5). To sort numbers, you must add a function that compare numbers
splice()	Adds/Removes elements from an array
	array.splice(index,howmany,element1,.....,elementX)
	index Required. An integer that specifies at what position to add/remove elements
	howmany Required. The number of elements to be removed. If set to 0, no elements will be removed
	element1, ..., elementX Optional. The new element(s) to be added to the array

Built-in Objects
Array

Method	Description
toString()	Converts an array to a string, and returns the result
	array.toString()
unshift()	Adds new elements to the beginning of an array, and returns the new length
	array.unshift(element1,element2, ..., elementX)
	element1,element2, ..., elementX Required. The element(s) to add to the beginning of the array
valueOf()	Returns the primitive values of an array
	array.valueOf()

Built-in Objects
Array

- Sort numbers

```
<script type="text/javascript">
function sortNumber(a,b)
{
      return a - b;           //(numerically and ascending)
      //return b - a;   (numerically and descending)
}
var n = [10, 5, 40, 25, 100, 1];
document.write(n.sort(sortNumber));
</script>
```


Built-in Objects
Array

- Example

```
<script type="text/javascript"><!--
var myvector= new Array("S","D","A","B");
print(myvector,"(initialise)");

function add()
{     myvector[myvector.length] = document.forms[0].text1.value;
      print(myvector, "(add)");
}

function concatenate()
{     var vector = myvector.join("+");
      document.writeln(vector+ "<br>(concatenate)");
}
```


Built-in Objects
Array

```
function invert()
{  myvector.reverse();
   print(myvector, "(invert)");
}

function shift()
{  myvector.shift();
   print(myvector,"(translate)");
}

function sort()
{  myvector.sort();
   print(myvector,"(sort)");
}
```


Built-in Objects
Array

```
function print(vector, comment)
{  for (var i = 0; i < vector.length; i++)
         document.writeln(vector[i]);
         document.writeln("<br>"+ comment);
}
-->
</script>
</head>
```

Built-in Objects Array

```
<body>
<form action="#">
<p>Add a new element: <input type="text" name="text1">
<input type="button" value="ADD" onclick="add()"><br>
<input type="button" value="JOIN('+')"
    onclick="concatenate()"><br>
<input type="button" value="INVERT" onclick="invert()"><br>
<input type="button" value="SHIFT" onclick="shift()"><br>
<input type="button" value="SORT" onclick="sort()"><br>
</p>
</form>
</body>
```


Built-in Objects Boolean

- The Boolean object is a wrapper around the primitive Boolean data type.

  booleanObjectName = new Boolean(value)
- If the Boolean object has no initial value or if it is 0, -0, null, "", false, undefined, or NaN, the object is set to false. Otherwise it is true (even with the string "false")!
- It is used to convert a non-boolean value into a boolean value.

Method	Description
toString()	Converts a Boolean value to a string, and returns the result
valueOf()	Returns the primitive value of a Boolean object

Built-in Objects Date

- JavaScript stores dates as the number of milliseconds since January 1, 1970, 00:00:00
- Four forms to create a Date object
  - new Date () *// Date object with current date and time*
  - new Date (milliseconds) *//milliseconds since 1970/01/01*
  - new Date (dateString) *//A string representing a date in the following form: "Month day, year hours:minutes:seconds." For example, Xmas95 = new Date("December 25, 1995 13:30:00"). If you omit hours, minutes, or seconds, the value will be set to zero.*
  - new Date (year, month, day, hours, minutes, seconds, milliseconds) *//parameters are integer values*
- Most parameters above are optional. Not specifying them, causes 0 to be passed in.

Built-in Objects Date

Method	Description
getDate()	Returns the day of the month (from 1-31)
getDay()	Returns the day of the week (from 0-6)
getFullYear()	Returns the year (four digits)
getHours()	Returns the hour (from 0-23)
getMilliseconds()	Returns the milliseconds (from 0-999)
getMinutes()	Returns the minutes (from 0-59)
getMonth()	Returns the month (from 0-11)
getSeconds()	Returns the seconds (from 0-59)
getTime()	Returns the number of milliseconds since midnight Jan 1, 1970

Built-in Objects Date

Method	Description
getUTCDate()	Returns the day of the month, according to universal time (from 1-31). UTC (Coordinated Universal Time) is an atomic timescale that approximates UT1.
getUTCDay()	Returns the day of the week, according to universal time (from 0-6)
getUTCFullYear()	Returns the year, according to universal time (four digits)
getUTCHours()	Returns the hour, according to universal time (from 0-23)
getUTCMilliseconds()	Returns the milliseconds, according to universal time (from 0-999)
getUTCMinutes()	Returns the minutes, according to universal time (from 0-59)
getUTCMonth()	Returns the month, according to universal time (from 0-11)
getUTCSeconds()	Returns the seconds, according to universal time (from 0-59)

Built-in Objects Date

Method	Description
parse()	Parses a date string and returns the number of milliseconds since midnight of January 1, 1970
	Date.parse(datestring)
	Datestring Required. A string representing a date
setDate()	Sets the day of the month (from 1-31)
setFullYear()	Sets the year (four digits)
setHours()	Sets the hour (from 0-23)
setMilliseconds()	Sets the milliseconds (from 0-999)
setMinutes()	Set the minutes (from 0-59)
setMonth()	Sets the month (from 0-11)
setSeconds()	Sets the seconds (from 0-59)
setTime()	Sets a date and time by adding or subtracting a specified number of milliseconds to/from midnight January 1, 1970

Built-in Objects Date

Method	Description
setUTCDate()	Sets the day of the month, according to universal time (from 1-31)
setUTCFullYear()	Sets the year, according to universal time (four digits)
setUTCHours()	Sets the hour, according to universal time (from 0-23)
setUTCMilliseconds()	Sets the milliseconds, according to universal time (from 0-999)
setUTCMinutes()	Set the minutes, according to universal time (from 0-59)
setUTCMonth()	Sets the month, according to universal time (from 0-11)
setUTCSeconds()	Set the seconds, according to universal time (from 0-59)

Built-in Objects Date

Method	Description
toDateString()	Converts the date portion of a Date object into a readable string
toLocaleDateString()	Returns the date portion of a Date object as a string, using locale conventions
toLocaleTimeString()	Returns the time portion of a Date object as a string, using locale conventions
toLocaleString()	Converts a Date object to a string, using locale conventions
toString()	Converts a Date object to a string
toTimeString()	Converts the time portion of a Date object to a string
toUTCString()	Converts a Date object to a string, according to universal time
UTC()	Returns the number of milliseconds in a date string since midnight of January 1, 1970, according to universal time
valueOf()	Returns the primitive value of a Date object

Built-in Objects
Date

```
<script type="text/javascript">
    var d=new Date();
    document.write("Original form: ");
    document.write(d + "<br>");
    document.write("Formatted form: ");
    document.write(d.toLocaleDateString());
</script>
```

Output

Original form: Mon May 17 16:46:50 UTC+0200 2010

Formatted form: lunedì 17 maggio 2010

Built-in Objects
Math

- The Math object allows performing mathematical tasks.
- The Math object includes several mathematical constants and methods.
- Math is not a constructor. All properties/methods of Math can be called by using Math as an object, without creating it.
- Note that all trigonometric methods of Math take arguments in radians.

```
var pi_value=Math.PI;
var sqrt_value=Math.sqrt(16);
```


Built-in Objects
Math

- Note: Case-Sensitive

Property	Description
E	Returns Euler's number (approx. 2.718)
LN2	Returns the natural logarithm of 2 (approx. 0.693)
LN10	Returns the natural logarithm of 10 (approx. 2.302)
LOG2E	Returns the base-2 logarithm of E (approx. 1.442)
LOG10E	Returns the base-10 logarithm of E (approx. 0.434)
PI	Returns PI (approx. 3.14159)
SQRT1_2	Returns the square root of 1/2 (approx. 0.707)
SQRT2	Returns the square root of 2 (approx. 1.414)

Built-in Objects
Math

Method	Description
abs(x)	Returns the absolute value of x
acos(x)	Returns the arccosine of x, in radians
asin(x)	Returns the arcsine of x, in radians
atan(x)	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians
atan2(y,x)	Returns the arctangent of the quotient of its arguments
ceil(x)	Returns x, rounded upwards to the nearest integer
cos(x)	Returns the cosine of x (x is in radians)
exp(x)	Returns the value of E^x

Built-in Objects
Math

Method	Description
floor(x)	Returns x, rounded downwards to the nearest integer
log(x)	Returns the natural logarithm (base E) of x
max(x,y,z,...,n)	Returns the number with the highest value
min(x,y,z,...,n)	Returns the number with the lowest value
pow(x,y)	Returns the value of x to the power of y
random()	Returns a random number between 0 and 1
round(x)	Rounds x to the nearest integer
sin(x)	Returns the sine of x (x is in radians)
sqrt(x)	Returns the square root of x
tan(x)	Returns the tangent of an angle

Built-in Objects
Number

- The Number object is an object wrapper for primitive numeric values.
- The Number object has properties for numerical constants, such as maximum value, not-a-number, and infinity. You cannot change the values of these properties and you use them as follows:

```
biggestNum = Number.MAX_VALUE
smallestNum = Number.MIN_VALUE
```

- Typically, you create a Number object when you want to add some properties to them through the prototype property

```
var num = new Number(value);
Number.prototype.newproperty=value
```


Built-in Objects
Number

Property	Description
constructor	Returns the function that created the Number object's prototype
MAX_VALUE	Returns the largest number possible in JavaScript
MIN_VALUE	Returns the smallest number possible in JavaScript
NEGATIVE_INFINITY	Represents negative infinity (returned on overflow)
POSITIVE_INFINITY	Represents infinity (returned on overflow)
prototype	Allows you to add properties and methods to an object

Built-in Objects
Number

Method	Description
toExponential(x)	Converts a number into an exponential notation
	number.toExponential(x)
	x Optional. An integer between 0 and 20 representing the number of digits in the notation after the decimal point. If omitted, it is set to as many digits as necessary to represent the value
toFixed(x)	Formats a number with x numbers of digits after the decimal point
	number.toFixed(x)
	x Optional. The number of digits after the decimal point. Default is 0 (no digits after the decimal point)
toPrecision(x)	Formats a number to x length
	number.toPrecision(x)
	x Optional. The number of digits. If omitted, it returns the entire number (without any formatting)
toString()	Converts a Number object to a string
valueOf()	Returns the primitive value of a Number object

Built-in Objects
Number

```
<script type="text/javascript">
    var num = new Number(13.3714);
    document.write(Number.MAX_VALUE+"<br>");
    document.write(Number.MIN_VALUE+"<br>");
    document.write(num.toExponential(4)+"<br>");
    document.write(num.toFixed(2)+"<br>");
    document.write(num.toPrecision(3)+"<br>");
    document.write(num.toString()+"<br>");
    document.write(num.valueOf());
</script>
```


Built-in Objects
String

- The String object is a wrapper around the string primitive data type. Do not confuse a string literal with the String object.
- String objects are created with new String().

```
s1 = "Hi"               //creates a string literal value
s2 = new String("Hi")   //creates a String object
```

- You can call any of the methods of the String object on a string literal value
  - JavaScript automatically converts the string literal to a temporary String object, calls the method, then discards the temporary String object.

Built-in Objects
String

- You can also use the String.length property with a string literal.
- A String object has two types of methods:
  - those that return a variation on the string itself, such as substring and toUpperCase,
  - those that return an HTML-formatted version of the string, such as bold and link.

Built-in Objects
String

Method	**Description**
charAt()	Returns the character at the specified index
charCodeAt()	Returns the Unicode of the character at the specified index
concat()	Joins two or more strings, and returns a copy of the joined strings
	string.concat(string2, string3, ..., stringX)
fromCharCode()	Converts Unicode values to characters
indexOf()	Returns the position of the first found occurrence of a specified value in a string
lastIndexOf()	Returns the position of the last found occurrence of a specified value in a string (-1 if the value to search for never occurs)
	string.lastIndexOf(searchstring, start)
	searchstring Required. The string to search for **Start** Optional. The start position in the string to start the search. If omitted, the search starts from position 0

Built-in Objects String

Method	Description
match()	Searches for a match between a regular expression and a string, and returns the matches *string*.match(regexp) *regexp* Required. A regular expression.
replace()	Searches for a match between a substring (or regular expression) and a string, and replaces the matched substring with a new substring *string*.replace(*regexp/substr*,*newstring*) *regexp/substr* Required. A substring or a regular expression. *newstring* Required. The string to replace the found value in parameter 1
search()	Searches for a match between a regular expression and a string, and returns the position of the match

Built-in Objects String

Method	Description
slice()	Extracts a part of a string and returns a new string *string*.slice(begin,end) *begin* Required. The index where to begin the extraction. First character is at index 0 *end* Optional. Where to end the extraction. If omitted, slice() selects all characters from the begin position to the end of the string
split()	Splits a string into an array of substrings *string*.split(separator, limit) *separator* Optional. Specifies the character to use for splitting the string. If omitted, the entire string will be returned *limit* Optional. An integer that specifies the number of splits

Built-in Objects String

Method	Description
substr()	Extracts the characters from a string, beginning at a specified start position, and through the specified number of character
substring()	Extracts the characters from a string, between two specified indices
toLowerCase()	Converts a string to lowercase letters
toUpperCase()	Converts a string to uppercase letters
valueOf()	Returns the primitive value of a String object

Built-in Objects String

```
<script type="text/javascript">
<!-- HTML comment
var str="Hello world!";
document.write(str +"<br>");
document.write(str.substring(1) +"<br>");
document.write(str.substring(3,7)+"<br>");
document.write(str.split('o',2)+"<br>");
document.write(str.toUpperCase()+"<br>");
document.write(str.toLowerCase());
-->
</script>
```

JavaScript RegExp

- A regular expression is an object that describes a pattern of characters.
  - A simple pattern can be one single character.
  - A more complicated pattern can consist of more characters, and can be used for parsing, format checking, substitution and more.
- Regular expressions are used to perform powerful pattern-matching and "search-and-replace" functions on text.

```
var txt=new RegExp(pattern,modifiers);
or more simply
var txt=/pattern/modifiers;
```

pattern specifies the pattern of an expression

modifiers specify if a search should be global, case-sensitive, etc.

Built-in Objects RegExp

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching

Method	Description
exec()	Tests for a match in a string. Returns the first match regexp.exec(str) regexp(str)
test()	Tests for a match in a string. Returns true or false regexp.test(str)

Built-in Objects RegExp

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character not between the brackets
[0-9]	Find any digit from 0 to 9
[a-z]	Find any character from lowercase a to lowercase z
[A-Z]	Find any character from uppercase A to uppercase Z
[a-Z]	Find any character from lowercase a to uppercase Z
adgk	Find the sequence of characters
(red|blue|green)	Find any of the alternatives specified

Built-in Objects RegExp

Metachar	Description
.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character
\s	Find a whitespace character
\S	Find a non-whitespace character
\b	Find a match at the beginning/end of a word
\B	Find a match not at the beginning/end of a word

Built-in Objects
RegExp

Metachar	Description
\0	Find a NUL character
\n	Find a new line character
\f	Find a form feed character
\r	Find a carriage return character
\t	Find a tab character
\v	Find a vertical tab character
\xxx	Find the character specified by an octal number xxx
\xdd	Find the character specified by a hexadecimal number dd
\uxxxx	Find the Unicode character specified by a hexadecimal number xxxx

Built-in Objects
RegExp

Quantifier	Description
n+	Matches any string that contains at least one n
n*	Matches any string that contains zero or more occurrences of n
n?	Matches any string that contains zero or one occurrences of n
n{X}	Matches any string that contains a sequence of X n's
n{X,Y}	Matches any string that contains a sequence of X or Y n's
n{X,}	Matches any string that contains a sequence of at least X n's
n$	Matches any string with n at the end of it
^n	Matches any string with n at the beginning of it
?=n	Matches any string that is followed by a specific string n
?!n	Matches any string that is not followed by a specific string n

Built-in Objects
RegExp

```
<script type="text/javascript">
    var str="100, 1000 or 10000?";
    var patt1=/\d{3,4}/g;
    document.write(str.match(patt1));
</script>
```

Output:
100,1000,1000

Built-in Objects
Global Properties and Functions

- The JavaScript global properties and functions can be used with all the built-in JavaScript objects.

Property	Description
Infinity	A numeric value that represents positive/negative infinity
NaN	"Not-a-Number" value
undefined	Indicates that a variable has not been assigned a value

Function	Description
decodeURI()	Decodes a URI
decodeURIComponent()	Decodes a URI component
encodeURI()	Encodes a URI
encodeURIComponent()	Encodes a URI component

Built-in Objects
Global Properties and Functions

Function	Description
escape()	Encodes a string
eval()	Evaluates a string and executes it as if it was script code. First, eval() determines if the argument is a valid string, then eval() parses the string looking for JavaScript code. If it finds any JavaScript code, it will be executed.
isFinite()	Determines whether a value is a finite, legal number
isNaN()	Determines whether a value is an illegal number
Number()	Converts an object's value to a number
parseFloat()	Parses a string and returns a floating point number
parseInt()	Parses a string and returns an integer
String()	Converts an object's value to a string
unescape()	Decodes an encoded string

Università di Pisa Francesco Marcelloni

Built-in Objects
Global Properties and Functions

```
<script type="text/javascript">
    eval("x=10;y=20;document.write(x*y)");
    document.write("<br>" + eval("2+2"));
    document.write("<br>" + eval(x+17));
</script>
```

Output:
200
4
27

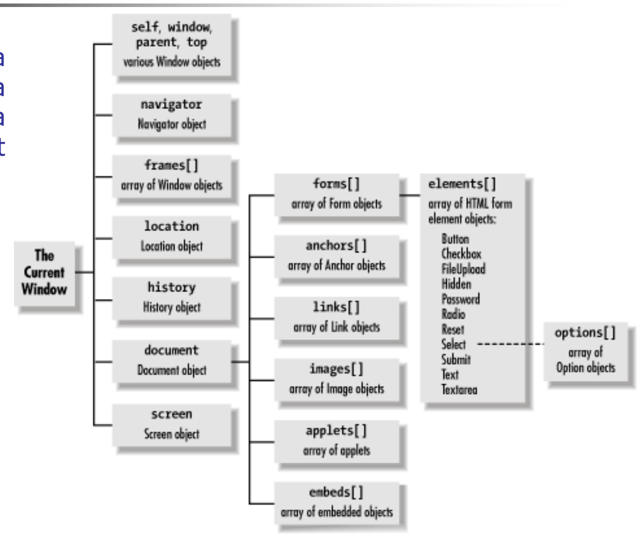
Università di Pisa Francesco Marcelloni

Client-side Objects

Università di Pisa Francesco Marcelloni

Client-side objects

When you load a document in a Browser, it creates a number of JavaScript Objects.



Client-side objects
Objects in a Page

Every page has the following objects

- navigator: has properties for the name and version of the Navigator being used, for the MIME types supported by the client, and for the plugins installed on the client.
- window: the top-level object; has properties that apply to the entire window. There is also a *window* object for each "child window" in a frames document.
- document: contains properties based on the content of the document, such as title, background color, links and forms.
- location: has properties based on the current URL.
- history: contains properties representing URLs the client has previously requested.

Client-side objects
Controlling objects

By controlling Objects, we can

- Open new browser windows, and determine their size and whether they should have scroll bars, or location windows
- Change the content of multiple frames at once, even rewriting the HTML in a frame without downloading a new file
- Add or remove text from forms including text areas and select boxes
- Control the background color of your Web pages using JavaScript
- etc...

Client-side objects
Window Object

- Every browser window that is currently open will have a corresponding window Object.
- If a document contain frames (<frame> or <iframe> tags), the browser creates one window object for the HTML document, and one additional window object for each frame.
- All the other Objects are children of one of the window Objects, except for navigator and screen.

Client-side objects
Window Object

- Every window has a history of the previous pages that have been displayed in that window, which are represented by the various properties of the history object.
- JavaScript maintains an idea of the current window, so that almost all references to sub-objects of the current window do not need to refer to it explicitly.
  - This is why all of our output has been done using document.write() rather than window.document.write().

Client-side objects
Window Object Properties

Property	Description
closed	Returns a Boolean value indicating whether a window has been closed or not
defaultStatus	Sets or returns the default text in the statusbar of a window
document	Returns the Document object for the window
frames	Returns an array of all the frames (including iframes) in the current window
history	Returns the History object for the window
innerHeight	Sets or returns the inner height of a window's content area
innerWidth	Sets or returns the inner width of a window's content area
length	Returns the number of frames (including iframes) in a window

Client-side objects
Window Object Properties

Property	Description
location	Returns the Location object for the window
name	Sets or returns the name of a window
navigator	Returns the Navigator object for the window
opener	Returns a reference to the window that created the window
outerHeight	Sets or returns the outer height of a window, including toolbars/scrollbars
outerWidth	Sets or returns the outer width of a window, including toolbars/scrollbars
pageXOffset	Returns the pixels the current document has been scrolled (horizontally) from the upper left corner of the window
pageYOffset	Returns the pixels the current document has been scrolled (vertically) from the upper left corner of the window

Client-side objects
Window Object Properties

Property	Description
parent	Returns the parent window of the current window
screen	Returns the Screen object for the window
screenLeft	Returns the x coordinate of the upper-left hand corner of the window (Explorer – Opera)
screenTop	Returns the y coordinate of the upper-left hand corner of the window (Explorer – Opera)
screenX	Returns the x coordinate of the upper-left hand corner of the window (Firefox – Chrome)
screenY	Returns the y coordinate of the upper-left hand corner of the window (Firefox – Chrome)
self	Returns the current window
status	Sets the text in the statusbar of a window
top	Returns the topmost browser window

Client-side objects
Window Object Properties

Method	Description
alert()	Displays an alert box with a message and an OK button
blur()	Removes focus from the current window
clearInterval()	Clears a timer set with setInterval()
clearTimeout()	Clears a timer set with setTimeout()
close()	Closes the current window
confirm()	Displays a dialog box with a message and an OK and a Cancel button
focus()	Sets focus to the current window

Client-side objects
Window Object Methods

Method	Description
moveBy()	Moves a window relative to its current position
moveTo()	Moves a window to the specified position
open()	Opens a new browser window
print()	Prints the content of the current window
prompt()	Displays a dialog box that prompts the visitor for input
resizeBy()	Resizes the window by the specified pixels
resizeTo()	Resizes the window to the specified width and height

Client-side objects
Window Object Methods

Method	Description
scrollBy()	Scrolls the content by the specified number of pixels
scrollTo()	Scrolls the content to the specified coordinates
setInterval()	Calls a function or evaluates an expression at specified intervals (in milliseconds)
setTimeout()	Calls a function or evaluates an expression after a specified number of milliseconds

Client-side objects
Window Object Methods

```
<title>Example</title>
<script type="text/javascript">
   var myWindow;
   function openWin()
   {  myWindow=window.open('','','width=200,height=100');
      myWindow.document.write("This is 'myWindow'");
   }

   function closeWin()
   { myWindow.close(); }

   function moveWin()
   {  myWindow.moveBy(250,250);
      myWindow.focus();}
</script>
</head>
```


Client-side objects
Window Object Methods

```
<body>
<div>
  <input type="button" value="Open 'myWindow'"
  onclick="openWin()" >
  <br><br>
  <input type="button" value="Move 'myWindow'"
  onclick="moveWin()" >
  <br><br>
  <input type="button" value="Close 'myWindow'"
  onclick="closeWin()" >
  </div>
</body>
</html>
```

Client-side objects
Navigator Object

- The navigator object contains information about the browser.
- The JavaScript runtime engine on the client automatically creates the navigator object.
- Use the navigator object to determine which version of the Navigator your users have, what MIME types the user's Navigator can handle, and what plugins the user has installed.
- All of the properties of the navigator object are read-only.

Client-side objects
Navigator Object

Property	Description
appCodeName	Returns the code name of the browser
appName	Returns the name of the browser
appVersion	Returns the version information of the browser
battery	Returns information about the battery charging status.
cookieEnabled	Determines whether cookies are enabled in the browser
platform	Returns for which platform the browser is compiled
userAgent	Returns the user-agent header sent by the browser to the server

Client-side objects
Navigator Object (example 1)

```
<script type="text/javascript">
document.write("Code Name: " + navigator.appCodeName +
  "<br>") ;
document.write("Name: " + navigator.appName + "<br>" );
document.write("Platform: " + navigator.platform + "<br>") ;
document.write("User Agent: " + navigator.userAgent + "<br>") ;
</script>
```

Output:
Code Name: Mozilla
Name: Microsoft Internet Explorer
Platform: Win32
User Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; GTB6.5; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)

Client-side objects
Navigator Object (example 2)

```
\\myscript5.js
var infobrowser = "<h1>Information on your browser</h1><p>";
for (var proprietyName in navigator)
  infobrowser += "<strong>" + proprietyName + ":</strong>" +
  navigator[proprietyName] + "<br>";
  infobrowser += "</p>";
document.writeln(infobrowser);
if (navigator.appName.indexOf("Netscape") != -1)
  window.alert("Your browser is Netscape");
else if (navigator.appName.indexOf("Microsoft Internet Explorer") !=
  -1)
  window.alert("Your browser is Internet Explorer");
```

Client-side objects
Navigator Object (example 2)

Information on your browser
appCodeName: Mozilla
appName: Microsoft Internet Explorer
appMinorVersion:0
cpuClass:x86
platform: Win32
plugins:[object]
opsProfile:null
userProfile:null
systemLanguage:en-us
userLanguage:it
appVersion:4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; GTB6.5; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)
userAgent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; GTB6.5; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)
onLine:true
cookieEnabled:true
mimeTypes:

Client-side objects
Screen Object

- Contains read-only properties describing the display screen and colours.

Property	Description
availHeight	Returns the height of the screen (excluding the Windows Taskbar)
availWidth	Returns the width of the screen (excluding the Windows Taskbar)
colorDepth	Returns the bit depth of the color palette for displaying images (the $\log_2$ of the number of colors)
height	Returns the total height of the screen
pixelDepth	Returns the color resolution (in bits per pixel) of the screen
width	Returns the total width of the screen

Client-side objects
Screen Object (example)

```
//myscript6.js
document.writeln("<h1>Screen Information</h1><p>");
printScreenInfo('width'); printScreenInfo('height');
printScreenInfo('colorDepth'); printScreenInfo('availWidth');
printScreenInfo('availHeight');
function printScreenInfo(name) {
document.writeln("<strong>"+ name + ":</strong>"+
  screen[name]+ "<br>");
}
```

Output:
width:1680
height:1050
colorDepth:32
availWidth:1680
availHeight:1016

Client-side objects
History Object (properties)

- The history object contains the URLs visited by the user (within a browser window).
- The history object is part of the window object and is accessed through the window.history property

length – contains the current length of the history list

- The history Object does not contain any values that reflect the actual URLs in the history list.
- Therefore, you cannot, for example, perform some action that reads the value of "URL number 3 in the history list"
- The history Object is designed for navigating the history list.

Client-side objects
History Object (Methods)

- back() – moves the user to the URL one place previous in the history list (previous to the current position).
- forward() - moves the user one URL forward, relative to current position, in the history list
- go(offset) - accepts an integer parameter, positive or negative, as offset.
  - If the parameter is a positive integer, the program will move the user that many places forward in the history list.
  - If the parameter is negative, it'll move the user that many places backward (previous to the current position) in the history list.
  - The current position is always place zero.
- go(substring) - searches for the newest history list URL that contains the specified string somewhere within its URL string
  - history.**go**("email.htm");

Client-side objects
History Object (Example)

```
<body>
 <div>
   Go to the page:<br>
   <button onclick="history.go(-1);">Back</button>
   <button onclick="history.go(0);">Current</button>
   <button onclick="history.go(+1);">Forward</button>
 </div>
```


Client-side objects
Location Object (properties)

The location Object contains information on the current URL.

- href - contains the string value of the entire URL
  - window.**open**(location.**href**, "windowName", "feature1,feature2, ...");

    open a new window, which connects to the same URL as the current window (so you can look at another portion of the same page at the same time you're looking at the current portion of the page)
  - location.**href** = "http://www.someISP.com/~me/myPage.htm";

    Launch a new page without opening a new window (you leave your current page, including other JavaScript programs).

Client-side objects
Location Object (properties)

- host - holds only the hostname and port of the current page's URL
  - Example:

    http://www.someISP.com:80/~me/myPage.htm

    location.host -> " www.someISP.com:80"
  - Note: Assigning a value to location.host will generate an error because it is nonsensical. Whereas you can assign an entire URL to location.href, which would then connect to that URL. You cannot connect to just a host.
- port - contains the value of the port number in the URL

Client-side objects Location Object (properties)

- hostname – returns the hostname portion of the URL.
- pathname - is the portion of the URL that describes the location of the Web document on the host computer.
  - Example:
    http://www.someISP.com/~me/myPage.htm
    location.pathname -> /~me/myPage.htm

  Assigning a value to this property,

  location.pathname = "~me/otherdocs/newdoc.htm";

  will cause the Web browser to load that document into the current window.

Client-side objects Location Object (properties)

- protocol – contains the leftmost portion of the URL.
  - By checking the location.protocol property, the JavaScript program can determine if the page currently resides in was delivered by HTTP, FTP, or NEWS.
- hash – contains the value following the hash mark
  - Some URLs contain special hash mark values following the pathname.

    http://www.someISP.com/~me/myPage.htm#item1

    The hash mark (#) specifies the name of an anchor to jump to in the Web page.

    location.hash -> item1

    You can use the location.**hash** property in Event Handlers to bring the user to specific locations within the current page.

Client-side objects Location Object (properties)

- search – contains the value following the question mark.
  - Some URLs contain search parameters following the pathname, denoted with a question mark (?).
  - A form entry is probably the most common use for a search parameter.

    http://www.someISP.com/~me/myProgram?formData

    location.search -> formData
- reload() – reloads the document that is currently displayed in the window of the location Object.
- assign(newURL) – loads a new document.
- replace(newURL) – replaces the current document with a new one.

Client-side objects Location Object (example)

```
<html>
<head>
<title> Location Object</title>
<script type="text/javascript" src="./myscript7.js">
</script>
</head>
<body>
<form action="#" method="get">
<p><strong>Insert your name:</strong><br>
<input type="text" name="Name">
<input type="button" onclick= "send()" value="SEND"></p>
</form>
</body>
</html>
```

Client-side objects Location Object (example)

```
//myscript7.js
display('URL',location.href);
display('Protocol',location.protocol);
display('Host name', location.hostname);
display('Local Address', location.hash);
display('Port', location.port);
display('Path', location.pathname);

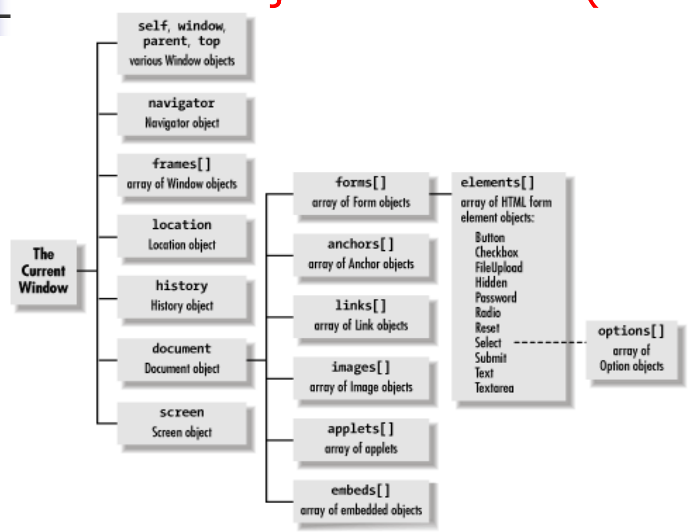
function send() {
location.search="&Name=" + document.forms[0].Name.value;}

function display(propriety, value) {
document.writeln("<p><strong>"+propriety+":</strong>"+value+"
  </p>");}
```


Client-side objects Document Object

- Each HTML document loaded into a browser window becomes a Document object.
- The Document object provides access to all HTML elements in a page, from within a script.
- Tip: The Document object is also part of the Window object, and can be accessed through the window.document property
- A document is the file of HTML codes that describe a given page.
  - A page is what appears within the browser window. So, every window is associated with a document object.

Client-side objects Document Object Collections (DOM 0)



Client-side objects Document Object (Properties)

Property	Description
activeElement	Returns the currently focused element.
body	Returns the <body> element of the current document.
cookie	Returns a semicolon-separated list of the cookies for that document or sets a single cookie.
defaultView	Returns a reference to the window object.
dir	Gets/sets directionality (rtl/ltr) of the document.
domain	Specifies the domain name of the server that served a document
head	Returns the <head> element of the document.
lastModified	Returns the date on which the document was last modified.
location	Returns the URI of the current document.

Client-side objects Document Object (Properties)

Property	Description
plugins	Returns a list of the available plugins.
readyState	Returns loading status of the document.
referrer	Returns the URI of the page that linked to this page.
title	Returns the title of the current document.
URL	Returns a string containing the URL of the current document.

- Note: document.URL property may be modified through URL redirection. Thus, location.href contains the requested URL while document.URL specifies the actual URL where it was found.

Client-side objects Document Object (Methods)

Method	Description
close()	Closes the output stream previously opened with document.open()
open()	Opens an output stream to collect the output from document.write() or document.writeln()
write()	Writes HTML expressions or JavaScript code to a document
writeln()	Same as write(), but adds a newline character after each statement

Client-side objects Document Object (Methods)

- Note: How is it possible to change the presentation style of the elements?
- Use the “style” property.
- Note: the names of the properties do not have hyphens and are in camelCase notation.
- For instance:
  - background-color -> backgroundColor
  - border-right-style -> borderRightStyle
- Example

  element.style.backgroundColor = ‘yellow’

Client-side objects Document Object (Methods)

- write(str) – outputs text to the client window’s document
  - document.write() "converts" everything to HTML
  - the browser then interprets the HTML and then renders the HTML
- writeln(str) – outputs text to the client window’s document (appends a newline character to the end of the output)
  - str - strings of text are surrounded by double (or single) quotes

Examples:

document.write("<h1>Thank you for ordering!</h1>");

document.write(example);

where example is a string variable

Client-side objects
Document Object (Methods)

- Note: write() method actually takes a variable number of arguments, rather than just one.
  - If more than one argument is given, each of the arguments is interpreted as a string and written in turn.

```
document.write("This is Part 1 ", "and this is Part 2 ", "and this is Part 3");
```

- Note: writeln() appends a newline character to the end of the output.
  - Keep in mind that these methods output their parameters as HTML. HTML ignores newline characters when it comes to outputting to the screen. HTML does not insert line breaks in screen output unless you specify a line break using either <br> or <p> tags or text resides between <pre> and </pre> tags.

Client-side objects
Document Object (Example)

```
<body>  <p>
   <a href="#"
     onclick="openW('yellow')">open yellow window</a><br>
   <a href="#"
     onclick="openW('red')">open red window</a><br>
   <a href="#"
     onclick="openW('blue')">open blue window</a>
  </p><p>
   <a href="#"
     onclick="openDoc('yellow')">open yellow page</a><br>
   <a href="#"
     onclick="openDoc('red')">open red page</a><br>
   <a href="#"
     onclick="openDoc('blue')">open blue page</a>
  </p>  </body>
```


Client-side objects
Document Object (Example)

```
document.writeln("<p>You are on '"+ document.URL + "'<p>");
function openW(color) {
  win = window.open("",color,"width=400,height=250");
  HTMLcode = '<html><head><meta charset="utf-8"><title>Page '+ color+ '</title></head>' +
   '<body><p>This is a ' + color + ' page </p></body></html>';
  win.document.writeln(HTMLcode);
  win.document.body.style.background = color;
  win.document.body.style.color = "white";
}
```


Client-side objects
Document Object (Example)

```
function openDoc(color) {
  doc = document.open("text/html","replace");
  HTMLcode = '<html><head><meta charset="utf-8"><title>Page '+ color+ '</title></head>' +
   '<body><p>This is a ' + color + ' page </p></body></html>';
  doc.writeln(HTMLcode);
  doc.body.style.backgroundColor = color;
  doc.body.style.color = "green";
  doc.close();
}
```

Client-side objects
Document Object (Methods)

Double and Single Quotes

- When writing HTML it is general practice to use double quotes for tag attributes, e.g.:

```
<img src="picture.gif" width="400" height="400">
```

- When writing JavaScript it is general practice to use single quotes for string literals:

```
<script type="text/javascript">
  document.write('<p>');
  </script>
```

In general these quoting styles then complement one another when outputting HTML using JavaScript

Client-side objects
How are objects named? (DOM 0)

- You can access only to a subset of all elements contained in a document, through the document object.
- The name of each Object is prefaced by the names of all the Objects that contain it, in order, separated by dots.
- For example, the window Object contains the document Object, and the document Object contains a form Object, and the form Object contains text input fields, buttons, select Objects, etc... — window.document.forms[0].elements[3]

Client-side objects
How are objects named? (DOM 0)

Leave off the word "window."

- All Client-side Object names officially start with the highest Object name, "window". But since all names start with "window", JavaScript allows you to leave that off.
  - document.forms[0].elements[3]

Client-side objects
How are objects named? (DOM 0)

Form elements

- The numerous Object types included in the Forms Array are collectively referred to as "elements" and they can all be referred to by means of the "Elements Array" (e.g.: elements[1]).
- Example: **document.forms[0].elements[1]**

Client-side objects
How to refer to properties or methods?

- Note: JavaScript is case sensitive
  - document.myform.Check1 is not the same "Object" as document.myform.check1
- The form Object has child objects named button1 and text1, corresponding to the button and text field in the form.
- These objects have their own properties based on their HTML attribute values, for example,
  - document.myform.button1.value is "Press Me"
  - document.myform.elements[2].name is "Button1"
  - document.myform.elements[0].name is "text1"

Client-side objects
How to refer to properties or methods?

- Three different ways to specify the value of an object:
  - document.myform.text1.value is "blahblah"
    NOTE: using just the object names
  - document.myform.elements[0].value is "blahblah"
    NOTE: using an object name & array notation (to specify the object instead of the object name)
  - document.forms[0].elements[0].value is "blahblah"
    NOTE: using just array notation

Client-side objects
How to refer to properties or methods?

```
<title>A Simple Document</title>
</head>
<body>
<p><a name="top" id="top">This is the top of the page</a><p>
<hr>
<form method="post" action="%20mailto:nobody@dev.null">
<p>Enter your name: <input type="text" name="me" size="70">
<input type="Submit" value="OK">
<input type="Reset" value="Oops"></p>
</form>
<hr>
<p>Click here to go to the <a href="#top">top</a> of the
  page</p>
</body>
</html>
```


Client-side objects
How to refer to properties or methods?

- The code creates an HTML page with an anchor at the top of the page and a link to that anchor at the bottom.
- In between is a simple form that allows the user to enter his name.
- We can also access the other HTML elements of this document using the following properties:
  - anchors
  - forms
  - links

Client-side objects How to refer to properties or methods?

document.title
<title>A very simple HTML page</title>
document.anchors[0]
This is the top of the page
document.anchors[0].name -> top

document.forms[0]
<form method= "post" action="mailto:nobody@dev.null">
document.forms[0].method -> post
document.forms[0].action -> mailto:nobody@dev.null

Client-side objects How to refer to properties or methods?

document.forms[0].elements[0]
<input type="text" name="me" size="70">
document.forms[0].elements[0].name -> me
document.forms[0].elements[0].type -> text

document.forms[0].elements[1]
<input type="Submit" value="OK">
document.forms[0].elements[1].value -> OK
document.forms[0].elements[1].type -> Submit

document.forms[0].elements[2]
<input type="Reset" value="Oops">
document.forms[0].elements[2].value -> Oops
document.forms[0].elements[2].type -> Reset

Client-side objects How to refer to properties or methods?

document.links[0]

top

document.links[0].href
file:///F:/Examples/example1.htm#top

Client-side objects Document

- Each object in the document has specific properties and methods.
- The access to each element using the approach described so far is not flexible, is limited and does not allow changing dymamically the structure of the document.
- The ability to change a Web page dynamically with a scripting language is made possible by the Document Object Model (DOM), which can connect any element on the screen to a JavaScript function.

Document Object Model DOM

- "The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."
- The DOM is separated into 3 different parts / levels:
  - Core DOM - standard model for any structured document
  - XML DOM - standard model for XML documents
  - HTML DOM - standard model for HTML documents
- The DOM defines the objects and properties of all document elements, and the methods (interface) to access them.

Document Object Model DOM

```
<!DOCTYPE html>
<html>
 <head>
  <title>Sample page</title>
 </head>
 <body>
  <h1>Sample page</h1>
  <p>This is a <a href="demo.html">simple</a> sample.</p>
  <!-- this is a comment -->
 </body>
</html>
```

```
├DOCTYPE: html
└html
 ├head
 │├#text: ⏎␣
 │├title
 ││└#text: Sample page
 │└#text: ⏎
 ├#text: ⏎
 └body
  ├#text: ⏎␣
  ├h1
  │└#text: Sample page
  ├#text: ⏎␣
  ├p
  │├#text: This is a
  │├a href="demo.html"
  ││└#text: simple
  │└#text: sample.
  ├#text: ⏎␣
  ├#comment: this is a comment
  └#text: ⏎␣⏎
```


DOM

The HTML DOM is:

- A standard object model for accessing and manipulating HTML documents HTML
  - A standard programming interface for HTML
  - Platform- and language-independent
  - A W3C standard
- In other words: The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

DOM

- According to the DOM, everything in an HTML document is a node.
  - The entire document is a document node
  - Every HTML element is an element node
  - The text in the HTML elements are text nodes
  - Every HTML attribute is an attribute node
  - Comments are comment nodes

DOM

```
<html>
  <head>  <title>DOM Tutorial</title>  </head>
  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>
</html>
```

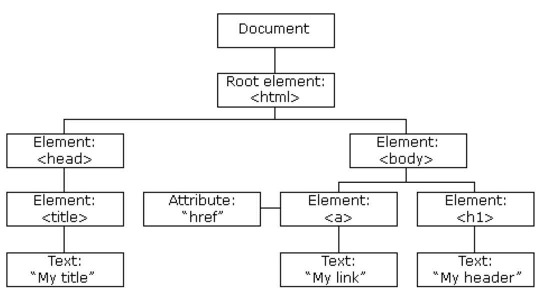
- The root node in the HTML above is <html>. All other nodes in the document are contained within <html>.
- The <html> node has two child nodes: <head> and <body>.
- The <head> node holds a <title> node. The <body> node holds a <h1> and <p> nodes.

DOM

- A common error in DOM processing is to expect an element node to contain text. However, the text of an element node is stored in a text node.
- In this example: <title>DOM Tutorial</title>, the element node <title> holds a text node with the value "DOM Tutorial".
  - "DOM Tutorial" is not the value of the <title> element!

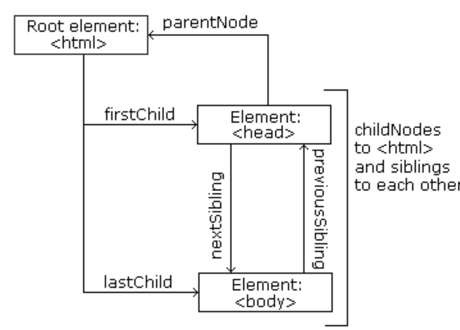
DOM Tree

- The HTML DOM views a HTML document as a tree-structure. The tree structure is called a node-tree.
- All nodes can be accessed through the tree.



DOM Tree

- Parent nodes have children. Children on the same level are called siblings (brothers or sisters).
- Every node, except the root, has exactly one parent node



DOM Tree

- <head> element: first child of the <html> element
- <body> element: last child of the <html> element
- <h1> element: first child of the <body> element
- <p> element: last child of the <body> element

- The programming interface of the DOM is defined by standard properties and methods.

DOM (Level 3) Tree

- Some attributes:
  - x.innerHTML - the text value of x (not a part of the W3C DOM specification, but supported by all major browsers)
    YOU MUST NOT USE INNERHTML
  - x.childNodes - the child nodes of x
  - x.parentNode - the parent of x
  - x.firstChild – the first child of x
  - x.lastChild – the last child of x
  - x.localName – the local part of the qualified name of x
  - x.namespaceURI – the namespace URI of this node of x
  - x.nodeName - the name of x
  - x.nodeValue - the value of x
  - x.nodeType – the type of x

- Note: In the list above, x is a node object (HTML element).

DOM (Level 3) Tree

- Some attributes:
  - x.nextSibling – next brother of x
  - x.previousSibling – previous brother of x
  - x.textContent – returns the text content of x and its descendants
  - x.attributes - the attributes nodes of x

- Note: In the list above, x is a node object (HTML element).

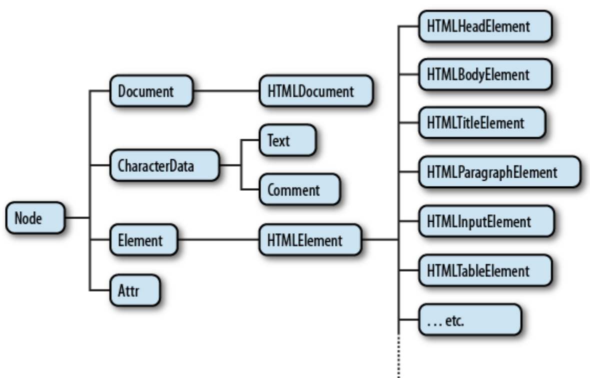
DOM (Level 3) Tree

- myNode.nodeType == Node.ELEMENT_NODE
  myNode is an object Element
- myNode.nodeType == Node.ATTRIBUTE_NODE
  Node is an attribute

Main types of nodes:
const unsigned **short** ELEMENT_NODE = 1
const unsigned **short** ATTRIBUTE_NODE = 2
const unsigned **short** TEXT_NODE = 3
const unsigned **short** ENTITY_NODE = 6 (XML documents)
const unsigned **short** PROCESSING_INSTRUCTION_NODE = 7
const unsigned **short** COMMENT_NODE = 8
const unsigned **short** DOCUMENT_NODE = 9

DOM (Level 3) Tree

As specified by the DOM API, each HTML element is represented by an object, hence the term "Object Model".



DOM (Level 3) Tree (Methods)

- x.getElementById(id) - gets the element with a specified id
- x.getElementsByTagName(name) - gets all elements with a specified tag name
- x.appendChild(node) - adds a child node to the end of the list of children of x
- x.compareDocumentPosition(node) – compares x with the node passed as a parameter, with regard to their position in the document and according to the document order. Returns a short integer which denotes how the node is positioned relatively to x.
- newx = x.cloneNode(deep) – clones x to create a new object: if deep == true, all the tree with x as root is cloned; otherwise only node x
- x.removeChild(node) - removes a child node from x and returns it

DOM (Level 3) Tree (Methods)

- x.replaceChild(newC,oldC) - replaces oldC with newC and returns oldC
- x.hasChildNodes() – returns true if the object has sons; false otherwise
- x.contains(node) - returns a Boolean value indicating whether a node is a descendant of x or not
- x.isEqualNode(node) – tests whether x is equal to node
- x.lookupNamespaceURI() - takes a prefix and returns the namespace URI associated with it on the given node if found (and null if not).

DOM (Level 3) Tree (Methods)

Read and set the attributes of elements

- x.getAttribute(string_name) – returns the value of the attribute string_attribute
- x.setAttribute(string_name, string_value) – set the value of the attribute string_name to string_value

```
function showAttributes() {
  var result;
  var elems = document.getElementsByTagName("input");
  for (var i=0;i<elems.length;i++) {
      result = "";
      for (var a in elems[i]) {
        aValue = elems[i].getAttribute(a);
        if (a=='type')       result += "- " + a + ": " + aValue + "\n";}
window.alert("Attributes of " + elems[i].tagName + ":\n" + result);
  }}
```

DOM Tree (get or modify the content of an element)

- Attention: The easiest way to get or modify the content of an element is by using the innerHTML property, but it is not W3C.

Two solutions:

- First Solution: Use the childNodes and NodeValue properties

```
<body>
<p id="intro">Hello World!</p>
…

function change()
{ txt=document.getElementById("intro").childNodes[0].nodeValue;
document.getElementById("intro").childNodes[0].nodeValue="hi";
document.write("<p>The text from the intro paragraph: " + txt +
  "<\/p>");
}
```


DOM Tree (get or modify the content of an element)

- Second Solution: use the firstChild and NodeValue properties

```
function change()
{
  txt=document.getElementById("intro").firstChild.nodeValue;
  document.getElementById("intro").firstChild.nodeValue="hi";
  document.write("<p>The text from the intro paragraph: " + txt
  + "<\/p>");
}
```

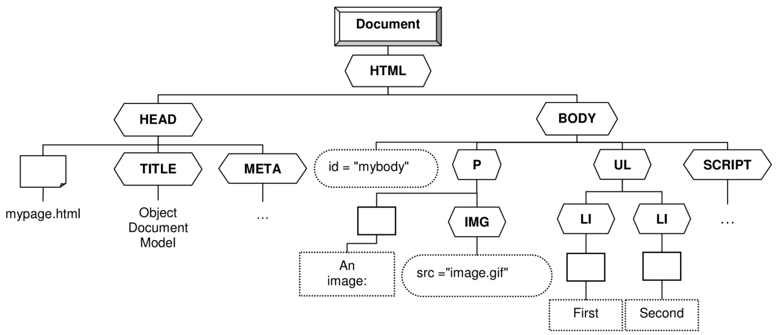

DOM Tree (Example)

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
<title> Object Model Example </title>
</head>
```


DOM Tree (Example)

```
<body id="mybody">
   <p id="mypar1">
    An image:<img src="image.gif">
   </p>
   <ul>
     <li>First
     </li>
     <li>Second
     </li>
   </ul>
   <script type="text/javascript" src="./myscript1.js"> </script>
  </body>
</html>
```

DOM Tree (Example)



DOM Tree
Access to a node

You can access a node in three ways:

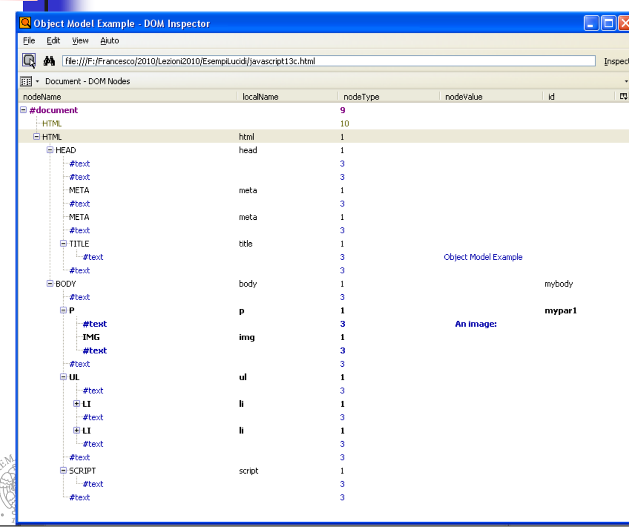
1. By using the getElementById() method
2. By using the getElementsByTagName() method
3. By navigating the node tree, using the node relationships

DOM Tree
Access to a node

1. Use of the getElementById method (myscript1.js)

```
var text =
   document.getElementById("mypar1").firstChild.nodeValue;
window.alert("The paragraph has the text '" + text +"'");
var body = document.getElementById("mybody");
var textNode = body.childNodes[0].firstChild;
      //Explorer (old versions)
if (!textNode) window.alert("Mozilla Firefox");
else window.alert("Microsoft Explorer");
text = textNode ? textNode.nodeValue :
body.childNodes[1].firstChild.nodeValue;          //Mozilla
window.alert("The paragraph has the text '" + text + "'");
```


DOM Tree
Access to a node



DOM Inspector (Mozilla Firefox)

Firefox (and the new versions of Explorer) add always a child node of text, also if the text is not present

DOM Tree
Access to a node

2. Using the getElementsByTagName() method (file myscript2.js)

```
var elems = document.getElementsByTagName("p");
var text =  elems[0].firstChild.nodeValue;
window.alert("The paragraph has the text '" + text +"'");
```


DOM Tree
Access to a node

3. Navigating the DOM tree (example 1 - file myscript3.js)

```
function visit(node,obj)
{  if (node.hasChildNodes())
 { for (var i=0; i<node.childNodes.length; i++)
  { obj.txt+="CHILD " + i + " of " + node.nodeName  +
          " - .#" + node.childNodes[i].nodeType+"." +
           node.childNodes[i].nodeName + "=" +
           node.childNodes[i].nodeValue +  "<br>";
          visit(node.childNodes[i],obj);     }}}
var node=document.documentElement;
var obj=new Object;
obj.txt="";
visit(node,obj);
document.write(obj.txt);
```


DOM Tree
Access to a node

3. Navigating the DOM tree (example 2 - file myscript4.js)

```
function visit(node,obj)
{  if (node==null) return;
   if (node.firstChild)
   {       obj.txt+=node.firstChild.nodeType + "." +
              node.firstChild.nodeName     + "=" +
              node.firstChild.nodeValue + "<br>";
           visit(node.firstChild,obj);
   }
  if (node.nextSibling)
  { obj.txt+=node.nextSibling.nodeType + "." +
          node.nextSibling.nodeName + "=" +
          node.nextSibling.nodeValue + "<br>";
    visit(node.nextSibling,obj);}
}
```


Events

Events

- Events are actions that can be detected by JavaScript.
- Every element on a web page has certain events which can trigger a JavaScript handler.
- The events are defined in the HTML tags
- Note: events are normally used in combination with functions, and the function will not be executed before the event occurs!
  - For example, we can use the onClick event of a button element to indicate that a function will run when a user clicks on the button

Event Object (Definition)

- The event object gives you information about an event that has occurred.
- The Event object represents the state of an event, such as the element in which the event occurred, the state of the keyboard keys, the location of the mouse, and the state of the mouse buttons.
- Events are normally used in combination with functions (Event handlers), and the function will not be executed before the event occurs!
- The event object contains properties that describe a JavaScript event, and is passed as an argument to an event handler when the event occurs.

Event Object (Definition)

Example

- mouse-down event. The event object contains
  - the type of event (in this case MouseDown),
  - the x and y position of the cursor at the time of the event,
  - a number representing the mouse button used,
  - a field containing the modifier keys (Control, Alt, Meta, or Shift) that were depressed at the time of the event.
- The properties used within the event object vary from one type of event to another.
  - This variation is provided in the descriptions of individual event handlers.

Event Object
To Associate Events with Code

- First approach – Events as HTML element attributes
  - The event is an attribute of an HTML element

    <tag eventName = "JavaScript code or call to handler">

    eventName starts with **on**
  - Example:

    <body onLoad = "myHandler()"

    <body onLoad = "instruction1;…; instructionN;">

Note: it is not possible to associate any event with any element since some events have no meaning or cannot arise for some element.

Event Object
To Associate Events with Code

- First Approach (example)

```
<body>
  <div style="position:absolute; left:300px; top:200px;"
     id="but">
   <button onmouseover="moveMouse()">Click</button>
  </div>
    <script    type="text/javascript"    src="./myscript10.js">
    </script>
 </body>
```


Event Object
To Associate Events with Code

- First Approach (example)

```
var but = document.getElementById('but');
var diff = [150, 0, -150, 0];
var i = 0;
function moveMouse()
{
 but.style.top= (parseInt(but.style.top)+ diff[(i+3)%4]) + "px";
 but.style.left=(parseInt(but.style.left)+ diff[(i++)%4])+ "px";
}
```


Event Object
To Associate Events with Code

- Second Approach – to set the value of the property corresponding to the event to a specific function

```
<div style="position:absolute; left:300px; top:200px;"
       id="but">    <button >Click</button>
  </div>
  <script type="text/javascript" src="./myscript11.js"> </script>

// myscript11.js
but.onmouseover=moveMouse; //Event handlers are function references
function moveMouse()
{
 but.style.top= (parseInt(but.style.top)+ diff[(i+3)%4]) + "px";
 but.style.left=(parseInt(but.style.left)+ diff[(i++)%4])+ "px";
}
```

Note: no parentheses

Event Object
To Associate Events with Code

- Third Approach – Specifying an event handler with a Function object. Function objects created with the Function constructor are evaluated each time they are used.

```
function dettaglia(elemento, evento) {
 var t = elemento.form.area_testo;
 var nome_elemento = elemento.name; var valore = " ";
 if ((elemento.type == "select-one") || (elemento.type == "select-multiple")){ for (var i = 0; i < elemento.options.length; i++)
             if (elemento.options[i].selected)
                    valore += elemento.options[i].value + " "; }
 else if (elemento.type == "textarea") valore = "...";
 else valore = elemento.value;
 var mess = evento + ": " + nome_elemento + ' (' + valore + ')\n';
 t.value += mess;
}
```

Event Object
To Associate Events with Code

- Third Approach – Specifying an event handler with a Function object.

```
// La funzione aggiungi_gestori installa un insieme di gestori di
// eventi su ogni elemento di un form f, senza controllare se
// l'elemento supporta tutti questi tipi di eventi

  function aggiungi_gestori(f) {
   var gestore_click = new Function("dettaglia(this, 'Click')");
   var gestore_change = new Function("dettaglia(this, 'Change')");
   var gestore_focus = new Function("dettaglia(this, 'Focus')");
   var gestore_blur  = new Function("dettaglia(this, 'Blur')");
   var gestore_select  = new Function("dettaglia(this, 'Select')");
   var gestore_dblclick  = new Function("dettaglia(this, 'dblClick')");
```


Event Object
To Associate Events with Code

- Third Approach – Specifying an event handler with a Function object.

```
for(var i = 0; i < f.elements.length; i++) {
    var e = f.elements[i];
    e.onclick    = gestore_click;
    e.onchange   = gestore_change;
    e.onfocus    = gestore_focus;
    e.onblur     = gestore_blur;
    e.onselect   = gestore_select;
    e.ondblclick = gestore_dblclick;
}
```


Event Object
To Associate Events with Code

- Fourth approach – addEventListener(type, listener[, useCapture])
- The EventTarget.addEventListener() method registers the specified listener on the EventTarget it is called on. The event target may be an Element in a document, the Document itself, a Window, or any other object that supports events
  - Type - A string representing the event type to listen for
  - Listener - the object that receives a notification when an event of the specified type occurs
  - useCapture - If true, useCapture indicates that the user wishes to initiate capture. After initiating capture, all events of the specified type will be dispatched to the registered listener before being dispatched to any EventTarget beneath it in the DOM tree.

Event Object
To Associate Events with Code

- Fourth approach – addEventListener(type, listener[, useCapture])

```
// Function to change the content of t2
function modifyText() {
  var t2 = document.getElementById("t2");
  if (t2.firstChild.nodeValue == "three") {
    t2.firstChild.nodeValue = "two";
  } else {
    t2.firstChild.nodeValue = "three";
  }
}
// add event listener to table
var el = document.getElementById("outside");
el.addEventListener("click", modifyText, false);
```

Events

- Main events

Attribute	The event occurs when...	IE	F	O	W3C
onblur	An element loses focus	3	1	9	Yes
onchange	The content of a field changes	3	1	9	Yes
onclick	Mouse clicks an object	3	1	9	Yes
ondblclick	Mouse double-clicks an object	4	1	9	Yes
onerror	An error occurs when loading a document or an image	4	1	9	Yes
onfocus	An element gets focus	3	1	9	Yes
onkeydown	A keyboard key is pressed	3	1	No	Yes

Events

Attribute	The event occurs when...	IE	F	O	W3C
onkeypress	A keyboard key is pressed or held down	3	1	9	Yes
onkeyup	A keyboard key is released	3	1	9	Yes
onmousedown	A mouse button is pressed	4	1	9	Yes
onmousemove	The mouse is moved	3	1	9	Yes
onmouseout	The mouse is moved off an element	4	1	9	Yes
onmouseover	The mouse is moved over an element	3	1	9	Yes
onmouseup	A mouse button is released	4	1	9	Yes
onresize	A window or frame is resized	4	1	9	Yes
onselect	Text is selected	3	1	9	Yes
onunload	The user exits the page	3	1	9	Yes

Events

- The events are applicable to specific elements
- For instance,
  - onBlur is applicable to only object window and all the elements of a form
  - onChange is applicable to only the fields <input>, <textarea> and <select> of a form
  - onMouseOut is applicable to elements links and areas
  - onResize is applicable to document, frame and window

Events

```
<script type="text/javascript">
function setStyle(x)
{ document.getElementById(x).style.background="yellow"; }
function resetStyle(x)
{ document.getElementById(x).style.background="white";}
</script>
</head>
<body>
<div>
 First name: <input type="text" onfocus="setStyle(this.id)"
 onblur="resetStyle(this.id)" id="fname"><br>
 Last name: <input type="text" onfocus="setStyle(this.id)"
 onblur="resetStyle(this.id)" id="lname"></div>
</body>
```

Event order

```
-------------------
| element1        |
|   -----------   |
|   |element2 |   |
|   -----------   |
|                 |
-------------------
```

- Two models
  - Capturing – the event on element1 takes place first (Netscape)
  - Bubbling – the event on element2 takes precedence (Microsoft)

Università di Pisa
Francesco Marcelloni

Event order

```
               | |
---------------| |-----------------
| element1     | |                |
|   -----------| |-----------     |
|   |element2  \ /          |     |
|   -------------------------     |
|        Event CAPTURING          |
-----------------------------------
```

- Event capturing
  - The event handler of element1 fires first, the event handler of element2 fires last

Università di Pisa
Francesco Marcelloni

Event order

```
               / \
---------------| |-----------------
| element1     | |                |
|   -----------| |-----------     |
|   |element2  | |          |     |
|   -------------------------     |
|        Event BUBBLING           |
-----------------------------------
```

- Event bubbling
  - The event handler of element2 fires first, the event handler of element1 fires last

Università di Pisa
Francesco Marcelloni

Event order

```
                 | |  / \
-----------------| |--| |-----------------
| element1       | |  | |                |
|   -------------| |--| |-----------     |
|   |element2    \ /  | |          |     |
|   --------------------------------     |
|        W3C event model                 |
------------------------------------------
```

- W3C model
  - W3C has decided to take a middle position in this struggle. Any event taking place in the W3C event model is first captured until it reaches the target element and then bubbles up again

Event Handler

- The Web developer can choose whether to register an event handler in the capturing or in the bubbling phase.
  - This is done through the addEventListener()
  - If its last argument is true the event handler is set for the capturing phase
  - If it is false the event handler is set for the bubbling phase

Event Handler

```
element1.addEventListener('click',doSomething2,true)
element2.addEventListener('click',doSomething,false)
```

If the user clicks on element2 the following happens:

1. The `click` event starts in the capturing phase. The event looks if any ancestor element of element2 has a `onclick` event handler for the capturing phase.
2. The event finds one on element1. `doSomething2()` is executed.
3. The event travels down to the target itself, no more event handlers for the capturing phase are found. The event moves to its bubbling phase and executes `doSomething()`, which is registered to element2 for the bubbling phase.
4. The event travels upwards again and checks if any ancestor element of the target has an event handler for the bubbling phase. This is not the case, so nothing happens.

Event Handler

```
element1.addEventListener('click',doSomething2,false)
element2.addEventListener('click',doSomething,false)
```

Now if the user clicks on element2 the following happens:

1. The `click` event starts in the capturing phase. The event looks if any ancestor element of element2 has a `onclick` event handler for the capturing phase and doesn't find any.
2. The event travels down to the target itself. The event moves to its bubbling phase and executes `doSomething()`, which is registered to element2 for the bubbling phase.
3. The event travels upwards again and checks if any ancestor element of the target has an event handler for the bubbling phase.
4. The event finds one on element1. Now `doSomething2()` is

Event Handler

- If you would like to stop the propagation
  - IE – window.event.cancelBubble = true
  - W3C model – a.stopPropagation();

```
function doSomething(e)
{
        if (!e) var e = window.event;
        e.cancelBubble = true;
        if (e.stopPropagation) e.stopPropagation();
}
```

Event Handler

- Current Target
  - During the capturing and bubbling phases the target does not change: it always remains a reference to element 2

```
element1.onclick = doSomething;
element2.onclick = doSomething;
```

  - How do you know which HTML element is currently handling the event?
  - Target/srcElement don't give a clue, they always refer to element2
  - In W3C has been added currentTarget (non in IE).
  - You can however use the this keyword.

Event Object Properties

- When an event occurs, the browsers make an Event object available to the handlers
- The object provides information regarding the event
- Differences between browsers
  - Typically, the Event object is passed to the handler except for Internet Explorer
    - In IE, the Event object is made available as a property of the object Window, that is, a global variable
  - Some properties have different names
    - For instance, to find the reference to the element where the event is arisen:
      - target (in FireFox)
      - srcElement (in IE)

How to write a code independent of the browser for handling events

- To access the event object
  - function handler(e) {
    e=(!e) ? window.event : e;

    If the browser is IE, the parameter e is not initialized. Thus, e has to be explicitly initialized to contain the Event object
- To refer to the object where the event is arisen
  - obj=(e.target != null) ? e.target : e.srcElement

Event Object Properties

Property	Description	IE	F	O	W3C
altKey	Returns whether or not the "ALT" key was pressed when an event was triggered	6	1	9	Yes
button	Returns which mouse button was clicked when an event was triggered	6	1	9	Yes
clientX	Returns the horizontal coordinate of the mouse pointer when an event was triggered	6	1	9	Yes
clientY	Returns the vertical coordinate of the mouse pointer when an event was triggered	6	1	9	Yes
ctrlKey	Returns whether or not the "CTRL" key was pressed when an event was triggered	6	1	9	Yes
metaKey	Returns whether or not the "meta" key was pressed when an event was triggered	6	1	9	Yes
relatedTarget	Returns the element related to the element that triggered the event	No	1	9	Yes
screenX	Returns the horizontal coordinate of the mouse pointer when an event was triggered	6	1	9	Yes
screenY	Returns the vertical coordinate of the mouse pointer when an event was triggered	6	1	9	Yes
shiftKey	Returns whether or not the "SHIFT" key was pressed when an event was triggered	6	1	9	Yes

Event Object Properties

Property	Description
bubbles	Returns a Boolean value that indicates whether or not an event is a bubbling event
cancelable	Returns a Boolean value that indicates whether or not an event can have its default action prevented
currentTarget	Returns the element whose event listeners triggered the event
eventPhase	Returns which phase of the event flow is currently being evaluated
target	Returns the element that triggered the event
timeStamp	Returns the time stamp, in milliseconds, from the epoch (system start or event trigger)
type	Returns the name of the event
isTrusted	Indicates whether or not the event was initiated by the browser or by a script

Università di Pisa
Francesco Marcelloni

Event Object Events

Event Handlers can be used in three different ways to trigger a function

1. Link Events

   Places an Event Handler as an attribute within an <a href = > tag

   For example:

```
<div>
<a href="#" onClick="alert('Ooo, do it again!');">
        Click on me!</a>
<a href="javascript:void('')" onClick="alert('Ooo, do
        it again!');"> Click on me! </a>
<a href="javascript:alert('Ooo, do it again!')" >
        Click on me!</a>
</div>
```


Università di Pisa
Francesco Marcelloni

Event Object (Events)

Note: No <script> tag.

- Anything that appears in the quotes of an onClick or an onMouseOver is automatically interpreted as JavaScript.
- href="#" tells the browser to look for the anchor #, but there is no anchor "#", so the browser goes to the top of the page since it could not find the anchor.
- <a href="javascript:void(' ')" tells the browser not to go anywhere –
  - it "deadens" the link when you click on it.
  - href="javascript: is the way to call a function when a link (hyperlink or an HREFed image) is clicked.

Università di Pisa
Francesco Marcelloni

Event Object (Events)

2. Actions within forms

```
function checkField(fld){
    if (fld.checkValidity() && fld.value>100)
    {
            alert("Correct number");
            fld.style.backgroundColor="white";
    }
    else
    { alert("Please enter a number greater than 100");
            fld.value=null;
            fld.style.backgroundColor="red";
            setTimeout(function()
             {document.getElementById('idname').focus();},1000);
    }
}
```

Event Object (Events)

```
function setStyle(fld)
{fld.style.backgroundColor="yellow"}
</script>
</head>

<body>
  <form id="frm1" action="form_action.asp">
  <p>Insert a Number: <input type="input" required
     pattern="^[0-9]+$" id="idname"
     onfocus="setStyle(this)"
     onchange="checkField(this)"></p>
   </form>
 </body>
</html>
```


Event Object (Events)

3. Body onLoad and unLoad
   - These Event Handlers are defined in the <body> or <frameset> tag of an HTML file and are invoked when the document or frameset are fully loaded or unloaded.
   - If you set a flag within the onLoad Event Handler, other Event Handlers can test this flag to see if they can safely run, with the knowledge that the document is fully loaded and all objects are defined.
   - Used to check the visitor's browser type and version, and load the proper version of the web page
   - Used to deal with cookies that should be set when a user enters or leaves a page
   - onload - allows launching a particular JavaScript function upon the completion of initially loading the document
   - onunload - is triggered when the user "unloads" or exits the document

Event Object (Example)

```
<head>
<script type="text/javascript">
var loaded = "false";
function doit() {
        alert('Everything is "loaded" and loaded = ' + loaded);
}
function bye(){alert("Bye bye");}
</script>
</head>
```


Event Object (Example)

```
<body onload="loaded='true';" onunload="bye()">
<form id="frm1" action="form_action.asp">
<p>
  <input type="button" value="Press me"
   onClick="if (window.loaded) doit();">
  </p>
</form>
</body>
```

Examples (keydown event) Move a figure on the page

```
<html>
 <head>
  <title>  Events  </title>
  <meta http-equiv="Content-Type"
        content="text/html; charset=us-ascii">
 </head>
 <body onload="set()">
  <div id="mydiv"
       style="position:absolute; left:450px; top:300px;">
    <img src="tower.jpg">
  </div>
        <script type="text/javascript"
    src="./myscript12.js">
</script>
 </body>
</html>
```


Examples (keydown event) Move a figure on the page

```
var elem = document.getElementById('mydiv');
function set() {document.onkeydown = onKeyHandler;}

function onKeyHandler(e) {
e = (!e) ? window.event : e;                              //Explorer -> !e
var key = (e.which != null) ? e.which : e.keyCode;      //Firefox -> e.which
switch (key){
case 37: move(-3, 0); break;        // left
case 38: move(0, -3); break;        // up
case 39: move(3,  0); break;        // right
case 40: move(0,  3); break;        // down
default: window.alert('stop');  }
}

function move(dx,dy) {
elem.style.left = parseInt(elem.style.left) + dx + "px";
elem.style.top  = parseInt(elem.style.top) + dy + "px"; }
```


Timer Events

setInterval("function()", delay)

Calls a function repeatedly, with a fixed time delay between each call to that function. The setInterval() method will continue calling the function until clearInterval() is called, or the window is closed.

delay is expressed in millisec.

setTimeout("function()", delay)

Calls a function after a specified number “delay” of milliseconds.

clearInterval(id_of_setinterval)

Clears a timer set with the setInterval() method. The ID value returned by setInterval() is used as the parameter for the clearInterval() method.

clearTimeout(id_of_settimeout)

Clears a timer set with the setTimeout() method. The ID value returned by setTimeout() is used as the parameter for the clearTimeout() method.

Timer Events

```
<!-- Head -->
<title>setInterval/clearInterval example</title>
 <script type="text/javascript">
var intervalID;
function changeColor()
{    intervalID = setInterval(flashText, 1000);   }

function flashText()
{ var elem = document.getElementById("my_box");
  if (elem.style.color == 'red')
  {   elem.style.color = 'blue';  }
  else
  {       elem.style.color = 'red';  }
}
```

Timer Events

```
function stopTextColor()
{  clearInterval(intervalID);}
</script>
</head>

<body onload="changeColor();">
<div id="my_box">
<p>Hello World</p>
<button onclick="stopTextColor();">Stop</button>
</div>
</body>
</html>
```


Examples

Dynamic Effects

- Dynamic effects are obtained by combining events and properties of the different objects
- Examples
  - Rollover – change of the images when the user passes over the images with the cursor of the mouse
  - Brief Animations
  - Sliding text
  - Gradual change of the background colour
  - Layer management

Rollover

```
<body>
  <div>
    <img src="a.jpg" height="100" width="100"
       alt="a" onmouseover="modify(0,'over')"
       onmouseout="modify(0,'out')">
    <img src="b.gif" height="100" width="100"
       alt="b" onmouseover="modify(1,'over')"
       onmouseout="modify(1,'out')">
  </div>
  <script type="text/javascript"   src="./myscript13.js"></script>
</body>
```

Rollover

```
//myscript13.js
var img1 = new Array(2); //
img1[0] = new Image();  img1[1] = new Image();
var img2 = new Array(2);  img2[0]= new Image();
img2[1]= new Image();
img1[0].src="a.jpg"; img1[1].src="b.gif"; //default images
img2[0].src="b.gif"; img2[1].src="a.jpg";//rollover images

function modify(i,type) {
  if (type == "over")
         document.images[i].src = img2[i].src; //mouseover
  else    document.images[i].src = img1[i].src;//mouseout
}
```


Sliding Text

```
<style type="text/css">
#slidText { color : red; background:black}
</style>
</head>
 <body onload="setInterval('slide()',100);">
  <form action='#'>
   <p> <input type="text"
        size="30"
        name="mytext“
        id = "slidText"
        style="border: solid;">   </p> </form>
  <script type="text/javascript"  src="./myscript12.js"></script>
 </body>
```


Sliding Text

```
//myscript14.js
var text = "Example of Sliding Text .............";
function slide() {
var firstchar = text.charAt(0);
text = text.slice(1,text.length) + firstchar;
document.forms[0].mytext.value = text}
```


Managing Layers

- Layers allow designers to (partially) superimpose an content on another content
- The element HTML div permits managing the layers using the following properties;
  - position: identifies the position of the element in the page
    - Values: absolute, fixed and relative
  - left, right, top, bottom: identify the position of the left-top or the right-bottom margin with respect to the reference container.
  - height and width: specify the width and height of the layer in pixels
  - z-index: specifies the layer (positive integer) on the z-axis
  - visibility: determines whether the element is visible or not.
    - Values: hidden and visible

Managing Layers Example

```
<link rel="stylesheet" href="mystyleJS.css"  type="text/css"
        media="screen">
 </head>
 <body>
  <h1>   Properties of the levels  </h1>
  <div id="div1" style= "position: absolute; left: 300px; top:150px;
        width:300px;    height:150px; z-index: 1">  Red </div>
  <div id="div2" style= "position: absolute; left: 350px; top:200px;
        width:300px; height:150px; z-index: 2">  Blu </div>
 <div id="div3" style= "position: absolute; left: 400px; top:250px;
        width:300px; height:150px; z-index: 3">   Green   </div>
```


Managing Layers Example

```
<div id="control">
      <form action="#" name="mymodule" id="mymodule">
       <p>
     1) Insert the block:<br>
          <select name="idiv">
           <option value="div1" selected> red </option>
         <option value="div2"> blu </option>
         <option value="div3"> green </option>
         </select><br>
     2) Insert the distance from the top:<br>
         <input type="text" size="5" name="top" value="150"><br>
//...
<input type="button" value="set"   onclick="set()">
//...
  <script type="text/javascript" src="./myscript16.js">
```


Managing Layers Example (myscript15.js)

```
var ERROR = false;
function read(i) {  value = 0;
  with (document.mymodule) {
       switch (i) {      case 1: return idiv.value; break;
                         case 2: value = top.value; break;
                         case 3: value = left.value; break;
                         case 4: value = width.value; break;
                         case 5: value = height.value; break;
                         case 6: value = level.value; }
  }
  value = parseInt(value);
  if ((value < 0) || isNaN(value))
  {    window.alert("Error in the field n." + i);
       ERROR = true;  } return value;  }
```


Managing Layers Example (myscript15.js)

```
function set() {  var i = read(1);
                  var x = read(2);
                  var y = read(3);
                  var w = read(4);
                  var h = read(5);
                  var z = read(6);
        if (!ERROR) {    var mydiv = document.getElementById(i);
                         mydiv.style.top = x + "px";
                         mydiv.style.left = y + "px";
                         mydiv.style.width = w + "px";
                         mydiv.style.height = h + "px";
                         mydiv.style.zIndex = z + "";   }
        ERROR = false;}
```

Managing Layers Example (mystyleJS.js)

```
div {     border:          outset;
          font-size:       xx-large;
          font-weight:     bolder;
          overflow:        hidden;}
#div1 {  background-color: red;}
#div2 {  background-color: blue; }
#div3 {  background-color: green; }
#control {      background-color: azure;  z-index:        0;
                width:          15em;  padding:         5px;
                font-size:       medium;  font-weight:      normal;
          }
```


Università di Pisa
Francesco Marcelloni

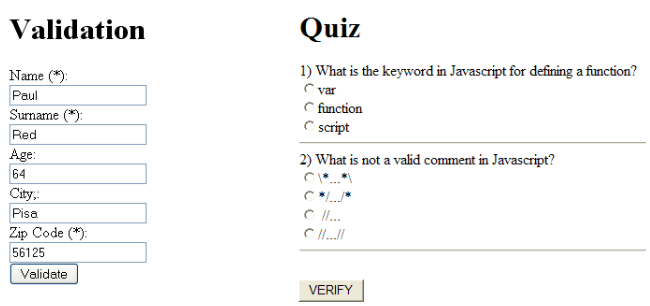
Validation of data inserted into a Form

- JavaScript allows verifying the data inserted into a form directly on the client-side
- Using handlers of appropriate events (onClick and onBlur), the data can be analysed and messages can be sent to the user
  - onClick – when the form is completed and sent to the server
  - onBlur – when the focus is passed on another element
- Typically, the verification is performed only on the input text fields

Università di Pisa
Francesco Marcelloni

Validation of data inserted into a Form

- Two examples
  - Validation of data
  - Assessment of the answers to a quiz



Università di Pisa
Francesco Marcelloni

Validation of data inserted into a Form

Validation of the data inserted by using the previous form

```
var fields = document.getElementsByTagName("input");
var shouldBeALPHANUMERICAL = [0, 1, 4];   //(1)
var shouldBeNUMERICAL      = [2, 4];
var shouldBeALPHABETICAL   = [0, 1, 3];
var shouldBe5NUMBERS     = [4];
var MESSAGES = [  "The following field does not have valid symbols: ",
                  "The following field is not exclusively numerical: ",
                  "The following field must have five digits: ",
                  "The following field must have only letters: ",
                  "...",
                  "OK"];
```

Validation of data inserted into a Form

```
var existALPHANUMERICAL    = /\w/;
var existNONALPHANUMERICAL = /\W/;
var existNONNUMERICAL     = /\D/;
var existNUMERICAL        = /\d/;
var exist5NUMERICAL       = /^\d{5}$/;
function error(idmess, field)
{  window.alert(MESSAGES[idmess] + field.name);
  field.focus();   field.select();
}
```


Validation of data inserted into a Form

```
function isTrue(COND, ELEM, BOOL, MESS) {
for (var i=0; i<ELEM.length; i++) {
    var j = ELEM[i];
    field = fields[j];
    if (COND.test(field.value) == BOOL) {
        error(MESS, field);
        return true;
                }
    }
    return false;
}
```


Validation of data inserted into a Form

```
function validate() {
if (isTrue(existALPHANUMERICAL,    shouldBeALPHANUMERICAL, false, 0))
  return;
if (isTrue(existNONALPHANUMERICAL, shouldBeALPHANUMERICAL, true,  0))
  return;
if (isTrue(existNONNUMERICAL,     shouldBeNUMERICAL,     true,  1))
  return;
if (isTrue(exist5NUMERICAL,       shouldBe5NUMBERS,     false, 2))    return;
if (isTrue(existNUMERICAL,        shouldBeALPHABETICAL,   true,  3))    return;
if (isTrue(existNONALPHANUMERICAL, shouldBeALPHABETICAL,   true,  3))
  return;
  window.alert(MESSAGES[MESSAGES.length-1]);
}
```


Assessment of the answers to a quiz

```
<form action="#" name="mymodule" id="mymodule">
<script type= "text/javascript">
   <!--
   for (var i=0; i<Queries.length; i++)
   {
        document.writeln("<div>" + (i+1) + ") "+ Queries[i] + "<br>");
        for (var j=0; j<Options[i].length; j++)
        document.writeln("<input type='radio' name='q" + i + "'>" +
                        Options[i][j] + "<br>");
        document.writeln("<hr></div>");
        }
     -->
</script>
<p>
<input type="button" value="VERIFY" onclick="check()"></p>
```

Assessment of the answers to a quiz

```
//myscript18.js
var Queries = [
"What is the keyword in Javascript for defining a function?",
"What is not a valid comment in Javascript?",
"What is the handler for the 'loss of active state' event?",
"Which operator can be used to instance an object?",
"To periodically call a function you use the method:"];
var Options = [  ["var","function","script"],
  ["\\*...*\\","*/.../*"," //...","//...//"],
  ["onFocus","onBlur","onClick"],
  ["create","new","add"],
  ["window.setInterval","window.setTimeout","date.setTime"]];
```


Assessment of the answers to a quiz

```
//myscript18.js
var Answers = [  0,1,0,  0,1,0,0,  0,1,0,  0,1,0,  1,0,0];
function check() {  var score = 0;
                    var answers =
  document.getElementsByTagName("INPUT");
  for (var i = 0; i < answers.length-1; i++)
        if (answers[i].checked)
                if (Answers[i]==1) score++;
                else score--;
        window.alert("Your score is: " + score);
        document.mymodule.reset();
}
```


Web Storage

Cookie

- A cookie is a piece of text stored by a user's web browser.
- A cookie can be used for
  - authentication,
  - storing site preferences,
  - shopping cart contents,
  - the identifier for a server-based session.
- A cookie consists of one or more name-value pairs containing bits of information, which may be encrypted for information privacy and data security purposes.
- The cookie is sent as an HTTP header by a web server to a web browser and then sent back unchanged by the browser each time it accesses that server.

Cookie

- Cookies may be set by the server with or without an expiration date.
  - Cookies without an expiration date exist until the browser terminates
  - Cookies with an expiration date may be stored by the browser until the expiration date passes.
  - Users may also manually delete cookies in order to save space or to avoid privacy issues.
- Cookies may be used to remember the information about the user who has visited a website in order to show relevant content in the future.
  - For example a web server may send a cookie containing the username last used to log in to a web site so that it may be filled in for future visits.

Cookie

```
function setCookie(name, value, expires, path, domain,
  secure )
{ // set time, it's in milliseconds
var today = new Date();
if ( expires ) { expires = expires * 1000 * 60 * 60 * 24;}
var expires_date = new Date( today.getTime() + expires);
document.cookie = name + "=" +escape( value ) +
( ( expires ) ? ";expires=" + expires_date.toGMTString() : "" )
  + ( ( path ) ? ";path=" + path : "" ) +
( ( domain ) ? ";domain=" + domain : "" ) +
( ( secure ) ? ";secure" : "" );
}
```


Cookie

Parameters:

- name – name of the cookie
- value – value of the cookie
- expire – the time the cookie expires
- path – path on the server in which the cookie will be available on. If set to '/', the cookie will be available within the entire domain.
- domain - The domain that the cookie is available to. To make the cookie available on all subdomains of example.com (including example.com itself) then you'd set it to '.example.com'.
- secure - Indicates that the cookie should only be transmitted over a secure HTTPS connection from the client. When set to **TRUE**, the cookie will only be set if a secure connection exists.

Cookie

```
function getCookie( check_name ) {
  // first we'll split this cookie up into name/value pairs
  // note: document.cookie only returns name=value, not the other
  components
  var a_all_cookies = document.cookie.split( ';' );
  var a_temp_cookie = '';
  var cookie_name = '';
  var cookie_value = '';
  var b_cookie_found = false; // set boolean t/f default f

  for ( i = 0; i < a_all_cookies.length; i++ )
  {       // now we'll split apart each name=value pair
          a_temp_cookie = a_all_cookies[i].split( '=' );

          // and trim left/right whitespace while we're at it
          cookie_name = a_temp_cookie[0].replace(/^\s+|\s+$/g, '');
```

Cookie

```
// if the extracted name matches passed check_name
if ( cookie_name == check_name )
{       b_cookie_found = true;
        // we need to handle case where cookie has no value
                but exists (no = sign, that is):
        if ( a_temp_cookie.length > 1 )
        { cookie_value =
unescape(a_temp_cookie[1].replace(/^\s+|\s+$/g, '') );
        }
// note: if cookie is initialized but no value, null is returned
        return cookie_value;
}
a_temp_cookie = null;   cookie_name = '';
}
```


Cookie

```
 if ( !b_cookie_found )
 { return null;}
}

function deleteCookie( name, path, domain ) {
name=prompt('Please enter the cookie to remove:','');
if ( getCookie( name ) ) document.cookie = name + "=" +
( ( path ) ? ";path=" + path : "") +
( ( domain ) ? ";domain=" + domain : "" ) +
";expires=Thu, 01-Jan-1970 00:00:01 GMT";
}

function deleteC()
{ name=prompt('Please enter the cookie to remove:','');
 deleteCookie(name,'/','');
}
```


Cookie

```
function checkCookie()
{ username=getCookie('username');
  if (username!=null && username!="")
  {  alert('Welcome again '+username+'!');  }
else
  {  username=prompt('Please enter your name:',"");
  if (username!=null && username!="")
    {    setCookie('username',username,100,'/');    }
  }
}
....
<body onLoad="checkCookie()">
<p><a href="javascript:deleteC()">
  Click to delete the cookie</a></p>
</body>
```


Session and Local Storage

- Cookies are extremely limited in size. Generally, only about 4KB.
- Cookies are transmitted back and forth from server to browser on every request scoped to that cookie.
  - security risks
  - network bandwidth consumption
- The same results could be achieved without involving a network or remote server and storing up to a few megabytes.
  - sessionStorage – values survive either across page loads in a single window or tab
  - localStorage – values survive across browser restarts

Session and Local Storage
Browser Support

```
function checkStorageSupport() {
//sessionStorage
if (window.sessionStorage) {
alert('This browser supports sessionStorage');
} else {
alert('This browser does NOT support sessionStorage');
}
//localStorage
if (window.localStorage) {
alert('This browser supports localStorage');
} else {
alert('This browser does NOT support localStorage');
}
}
```


Session and Local Storage
Browser Support

- Setting a value

```
window.sessionStorage.setItem('myFirstKey', 'myFirstValue');
```

- Retrieve a value

```
window.sessionStorage.getItem('myFirstKey')
```

- setItem and getItem calls can be avoided entirely by simply setting and retrieving values corresponding to the key-value pair directly on the sessionStorage object.

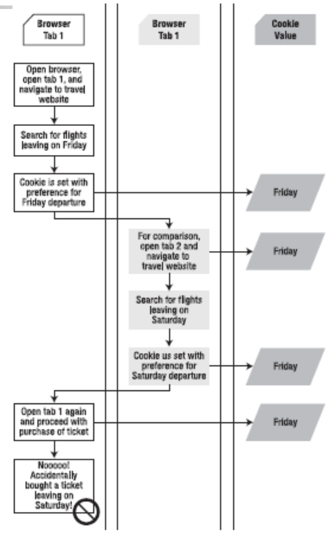
```
window.sessionStorage.myFirstKey = 'myFirstValue';
window.sessionStorage.myFirstKey;
```


Session and Local Storage
How long do the values persist?

- Objects in sessionStorage
  - These objects will persist as long as the browser window (or tab) is not closed.
  - Session storage is perfect for short-lived processes that would normally be represented in wizards or dialogs.
  - Solves a problem that has plagued many web-applications: scoping of values.

Session and Local Storage
How long do the values persist?

- The figure shows the problem of value scoping with cookies
- Solution: sessionStorage allows temporary values like a departure date to be saved across pages that access the application but not leak into other windows where the user is also browsing for flights. Therefore, those preferences will be isolated to each window where the corresponding flights are booked.



Local Versus Session Storage

- localStorage – values survive across browser restarts

sessionStorage	localStorage
Values persist only as long as the window or tab in which they were stored.	Values persist beyond window and browser lifetimes.
Values are only visible within the window or tab that created them.	Values are shared across every window or tab running at the same origin.

- The storage interface

```
interface Storage {
readonly attribute unsigned long length;
getter DOMString key(in unsigned long index);
getter any getItem(in DOMString key);
setter creator void setItem(in DOMString key, in any data);
deleter void removeItem(in DOMString key);
void clear();
};
```


Storage

- length – specifies how many key-value pairs are currently stored in the storage object.
- key(index) – allows retrieval of a given key.
- getItem(key) – is one way to retrieve the value based on a given key.
- setItem(key, value) – will put a value into storage under the specified key name, or replace an existing value if one already exists under that key name.
- removeItem(key) – If a value is currently in storage under the specified key, this call will remove it. If no item was stored under that key, no action is taken.
- clear() – removes all values from the storage list.

Communicating Web Storage Updates

- To register to receive the storage events of a window's origin, simply register an event listener, for example:

```
window.addEventListener("storage", displayStorageEvent, true);
```

- The StorageEvent Interface

```
interface StorageEvent : Event {
readonly attribute DOMString key;
readonly attribute any oldValue;
readonly attribute any newValue;
readonly attribute DOMString url;
readonly attribute Storage storageArea;
};
```


Storage Event

- key – contains the key value that was updated or removed in the storage.
- oldValue – contains the previous value corresponding to the key before it was updated, and the newValue contains the value after the change.
- url – points to the origin where the storage event occurred.
- storageArea – provides a convenient reference to the sessionStorage or localStorage where the value was changed. This gives the handler an easy way to query the storage for current values or make changes based on other storage changes.

Display Content of a Storage Event

```
// display the contents of a storage event
function displayStorageEvent(e) {
var logged = "key:" + e.key + ", newValue:" + e.newValue + ", oldValue:" +
e.oldValue +", url:" + e.url + ", storageArea:" + e.storageArea;
alert(logged);
}

// add a storage event listener for this origin
window.addEventListener("storage", displayStorageEvent, true);
```


Example

```
<body>
  Key:  <input type="text" id="key"><br>
  Value: <input type="text" id="value"><br>
  <input type="button" id="add" value="Add to Storage"> 
  <input type="button" id="remove" value="Remove from Storage"> 
  <input type="button" id="clear" value="Clear Storage"><br>
  Contents of Local Storage:<br>
  <div id="content"></div>
</body>
```

Note that the example does not work correctly in Chrome if the page runs from the file protocol (file:///).

Progettazione Web

PHP

Francesco Marcelloni

Dipartimento di Ingegneria dell'Informazione
Università di Pisa
ITALY

What is PHP

- PHP is recursive acronym for PHP: Hypertext Preprocessor
- PHP is a server-side scripting language
- PHP scripts are executed on the server
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)
- PHP is an open source software
- PHP is free to download and use

What is a PHP file

- PHP files can contain text, HTML tags and scripts
- PHP files are returned to the browser as plain HTML
- PHP files have a file extension of ".php", ".php3", or ".phtml"

- PHP+MySQL
  - PHP combined with MySQL are cross-platform (you can develop in Windows and serve on a Unix platform)

Why PHP?

- PHP runs on different platforms (Windows, Linux, Unix, etc.)
- PHP is compatible with almost all web servers used today (Apache, IIS, etc.)
- PHP is FREE to download from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side

Where to Start?

- To get access to a web server with PHP support, you can:
  - Install Apache (or IIS) on your own server, install PHP, and MySQL
  - Or find a web hosting plan with PHP and MySQL support

Università di Pisa Francesco Marcelloni

How is PHP included in HTML pages?

1. <?php echo 'if you want to serve XHTML or XML documents, do it like this'; ?>

2. <script language="php">
   echo 'some editors (like FrontPage) don\'t
   like processing instructions';
   </script>

3. <? echo 'this is the simplest, an SGML processing instruction'; ?>
   <?= expression ?> This is a shortcut for "<? echo expression ?>"

Tags in 1. and 2. are both always available. Tag 1. is the most commonly used, and recommended, of the two.

Short tags (example 3.) are only available when they are enabled via the short_open_tag php.ini configuration file directive, or if PHP was configured with the --enable-short-tags option.

Università di Pisa Francesco Marcelloni

PHP Syntax

- A PHP scripting block can be placed anywhere in the document.
- A PHP file normally contains HTML tags, just like an HTML file, and some PHP scripting code

```
<?php if ($expression) { ?>
   <strong>This is true.</strong>
<?php } else {  ?>
   <strong>This is false.</strong>
   <?php } ?>
```

- When PHP hits the ?> closing tags, it simply starts outputting whatever it finds (except for an immediately following newline) until it hits another opening tag.

Università di Pisa Francesco Marcelloni

PHP Example

```
<!DOCTYPE html>
<html lang="en">
  <head>
  <meta charset="utf-8">
    <title> PHP Example </title>
  </head>
  <body>
      <p><?php   echo "Hi, I'm a PHP script!";  ?></p>
    </body>
</html>
```

- NOTE: The file must have a .php extension. If the file has a .html extension, the PHP code will not be executed.

Instruction Separation and Comments

- NOTE: Each code line in PHP must end with a **semicolon**. The closing tag of a block of PHP code automatically implies a semicolon; you do not need to have a semicolon terminating the last line of a PHP block.
- NOTE: The closing tag for the block will include the immediately trailing newline if one is present.
- NOTE: In PHP, comments are expressed as:
  - // to make a single-line comment
  - /* to make a large comment block */
  - # to make a single comment as in Unix shell

PHP Variables

- All variables in PHP start with a $ sign symbol.
  $var_name = value;
- In PHP, a variable does not need to be declared before adding a value to it.
- PHP automatically converts the variable to the correct data type, depending on its value.
- A variable name is case-sensitive and must start with a letter or an underscore "_"
- A variable name can only contain alpha-numeric characters and underscores (a-z, A-Z, 0-9, and _)

PHP Types

PHP supports eight primitive types.

- Four scalar types:
  - boolean
  - integer
  - float (floating-point number)
  - string
- Two special types:
  - NULL
  - resource
- Two compound types:
  - array
  - object

PHP Types
Boolean and Integer

- Boolean
  - To specify a boolean literal, use the keywords TRUE or FALSE. Both are case-insensitive.
- Integer
  - Integers can be specified in decimal (base 10), hexadecimal (base 16), or octal (base 8) notation, optionally preceded by a sign (- or +).
  - To use octal notation, precede the number with a 0 (zero). To use hexadecimal notation precede the number with 0x.
  - Integer size can be determined using the constant **PHP_INT_SIZE**, and maximum value using the constant **PHP_INT_MAX**
  - If PHP encounters a number beyond the bounds of the integer type, it will be interpreted as a float instead. Also, an operation which results in a number beyond the bounds of the integer type will return a float instead.

PHP Types
Floating-Point Numbers

- Floating point numbers

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
?>
```

  - The size of a float is platform-dependent, although a maximum of ~1.8e308 with a precision of roughly 14 decimal digits is a common value (the 64 bit IEEE format).

PHP Types
NULL and Resource

- NULL

  The special NULL value represents a variable with no value. NULL is the only possible value of type NULL.

  A variable is considered to be null if:
  - the constant NULL has been assigned to it
  - it has not been set to any value yet.
  - it has been unset().

    The function is_null() allows evaluating whether the value of a variable is NULL
- Resource
  - A resource is a special variable, holding a reference to an external resource. Resources are created and used by special functions. For instance, dbase_open() opens a database.

PHP Types
String

- String
  1. Single quoted

     echo 'This is great
';
     - To specify a literal single quote, escape it with a backslash (\). To specify a literal backslash before a single quote, or at the end of the string, double it (\\).
  2. Double quoted
     - The most important feature of double-quoted strings is the fact that variable names will be expanded.
     - Simple syntax

       If a dollar sign ($) is encountered, the parser will greedily take as many tokens as possible to form a valid variable name. Enclose the variable name in curly brackets to explicitly specify the end of the name.

PHP Types
String

- Complex Curly Syntax

  Any scalar variable, array variable or object property with a string representation can be included via this syntax.
  - Simply write the expression the same way as it would appear outside the string, and then wrap it in { and }.
  - Since { can not be escaped, this syntax will only be recognized when the $ immediately follows the {.
  - Use {\$ to get a literal {$.
- PHP uses the same escape sequences ("\") as in C++

  \n, \t, \\, \$, \", \0, *\[0-7]{1,3}, \x[0-9A-Fa-f]{1,2}*

PHP Types
String

- Complex Curly Syntax (example)

```
<?php
$great = 'fantastic';

// Won't work, outputs: This is { fantastic}
echo "This is { $great}";

// Works, outputs: This is fantastic
echo "This is {$great}";

// Works
echo "This square is {$square->width}00 centimeters
  broad.";

// Works
echo "This works: {$arr[4][3]}";
?>
```


PHP Types
String

3. Heredoc

   A third way to delimit strings is the heredoc syntax: <<<. After this operator, an identifier is provided, then a newline. The string itself follows, and then the same identifier again to close the quotation. Heredoc text behaves just like a double-quoted string.

```
<?php
$str = <<<EOD
Example of string
spanning multiple lines
using heredoc syntax.
EOD;
echo $str;
?>
```

Output: Example of string spanning multiple lines using heredoc syntax.

PHP Types
String

- Characters within strings may be accessed and modified using square array brackets (or also braces)

```
<?php
// Get the first character of a string
$str = 'This is a test.';
$first = $str[0];

// Get the last character of a string.
$str = 'This is still a test.';
$last = $str[strlen($str)-1];

// Modify the last character of a string
$str = 'Look at the sea';
$str{strlen($str)-1} = 'e';
?>
```


PHP Types
String

- Concatenation operator ('.') - returns the concatenation of its right and left arguments.
- Concatenating assignment operator ('.=') - appends the argument on the right side to the argument on the left side.

```
<?php
$a = "Hello ";
$b = $a . "World!"; // now $b contains "Hello World!"
$a = "Hello ";
$a .= "World!";     // now $a contains "Hello World!"
?>
```

PHP Types
String

- The strpos() function is used to search for character within a string. If no match is found, it will return FALSE

```
<?php
echo strpos("Hello world!","world");
?>
```

- The position of the string "world" in our string is position 6 (the first position in the string is 0).

PHP Types
String Functions

- string chr (int $ascii) - Returns an one-character string containing the character specified by ascii.
- int ord (string $string) - Returns the ASCII value of the first character of string.
- void echo (string $arg1 [, string $...]) – Outputs all parameters. If you want to pass more than one parameter to echo(), the parameters must not be enclosed within parentheses.
  Es: echo $first, "<br>";
- array explode (string $delimiter , string $string [, int $limit]) - Returns an array of strings (at most limit), each of which is a substring of string formed by splitting it on boundaries formed by the string delimiter.

PHP Types
String Functions

- string implode (string $glue , array $pieces) - Join array elements with a glue string.

```
<?php
$array = array('lastname', 'email', 'phone');
$comma_separated = implode(",", $array);
echo "$comma_separated <br>";   // lastname,email,phone
$array = explode(",", $comma_separated);
for($i = 0, $size = sizeof($array); $i < $size; ++$i)
    echo "$array[$i] ";        // lastname email phone
?>
```


PHP Types
String

- string number_format (float $number , int $decimals = 0 , string $dec_point = '.' , string $thousands_sep = ',') - generates the representation in string of a real number
  - Only one parameter: number will be formatted without decimals, but with a comma (",") between every group of thousands.
  - Two parameters: number will be formatted with decimals with a dot (".") in front, and a comma (",") between every group of thousands.
  - Four parameters: number will be formatted with decimals, dec_point instead of a dot (".") before the decimals and thousands_sep instead of a comma (",") between every group of thousands.

PHP Types
String

```
<?php
$number = 1234.56;
// english notation (default)
$english_format_number = number_format($number);
// 1,235
// French notation
$nombre_format_francais = number_format($number, 2, ',', ' ');
// 1 234,56
$number = 1234.5678;
// english notation without thousands separator
$english_format_number = number_format($number, 2, '.', '');
// 1234.57
?>
```


PHP Types
String

- void parse_str (string $str [, array &$arr])
  Parses str as if it were the query string passed via a URL and sets variables in the current scope.

```
<?php
$str = "first=value&arr[]=foo+bar&arr[]=baz";
parse_str($str);
echo "$first ";                          // value
echo "$arr[0] ";                         // foo bar
echo "$arr[1]<br>";                      // baz

parse_str($str, $output);
echo "{$output['first']} ";              // value
echo "{$output['arr'][0]} ";             // foo bar
echo "{$output['arr'][1]}<br>";          // baz

?>
```


PHP Types
String

- int strncmp (string $str1 , string $str2 , int $len)
  This function is similar to strcmp(), with the difference that you can specify the (upper limit of the) number of characters from each string to be used in the comparison.
  Returns
  - < 0 if str1 is less than str2;
  - > 0 if str1 is greater than str2,
  - 0 if they are equal.

PHP Types
Array

- An array in PHP is actually an ordered map.
  - A map is a type that associates values to keys.
- Optimized for several different uses:
  - array, list (vector), hash table (an implementation of a map), dictionary, collection, stack, queue.
- Since array values can be other arrays, trees and multidimensional arrays are also possible.
- An array can be created by the *array()* language construct. It takes as parameters any number of comma-separated key => value pairs.

PHP Types
Array

```
array(  key =>  value
     , ...
     )
```

- key may only be an integer or string
  (booleans are converted into integers and NULL is managed as an empty string)
- value may be any value of any type

```
<?php
    $arr = array("foo" => "bar", 12 => true);
    echo $arr["foo"];      // bar
    echo $arr[12];         // 1
?>
```


PHP Types
Array

- If a key is not specified for a value, the maximum of the integer indices is taken and the new key will be that value plus 1.
- If a key that already has an assigned value is specified, that value will be overwritten.

```
<?php
// This array is the same as ...
array(5 => 43, 32, 56, "b" => 12, "a");

// ...this array
array(5 => 43, 6 => 32, 7 => 56, "b" => 12, 8 =>"a");
?>
```


PHP Types
Array

- A numeric array stores each array element with a numeric index.
- Two methods to create a numeric array
  1. `$cars=array("Saab","Volvo","BMW","Toyota");`
  2. ```
     $cars[0]="Saab";
     $cars[1]="Volvo";
     $cars[2]="BMW";
     $cars[3]="Toyota";
     ```


PHP Types
Array

```
<?php
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
// or
// $ages['Peter'] = 32;
// $ages['Quagmire'] = 30;
// $ages['Joe'] = 34;
echo "Peter is " . $ages['Peter'] . " years old.";
?>
```

Output: Peter is 32 years old.
```

PHP Types
Array

- An existing array can be modified by explicitly setting values in it.
  - This is done by assigning values to the array, specifying the key in brackets. The key can also be omitted.

```
<?php
$arr = array(5 => 1, 12 => 2);

$arr[] = 56;          // This is the same as $arr[13] = 56;
                      // at this point of the script

$arr["x"] = 42;       // This adds a new element to
                      // the array with key "x"

unset($arr[5]);       // This removes the element from the array

unset($arr);          // This deletes the whole array
?>
```


PHP Types
Array

- To fill an array with *num* entries of the value of the *value* parameter, keys starting at the *start_index* parameter.

array array_fill (int $start_index , int $num , mixed $value)

If start_index is negative, the first index of the returned array will be start_index and the following indices will start from zero

```
<?php
$a = array_fill(5, 6, 'banana');
$b = array_fill(-2, 4, 'pear');
print_r($a);          //displays in a way that's readable by humans
echo '<br>';
print_r($b);
?>
```

Array ([5] => banana [6] => banana [7] => banana [8] => banana [9] => banana [10] => banana)
Array ([-2] => pear [0] => pear [1] => pear [2] => pear)

PHP Types
Multidimensional Array

- In a multidimensional array, each element in the main array can also be an array and each element in the sub-array can be an array, and so on.

```
<?php
$arr = array("somearray" => array(6 => 5, 13 => 9, "a" =>
  42));
echo "{$arr["somearray"][6]}<br>";    // 5
echo "{$arr["somearray"][13]}<br>";   // 9
echo "{$arr["somearray"]["a"]}<br>";  // 42
?>
```


PHP Types
Array Functions

- array array_combine (array $keys , array $values) - Creates an array by using the values from the keys array as keys and the values from the values array as the corresponding values.
- array array_keys (array $input [, mixed $search_value [, bool $strict = false]]) - Returns the keys, numeric and string, from the input array.
- array array_merge (array $array1 [, array $array2 [, array $...]]) – Merges the elements of one or more arrays together so that the values of one are appended to the end of the previous one. It returns the resulting array.
  - If the input arrays have the same string keys, then the later value for that key will overwrite the previous one.
  - If the arrays contain numeric keys, the later value will *not* overwrite the original value, but will be appended.

PHP Types
Array Functions

- bool asort (array &$array [, int $sort_flags = SORT_REGULAR]) - sorts an array such that array indices maintain their correlation with the array elements they are associated with.

```
<?php
$fruits = array("d" => "lemon", "a" => "orange", "b" =>
  "banana", "c" => "apple");
asort($fruits);
foreach ($fruits as $key => $val) {
   echo "$key = $val\n";}
$num = array (4,2,7,3);
asort($num);
foreach ($num as $key => $val) {
   echo "$key = $val\n";}
?>
```

Output:
c = apple
b = banana
d = lemon
a = orange

1 = 2
3 = 3
0 = 4
2 = 7

PHP Types
Array Functions

- bool sort (array &$array [, int $sort_flags = SORT_REGULAR])- sorts an array. Elements will be arranged from lowest to highest when this function has completed.
  - # SORT_REGULAR - compare items normally (don't change types)
  - # SORT_NUMERIC - compare items numerically
  - # SORT_STRING - compare items as strings

```
<?php
$fruits = array("lemon", "orange", "banana", "apple");
sort($fruits);
foreach ($fruits as $key => $val) {
   echo "fruits[" . $key . "] = " . $val . "\n";
}
?>
```

Output:
fruits[0] = apple
fruits[1] = banana
fruits[2] = lemon
fruits[3] = orange

PHP Types
Array Functions

- bool ksort (array &$array [, int $sort_flags = SORT_REGULAR]) - sorts an array by key, maintaining key to data correlations.

```
<?php
$fruits = array("d"=>"lemon", "a"=>"orange", "b"=>"banana",
  "c"=>"apple");
ksort($fruits);
foreach ($fruits as $key => $val) {
   echo "$key = $val\n";
}
?>
```

Output:
a = orange
b = banana
c = apple
d = lemon

PHP Types
Array Functions

- array compact (mixed $varname [, mixed $...]) - creates an array containing variables and their values. For each of these, compact() looks for a variable with that name in the current symbol table and adds it to the output array such that the variable name becomes the key and the contents of the variable become the value for that key.

- int extract (array $var_array [, int $extract_type = EXTR_OVERWRITE [, string $prefix]]) - import variables from an array into the current symbol table. Checks each key to see whether it has a valid variable name. It also checks for collisions with existing variables in the symbol table. Returns the number of variables successfully imported.

PHP Types
Array Functions

```
$city  = "San Francisco";
$state = "CA";
$event = "SIGGRAPH";
$location_vars = array("city", "state");
$result = compact("event", "nothing_here", $location_vars);
print_r($result);
$var_array = array("color" => "blue",
                   "size"  => "medium", "shape" => "sphere");
extract($var_array);
echo "<br>$color, $size, $shape\n";
?>
Array ( [event] => SIGGRAPH [city] => San Francisco
  [state] => CA )
  blue, medium, sphere
```


PHP Types
gettype

```
<?php
$a_bool = TRUE;         // a boolean
$a_str  = "foo";        // a string
$a_str2 = 'foo';        // a string
$an_int = 12;           // an integer

echo gettype($a_bool);          // prints out:  boolean
echo gettype($a_str);           // prints out:  string

// If this is an integer, increment it by four
if (is_int($an_int)) {    $an_int += 4;}

// If $a_bool is a string, print it out (does not print out anything)
if (is_string($a_bool)) {    echo "String: $a_bool";}
?>
```


PHP Types
settype

- To forcibly convert a variable to a certain type, either cast the variable or use the settype() function on it. It is forbidden to convert a variable to NULL or resouce types.

```
<?php
$foo = "5bar";        // string
$bar = true;          // boolean

settype($foo, "integer");     // $foo is now 5   (integer)
settype($bar, "string");      // $bar is now "1" (string)
?>
```


Variable scope

- Any variable used inside a function is by default limited to the local function scope

```
<?php
$a = 1;              /* global scope */
function test()
{
    echo $a;         /* reference to local scope variable */
}
test();
?>
```

- Note: This script will not produce any output because the echo statement refers to a local version of the $a variable (undefined variable)

Variable scope
Accessing Global Variables

Two ways for accessing global variables.

- Use of global keyword

```
<?php
$a = 1;
$b = 2;
function sum()
{    global $a, $b;
     $b = $a + $b;
}
sum();
echo $b;
?>
```

- Note: The above script will output 3.

Variable scope
Accessing Global Variables

- Use of the special PHP-defined $GLOBALS array

```
<?php
$a = 1;
$b = 2;
function Sum()
{    $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
}
Sum();
echo $b;
?>
```

- The $GLOBALS array is an associative array with the name of the global variable being the key and the contents of that variable being the value of the array element.

Variable scope
Static Variables

- A static variable exists only in a local function scope, but it does not lose its value when program execution leaves this scope.

```
<?php
function test()
{
    static $a = 0;
    echo $a;
    $a++;
}
?>
```

- $a is initialized only in the first call of the function and, every time the test() function is called, it will print the value of $a and increment it.

Predefined Variable
Superglobals

- Superglobals are built-in variables that are always available in all scopes
  - $GLOBALS — References all variables available in global scope
  - $_SERVER — an array containing server and execution environment information (headers, paths and script locations)
  - $_GET — An associative array of variables passed to the current script via the URL parameters
  - $_POST — An associative array of variables passed to the current script via the HTTP POST method.
  - $_FILES — An associative array of items uploaded to the current script via the HTTP POST method.
  - $_REQUEST — An associative array that by default contains the contents of $_GET, $_POST and $_COOKIE
  - $_SESSION — An associative array containing session variables available to the current script.

Predefined Variable Superglobals

- $_ENV — An associative array of variables passed to the current script via the environment method (provided by the shell under which PHP is running)
- $_COOKIE — An associative array of variables passed to the current script via HTTP Cookies
- $php_errormsg — $php_errormsg is a variable containing the text of the last error message generated by PHP. This variable will only be available within the scope in which the error occurred, and only if the track_errors configuration option is turned on (it defaults to off)
- $HTTP_RAW_POST_DATA — Raw POST data
- $http_response_header — HTTP response headers
- $argc — The number of arguments passed to script when running from the command line – php script.php arg1 arg2
- $argv — Array of arguments passed to the script

References

- By default, variables are always assigned by value.
- PHP also offers another way to assign values to variables: assign by reference.

```
<?php
$foo = 'Bob';                 // Assign the value 'Bob' to $foo
$bar = &$foo;                 // Reference $foo via $bar.
$bar = "My name is $bar";     // Alter $bar...
echo $bar;
echo $foo;                    // $foo is altered too.
?>
```

- There are three basic operations performed using references: assigning by reference, passing by reference, and returning by reference.

References

Function unset() allows separating the variable from its reference

```
<?php
  $a = 1000;
  $b =& $a;
  $b = 50;
  echo "$b<br>";        //50
  echo "$a<br>";        //50
  unset($b);
  $a = 0;
  echo "$a<br>";        //0
  echo "$b<br>";        //The variable is undefined
?>
```


Constants

- A valid constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores.

```
<?php
    define("FOO",     "something");
    define("FOO2",    "something else");
    define("FOO_BAR", "something more");
?>
```

- PHP provides a large number of predefined constants to any script which it runs.

Constants

Name	Description
__LINE__	The current line number of the file.
__FILE__	The full path and filename of the file. If used inside an include, the name of the included file is returned.
__DIR__	The directory of the file. If used inside an include, the directory of the included file is returned.
__FUNCTION__	The function name. The constant returns the function name as it was declared (case-sensitive).
__CLASS__	The class name. The constant returns the class name as it was declared (case-sensitive).
__METHOD__	The class method name. (Added in PHP 5.0.0) The method name is returned as it was declared (case-sensitive).

PHP Operators

- The PHP operators are practically equal to the C++ operators.
- In the following, we will show only the main differences

PHP Operators
Comparison Operators

- We show only the particularities

Example	Name	Result
$a = = $b	Equal	TRUE if $a is equal to $b.
$a = = = $b	Identical	TRUE if $a is equal to $b, and they are of the same type. (introduced in PHP 4)
$a != $b	Not equal	TRUE if $a is not equal to $b.
$a <> $b	Not equal	TRUE if $a is not equal to $b.
$a != = $b	Not identical	TRUE if $a is not equal to $b, or they are not of the same type. (introduced in PHP 4)

PHP Operators
Error Control Operators

- PHP supports one error control operator: the at sign (@). When prepended to an expression in PHP, any error messages that might be generated by that expression will be ignored.
- If the track_errors feature is enabled, any error message generated by the expression will be saved in the variable $php_errormsg.

```
<?php
/* Intentional file error */
$my_file = @file ('non_existent_file') ;
$value = @$cache[$key];
// will not issue a notice if the index $key doesn't exist.
?>
```

PHP Operators
Execution Operators

- PHP supports one execution operator: backticks (` `). Note that these are not single-quotes!
- PHP will attempt to execute the contents of the backticks as a shell command; the output will be returned (i.e., it won't simply be dumped to output; it can be assigned to a variable).
- Use of the backtick operator identical to shell_exec().

```
<?php
// $output = `ls -al`;              Unix
   $output = `dir`;
  echo "<pre>$output</pre>";
?>
```

The backtick operator is disabled when safe mode is enabled or shell_exec() is disabled.

PHP Operators
Logical Operators

Example	Name	Result
$a and $b	And	TRUE if both $a and $b are TRUE.
$a or $b	Or	TRUE if either $a or $b is TRUE.
$a xor $b	Xor	TRUE if either $a or $b is TRUE, but not both.
! $a	Not	TRUE if $a is not TRUE.
$a && $b	And	TRUE if both $a and $b are TRUE.
$a || $b	Or	TRUE if either $a or $b is TRUE.

- The two different variations of "and" and "or" operate at different precedences:
  - $e = false || true;   // Acts like: ($e = (false || true))
  - $f = false or true;   // Acts like: (($f = false) or true)

PHP Operators
Array Operators

Example	Name	Result
$a + $b	Union	Union of $a and $b.
$a == $b	Equality	TRUE if $a and $b have the same key/value pairs.
$a === $b	Identity	TRUE if $a and $b have the same key/value pairs in the same order and of the same types.
$a != $b	Inequality	TRUE if $a is not equal to $b.
$a <> $b	Inequality	TRUE if $a is not equal to $b.
$a !== $b	Non-identity	TRUE if $a is not identical to $b.

The + operator returns the right-hand array appended to the left-hand array; for keys that exist in both arrays, the elements from the left-hand array will be used, and the matching elements from the right-hand array will be ignored.

```
var_dump($a == $b);             // bool(true)
var_dump($a === $b);            // bool(false)
//var_dump displays structured information
```


PHP Operators
Type Operators

- instanceof is used to determine whether a PHP variable is an instantiated object of a certain class

```
<?php
class MyClass {}
class NotMyClass {}

$a = new MyClass;
var_dump($a instanceof MyClass);
var_dump($a instanceof NotMyClass);
?>
```

bool(true)
bool(false)

PHP Statements

- The PHP statements are practically equal to the C++ statements, except for some particular use:
  - if
  - switch
  - while
  - do-while
  - for
  - break
  - continue

- In the following, we highlight the main differences

PHP Statements
If

if (condition)
  code to be executed if condition is true;
elseif (condition)
  code to be executed if condition is true;
else
  code to be executed if condition is false;

```
<?php
$d=date("D");  //Returns a string formatted according
               // to the given format string
if ($d=="Fri")  echo "Have a nice weekend!";
elseif ($d=="Sun")  echo "Have a nice Sunday!";
else  echo "Have a nice day!";
?>
```


PHP Statements
For

```
<?php
for ($i=1; $i<=5; $i++) {  echo "The number is " . $i . "<br>"; }
?>

<?php
$people = array(
      array('name' => 'Kalle', 'salt' => 856412),
      array('name' => 'Pierre', 'salt' => 215863)
      );

for($i = 0, $size = sizeof($people); $i < $size; ++$i)
{   $people[$i]['salt'] = rand(000000, 999999);}
print_r($people);
?>
  Array ( [0] => Array ( [name] => Kalle [salt] => 64086 )
         [1] => Array ( [name] => Pierre [salt] => 49713 ) )
```


PHP Statements
Foreach

Foreach gives an easy way to iterate over arrays.
- Note: foreach works only on arrays
- Two possible syntaxes (the second is an extension of the first)

foreach (array_expression **as** $value)
    statement
\\ On each loop, the value of the current element is assigned to
\\ $value and the internal array pointer is advanced by one

foreach (array_expression **as** $key => $value)
  statement
\\ On each loop, the current element's key will be assigned to the
\\ variable $key and the internal array pointer is advanced by one

PHP Statements
Foreach

- Unless the array is referenced, foreach operates on a copy of the specified array and not the array itself.
- You can easily modify array's elements by preceding $value with &. This will assign reference instead of copying the value.

```
<?php
$arr = array(1, 2, 3, 4);
foreach ($arr as &$value) {
    $value = $value * 2;
}
// $arr is now array(2, 4, 6, 8)
$value=1000;   //Note! $value refers to the last element
print_r($arr);    // $arr is now array(2, 4, 6, 1000)
unset($value); // break the reference with the last element
?>
```


PHP Statements
Break

- break accepts an optional numeric argument which tells it how many nested enclosing structures are to be broken out

```
$i = 0;
while (++$i) {
    switch ($i) {
    case 5:
        echo "At 5<br>";
        break 1;            /* Exit only the switch. */
    case 10:
        echo "At 10; quitting<br>";
        break 2;            /* Exit the switch and the while. */
    default:
        break;
    }
}
```


PHP Statements
Continue

- continue accepts an optional numeric argument which tells it how many levels of enclosing loops it should skip to the end
- Note: continue applies to switch and acts similar to break.

```
$i = 0;
while ($i++ < 5) {
    echo "Outer<br>";
    while (1) {
        echo "  Middle<br>";
        while (1) {
            echo "  Inner<br>";
            continue 3;
        }
        echo "This never gets output.<br>";
    }
    echo "Neither does this.<br>";
}
```


PHP Statements
Alternative syntax

- PHP offers an alternative syntax for some of its control structures; namely, if, while, for, foreach, and switch.
- The basic form of the alternate syntax is to change the opening brace to a colon (:) and the closing brace to, respectively:
  - endif;
  - endwhile;
  - endfor;
  - endforeach;
  - or endswitch;.

PHP Statements
Alternative syntax

Example:

```
<?php
$a=6;
if ($a == 5):
   echo "a equals 5";
   echo "...";
elseif ($a == 6):
   echo "a equals 6";
   echo "!!!";
else:
   echo "a is neither 5 nor 6";
endif;
?>
```


PHP Statements
Include and Require

- The include() and require() statements include and evaluate the specified file.
- Files are included based on the file path given or, if none is given, the include_path specified.
- If the file isn't found in the include_path, include() and require() will finally check in the calling script's own directory and the current working directory before failing.
- The include() construct will emit a warning if it cannot find a file; this is different behavior from require(), which will emit a fatal error.

PHP Statements
Include and Require

vars.php

```
<?php
$color = 'green';
$fruit = 'apple';
?>
```

test.php

```
<?php
echo "A $color $fruit";     // A
include 'vars.php';
echo "A $color $fruit";     // A green apple
?>
```


PHP Statements
Include and Require

- It is possible to execute a return() statement inside an included file in order to terminate processing in that file and return to the script which called it. Also, it's possible to return values from included files.

return.php

```
<?php
$var = 'PHP';
return $var;
?>
```

noreturn.php

```
<?php
$var = 'PHP';
?>
```

PHP Statements Include and Require

```
testreturns.php
<?php
$foo = include 'return.php';
echo "$foo<br>";                    // prints 'PHP'
$bar = include 'noreturn.php';
echo $bar;                    // prints 1 (the include was successful)
?>
```

NOTE: The include_once() and require_once() statements include and evaluate the specified file during the execution of the script. This is a behavior similar to the include() and require() statements, with the only difference being that if the code from a file has already been included, it will not be included again.

PHP Functions

- Function definition

```
<?php
function foo($arg_1, $arg_2, /* ..., */ $arg_n)
{
  //....
  echo "Example function.\n";
  return $retval;
}
?>
```

- Functions need not be defined before they are referenced, except when a function is conditionally defined

PHP Functions

- Conditionally defined function

```
<?php
$makefoo = true;
/* We can't call foo() from here   since it doesn't exist yet,
  but we can call bar() */
bar();
if ($makefoo) {
  function foo()
  {    echo "I don't exist until program execution reaches me.\n";
  }
}
/* Now we can safely call foo()  since $makefoo evaluated to true */
if ($makefoo) foo();
function bar()
{  echo "I exist immediately upon program start.\n";}
?>
```


PHP Functions

- Functions within functions

```
<?php
function foo()
{  function bar()
  {    echo "I don't exist until foo() is called.\n";  }
}

/* We can't call bar() yet   since it doesn't exist. */

foo();

/* Now we can call bar(),   foo()'s processesing has
   made it accessible. */
bar();

?>
```

PHP Functions

- All functions and classes in PHP have the global scope - they can be called outside a function even if they were defined inside and vice versa.
- PHP does not support function overloading, nor is it possible to undefine or redefine previously-declared functions.
- Function names are case-insensitive

PHP Functions
Arguments

- PHP supports passing arguments by value (the default), passing by reference, and default argument values. Variable-length argument lists are also supported
- Passing arrays (by value) to functions

```
<?php
 $arr=array(2,3);
 takes_array($arr);          \\ 2 + 3 = 5
 takes_array($arr);          \\ 2 + 3 = 5
function takes_array($input)
{
  echo "$input[0] + $input[1] = ", $input[0]+$input[1], "<br>";
  $input[0]=10;
}
?>
```


PHP Functions
Arguments

- Passing arrays (by reference) to functions

```
<?php
 $arr=array(2,3);
 takes_array($arr);          \\ 2 + 3 = 5
 takes_array($arr);          \\ 10 + 3 = 13
function takes_array(&$input)
{
  echo "$input[0] + $input[1] = ", $input[0]+$input[1], "<br>";
  $input[0]=10;
}
?>
```


PHP Functions
Arguments

- Passing strings (by value) to functions

```
<?php
 function add_some_extra($string)
 {  $string .= 'and something extra.';}
 $str = 'This is a string, ';
 add_some_extra($str);
 echo $str;   // outputs 'This is a string, '
?>
```

- Passing strings (by reference) to functions

```
<?php
 function add_some_extra(&$string)
 //..
 echo $str;     // outputs 'This is a string, and something extra.'
?>
```

PHP Functions Arguments

- Default arguments

```
<?php
function makecoffee($type = "cappuccino")
{
   return "Making a cup of $type. <br>";
}
echo makecoffee();            \\  Making a cup of cappuccino.
echo makecoffee(null);        \\  Making a cup of .
echo makecoffee("espresso");  \\  Making a cup of espresso.
?>
```

The default value must be a constant expression, not (for example) a variable, a class member or a function call.

PHP Functions Arguments

- Note that when using default arguments, any defaults should be on the right side of any non-default arguments; otherwise, things will not work as expected.

```
<?php
function makeyogurt($type = "acidophilus", $flavour)
{
   return "Making a bowl of $type $flavour.\n";
}
echo makeyogurt("raspberry");   // won't work as expected
?>
```

Warning: Missing argument 2 for makeyogurt(), called in c:\tia\www\php34.php on line 16 and defined in **c:\tia\www\php34.php** on line **11**

Making a bowl of raspberry .

PHP Functions Arguments

- PHP 4 and above has support for variable-length argument lists in user-defined functions.
- This is really quite easy, using the func_num_args(), func_get_arg(int), and func_get_args() functions.

```
<?php
function foo()
{   $numargs = func_num_args();
   echo "Number of arguments: $numargs<br>";
   $arg_list = func_get_args();
   for ($i = 0; $i < $numargs; $i++) {
      echo "Argument $i is: " . $arg_list[$i] . "<br>";  }
}
foo(1, 2, 3);
?>
```

Number of arguments: 3
Argument 0 is: 1
Argument 1 is: 2
Argument 2 is: 3

PHP Functions Returning values

- Values are returned by using the optional return statement.
- Any type may be returned, including arrays and objects.

```
<?php
function small_numbers()
{   return array (0, 1, 2);
}
$arr = small_numbers();
print_r($arr);
?>
```

Array ([0] => 0 [1] => 1 [2] => 2)

PHP Functions
Variable Functions

- If a variable name has parentheses appended to it, PHP will look for a function with the same name as whatever the variable evaluates to, and will attempt to execute it.

```
<?php
function foo() {    echo "In foo()<br>";}
function bar($arg = '')
{    echo "In bar(); argument was '$arg'.<br>";}
// This is a wrapper function around echo
function echoit($string) {    echo $string;}
$func = 'foo';
$func();                    // This calls foo()
$func = 'bar';
$func('test');              // This calls bar()
$func = 'echoit';
$func('test');              // This calls echoit()
?>
```


PHP Functions
Built-in Functions

- In PHP, there are more than 700 built-in functions.
- Functions are organized in categories in the function reference.
- For instance:
  - Affecting PHP's Behaviour
  - Audio Formats Manipulation
  - Authentication Services
  - Date and Time Related Extensions
  - Compression and Archive Extensions

PHP Functions
Date/Time Functions

- string date (string $format [, int $timestamp])

  Returns a string formatted according to the given format string using the given integer timestamp (the number of seconds since January 1 1970 00:00:00 UTC) or the current time if no timestamp is given.

Format character	Description	Example returned values
d	Day of the month, 2 digits with leading zeros	01 to 31
D	A textual representation of a day, three letters	Mon through Sun
j	Day of the month without leading zeros	1 to 31
l (lowercase 'L')	A full textual representation of the day of the week	Sunday through Saturday
…	…	…

PHP Functions
Date

The timestamp can be fixed by using function

- int strtotime (string $time [, int $now])

  Parse about any English textual datetime description into a Unix timestamp

```
<?php
echo date("F j, Y, g:i a") . "<br>";        // January 7, 2011, 1:02 pm
echo date("m.d.y")  . "<br>";               // 01.07.11
echo date("j, n, Y") . "<br>";              // 7, 1, 2011
echo date("Ymd") . "<br>";                  // 20110107
echo date("F j, Y, g:i a", strtotime("10 September 2000")) . "<br>";
    // September 10, 2000, 12:00 am
echo date("m.d.y")  . "<br>";               // 01.07.11
echo date("F j, Y, g:i a", strtotime("+1 day")) . "<br>";
   // January 8, 2011, 1:02 pm
?>
```

PHP Functions
Date/Time Functions

- array getdate ([int $timestamp = time()])
  Returns an associative array containing the date information of the timestamp, or the current local time if no timestamp is given.

  The keys to access to the values are: seconds, minutes, hours, mday, wday, mon, year,yday, weekday (full text representation of the day of the week), month (full textual representation of a month).

- int time (void)
  Returns the current time measured in the number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT).

PHP Functions
FileSystem Functions

- resource fopen (string $filename , string $mode [, bool $use_include_path = false [, resource $context]])
  fopen() binds a named resource, specified by filename, to a stream.
  If *filename* is of the form "scheme://...", it is assumed to be a URL and PHP will search for a protocol handler (also known as a wrapper) for that scheme.
  If PHP has decided that filename specifies a local file, then it will try to open a stream on that file.
  If PHP has decided that filename specifies a registered protocol, and that protocol is registered as a network URL, PHP will check to make sure that allow_url_fopen is enabled.
  Modes: ‘r’, ‘r+’, ‘w’, ‘w+’, ‘a’, ‘a+’.
  ‘+’ indicates reading and writing

PHP Functions
FileSystem Functions

- bool copy (string $source , string $dest)
  Makes a copy of the file source to dest.
- array file (string $filename)
  Reads an entire file into an array. Each element of the array corresponds to a line in the file, with the newline still attached.
- bool flock (resource $handle , int $operation)
  allows you to perform a simple reader/writer model which can be used on virtually every platform(all accessing programs have to use the same way of locking)
  handle - A file system pointer resource that is typically created using fopen().
  operation is one of the following:
  * LOCK_SH to acquire a shared lock (reader).
  * LOCK_EX to acquire an exclusive lock (writer).
  * LOCK_UN to release a lock (shared or exclusive).

PHP Functions
FileSystem Functions

- int fwrite (resource $handle , string $string [, int $length])
  writes the contents of string to the file stream pointed to by handle.
- string fread (resource $handle , int $length)
  reads up to length bytes from the file pointer referenced by handle. Reading stops as soon as one of the following conditions is met:
  - length bytes have been read
  - EOF (end of file) is reached
  - a packet becomes available or the socket timeout occurs (for network streams)
  - 8192 bytes have been read (after opening userspace stream)

PHP Functions
FileSystem Functions

```
<?php
  $fp = fopen("lock.txt", "r+");
  if (flock($fp, LOCK_EX)) {                 // do an exclusive lock
    ftruncate($fp, 0);                       // truncate file
    fwrite($fp, "Write something here\n");
    flock($fp, LOCK_UN);                     // release the lock
  }
  else {    echo "Couldn't get the lock!";}
  fclose($fp);
  $fp = fopen("lock.txt", "r");
  echo fread($fp, filesize('lock.txt'));     // Write something here
  fclose($fp);
?>
```

Write something here

PHP Functions
Regular Expressions

- A regular expression is a pattern that is matched against a subject string from left to right. Most characters stand for themselves in a pattern, and match the corresponding characters in the subject.
- The expression must be enclosed in the delimiters, a forward slash (/), for example. Delimiters can be any non-alphanumeric, non-whitespace ASCII character except the backslash (\) and the null byte.
- The ending delimiter may be followed by various modifiers that affect the matching.

PHP Functions
Regular Expressions

Some examples of modifiers

- i - letters in the pattern match both upper and lower case letters
- A - If this modifier is set, the pattern is forced to be "anchored", that is, it is constrained to match only at the start of the string which is being searched (the "subject string").
- s - If this modifier is set, a dot metacharacter in the pattern matches all characters, including newlines. Without it, newlines are excluded.

PHP Functions
Regular Expressions

Some functions

- int preg_match (string $pattern , string $subject [, array &$matches)

  Searches $subject for a match to the regular expression given in $pattern.
  If matches is provided, then it is filled with the results of search.
- int preg_match_all (string $pattern , string $subject [, array &$matches)

  Searches subject for all matches to the regular expression given in pattern
- array preg_split (string $pattern , string $subject)

  Split the given string by a regular expression. The function returns an array containing substrings of subject split along boundaries matched by pattern

PHP Functions Regular Expressions

```
<?php
/* The \b in the pattern indicates a word boundary, so only the
  distinct word "the" is matched */
  $str="PHP is the scripting language of the web.";
  if (preg_match("/\bthe\b/i", $str,$output)) {
   echo "A match was found.<br>"; print_r($output);}
  else {    echo "A match was not found.";}
  if (preg_match_all("/\bthe\b/i", $str,$output)) {
   echo "<br>A match was found.<br>"; print_r($output);}
  else {   echo "A match was not found.";}
  echo "<br>";
  $result=preg_split("/[\s,]+/",  $str);
  if ($result) print_r($result);
?>
```


PHP Functions Regular Expressions

Output

```
A match was found.
Array ( [0] => the )
A match was found.
Array ( [0] => Array ( [0] => the [1] => the ) )
Array ( [0] => PHP [1] => is [2] => the [3] => scripting [4] =>
  language [5] => of [6] => the [7] => web. )
```


PHP Functions Other Functions

- mixed print_r (mixed $expression)

  print_r — Prints human-readable information about a variable

```
<pre>
<?php
$a = array ('a' => 'apple', 'b' => 'banana', 'c' => array ('x', 'y', 'z'));
print_r ($a);
?>
</pre>
```


PHP Functions Other Functions

Output

```
<pre>
Array
(
    [a] => apple
    [b] => banana
    [c] => Array
        (
            [0] => x
            [1] => y
            [2] => z
        )
)
</pre>
```

PHP Classes

- Syntax equal to Java
- The visibility of a property or method can be defined by prefixing the declaration with the keywords public, protected or private.
- This declaration may include an initialization, but this initialization must be a constant value (that is, evaluated at compile time)
- Methods declared without any explicit visibility keyword are defined as public.
- PHP 5 allows developers to declare constructor methods for classes.

PHP Classes
Constructors and Destructors

```
class MyDestructableClass {
  private $name="Default: ";
  function __construct() {
      print "In constructor\n";
      $this->name .= "MyDestructableClass";
  }
  function __destruct() {
      print "Destroying " . $this->name . "\n";
  }
}
$obj = new MyDestructableClass;
?>
```

In constructor
Destroying Default: MyDestructableClass

PHP Classes

- NOTE: For backwards compatibility, if PHP 5 cannot find a __construct() function for a given class, it will search for the old-style constructor function, by the name of the class.
- NOTE: “$this” is not a pointer but rather a reference
- NOTE: if we do not desire to receive error alerts when using new, adopt the operator @, that is, @new

PHP Classes

```
<?php
class Stack {
  private $top, $vett;
  function Stack() {
      print "In Stack\n";
      $this->top =-1;
  }
  function push($elem) {
    $this -> vett[++$this->top]=$elem;
  }
```

PHP Classes

```
  function pop() {
   if ($this->top>-1)
     return $this -> vett[$this->top--];
   return null;
    }
}
$mystack = new Stack;
$mystack->push(15);
echo $mystack->pop()."<br>";
echo $mystack->pop()."<br>";
?>
```

In Stack 15

PHP Classes Inheritance

```
class Foo
{
   public function printItem($string)
   {      echo 'Foo: ' . $string . "<br>";   }
   public function printPHP()
   {      echo 'PHP is great.' . "<br>";   }
}

class Bar extends Foo
{
   public function printItem($string)
   {      echo 'Bar: ' . $string . "<br>";   }
}
```


PHP Classes Inheritance

```
$foo = new Foo();
$bar = new Bar();
$foo->printItem('baz'); // Output: 'Foo: baz'
$foo->printPHP();       // Output: 'PHP is great'
$bar->printItem('baz'); // Output: 'Bar: baz'
$bar->printPHP();       // Output: 'PHP is great'
?>
```


PHP Classes Scope Resolution Operator

```
<?php
class MyClass {
   const CONST_VALUE = 'A constant value';
}
echo MyClass::CONST_VALUE;
?>
```

PHP Classes
Scope Resolution Operator

```php
<?php
class BaseClass {
   function __construct() {print "In BaseClass constructor<br>";   }
}
class SubClass extends BaseClass {
   function __construct() {
       parent::__construct();        // explicit call
       print "In SubClass constructor <br>";
   }
}
$obj = new BaseClass();            //In BaseClass constructor
$obj = new SubClass();             //In BaseClass constructor
                                   //In SubClass constructor
?>
```


PHP Classes
Scope Resolution Operator

```php
<?php
class MyClass
{    protected function myFunc() {
        echo "MyClass::myFunc()<br>";    }
}
class OtherClass extends MyClass
{    // Override parent's definition
    public function myFunc()
    {        // But still call the parent function
        parent::myFunc();    echo "OtherClass::myFunc()<br>";    }
}
$class = new OtherClass();
$class->myFunc();         //MyClass::myFunc()
                          //OtherClass::myFunc()
?>
```


PHP Classes
Static Keyword

- Declaring class members or methods as static makes them accessible without needing an instantiation of the class.
- A member declared as static can not be accessed with an instantiated class object (though a method can).

```php
<?php
class Foo
{   public static $my_static = 0;
    public function staticValue() {
        return ++self::$my_static;    }
}

class Bar extends Foo
{    public function fooStatic() {
        return ++parent::$my_static;
    }
}
```


PHP Classes
Static Keyword

```php
print Foo::$my_static . "<br>";            //0

$foo = new Foo();
print $foo->staticValue() . "<br>";        //1
print $foo->my_static . "<br>";  // Undefined "Property" my_static

// $foo::my_static is not possible

print Bar::$my_static . "<br>";            //1
$bar = new Bar();
print $bar->fooStatic() . "<br>";          //2
?>
```

PHP Classes Object Serialization

- serialize() returns a string containing a byte-stream representation of any value that can be stored in PHP.
- unserialize() can use this string to recreate the original variable values. Using serialize to save an object will save all variables in an object.
- PHP checks if your class has a function with the magic name __sleep and will attempt to call it prior to serialization so as to allow the object to do any last minute clean-up, etc. prior to being serialized. Likewise, when the object is restored using unserialize() the __wakeup member function is called.

```php
<?php
class A {
     public $one = 100;
     public function show_one() {  echo $this->one;  }
 }
?>
```


PHP Classes Object Serialization

```php
// page1.php:
  include("classa.inc");
  $a = new A;
  $s = serialize($a);
 // store $s somewhere where page2.php can find it.
 file_put_contents('store', $s);

// page2.php:
// this is needed for the unserialize to work properly.
 include("classa.inc");
 $s = file_get_contents('store');
 $a = unserialize($s);
 // now use the function show_one() of the $a object.
 $a->show_one();
```


Forms and User Input

- Any form element in an HTML page will automatically be available to PHP scripts
- The use of the global associative arrays $_GET , $_POST and $_REQUEST allows retrieving information from forms, like user input.
  - $_GET - An associative array of variables passed to the current script via the URL parameters.
  - $_POST - An associative array of variables passed to the current script via the HTTP POST method.
  - $_REQUEST - An associative array that by default contains the contents of $_GET, $_POST and $_COOKIE.

Forms and User Input

```html
<head>
   <title>Form Example</title>
 </head>
 <body>
   <form action="elab.php" method="post">
     <p>
      Name:<br>
      <input type="text" name="username"><br>
      E-mail:<br>
      <input type="text" name="email"><br>
      <input type="submit" name="submit"
        value="SUBMIT">
     </p>
   </form>
 </body>
```

Forms and User Input

```
// file elab.php
<?php
  print $_POST['username'];
  print $_REQUEST['username'];
  ?>

<form action="elab.php" method="get">

// file elab.php
<?php
  print $_GET['username'];
  print $_REQUEST['username'];
  ?>
```


Forms and User Input

bool import_request_variables (string $types [, string $prefix])

- Imports GET/POST/Cookie variables into the global scope.
  - Types parameter – you can specify which request variables to import. You can use 'G', 'P' and 'C' characters respectively for GET, POST and Cookie. These characters are not case sensitive.
  - $prefix parameter - is prepended before all variable's name imported into the global scope. So if you have a GET value named "userid", and provide a prefix "pref_", then you'll get a global variable named $pref_userid.

Forms and User Input

```
// file elab.php

<?php
  import_request_variables('p', 'p_');
  print $p_username . "<br>";
  print $p_email;
  ?>
```


PHP Forms and User Input

- Form Validation
  - User input should be validated on the browser whenever possible (by client scripts). Browser validation is faster and reduces the server load.
  - You should consider server validation if the user input will be inserted into a database. A good way to validate a form on the server is to post the form to itself, instead of jumping to a different page. The user will then get the error messages on the same page as the form. This makes it easier to discover the error.

Get or Post

- Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send (max. 100 characters).

- Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.
- However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

Cookie

- A cookie is a small piece of data sent from a website and stored in a user's web browser while the user is browsing that website.
  - A web server specifies a cookie to be stored by sending an HTTP header called Set-Cookie.

  Set-Cookie: value[; expires=date][; domain=domain][; path=path][; secure]
  - When a cookie is present, and the optional rules allow, the cookie value is sent to the server with each subsequent request. The cookie value is stored in an HTTP header called Cookie and contains just the cookie value without any of the other options

  Cookie: value

Cookie

bool setcookie (string $name [, string $value [, int $expire = 0 [, string $path [, string $domain [, bool $secure = false [, bool $httponly = false]]]]]])

- setcookie() defines a cookie to be sent along with the rest of the HTTP headers.
  - Like other headers, cookies must be sent before any output from your script (this is a protocol restriction). This requires that you place calls to this function prior to any output, including <html> and <head> tags as well as any whitespace.

Cookie

Expire

- Indicates when the cookie should no longer be sent to the server and therefore may be deleted by the browser.
- Without the expires option, a cookie has a lifespan of a single session. A session is defined as finished when the browser is shut down, so session cookies exist only while the browser remains open.

Cookie

Domain

- Indicates the domain(s) for which the cookie should be sent.
- By default, domain is set to the host name of the page setting the cookie, so the cookie value is sent whenever a request is made to the same host name.
- For example, the default domain for a cookie set might be www.ing.unipi.it.
- The domain option is used to widen the number of domains for which the cookie value will be sent.
- Consider the case of a large network such as Yahoo! that has many sites in the form of name.yahoo.com (e.g., my.yahoo.com, finance.yahoo.com, etc.). A single cookie value can be set for all of these sites by setting the domain option to simply yahoo.com.

Cookie

Path

- Indicates a URL path that must exist in the requested resource before sending the Cookie header. This comparison is done by comparing the option value character-by-character against the start of the request URL. If the characters match, then the Cookie header is sent.

  Ex:

  Set-Cookie: name=Nicholas; path=/blog
- The path option would match /blog, /blogroll, etc.; anything that begins with /blog is valid. Note that this comparison is only done once the domain option has been verified. The default value for the path option is the path of the URL that sent the Set-Cookie header.

Cookie

Secure

- just a flag and has no additional value specified. A secure cookie will only be sent to the server when a request is made using SSL and the HTTPS protocol.
- confidential or sensitive information should never be stored or transmitted in cookies as the entire mechanism is inherently insecure. By default, cookies set over an HTTPS connection are automatically set to be secure.

Cookie

```
<?php
$value = 'something from somewhere';
setcookie("TestCookie", $value, time()+3600, "/~rasmus/",
  ".example.com", 1);
  /* expire in 1 hour, available within the /~rasmus/ directory and
  all sub-directories, available on all subdomains of example.com
  and set if a secure connection exists*/
?>
```

Cookie

To see the contents of the test cookie in a script

```
<?php
// Print an individual cookie
echo $_COOKIE["TestCookie"];
// Another way to debug/test is to view all cookies
print_r($_COOKIE);
?>
```

Output

```
something from somewhere
Array ( [TestCookie] => something from somewhere )
```


Cookie

- Cookies are usually set in an HTTP header (although JavaScript can also set a cookie directly on a browser)

An example of header returned by a PHP program:

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name=xyz; expires=Friday, 04-Feb-07 22:03:38 GMT;
            path=/; domain=tutorialspoint.com
Connection: close
Content-Type: text/html
```


Cookie

- If the browser is configured to store cookies, it will then keep this information until the expiry date. If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server.

An example of header used by a browser:

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name=xyz
```


Cookie

- Example of use of the cookies

```
<html>
  <head>
    <title>Accessing Cookies with PHP</title>
  </head>
  <body>
    <?php
      echo $_COOKIE["name"]. "<br />";
      /* is equivalent to */
      echo $HTTP_COOKIE_VARS["name"]. "<br />";
      echo $_COOKIE["age"] . "<br />";
      /* is equivalent to */
      echo $HTTP_COOKIE_VARS["name"] . "<br />";
    ?>
  </body>
</html>
```

Cookie

- Example of use of the cookies

```
<html>
  <head>
    <title>Accessing Cookies with PHP</title>
  </head>
  <body>
    <?php
     if( isset($_COOKIE["name"]))
        echo "Welcome " . $_COOKIE["name"] . "<br />";
     else
        echo "Sorry... Not recognized" . "<br />";
    ?>
  </body>
</html>
```


Cookie

- Deleting cookie with PHP

```
<?php
  setcookie( "name", "", time()- 60, "/","", 0);
  setcookie( "age", "", time()- 60, "/","", 0);
?>
<html>
  <head>
    <title>Deleting Cookies with PHP</title>
  </head>
  <body>
    <?php echo "Deleted Cookies" ?>
  </body>
</html>
```


Sessions

- Session support allows preserving certain data across subsequent accesses.
  - This enables you to build more customized applications and increase the appeal of your web site.
- A visitor accessing your web site is assigned a unique id, the so-called session id. This is either stored in a cookie on the user side or is propagated in the URL.
- The session support allows you to register arbitrary numbers of variables to be preserved across requests.
  - When a visitor accesses your site, PHP will check automatically (if session.auto_start is set to 1) or on your request (explicitly through session_start() or implicitly through session_register()) whether a specific session id has been sent with the request.
  - If this is the case, the prior saved environment is recreated.

Sessions

```
//login.php
<?php
 session_start();
 $_SESSION['count'] = 0;
?>
 <p>Welcome :)<br>
 <a href="page.php">GO</a>.
 </p>
```

session_start() creates a session or resumes the current one based on a session identifier passed via a GET or POST request, or passed via a cookie

Sessions

```
//page.php
<?php
 session_start();
 if (!isset($_SESSION['count']))
       //Determine if a variable is set and is not NULL.
 {  echo "<p>You have to login</p>";  }
 else {  $_SESSION['count']++;
 echo "<p>Hi, you have visited this page " .
 $_SESSION['count'] . " times.</p>";
 echo "<p>
 <a href=\"page.php\">GO</a>, or
 <a href=\"logout.php\">Exit</a>.</p>";}
?>
```


Sessions

```
//logout.php
<?php
 session_start();  unset($_SESSION['count']);
?>
<p>Good Bye :(</p>
```


Sessions

```
//form.php
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8">
  <title>Login</title>
   <style type="text/css"> p { font-size: 30pt;} </style>
 </head>
 <body>
  <form method="post" action="login.php">
  <p>Username: <input type="text" name="username"></p>
  <p>Password: <input type="text" name="password"></p>
  <p><input type="submit" value="Let me in"></p>
  </form>
  </body>
  </html>
```


Sessions

```
//login.php
<!DOCTYPE html>
 <html lang="en">
 <head>
  <meta charset="utf-8">
  <title>Login</title>
  </head>
  <body>
  <?php
  // Check if username and password are correct
  if ($_POST["username"] == "php" && $_POST["password"] ==
  "php") {
```

Sessions

```
//login.php
// If correct, we set the session to YES
    session_start();
    $_SESSION["Login"] = "YES";
    echo "<h1>You are now logged correctly in</h1>";
    echo "<p><a href='document.php'>Link to protected
  file</a></p>";
  }
```


Sessions

```
//login.php
else {
   // If not correct, we set the session to NO
    session_start();
    $_SESSION["Login"] = "NO";
    echo "<h1>You are NOT logged correctly in </h1>";
    echo "<p><a href='document.php'>Link to protected
  file</a></p>";
  }
  ?>
  </body>
  </html>
```


Sessions

```
//document.php
<?php
  // Start up your PHP Session
  session_start();
  // If the user is not logged in send him/her to the login form
  if ($_SESSION["Login"] != "YES") {
    header("Location: form.php");
  }
  ?>
```

See laboratories 8 and 9 for more details on the use of the Session variables

Sessions

```
//document.php
<!DOCTYPE html>
 <html lang="en">
  <head>
  <meta charset="utf-8">
    <title>Protected File</title>
    <style type="text/css"> p { font-size: 30pt;} </style>
  </head>
  <body>
  <h1>This document is protected</h1>
  <p>You can only see it if you are logged in.</p>
  <a href='logout.php'>Exit</a> </p>
  </body>
  </html>
```

Sessions

```
//logout.php
<!DOCTYPE html>
<?php
  session_start();  unset($_SESSION['Login']);
?>
  <html lang="en">
  <head>
   <meta charset="utf-8">
    <title>Logout</title>
    <style type="text/css"> p { font-size: 30pt;} </style>
  </head>
  <body>
   <p>Good Bye :(</p>
   </body>
   </html>
```


PHP and mysql

- Connecting to MySQL server

mysqli::__construct() ([string $host = ini_get("mysqli.default_host") [, string $username = ini_get("mysqli.default_user") [, string $passwd = ini_get("mysqli.default_pw") [, string $dbname = "" [, int $port = ini_get("mysqli.default_port") [, string $socket = ini_get("mysqli.default_socket")]]]]]])

Opens a connection to the MySQL server.

Returns an object which represents the connection to a MySQL Server.

PHP and mysql

- Connecting to MySQL server

```
<?php
$mysqli= new mysqli($nomeHost, $nomeUtente, $password, $db);
if ($mysqli->connect_error) {
   die('Connect Error (' . $mysqli->connect_errno . ') '
          . $mysqli->connect_error);
}
echo 'Success... ' . $mysqli->host_info . "\n";

$mysqli->close();
?>
```

Success... MySQL host info: localhost via TCP/IP

PHP and mysql

- Selecting a database

bool mysqli::select_db (string $database_name)

Selects the default database to be used when performing queries against the database connection.

Every subsequent call to mysql_query() will be made on the active database.

```
// make pweb the current db
$mysqli->select_db($nomeDb) or
   die ('Can\'t use pweb: ' . mysql_error());
```

PHP and mysql

- Send a query

  mixed mysqli::query (string $query)

  Performs a query against the database.
  - For SELECT, SHOW, DESCRIBE, EXPLAIN and other statements returning resultset, returns a mysqli_result object on success, or FALSE on error.
  - For other type of SQL statements, INSERT, UPDATE, DELETE, DROP, etc, returns TRUE on success or FALSE on error.
  - The returned result resource should be passed to mysqli_fetch_array(), and other functions for dealing with result tables, to access the returned data.

PHP and mysql

```
<?php
 $nomeHost = "localhost";
 $nomeUtente = "root";
 $password = "";
 $nomeDb = "pweb";
 $mysqli= new mysqli($nomeHost, $nomeUtente, $password,
 $nomeDb);
 if ($mysqli->connect_error) {
   die('Connect Error (' . $mysqli->connect_errno . ') '
          . $mysqli->connect_error);
 }
 echo 'Success... ' . $mysqli->host_info . "\n";
```


PHP and mysql

```
// Formulate Query
// This is the best way to perform an SQL query
$query = "SELECT * FROM studenti";

// Perform Query
$result = $mysqli->query($query);

/* Check result
   This shows the actual query sent to MySQL, and the error.
   Useful for debugging.*/
checkResult($result,$query);

//show results
showResult($result);
```


PHP and mysql

```
 // Free the resources associated with the result set
 // This is done automatically at the end of the script
 $result->close();
 $mysqli->close();

function checkResult($result, $query)
{ if (!$result) {
   $message  = 'Invalid query: ' . $mysqli->error . "\n";
   $message .= 'Whole query: ' . $query;
   die($message);
   }
}
```

PHP and mysql

```
function showResult($result)
{       while ($row = $result->fetch_assoc()) {
        echo $row['MATRICOLA']. " ";
        echo $row['NOME'] . "<br>";
        }
        echo "<br>";
}
?>
```

1111 ROMOLO
2222 REMO
3333 CESARE
4444 PIPPO

mysqli_result::fetch_assoc() - Returns an associative array that corresponds to the fetched row and moves the internal data pointer ahead.

PHP and mysql

```
$name  = 'ROMOLO';
$query = sprintf("SELECT * FROM studenti WHERE NOME =%s",
filterSQLQuery($name));
$result = $mysqli->query($query);
checkResult($result,$query);
showResult($result);

$mat  = 3333;
$query = sprintf("SELECT * FROM studenti WHERE MATRICOLA
=%s",  filterSQLQuery($mat));
$result = $mysqli->query($query);
checkResult($result,$query);
showResult($result);
```


PHP and mysql

```
function filterSQLQuery($string) {
  if (get_magic_quotes_gpc())
 /*When magic_quotes are on for GPC (Get/Post/Cookie) operations,
all ', ", \ and NUL's are escaped with a backslash automatically –
This feature has been removed in PHP 5.4.0 and the function returns
always false*/
        $string = stripslashes($string);
        // removes slashes (\)
if (!is_numeric($string))
        $string = "'" . mysqli_real_escape_string($string) . "'";
/*Escapes special characters in a string for use in an SQL statement
in such a way that the SQL statements are sure ;*/
return $string }
```

1111 ROMOLO

3333 CESARE

SQL Injection Attack

- Exploiting SQL code is possible to obtain private information or modify a database

Esempio

Let us assume to have a login form which asks for username and passwd

These data are used to perform sql query:

```
$miaQuery="SELECT COUNT(*) FROM utenti
  WHERE username=' " .$_POST['username']. " '
  AND password=' " .$_POST['password'].  " ' ";
```

username password

SQL Injection Attack

username GIANNI password ` OR ''='

Input the following data:
SELECT COUNT(*)
WHERE username='GIANNI'
AND password= ' ' OR ' ' = ' ';

The query results to be:
SELECT COUNT(*) FROM utenti
WHERE username='GIANNI'
AND password= ' ' OR ' ' = ' ';

Same effect as knowing the GIANNI's password. By using mysqli_real_escape_string, each quote is escaped.

SQL Injection Attack

Other examples:

username ` OR 1=1-- password

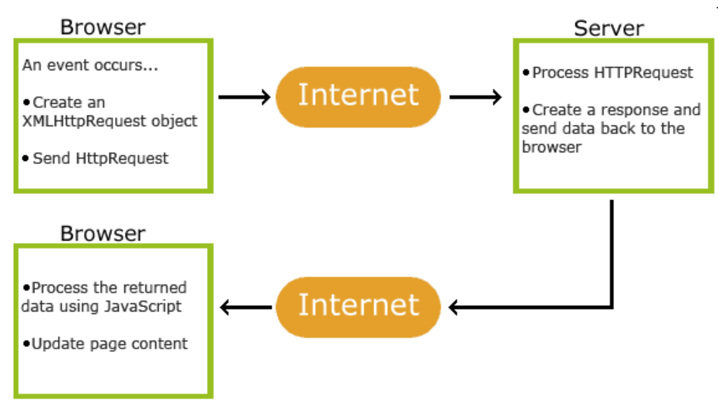
SELECT COUNT(*)
WHERE username= ' ' OR 1=1 --'
AND password= ' `;

The text (in italic) after -- is not considered since it indicates an SQL comment.
The query will return the number of rows in the table.

AJAX

- Asynchronous JavaScript and XML (AJAX) is a group of interrelated web development techniques used on the client-side to create interactive web applications.
- With Ajax, web applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page.
- Instead of using normal HTTP POST or GET requests (such as when using forms or hyperlinks), data is usually retrieved using the XMLHttpRequest object

AJAX



AJAX

- XMLHttpRequest object
  - used to send HTTP or HTTPS requests directly to a web server and load the server response data directly back into the script.
  - The data might be received from the server as XML text or as plain text.
  - Data from the response can be used directly to alter the DOM of the currently active document in the browser window without loading a new web page document.

AJAX

Three steps

- Create an XMLHttpRequest object

```
try { xmlHttp=new XMLHttpRequest(); }
catch (e) {
   try { xmlHttp=new ActiveXObject("Msxml2.XMLHTTP"); }
  //IE (recent versions)
  catch (e) {
  try { xmlHttp=new ActiveXObject("Microsoft.XMLHTTP"); }
  //IE (older versions)
    catch (e) {
      window.alert("Your browser does not support AJAX!");
  return false;
              }
      }
  }
```


AJAX

- Write an event handler for sending requests for data to the server

```
xmlHttp.open("GET","data.php",true);
xmlHttp.onreadystatechange = useHttpResponse;
xmlHttp.send(null);
```

open(method,url,async)

Specifies the type of request, the URL, and if the request should be handled asynchronously or not.

method: the type of request: GET or POST

url: the location of the file on the server

async: true (asynchronous) or false (synchronous)

onreadystatechange

Stores a function (or the name of a function) to be called automatically each time the readyState property changes

AJAX

send(string)

Sends the request off to the server.

string: Only used for POST requests

Example:

```
xmlhttp.open("POST","ajax_test.php",true);
xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");
xmlhttp.send("fname=Henry&lname=Ford");
```

setRequestHeader(header,value)

Adds HTTP headers to the request.

header: specifies the header name

value: specifies the header value

AJAX

- Write a function useHttpResponse which will establish when the server has completed the request and will do something useful with the returned data

```
function useHttpResponse() {
 if (xmlHttp.readyState == 4)
   document.mymodule.day.value = xmlHttp.responseText;
}
```

readyState

Holds the status of the XMLHttpRequest. Changes from 0 to 4:

0: request not initialized

1: server connection established

2: request received

3: processing request

4: request finished and response is ready

AJAX

responseText

get the response data as a string

responseXML

get the response data as XML data

AJAX

- Example (submit.html)

```
<html lang="en">
 <head>
 <meta charset="utf-8">
<title>Example</title>
</head>
<body>
<script type="text/javascript" src="./myajax.js"></script>
<form action="#" name="mymodule" id="mymodule">
<p>Name: <input type="text" onkeyup="ajaxHandler();"
  name="name">
  Date: <input type="text" name="day"></p>
</form>
</body>
</html>
```

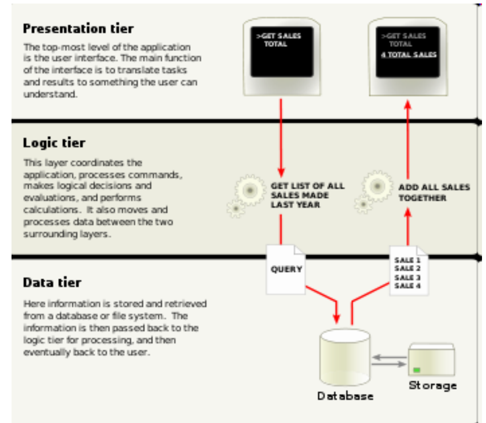

AJAX

```
\\date.php
<?php   $oggi = date("D M Y G:i:s");   echo($oggi); ?>

\\myajax.js
function ajaxHandler() {
    var xmlHttp;
    try { xmlHttp=new XMLHttpRequest(); }
    catch (e)
   {      try { xmlHttp=new ActiveXObject("Msxml2.XMLHTTP"); }
          catch (e)
          {  try { xmlHttp=new ActiveXObject("Microsoft.XMLHTTP"); }
               catch (e)
              {       window.alert("Your browser does not support AJAX!");
                      return;
               }
           }
      }
```

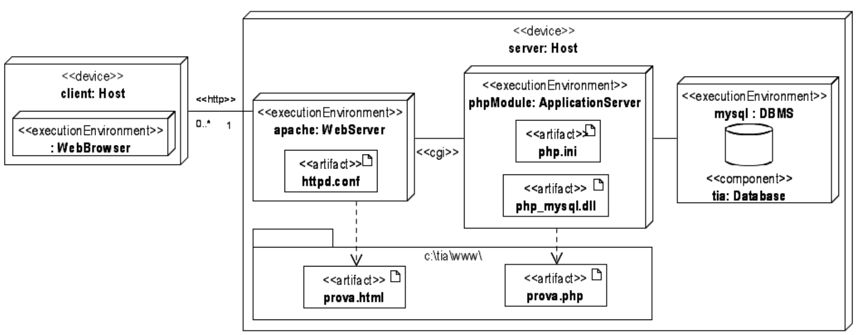
AJAX

```
xmlHttp.open("GET","date.php",true);
xmlHttp.onreadystatechange = useHttpResponse;
xmlHttp.send(null);

function useHttpResponse()
{     if(xmlHttp.readyState == 4)
      document.mymodule.day.value = xmlHttp.responseText;
  }
}
```


Three-tier architecture



Three-tier architecture UML - Deployment Diagram



Three-tier architecture

- The Common Gateway Interface (CGI) is a standard (see RFC 3875: CGI Version 1.1) that defines how web server software can delegate the generation of web pages to a text-based application.
- CGI is a standard way for a Web server to pass a Web user's request to an application program and to receive data back to forward to the user.
- httpd.conf – configuration file for determining for instance the root
- php.ini – configuration file for selecting the DBMS and the libraries

Three-tier architecture

- In the web development field, three-tier is often used to refer to websites, commonly electronic commerce websites, which are built using three tiers:
  - A front-end web server serving static content, and potentially some cached dynamic content. In web based application, Front End is the content rendered by the browser. The content may be static or generated dynamically.
  - A middle dynamic content processing and generation level application server, for example Java EE, ASP.NET, PHP platform.
  - A back-end database, comprising both data sets and the database management system or RDBMS software that manages and provides access to the data.

Designing Web Applications

HTTP

Francesco Marcelloni

Department of Information Engineering
University of Pisa
ITALY

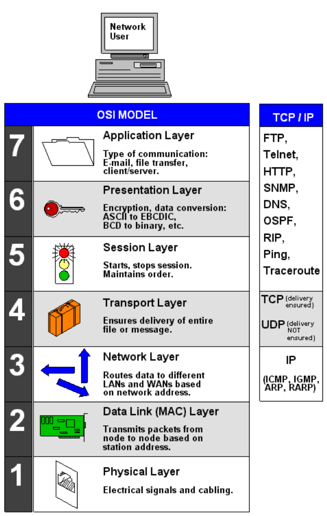
Acknowledgement

I acknowledge that some slides have been adapted from (or are just identical to) the slides provided with the following book:

Thee Computer Networking: A Top Down Approach , 5th edition. Jim Kurose, Keith Ross Addison-Wesley, April 2009.

TCP/IP Stack

From Computer Desktop Encyclopedia
© 2000 The Computer Language Co. Inc.



HTTP Overview

First some jargon

- Web page consists of objects
- Objects can be HTML files, JPEG images, Java applets, audio files,…
- Web page consists of base HTML-file which includes several referenced objects
- Each object is addressable by a URI
- Example URI:

```
www.someschool.edu/someDept/pic.gif
```

$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$

HTTP Overview

http://guest:secret@www.ietf.org:80/html.charters/wg-dir.html?sess=1#Applications_Area

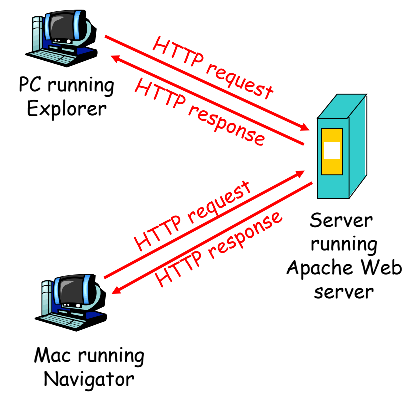
- protocol --- http
- username ------- guest
- password ----------- secret
- host ----------------- www.ietf.org
- port ---------------------------- 80
- path ------------------------------ /html.charters
- file -- wg-dir.html
- query --- sess=1
- fragment --- Applications_Area

Università di Pisa Francesco Marcelloni

HTTP Overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, "displays" Web objects
  - *server:* Web server sends objects in response to requests



Università di Pisa Francesco Marcelloni

HTTP Overview

Uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"

- server maintains no information about past client requests

aside

Protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

Università di Pisa Francesco Marcelloni

HTTP Connections

Nonpersistent HTTP

- At most one object is sent over a TCP connection.

Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server.

Nonpersistent HTTP

Suppose user enters URL (contains text, references to 10 jpeg images)

`www.someSchool.edu/someDepartment/home.index`

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

Nonpersistent HTTP (continued)

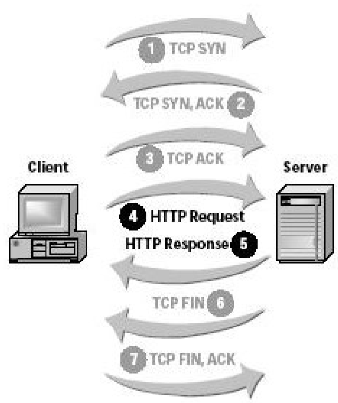
4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

time

Nonpersistent HTTP (continued)



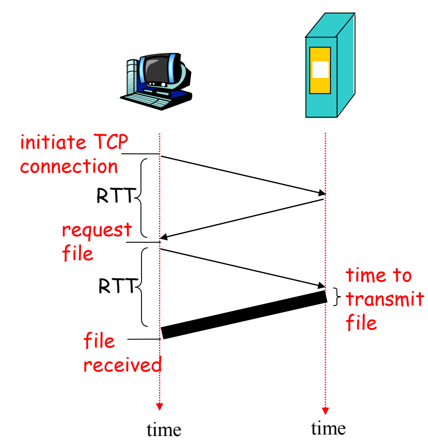
Nonpersistent HTTP: Response Time

Definition of Round-Trip Time (RTT): time for a small packet to travel from client to server and back.

Response time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time

total = 2RTT+transmit time



Persistent HTTP

Nonpersistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for *each* TCP connection (for each connection, TCP buffers and TCP variables have to be allocated in the client and in the server)
- browsers often open parallel TCP connections to fetch referenced objects

Persistent HTTP

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- Pipelining version:
  - client sends requests as soon as it encounters a referenced object
  - as little as one RTT for all the referenced objects

Università di Pisa Francesco Marcelloni

Persistent HTTP

Two versions of persistent connections

- Without pipelining
  - The client issues a new request only when the previous response has been received
  - The client experiences one RTT in order to request and receive each of the referenced objects
  - After the server sends an object over the persistent TCP connection, the connection idles while it waits for another request

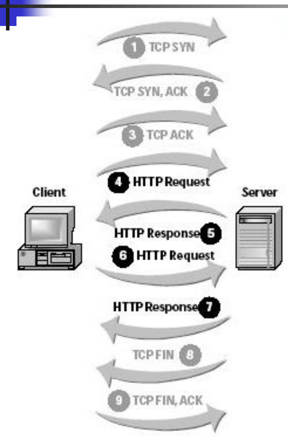
Università di Pisa Francesco Marcelloni

Persistent HTTP

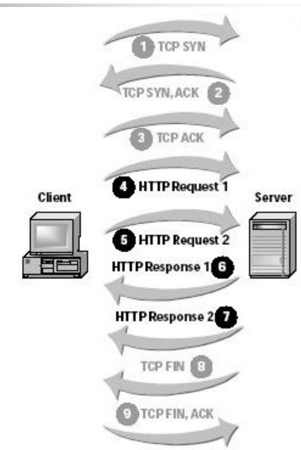
- With pipelining (default mode in HTTP/1.1)
  - The client issues a new request as soon as it encounters a reference.
  - It is possible for only one RTT to be expended for all the referenced objects
  - The pipelined TCP connection remains idle for a smaller fraction of time

Università di Pisa Francesco Marcelloni

Persistent HTTP



Without Pipelining



With Pipelining

HTTP Request Message

- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)

request line

(HTTP commands)

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

header lines

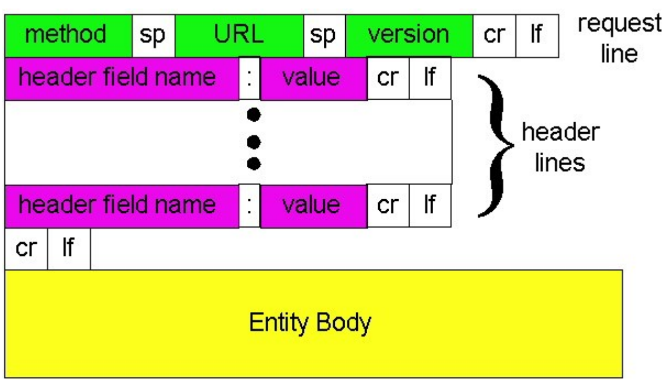
Carriage return, line feed indicates end of message

(extra carriage return, line feed)

HTTP Request Message

- Connection: close
  - The browser is telling the server that it does not want to use persistent connections (when it wants, it uses Keep-Alive)
- User-agent
  - Specifies the user agent, that is, the browser that is making the request to the server
  - Useful: the server could send different versions of the same object to different types of user agent
- Accept-language
  - The user prefers to receive a French version of the object, if such an object exists on the server; otherwise the server should send its default version

HTTP Request Message General Format



HTTP Request Message

- Web page often include form input

Post method:

- Input is uploaded to server in entity body

GET method:

- Input is uploaded in URL field of request line:

`www.somesite.com/animalsearch?monkeys&banana`

HTTP Request Message

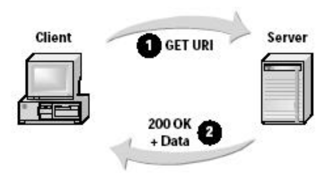
HTTP/1.0
- GET
- POST
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1
- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field
- TRACE
  - invoke a remote, application-layer loop- back of the request message

Università di Pisa Francesco Marcelloni

HTTP Request Message

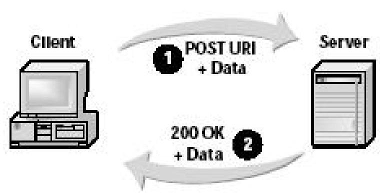
- GET
  - Input in the URL field. No entity body
  - Sure and idempotent (the side-effects of N > 0 identical requests is the same as for a single request)
  - Only ASCII characters
  - Browser and proxy can cache the responses of the server



Università di Pisa Francesco Marcelloni

HTTP Request Message

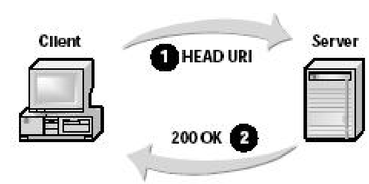
- POST
  - The URI specifies the script which should process the information
  - The entity body contains what the user entered into the form fields
  - The script is called at each form submission (no cache)



Università di Pisa Francesco Marcelloni

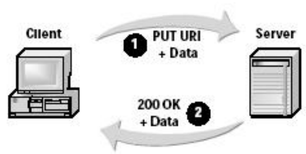
HTTP Request Message

- HEAD
  - Similar to the GET method
  - When a server receives a request with the HEAD method, it responds with an HTTP message but it leaves out the requested object
  - Often used for debugging



HTTP Request Message

- PUT
  - Used in conjunction with Web publishing tools
  - Allows a user to upload an object to a specific path (directory) on a specific Web server (typically the object is a file)
  - If the Request-URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server.
  - Used by applications which need to upload objects to Web servers

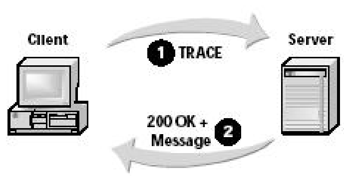


HTTP Request Message

- DELETE
  - Allows a user, or an application, to delete an object on a Web server

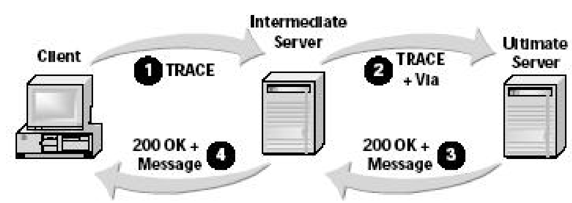
HTTP Request Message

- TRACE
  - The TRACE method is used to invoke a remote, application-layer loop- back of the request message.
  - The final recipient (origin server or proxy) of the request SHOULD reflect the message received back to the client as the entity-body of a 200 (OK) response.
  - TRACE allows the client to see what is being received at the other end of the request chain and use that data for testing or diagnostic information.



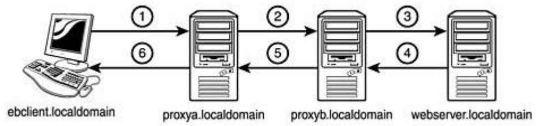
HTTP Request Message

- TRACE (example)



HTTP Request Message

- TRACE (example)



1

TRACE / HTTP/1.1
Host: webserver.localdomain

HTTP Request Message

- TRACE (example)

2

TRACE / HTTP/1.1
Host: webserver.localdomain
Via: 1.1 proxya.localdomain

3

TRACE / HTTP/1.1
Host: webserver.localdomain
Via: 1.1 proxya.localdomain, 1.1proxyb.localdomain

HTTP Request Message

- TRACE (example)

4

HTTP/1.1 200 OK
Date: Tue, 21 May 2002 12:34:56 GMT
Server: Apache/1.3.22 (Unix)
Content-Type: message/http

TRACE / HTTP/1.1
Host: webserver.localdomain
Via: 1.1 proxya.localdomain, 1.1proxyb.localdomain

HTTP Request Message

- TRACE (example)

5

HTTP/1.1 200 OK
Date: Tue, 21 May 2002 12:34:56 GMT
Server: Apache/1.3.22 (Unix)
Content-Type: message/http
Via:1.1 proxyb.localdomain

TRACE / HTTP/1.1
Host: webserver.localdomain
Via: 1.1 proxya.localdomain, 1.1proxyb.localdomain

HTTP Request Message

- TRACE (example)

6

```
HTTP/1.1 200 OK
Date: Tue, 21 May 2002 12:34:56 GMT
Server: Apache/1.3.22 (Unix)
Content-Type: message/http
Via:1.1 proxyb.localdomain, 1.1 proxya.localdomain

TRACE / HTTP/1.1
Host: webserver.localdomain
Via: 1.1 proxya.localdomain, 1.1proxyb.localdomain
```


HTTP Response Message

status line (protocol, status code, status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```


HTTP Response Status Codes

In first line in server->client response message.

A few sample codes:

200 OK
- request succeeded, requested object later in this message

301 Moved Permanently
- requested object moved, new location specified later in this message (Location:)

400 Bad Request
- request message not understood by server

404 Not Found
- requested document not found on this server

505 HTTP Version Not Supported

HTTP Response Message Header Lines

- Date: indicates the time and the date when the HTTP response was created and sent by the server
- Content-Type: indicates that the object in the entity body is HTML text (The object type is officially indicated by the Content-Type: header and not by the file extension)

Trying HTTP

1. Telnet to your favorite Web server:

```
telnet localhost 80
```

Opens TCP connection to port 80 (default HTTP server port) at localhost. Anything typed in sent to port 80 at localhost

2. Type in a GET HTTP request:

```
GET /form.html HTTP/1.1
Host: localhost
```

By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. Look at response message sent by HTTP server!

Trying HTTP (get.php)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="en">
 <head>
  <meta http-equiv="Content-type" content="text/html;
  charset=ISO-8859-1">
   <title>send data using the GET method without forms and
  opening a new page</title>
         </head>
         <body>
```


Trying HTTP

```
/* Redirect to a different page in the current directory that was
  requested */
<?php
        $host  = $_SERVER['HTTP_HOST'];
        $uri   = rtrim(dirname($_SERVER['PHP_SELF']), '/\\');
        //rtrim — Strip whitespace (or other characters) from the
  end of a string
        $extra = 'receiveGET.php?name=Bob&age=21';
        header("Location: http://$host$uri/$extra");
        //Send a raw HTTP header
        exit;
        ?>
        </body>
</html>
```


Trying HTTP (receiveGet.php)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html;
 charset=iso-8859-1">
  <title>Receive data and forward to the client</title>
 </head>
 <body>
  <p> Hi <?php echo htmlspecialchars($_GET['name']); ?>,<br>
    You are <?php echo (int)$_GET['age']; ?> old.
  </p>
 </body>
</html>
```

htmlspecialchars — Convert special characters to HTML entities

Trying HTTP (post.php)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="en">
 <head>
 <meta http-equiv="Content-type" content="text/html;
 charset=ISO-8859-1">
  <title>send data using the POST method without forms and
 opening a new page</title>
  </head>
        <body>
        <p>
```


Trying HTTP (post.php)

```
<?php
 $data =
 sendPost('/receivePOST.php','name=Bob&age=21','localhost');
 echo $data;

 function sendPost($uri,$postdata,$host){
        $flow = fsockopen($host, 80, $errno, $errstr);
        //Initiates a socket connection to the resource specified by
 hostname.
        if (!$flow) {
                echo "$errstr ($errno)<br>\n";
                echo $flow;      }
        else {
```


Trying HTTP (post.php)

```
$requestHTTP ="POST $uri  HTTP/1.1\r\n";
$requestHTTP.="Host: $host\r\n";
$requestHTTP.="User-Agent: PHP Script\r\n";
$requestHTTP.= "Content-Type: application/x-www-form-
  urlencoded\r\n";
$requestHTTP.="Content-Length: ".strlen($postdata)."\r\n";
$requestHTTP.="Connection: close\r\n\r\n";
$requestHTTP.=$postdata;
$replyHTTP = NULL;
fwrite($flow, $requestHTTP);
while (!feof($flow))
   $replyHTTP.=fgets($flow, 128);
$replyHTTP = explode("\r\n\r\n",$replyHTTP);
$headerReplyHTTP = $replyHTTP[0];
$contentsReplyHTTP    = $replyHTTP[1];
```


Trying HTTP (post.php)

```
if(!(strpos($headerReplyHTTP,"Transfer-Encoding:
  chunked")===false))
{  $temp=explode("\r\n",$contentsReplyHTTP);
  for($i=0;$i<count($temp);$i++)
  if($i==0 || ($i%2==0))
     $temp[$i]="";
  $contentsReplyHTTP=implode("",$temp);
  //implode — Join array elements with a string
   }
   fclose($flow);
return rtrim($contentsReplyHTTP);
   }
 } ?>
</p>
</body>
</html>
```

Trying HTTP (receivePost.php)

Example of chunked transfer-encoding

HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked
25\r\n
This is the data in the first chunk\r\n
1C\r\n
and this is the second one\r\n
3\r\n
Con\r\n
8\r\n
Sequence\r\n
0\r\n

Trying HTTP (receivePost.php)

Hi <?php echo htmlspecialchars($_POST['name']); ?>,
 You are <?php echo (int)$_POST['age']; ?> old

Header Lines

- How does a browser decide which header lines to include in a request message?
  - Browser type and version
  - User configuration of the browser
  - If the browser has a cached, but possibly out-of-date version of the object
- How does a Web server decide which header lines to include in a response message?
  - Server type and version
  - Server configuration

Header Lines

- HTTP/1.0 defines 16 different header lines (no mandatory)
- HTTP/1.1 defines 51 different header lines (only one Host is mandatory) in the request messages to avoid that the server replies 400 Bad Request.

Authorization and Cookies

- HTTP server is stateless
  - Simplifies server design
  - High performance Web servers that can handle thousands of simultaneous TCP connections
- Sometimes it is desirable for a Web site to identify users
  - To restrict user access
  - To serve a content as a function of the user identity

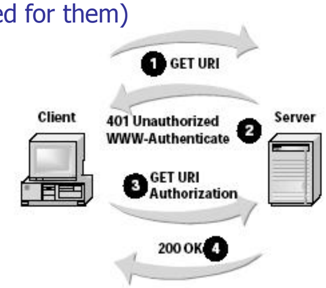
Authorization

- Requesting and receiving authorizations is often done by using special HTTP headers and status codes
- Scenario
  1. Client requests an object from a server and the server requires an authorization
  2. Client sends an ordinary request message with no special header lines
  3. The server responds with an empty entity body and 401 Authorization Required status code. In the response message the server includes the WWW-Authenticate: header

Authorization

4. The client prompts the user for a username and passwd
5. The client resends the request message including an Authorization: header line
6. After obtaining the first object, the client continues to send username and password in subsequent requests for objects on the server (username and passwords are cached and therefore the user is not prompted for them)

- Note: this is a very weak form of authorization
  - One can sniff (read and store) all the packets and therefore steal the login password.



Cookies

Many major Web sites use cookies

Four components:

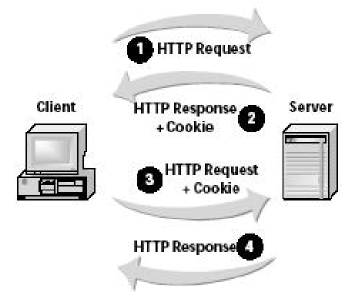
1) cookie header line of HTTP *response* message
2) cookie header line in HTTP *request* message
3) cookie file kept on user's host, managed by user's browser
4) back-end database at Web site

Example:

- Susan always access Internet always from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

Cookies

Cookies



Cookies

What cookies can bring:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

aside

Cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

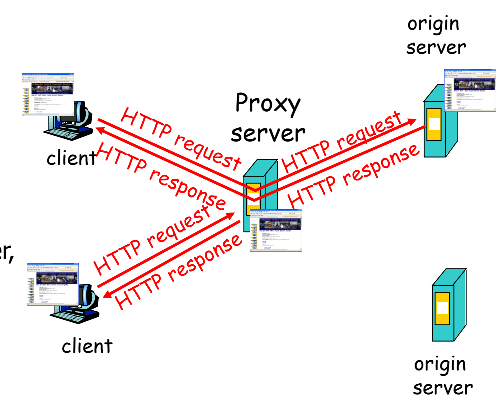
How to keep "state":

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

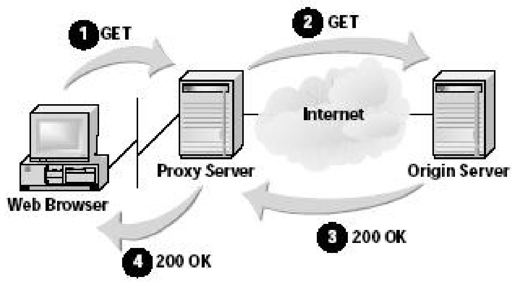
Web caching

Goal: satisfy client request without involving origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



More about Web caching



More about Web caching

- cache acts as both client and server
- typically cache is installed by the Internet Service Providers (university, company, residential ISP)

Why Web caching?

- reduce response time for client request
- reduce traffic on an institution’s access link.
- Internet dense with caches: enables “poor” content providers to effectively deliver content (but so does P2P file sharing)

Caching Example

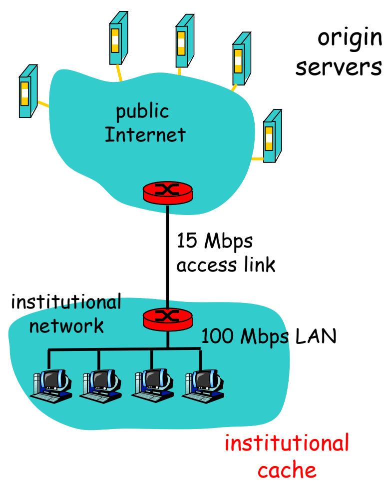
Assumptions

- average object size = 1,000,000 bits
- avg. request rate from institution’s browsers to origin servers = 15/sec
- delay from institutional router to any origin server and back to router (Internet delay) = 2 sec

Consequences

- utilization on LAN = 15%
- utilization on access link = 100%
- total delay = Internet delay + access delay + LAN delay

= 2 sec + minutes + milliseconds



Caching Example

Some Computation

Lan Traffic Intensity

(15 requests/sec) * (1Mb/request)/100Mbps = 0.15

An intensity of 0.15 results in, at most, tens of milliseconds of delay

Access Link Intensity

(15 requests/sec) * (1Mb/request)/15Mbps = 1

An intensity of 1 results in a very large delay of the order of minutes

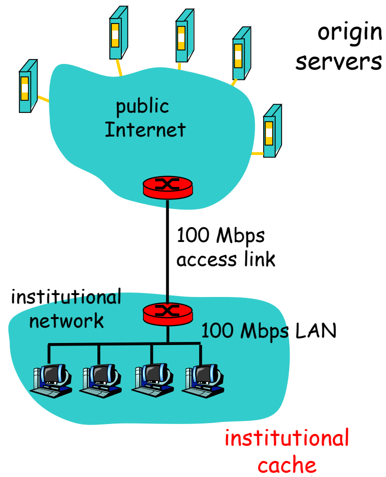
Caching Example (continued)

possible solution

- increase bandwidth of access link to, say, 100 Mbps

consequence

- utilization on LAN = 15%
- utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay
  = 2 sec + msecs + msecs
- often a costly upgrade



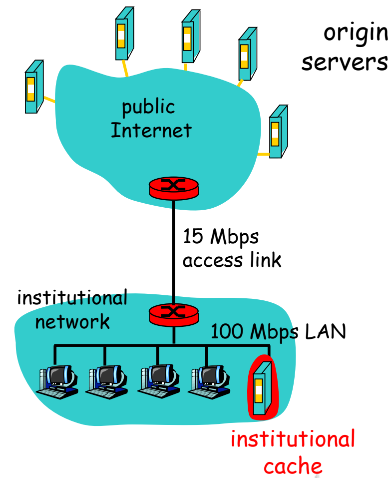
Caching Example (continued)

possible solution: install cache

- suppose hit rate is 0.4

consequence

- 40% requests will be satisfied almost immediately (order of milliseconds)
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- total avg delay = Internet delay + access delay + LAN delay = .6*(2.01) secs + .4*milliseconds < 1.4 secs



Problem

- Web caches can reside in a client (managed by the user's browser) or in an intermediate network cache server
- Problem: a copy of an object residing in the cache may be *stale* (object housed in the Web server may have been modified since the copy was cached at the client)
- Solution: the conditional GET
  - The request message uses the GET method
  - The request message includes an If-Modified-Since: header line

Conditional Cache

- Goal: don't send object if cache has up-to-date cached version
- cache: specify date of cached copy in HTTP request
  If-modified-since: <date>
- server: response contains no object if cached copy is up-to-date:
  HTTP/1.0 304 Not Modified