Part 1
Fundamentals

Contents (8 sections)

1.1 A Brief Introduction to the Internet

- **Internet history**
- **Internet protocols**

1.1 Internet History (pp.2)

1.1.1 Origins (three phases)

- ARPAnet - late 1960s and early 1970s
  - Network reliability
  - For ARPA-funded research organizations
- BITnet, CSnet - late 1970s & early 1980s
  - email and file transfer for other institutions
- NSFnet eventually became known as the Internet

Notes:

ARPA : Advanced Research Project Agency

1.1 Internet History

- NSFnet - 1986
  - Originally for non-DOD funded places
  - Initially connected five supercomputer centers
  - By 1990, it had replaced ARPAnet for non-military uses
  - Soon became the network for all (by the early 1990s)

Notes:

DOD (Department of Defense)

1.1 Internet History (pp.3)

1.1.2 What's the Internet ?

- A world-wide network of computers
- At the lowest level, since 1982, all connections use TCP/IP
- TCP/IP hides the differences among devices connected to the Internet
- It allows a program on one computer to communicate with a program on another computer.

Notes:

TCP (Transfer Control Protocol)

IP (Internet Protocol)

1.1 Internet Protocols

1.1.3 Internet Protocol (IP) Addresses(1982)

- Every node has a unique numeric address
- Form: 32-bit binary number
  - New standard, IPv6, has 128 bits (1998)
- Organizations are assigned groups of IPs for their computers. For example, millions of IPs are assigned to DoD.
- Several different protocols had been invented and were being used on the Internet, all with different user interfaces (Telnet, FTP, Usenet, Mailto)
- For example: a href="mailto:hjymail@163.com"

1.1 Internet Protocols (pp.4)

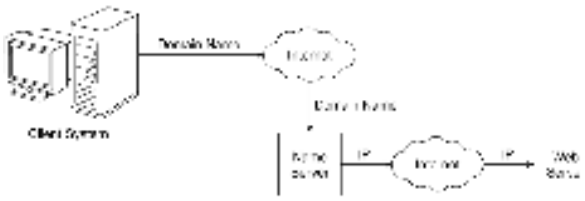
1.1.4 Domain names (域名)

- People have difficulty to remember the numeric IP addresses. So textual names are invented.
- Form: host-name.domain-names
- For example: db.cs.berkeley.edu
- First domain is the smallest; last is the largest
- Last domain specifies the type of organization
- Fully qualified domain name - the host name and all of the domain names
- Fully qualified domain name must be unique.

1.1 Internet Protocols

- Domain Name Servers (DNS) - convert fully qualified domain names to IPs.
- Name servers are a collection of computers and are responsible for their own organizations.
- How it works can be illustrated by Figure 1.1.

Figure 1.1 Domain name conversion



1.1 Internet Protocols

- Clients and Servers are programs that communicate with each other over the Internet
- A Server runs continuously, waiting to be contacted by a client
  - Each Server provides certain services
  - Services include providing web pages
- A Client will send a message to a Server requesting the service provided by that server
  - The client will usually provide some information, parameters, with the request

1.2 World-Wide Web (pp.6)

- A possible solution to the proliferation of different protocols being used on the Internet

1.2.1 Origins

- Tim Berners-Lee at CERN proposed the Web in 1989
  - Purpose: to allow scientists to have access to many databases of scientific works through their own computers
- Document form: hypertext
- Pages? Documents? Resources?
  - We'll call them documents
- Hypermedia – more than just text – images, sound, etc.

1.2 The World-Wide Web

1.2.2 Web or Internet?

- The internet is a collection of computers and devices which communicate with each other through different kinds of protocols (hardware).
- The Web uses one of the protocols, http, that runs on the Internet--there are several others (telnet, mailto, etc.). It cannot work properly without the internet.
- http- HyperText Transfer Protocol

1.3 Web Browsers

- Browsers are clients - always initiate a communications, servers react (although sometimes servers require responses)
- Most requests are for existing documents, using HyperText Transfer Protocol (HTTP)
- But some requests are for program execution, with the output being returned as a document

Notes:

NCSA: National Center for Supercomputer Applications

1.4 Web Servers

- Provide responses to browser requests, either existing documents or dynamically built documents
- Browser-server connection is now maintained through more than one request-response cycle
- All communications between browsers and servers use Hypertext Transfer Protocol (HTTP)

1.4.1 Web Server Operation

- Web servers run as background processes in the operating system
  - Monitor a communications port on the host, accepting HTTP messages when they appear
- Currently, the most common server configuration is Apache on some versions of UNIX.

1.4.2 Web Server Operation Details

- Web servers have two main directories:
  1. Document root (servable documents)
  2. Server root (server system software)
- Document root is accessed indirectly by clients
  - Its actual location is set by the server configuration file
  - Requests are mapped to the actual location

1.4.3 Apache

- It is by far the most widely used web servers
  - open source
  - fast
  - reliable

1.4.4 IIS

- IIS
  - Operation is maintained through a program with a GUI interface
  - In contrast, Apache is maintained by editing a configuration file

1.5 Uniform Resource Locators (URLs)

- URLs are used to identify documents (resources) on the internet. Different resources are identified by different kinds of URLs.
- General form:

  schema:object-address
  - The scheme is often a communications protocol, such as http, telnet, file or ftp
  - For the http protocol, the object-address is: fully qualified domain name/doc path
  - For the file protocol, only the doc path is needed

1.5 URLs

- Host name may include a port number, as in zeppo:80 (80 is the default, so this is silly)
- URLs cannot include spaces or any of a collection of other special characters (semicolons, colons, ampersands). Otherwise, the special characters must be coded in a special way.

1.6 Multipurpose Internet Mail Extensions (MIME)

- Originally, it is developed to allow different kinds of files to be sent by mail.
- Used to specify to the browser the form of a file returned by the server (attached by the server to the beginning of the document)

1.6 Multipurpose Internet Mail Extensions (MIME)

- Type specifications
  - Form:

    type/subtype
  - The most common types are text, image, and video.
  - The most common subtypes of text are plain and html
  - The most common subtypes of images are jpeg and gif
  - Examples: text/plain, text/html, image/gif, image/jpeg
  - In the remainder of the book, we refer to the type/subtype as *document type*

  .

1.6 Multipurpose Internet Mail Extensions (MIME)

- Server gets type from the requested file name's suffix (.html implies text/html)
- Browser gets the type explicitly from the server (This is in the header field of http protocol)

1.7 The HyperText Transfer Protocol

- The protocol is used by *all* Web communications.
- HTTP consists of two phases, the request and the response.
- Each http communication mainly consists of two parts, a header and a body.
- The header contains the information about the communications, while the body contains the data which are to be transferred between servers and browsers.

1.7.1 Request Phase

General Form:

1. HTTP method URL HTTP version
2. Header fields
3. blank line
4. Message body

- An example of the first line of a request:
  GET /degrees.html HTTP/1.1

1.7 The HyperText Transfer Protocol: Methods

- Get - fetch a document
- Post - execute the document, using the data in body
- Head - fetch just the header of the document
- Put - store a new document on the server
- Delete - remove a document from the server

1.7.1 Request Phase

- Following the first line of an http communication is any number of header fields, most of which are optional.
- The format of a header field is the *field name* followed by a *colon* and the *value* of the field.
- Four categories of the header fields:
  - General: For general information, such as date
  - Request: included in the request headers
  - Response: for response headers
  - Entity: used in both request and response headers.

HTTP Headers

- One common request fields:

  Accept: text/plain

  Accept: text/*

  If-Modified-since: date

1.7.2 HTTP Response

- Form:
  Status line
  Response header fields
  blank line
  Response body
- Status line format:
  HTTP version status code explanation
- Example: HTTP/1.1 200 OK
  (Current version is 1.1)

1.7.2 HTTP Response

- Status code is a three-digit number; first digit specifies the general status
  - 1 => Informational
  - 2 => Success
  - 3 => Redirection
  - 4 => Client error
  - 5 => Server error
- The header fields can contain several lines of information about the response, each in the format of a field.
- The only essential field is Content-type

HTTP Headers

- Common response fields:

  Content-length: 488

  Content-type: text/html

1.7.2 HTTP Response Example (status line and header)

HTTP/1.1 200 OK

Date: Tues, 18 May 2004 16:45:13 GMT

Server: Apache (Red-Hat/Linux)

Last-modified: Tues, 18 May 2004 16:38:38 GMT

Etag: "841fb-4b-3d1a0179"

Accept-ranges: bytes

Content-length: 364

Connection: close

Content-type: text/html, charset=ISO-8859-1

1.8 The Web Programmer's Toolbox

- This section provides an overview of the programming languages.
- XHTML is a markup language, and XML is a meta-markup language
- PHP is server-side languages
- JavaScript is most often a client-side language, although it can also be used as server-side language.

1.8.1 XHTML

- To describe the general form and layout of documents, and it is not a programming language.
- An XHTML document is a mix of content and controls
  - Controls are *tags* and their *attributes*
    - Tags often delimit content and specify something about how the content should be arranged in the document
    - Attributes provide additional information about the content of a tag
    - For example: <img src="redhead.jpg">

1.8.1 Creating XHTML documents

- XHTML editors - make document creation easier
  - Shortcuts to typing tag names, spell-checker,
- WYSIWYG XHTML editors
  - Need not know XHTML to create XHTML documents
- Commonly used tools for editing XHTML
  - Frontpage
  - Dreamweave
  - Editplus
- Example:1-1.html

1.8.4 XML

- A meta-markup language
- Used to create a new markup language for a particular purpose or area
- Because the tags are designed for a specific area, they can be meaningful
- No presentation details
- A simple and universal way of representing data of any textual kind
- Example:1-2.xml

1.8 JavaScript

- A client-side HTML-embedded scripting language
- Only related to Java through syntax
- Dynamically typed and not object-oriented
- Provides a way to access elements of HTML documents and dynamically change them
- Example:1-3.html

1.8.6 Java

- General purpose object-oriented programming language
- Based on C++, but simpler and safer
- Our focus is on Servlets, and JSP

1.8 Perl (optional)

- Provides server-side computation for HTML documents, through CGI
- Perl is good for CGI programming because:
  - Direct access to operating systems functions
  - Powerful character string pattern-matching operations
  - Access to database systems
- Perl is highly platform independent, and has been ported to all common platforms
- Perl is not just for CGI

1.8.8 PHP

- A server-side scripting language
- Great for form processing and database access through the Web
- Example:1-4.php

Homework

- Install the wamp (Windows, Apache, Mysql, PHP) or lamp (Linux, Apache, Mysql, PHP)
- Editing the examples

Part 2

Introduction to XHTML

Contents

2.1 Origins and Evolution of HTML

- HTML was derived from SGML
- Original intent of HTML: General layout of documents that could be displayed by a wide variety of computers
- Recent versions:
  - HTML 4.0 – 1997
    - » Introduced many new features and deprecated many older features
  - HTML 4.01 - 1999 - A cleanup of 4.0
  - XHTML 1.0 - 2000
    - » Just 4.01 defined using XML, instead of SGML

SGML: Standard generalized markup language

2.1 Origins and Evolution of HTML

- Reasons to use XHTML, rather than HTML:

1. HTML has lax syntax rules, leading to sloppy and sometime ambiguous documents
   - – XHTML syntax is much more strict, leading to clean and clear documents in a standard form
2. HTML processors do not even enforce the few syntax rule that do exist in HTML
3. The syntactic correctness of XHTML documents can be validated

2.2 Basic Syntax

- Elements are defined by tags (markers)
  - Tag format:
    - » Opening tag: <name>
    - » Closing tag: </name>
  - The opening tag and its closing tag together specify a container for the *content* they enclose
  - Example:

 This is extremely simple.

2.2 Basic Syntax

- Not all tags have content
  - If a tag has no content, its form is <name />
- The container and its content together are called an *element*
- If a tag has attributes, they appear between its name and the right bracket of the opening tag

For example,

2.2 Basic Syntax

- Comment form: <!-- … -->
- Browsers ignore *comments, unrecognizable tags, line breaks, multiple spaces*, and *tabs*
- Tags are suggestions to the browser, even if they are not recognized by the browser

2.3 HTML Document Structure

- Every XHTML document begin with <html>
- <html>, <head>, <title>, and <body> are usually required in every document

2.3 HTML Document Structure

- The whole document must have <html> as its root
- A document consists of a *head* and a *body*
- The <title> tag is used to give the document a title, which is normally displayed in the browser's window title bar (at the top of the display)
- Example:2-test.html

2.4 Basic Text Markup-2.4.1 Paragraphs

- Text is normally placed in *paragraph* elements
- *Paragraph Elements*
  - The <p> tag breaks the current line and inserts a blank line - the new line gets the beginning of the content of the paragraph
  - The browser puts as many words of the paragraph's content as will fit in each line

  Example: 2-lamb.html

2.4 Basic Text Markup-2.4.1 Paragraphs

- codes of greet.html

```
<html>
  <head> <title> Our first document </title>
  </head>
  <body>
    <p>
      Greetings from your Webmaster!
    </p>
  </body>
</html>
```


2.4 Basic Text Markup-2.4.3 Line breaks

- Line breaks
  - The effect of the
 tag is the same as that of <p>, except for the blank line
  - No closing tag!
- Example of paragraphs and line breaks

```
On the plains of hesitation <p> bleach the
bones of countless millions </p>
who, at the dawn of victory <br /> sat down
to wait, and waiting, died.
```

- Typical display of this text (2-linebreak.html)

```
On the plains of hesitation

bleach the bones of countless millions

who, at the dawn of victory
sat down to wait, and waiting, died.
```

2.4 Basic Text Markup-2.4.4 Headings

- Headings
  - Six sizes, 1 - 6, specified with <h1> to <h6>
  - 1, 2, and 3 use font sizes that are larger than the default font size
  - 4 uses the default size
  - 5 and 6 use smaller font sizes
- Example: 2-headings.html

2.4 Text Markup-2.4.5 Block Quotations

- Blockquote
  - Content of <blockquote>
  - To set a block of text off from the normal flow and appearance of text
  - Browsers often indent, and sometimes italicize the content
- Please analyze the display style of Fig 2.8 (2-blockquotes.html)

2.4 Text Markup -2.4.6 Font styles and sizes

- Font Styles and Sizes (can be nested)
  - Boldface - <b>
  - Italics - <i>
  - Larger - <big>
  - Smaller - <small>
  - Monospace - <tt>
- Examples of Figure 2.9 and Figure 2.10

2.4 Text Markup -2.4.6 Font styles and sizes

- Superscripts and subscripts
  - Subscripts with <sub>
  - Superscripts with <sup>
- Example: x₂³
  Display: x_2^3

2.4 Text Markup -2.4.7 Character entities

- Character Entities

Char.	EntityMeaning
&	&Ampersand
<	<Less than
>	>Greater than
”	"Double quote
’	'Single quote
¼	¼One quarter
½	½One half
¾	¾Three quarters
°	°Degree
(space)	Non-breaking space

2.4 Text Markup -2.4.(8-9)

- Horizontal rules
  - <hr /> draws a line across the display, after a line break
  - Typically, the line is 3 pixels thick
- The meta element (for search engines) used to provide additional information about a document, with attributes
  - The two attributes that are used to provide information are *name* and *content*
  - The user makes up a name as the value of the *name* attribute and specifies information through the *content* attribute.
- Example 2-meta.html

2.5 Images

- Images are inserted into a document with the <img /> tag with the *src* attribute
  - –The *alt* attribute can be used
    - » Purposes:
      1. Non-graphical browsers
      2. Browsers with images turned off

    <img src = "comets.jpg"
    alt = "Picture of comets" />

2.5 Images

- The <img> tag has 30 different attributes, including ***width*** and ***height*** (in pixels)
- Portable Network Graphics (PNG)
  - – Relatively new
  - – Should eventually replace both *gif* and *jpeg*
- Example 2-boy.html

2.6 Hypertext Links

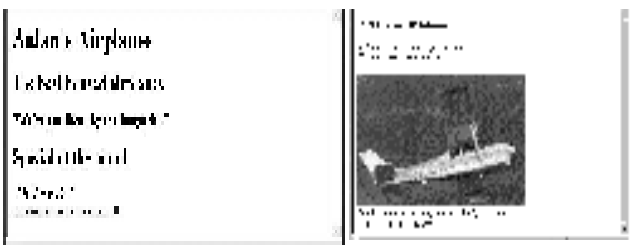
- Hypertext is the essence of the Web!
- A link is specified with the ***href*** (hypertext reference) attribute of <a> (the anchor tag)
  - The content of <a> is the visual link in the document
  - If the target is a document in the same directory, the target is the document's file name
  - Note: Relative addressing of targets is easier to maintain and more portable than absolute addressing

2.6 Hypertext Links

```
<html>
 <head> <title> Links </title>
 </head>
 <body>
  <h1> Aidan's Airplanes </h1>
  <h2> The best in used airplanes </h2>
  <h3> "We've got them by the hangarful"
  </h3>
  <h2> Special of the month </h2>
  <p>
   1960 Cessna 210 <br />
   <a href = "C210data.html">
    Information on the Cessna 210 </a>
  </p>
 </body>
</html>
```

Example 2-link.html and 2-linksub.html

2.6 Hypertext Links



2.6 Hypertext Links

- If the target is not at the beginning of the document, the target spot must be marked
- Target labels can be defined in many tags with the *id* attribute, as in
  <h1 id = "baskets"> Baskets </h1>
- The link to an *id* must be preceded by a pound sign (#);
  - ü If the *id* is in the same document, the target could be

    What about baskets?
  - ü If the target is in a different document, the document reference must be included as follows
     Baskets

2.6 Hypertext Links

- Links can have images:

```
<a href = "c210data.html“>
<img src = "smallplane.jpg”
    height=“20”
    width=“20”
    alt = "Small picture of an airplane “
/>
  Info on C210 </a>
```

- Example 2-linkimg.html

Class exercises

- Create an html document with the following features
- (1) Display your name
- (2) There are three paragraphs, and each paragraph has a heading and the content. The paragraphs are used to introduce *you*, *your mother* and *your father*. You can just make up the content.
- (3) There is a link in the paragraph of *you* to NJUPT home page
- (4)There are at least one line of comments to introduce the content.

2.7 Lists

- Contents
  - 2.7.1 Unordered list
  - 2.7.2 Ordered list
  - 2.7.3 Definition list

2.7 Lists -2.7.1 Unordered list

- The list is the content of the <ul> tag
- Each item in a list is specified with an <li> tag. When the item is displayed, it is preceded with a bullet.
- Example 2-unordered.html.

2.7 Lists -2.7.2 Ordered list

- The list is the content of the <ol> tag
- Each item in the display is preceded by a sequence value, such as 1, 2, 3, etc.
- Example 2-ordered.html.

2.7 Lists -2.7.2 Ordered list

- The list is the content of the <ol> tag
- Each item in the display is preceded by a sequence value, such as 1, 2, 3, etc.
- Example 2-ordered.html.

2.7 Lists-2.7.3 Definition list

- Definition lists are used to specify lists of terms and their definitions, such as glossaries.
  - List is the content of the <dl> tag
  - Terms being defined are the content of the <dt> tag
  - The definitions themselves are the content of the <dd> tag
- Example 2-definition.html

2.8 Tables

- A table is a matrix of cells, each possibly having content
- The cells can include almost any element
- Some cells have row or column labels and some have data
- A table is specified with a <table> tag
- A *border* attribute in the <table> tag specifies a border
- If *border* is set to "border", the browser's default border width applies
- The *border* attribute can be set to a number, which will be the border width
- Without the *border* attribute, the table will have no lines!

2.8 Tables

- Tables are given titles with the <caption> tag, which can immediately follow the <table> tag
- Each row of a table is specified as the content of a <tr> tag
- The row headings are specified as the content of a <th> tag
- The contents of a data cell is specified as the content of a <td> tag
- Example 2-table.html

2.8 Tables-2.8.2 *rowspan* and *colspan*

- A table can have two levels of column labels
  - If so, the *colspan* attribute must be set in the <*th*> or <*td*> tag to specify that the label or data must span some number of columns
- Example

```
<tr>
  <th colspan = "3"> Fruit Juice Drinks </th>
</tr>
<tr>
      <th> Orange </th>
      <th> Apple </th>
      <th> Screwdriver </th>
    </tr>
```

2.8 Tables -2.8.3 *align* and *valign*

- The *align* attribute controls the horizontal placement of contents in a table cell
  - Values are left (for data default), right, and center (for header default)
  - *align* is an attribute of <tr>, <th>, and <td> elements
- The *valign* attribute controls the vertical placement of contents of a table cell
  - Values are top, bottom, and center (default)
  - *valign* is an attribute of <th> and <td> elements

  à show 2-cell_align.html and display it

2.8 Tables-2.8.4 *cellpadding* and *cellspacing*

- The cellspacing attribute of <table> is used to specify the distance between cells in a table
- The cellpadding attribute of <table> is used to specify the spacing between the content of a cell and the inner walls of the cell
- Example 2-space_pad.html

2.8 Tables

Example 2-tablespecial.html

```
<table cellspacing = "50">
    <tr>
      <td> Colorado is a state of …
      </td>
      <td> South Dakota is somewhat…
      </td>
    </tr>
 </table>
```

2.8 Tables-2.8.5 Table sections

- Table Sections
  - Header, body, and footer, which can be specified with the tag: *thead, tbody,* and *tfoot,* respectively.
- Example 2-tablesection.html. Please also pay attention to the *CSS*.

Class exercises

- Exercise 9 (pp.91)

2.9 Forms

- A form is the usual way that information is sent from a browser to a server
- HTML has tags to create a collection of objects that implement this information gathering
  - The objects are called *widgets* (e.g., radio buttons and checkboxes)
- When the submit button of a form is clicked, the form's values are sent to the server

2.9 Forms

- All of the widgets, or components of a form are defined in the content of a <form> tag
  - The only required attribute of <form> is *action*, which specifies the URL of the application that is to be called when the submit button is clicked. For example,

    action ="http://www.cs.ucp.edu/cgi-bin/survey.php"

    » If the form has no *action*, the value of *action* is the empty string
- The *method* attribute of <form> specifies one of the two possible techniques of transferring the form data to the server, *get* and *post*
  - *get* and *post* are discussed later.
  - *get* is the default method.

2.9 Forms-2.9.2 <input> tag

- Widgets
  - Many are created with the <input> tag
    - » The *type* attribute of <input> specifies the kind of widget being created

    1.Text
    - Creates a horizontal box for text input
    - Default size is 20; it can be changed with the *size* attribute
    - If more characters are entered than will fit, the box is scrolled (shifted) left
    - <input type = "text" name = "Jack" size = "25" />

2.9 Forms -2.9.2 <input> tag

– If you don't want to allow the user to type more characters than will fit, set ***maxlength***, which causes excess input to be ignored

```
<input type = "text" name = "Phone"
    size = "12" maxlength="12">
```

Example:2-textpassword.html

2. Checkboxes - to collect multiple choice input (type="checkbox")

– Every checkbox requires a ***value*** attribute, which is the widget's value in the form data when the checkbox is 'checked'.

» A checkbox that is not 'checked' contributes no value to the form data.

2.9 Forms -2.9.2 <input> tag

– By default, no checkbox is initially 'checked'

– To initialize a checkbox to 'checked', the ***checked*** attribute must be set to "checked"

– Example 2-checkbox.html

3. Radio Buttons - collections of checkboxes in which only one button can be 'checked' at a time (type="radio")

» Every button in a radio button group MUST have the same ***name***

» The ***value***s are usually different

» Example 2-radio.html

2.9 Forms -2.9.2 <input> tag

```
<form action = "">
 <p>
 <input type = "radio"  name = "age"
  value = "under20" checked = "checked"> 0-19
 <input type = "radio"  name = "age"
      value = "20-35"> 20-35
 <input type = "radio"  name = "age"
      value = "36-50"> 36-50
 <input type = "radio"  name = "age"
      value = "over50"> Over 50
 </p>
</form>
```


2.9 Forms -2.9.3 Menus

- Menus - created with <select> tags
- There are two kinds of menus, those that behave like checkboxes and those that behave like radio buttons (the default).
- The ***name*** attribute of <select> is required.
- The ***size*** attribute of <select> can be included to specify the number of menu items to be displayed (the default is 1).

2.9 Forms -2.9.3 Menus

– Each item of a menu is specified with an <option> tag, whose pure text content is the value of the item.

– An <option> tag can include the ***selected*** attribute. When it is assigned "selected", it specifies that the item is preselected.

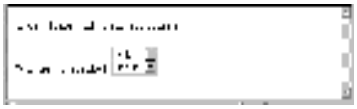
```
<form action = "">
 <p>
  With size = 1 (the default)
  <select name = "groceries">
   <option> milk </option>
   <option> bread </option>
   <option> eggs </option>
   <option> cheese </option>
  </select>
 </p>
</form>
```


2.9 Forms -2.9.3 Menus

- Widgets (continued)

- After clicking the menu:

- After changing size to 2:



2.9 Forms -2.9.4 <textarea> tag

- Text areas - created with <textarea> tag
  - Usually include the *rows* and *cols* attributes to specify the size of the text area
  - Default text can be included as the content of <textarea>
  - Scrolling is implicit if the area is overfilled
- Example:

```
<p>Please provide your employment aspirations</p>
<form action = "">
  <p>
  <textarea name = "aspirations"  rows = "3"
        cols = "40">
  (Be brief and concise)
  </textarea>
  </p>
</form>
```

2.9 Forms -2.9.5 submit and reset buttons

- Both are created with <input>

  <input type = "reset" value = "Reset Form">

  <input type = "submit" value = "Submit Form">
- Submit has two actions:
  1. Encode the data of the form
  2. Request that the server execute the server-resident program specified as the value of the *action* attribute of <form>
  3. A submit button is required in every form

  --> show popcorn.html and display it

Class exercises

- **Exercise 16 of page 92**

2.11 Syntactic Differences HTML & XHTML

- Case sensitivity
- Closing tags
- Quoted attribute values
- Explicit attribute values
- id and name attributes
- Element nesting

Homework

- Exercises 1,2,3,9,16.
- First, you should code and validate the html document in WAMP environment. Then, the codes are written to homework notebook.

Part 3
Cascading Style Sheets

Content

3.1 Introduction

- CSS provides the means to control and change presentation of HTML documents.
- Style sheets allow you to impose a standard style on a whole document, or even a whole collection of documents.
- Style is specified for a tag by the values of its properties.
- CSS is not technically HTML, but can be embedded in HTML documents

3.2 Levels of Style Sheets

- There are three levels of style sheets:
  - Inline level - specified for a specific occurrence of a tag and apply only to that tag.
    – This is fine-grain style, which defeats the purpose of style sheets - uniform style.
  - Document level - apply to the whole document in which they appear.
  - External level - can be applied to any number of documents.
- When more than one style sheet applies to a specific tag in a document, the lowest level style sheet has precedence.

3.2 Levels of Style Sheets

- *Inline style* sheets appear in the tag itself

```
<p style="color：#ff0000">inline style sheet</p>
```

- *Document-level* style sheets appear in the head of the document

```
<style type="text/css">
        p{color:red;font-weight:bold}
</style>
```

- *External style* sheets are in separate files, potentially on any server on the Internet
  - Written as text files

3.2 Levels of Style Sheets

- A <link> tag is used to specify that the browser is to fetch and use an external style sheet file

```
<link rel = "stylesheet"  type = "text/css"
    href = "http://www.wherever.org/termpaper.css">
</link>
```

- Example 3-external.html

3.3 Style Specification Formats

- Format depends on the level of the style sheet
- Inline:
  - Style sheet appears as the value of the `style` attribute of a tag
  - General form:

```
style = "property_1: value_1;
         property_2: value_2;
         …
         property_n: value_n"
```

  - Example 3-inline.html

3.3 Style Specification Formats-document level

- Style sheet appears as a list of rules that are the content of a <style> tag
- The <style> tag must include the *type* attribute, setting to "text/css"

3.3 Style Specification Formats-document level

- General form:

```
<style type = "text/css">
    rule list
</style>
```

- Form of the rules:
  selector {list of property/values}
  - Each property/value pair has the form
    property: value
  - Pairs are separated by semicolons, just as in the value of a <style> tag
- Example 3-document.html

3.3 **Style Specification Formats-**External

- It is a list of style rules, as in the content of a <style> tag for document-level style sheets
- But the rules are in a separate file with suffix .css
- Example 3-external.html

3.4 Selector Forms

- 3.4.1 simple selector forms
- 3.4.2 class selectors
- 3.4.3 generic selectors
- 3.4.4 id selectors
- 3.4.5 pseudo class

3.4.1 Simple selector form

- The selector is a tag name or a list of tag names, separated by commas

```
h1, h3 {font-size:30pt}
p {font-size:20pt}
```

- Contextual selectors

```
body b i{font-size:20pt}
```

3.4.2 Class Selectors

- It allows different occurrences of the same tag to use different style specifications.
- A style class has a name, which is attached to a tag name.

  p.narrow {property/value list}

  p.wide {property/value list}
- The class you want on a particular occurrence of a tag is specified with the *class* attribute of the tag
- For example:

  ...

  ...

  ...

3.4.3 Generic Selectors

- A generic class can be defined if you want a style to apply to more than one kind of tag
- A generic class must be named, and the name must begin with a period
- Example:

  .really-big { … }
- Use it as if it were a normal style class

  <h1 class = "really-big"> … </h1>

  ...

   …

3.4.4 id Selectors

- An id selector allow the application of a style to one specific element.
- General form:

  #specific-id {property-value list}
- Example:

  #section14 {font-size: 20}

  <h2 id="section14">1.4 Example</h2>

3.4.5 Pseudo Classes

- Pseudo classes are styles that apply when something happens, rather than because the target element simply exists
- Names begin with colons
- Example pseudo.html
  - hover classes apply when the mouse cursor is over the element
  - focus classes apply when an element has focus

3.4.5 Pseudo Class Example

```
<!-- pseudo.html -->
<head> <title> Checkboxes </title>
    <style type = "text/css">
      input:hover {color: red;}
      input:focus {color: blue;}
    </style>
  </head>
  <body>
    <form action = "">
      <p>
        Your name:
        <input type = "text" />
      </p>
    </form>
  </body>
 </html>
```


3.5 Properties

- There are 60 different properties in 7 categories:
  - Fonts
  - Lists
  - Alignment of text
  - Colors
  - Backgrounds
  - Special sections

3.6 Font Properties

- *font-family*
  - Value is a list of font names - browser uses the first in the list it has
    - *font-family: Arial, Helvetica, Courier*
  - If a font name has more than one word, it should be single-quoted.
    - *font-family: 'Times New Roman'*

3.6.1 Font Properties

- *font-size*
  - Possible values: a length number or a name, such as *smaller*, *xx-large*, etc.
  - font-size:10pt
- *font-style*
  - *italic, normal*

3.6.1 Font Properties

- *font-weight* - degrees of boldness
  - *bolder, lighter, bold, normal*
    - Could specify as a multiple of 100 (100 – 900)
- *font*
  - For specifying a list of font properties

    font: bolder 14pt Arial Helvetica
  - Order must be: style, weight, size, family(s)

3.6 Font Properties

- Example fonts.html and fonts2.html
- The *text-decoration* property
  - *line-through, overline, underline, none*
  - Example 3-decoration.html

3.7 List properties

- *list-style-type* property of an unordered lists
  - Bullet can be a *disc* (default), a *square*, a *circle* or *none*
  - Set it in either the <ul> or <li> tag
    - In <ul>, it applies to all list items
- Example

```
<h3> Some Common Single-Engine Aircraft </h3>
<ul style = "list-style-type: square">
    <li> Cessna Skyhawk </li>
    <li> Beechcraft Bonanza </li>
    <li> Piper Cherokee </li>
</ul>
```


3.7 List properties

- In <li>, *list-style-type* applies to just that item
- Example

```
<h3> Some Common Single-Engine Aircraft </h3>
<ul>
    <li style = "list-style-type: disc">
        Cessna Skyhawk </li>
    <li style = "list-style-type: square">
        Beechcraft Bonanza </li>
    <li style = "list-style-type: circle">
        Piper Cherokee </li>
</ul>
```

3.7 List properties

- *List-style-image* property uses an image for the bullets in an unordered list
- Example:

```
<li style = "list-style-image: url(bird.jpg)">…</li>
```

- On ordered lists *list-style-type* property can be used to change the sequence values

Property value	Sequence type	First four
decimal	Arabic numerals	1, 2, 3, 4
upper-alpha	Uppercase letters	A, B, C, D
lower-alpha	Lowercase letters	a, b, c, d
upper-roman	Uppercase Roman	I, II, III, IV
lower-roman	Lowercase Roman	i, ii, iii, iv

- Example 3-sequence_types.html

3.8 Colors

- The *color* property specifies the foreground color of elements

```
<style type = "text/css">
   th.red {color: red}
   th.orange {color: orange}
</style>  …
<table border = "5">
  <tr>
    <th class = "red"> Apple </th>
    <th class = "orange"> Orange </th>
  </tr>
</table>
```

- The *background-color* property specifies the background color of elements

3.9 Alignment of text

- The *text-indent* property allows the indentation of the first line
  - Takes either a length or a % value
  - *text-indent:0.5in*
- The *text-align* property is used to arrange text horizontally.
- The *text-align* property has the possible values, *left* (the default), *center*, *right*, or *justify*

3.9 Alignment of text

- *float* :Sometimes we want text to flow around another element - the *float* property
  - The *float* property has the possible values, *left*, *right*, and *none* (the default)
  - If we have an element on the right, with text flowing on its left, we use the default *text-align* value (*left*) for the text and the *right* value for *float* on the element we want on the right

```
<img src = "c210.jpg"    style = "float: right" />
```

  - The text with the default alignment - *left*
  - Example 3-float.html

3.10.2 Margins

- The *margin* property is the space between the border of an element and the element's neighbor
- It can also be refined into four properties, margin-left, margin-right, margin-bottom, and margin-top
- Example 3-margin.html

3.11 Background images

- *background-image* property is used to place an image in the background of an element.
- Example 3-back.html

3.11 Background images

- *background-image* property is used to place an image in the background of an element.
- Example 3-back.html

3.12 The and tags

- and
- In many situations, we want to apply special font properties to less than a whole paragraph of text.
  - Solution: a new tag to define an element in the content of a larger element -
  - The default meaning of is to leave the content as it is

```
<p>
  Now is the <span> best time </span> ever!
</p>
```


3.12 The and tags

- Use to apply a document style sheet to its content

```
<style type = "text/css">
   .bigred {font-size: 24pt;font-family: Ariel; color: red}
</style>  ….
  <p>
    Now is the
      <span class = "bigred">
    best time </span> ever!
  </p>
```

3.12 The and tags

- Another tag that is useful for style specifications:
  - Used to create document sections (or divisions) for which style can be specified.
    - e.g., A section of heading and paragraphs for which you want some particular style.
- Example 3-div.html

Class exercises

- **Exercise 5 (pp.127)**

Part 4

Basics of JavaScript

Contents

4.1 Overview of JavaScript: Origins

- Origin: developed by Netscape
- Supported by Netscape, Mozilla, Internet Explorer
- JavaScript Components
  - Core:
    - The heart of the language: operators, expressions, etc.
  - Client-side:
    - Library of objects supporting browser control and user interaction such as button, form, etc.
  - Server-side:
    - Library of objects that support use in web servers

4.1 Uses of JavaScript

- Client-side JavaScript is used far more frequently than server-side JavaScript.
- Client-side JavaScript is embedded in HTML, Example 4-1.html
- Advantages:

(1)Provide alternative to server-side programming
  - Servers are often overloaded
  - Client processing has short reaction time

4.1 Uses of JavaScript

(2)JavaScript can work with forms

(3)JavaScript can interact with the internal model of the web page (Document Object Model)

(4)JavaScript is used to provide more complex user interface

- Resources
  - http://www.protopage.com/ is an interesting example
  - http://300mb.us is also a useful self-publishing sites.

4.1 Event-driven Computation

- Users actions, such as mouse clicks and key presses, are referred to as *events.*
- JavaScript task: The main task of JavaScript programs is to respond to *events.*
- For example, a JavaScript program could validate data in a form before it is submitted to a server.

4.1 XHTML/JavaScript Documents

- When JavaScript is embedded in an XHTML document, the browser must *interpret* it.
- JavaScript positions: Two locations for JavaScript serve different purposes
  - JavaScript in the *head* element contains functions which will be called from other locations.
  - JavaScript in the *body* element will be executed once as the page is loaded.
  - Example 4-2.html

4.2 Object Based and JavaScript

- JavaScript is *object-based*
  - JavaScript defines objects that encapsulate both data and methods.
  - However, JavaScript does not have true inheritance nor subtyping.

4.2 JavaScript Objects

- Objects are collections of *properties*, which are either *data properties* or *method properties*
  - *Data properties* are either primitive values or references to other objects.
  - *Method property* is often referred as *method*.
- The root object in JavaScript is *Object*, which is the ancestor of all objects in a JavaScript program.
  - *Object* has no data properties, but several *method properties*

4.3 General Syntactic Characteristics

- Directly embedded

```
<script type="text/javascript">
    …Javascript here…
</script>
```

- Indirect reference

```
<script type="text/javascript" src="tst_number.js"/>
```

  - This is the preferred approach.
- Note: the double-quotation marks must be input in English format.

4.3 General Syntactic Characteristics

- Identifiers or names
  - Start with dollar sign($), underscore(_) or a letter
  - Continue with $, _, letter or digit
  - Case sensitive
- 25 Reserved words (refer to table 4.1 in pp. 135)
- Comments
  - //
  - /* … */

4.3 Statement Syntax

- Statements can be terminated with a semicolon.
- However, the interpreter will insert the semicolon if it is missing at the end of a line and the statement seems to be complete.
  - Can be a problem:

```
return
x;
```

- If a statement must be continued to a new line, ***make sure*** that the first line does not make a complete statement by itself.

4.4.1 Primitive Types

- Five primitive types
  - Number
  - String
  - Boolean
  - Undefined
  - Null
- Five classes corresponding to the five *primitive types*
  - *Wrapper* objects for primitive values, which contain methods and properties relevant to the primitive types
  - Primitive values are *coerced* to the wrapper class as necessary, and vice-versa

4.4.1 Primitive and Object Storage

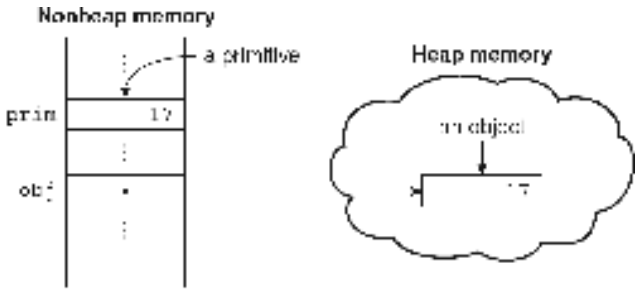


Figure 4.1 Primitives and objects

4.4.2 Numeric and String Literals

- Number values are represented internally as double-precision floating-point values.
  - Number literals can be either integer or float.
  - Float values may have a decimal and an exponent.
- A String literal is delimited by single or double quotes.
  - There is no difference between single and double quotes.
  - Certain characters may be *escaped* in strings
    - \’ or \” to use a quote in a string delimited by the same quotes, \\ to use a literal backslash
  - The empty string ‘’ or “” has no characters.

4.4.3 Other Primitive Types

- Null
  - A single value, *null*
  - *null* is a reserved word
  - A variable that is used but has not been declared nor been assigned a value has a *null* value.
  - Using a *null* value usually causes an error.

4.4.3 Other Primitive Types

- Undefined
  - A single value, *undefined*
  - However, *undefined* is not, itself, a reserved word.
  - The value of a variable that is declared but not assigned a value.
- Boolean
  - Two values: *true* and *false*

4.4.4 Declaring Variables

- Type: JavaScript is *dynamically typed*, that is, variables do not have declared types.
  - A variable can hold different types of values at different times during program execution.
- Declaring method: A variable is declared using the keyword *var or assigning it a value:*

```
var counter, index;
pi = 3.14159265,
quarterback = "Elway",
stop_flag = true;
```

4.4.5 Numeric Operators

- Standard arithmetic
  - + * - / %
- Increment and decrement
  - -- ++
  - Increment and decrement differ in effect when it is used before and after a variable.
    - Assume that a initially has the value 7,
    - (++a) * 3 has the value 24
    - (a++) * 3 has the value 21
    - a has the final value 8 in either case.

4.4.5 Numeric Operator

Operators	Associativity
++, --, unary -	Right
*, /, %	Left
+, -	Left

4.4 Numeric Operator

```
var a = 2,
b = 4,
c,
d;
c = 3 + a * b;
// * is first, so c is now 11 (not 24)
d = b / a / 2;
// / associates left, so d is now 1 (not 4)
```


4.4.6 The Math Object

- Provides a collection of properties and methods useful for number values
- This includes the trigonometric functions such as
  - *sin* and *cos*
  - *floor*
  - *round*
  - *max*
- When used, the methods must be qualified, as in Math.sin(x).

4.4.7 The Number Object

- Number object includes a collection of useful properties that has constant values.
- Properties
  - *MAX_VALUE*
  - *NaN*
  - *POSITIVE_INFINITY*
  - *NEGATIVE_INFINITY*
  - *PI*
- Operations resulting in errors return *NaN*
  - Use *isNaN*(a) to test if a is *NaN*
- *toString*() method converts a number to string

4.4.8 String Catenation

- The operator + denotes the string catenation operation.
- In many cases, other types are automatically converted to string.
- For example:

```
var first="a tiger";
var last=first+ "is big";
```

4.4.9 Implicit Type Conversion

- JavaScript attempts to convert values in order to be able to perform operations
  - "August " + 1977 causes the number to be converted to string and a concatenation is to be performed
  - 7 * "3" causes the string to be converted to a number and a multiplication is to be performed

4.4.10 Explicit Type Conversion

- Explicit conversion of string to number
  - Number(aString)
  - aString – 0
- *parseInt* and *parseFloat* try to find the integer or float from the beginning of a string. If failed, NaN is returned.
  - For example: parseFloat("10")

4.4.11 String Properties and Methods

- One property: *length*
  - Note to Java programmers, this is not a method!
  - For example

    var str="George";

    var len=str.length;
- Character positions in strings begin at index 0 (next slide)

4.4.11 String Methods

Method	Parameters	Result
charAt()	A number	Returns the character in the String object that is at the specified position
indexOf()	One-character string	Returns the position in the String object of the parameter
Substring()	Two numbers	Returns the substring of the String object from the first parameter position to the second
toLowerCase()	None	Converts any uppercase letters in the string to lowercase
toUpperCase()	None	Converts any lowercase letters in the string to uppercase

4.4.12 The `typeof` Operator

- Two syntactic forms
  - *typeof x*
  - *typeof(x)*
- Returns "*number*" or "*string*" or "*boolean*" for primitive types
- Returns "*object*" for an object or null

4.4.13 Assignment Statements

- Plain assignment indicated by =
- Compound assignment with
  - += -= /= *= %= …
- a += 7 means the same as a = a + 7

4.4.14 The Date Object

- A Date object represents a *time stamp*, a point in time
- A Date object is created with the *new* operator
  - var now= new Date();
  - This creates a Date object for the time at which it was created

4.5 Screen Output and Keyboard Input

- The Window object represents the window in which the script is being displayed.
- The Document object represents the document being displayed using DOM.
- Window has two properties
  - *window* refers to the Window object itself.
  - *document* refers to the Document object.
- Note: The Window object is the default object for JavaScript. So properties and methods may be used without the class name.

4.5 Screen Output and Keyboard Input

- The *write* method of the *document* object write its parameters to the browser window.
- The output is interpreted as HTML by the browser.
- If a line break is needed in the output, interpolate
 into the output.

4.5 The *alert* Method

- The *alert* method opens a dialog box with a message.
- The output of the alert is *not* XHTML, so use new lines \n rather than

```
alert("The sum is:" + sum +"\n");
```

4.5 The *confirm* Method

- The *confirm* method displays a message provided as a parameter.
  - The confirm dialog has two buttons: OK and Cancel.
- If the user presses OK, *true* is returned by the method.
- If the user presses Cancel, *false* is returned.

```
var question =
   confirm("Do you want to continue this download?");
```

4.5 The *prompt* Method

- This method displays its string argument in a dialog box.
  - A second argument provides a default content for the user entry area.
- The dialog box has an area for the user to enter text.
- The method returns a String with the text entered by the user.

```
name = prompt("What is your name?", "");
```

4.5 Example of Input and Output

- roots.html

4.6.1 Control Expressions

- A control expression has a Boolean value.
  - An expression with a non-Boolean value used in a control statement will have its value converted to Boolean automatically.
- Comparison operators
  - == != < <= > >=
  - === compares identity of values or objects
  - 3 == '3' is true due to automatic conversion
  - 3 === '3' is false
- Boolean operators
  - && || !

4.6.2 Selection Statements

- The *if-then* and *if-then-else* are similar to those that are in other programming languages, especially C/C++/Java.
- For example:

```
if(a>b)
 document.write("a is greater than b <br/>");
else
{
  a=b;
  document.write("a is less than or equal to b");
}
```


4.6.3 *switch* Statement Syntax

```
switch (expression)
{
case value_1:
        // statement(s)
case value_2:
        // statement(s)
...
default:
        // statement(s)
}
```


4.6.3 *switch* Statement Semantics

- The *expression* is evaluated. The value of the *expression* is compared to the value in each *case* in turn.
- If no *case* matches, execution begins at the *default* case.
- Otherwise, execution continues with the statement following the *case.*
- Execution continues until either the end of the *switch* is encountered or a *break* statement is met

4.6.4 Loop Statements

- Loop statements in JavaScript are similar to those in C/C++/Java.
- while

  while (*control expression*)
  statement or compound statement
- for

  for (*initial expression; control expression; increment expression*)
  statement or compound statement
- do/while

  do *statement or compound statement*
  while (*control expression*)

4.6.4 `while` Statement Semantics

- while (*control expression*)
  statement or compound statement
- The *control expression* is evaluated. If the *control expression* is *true*, then the statement is executed.
- These two steps are repeated until the *control expression* becomes *false.*
- At that point the while statement is finished

4.6 `for` Statement Semantics

for (initial expression; control expression; increment expression)
  statement or compound statement

- The *initial expression* is evaluated
- The *control expression* is evaluated
- If the *control expression* is *true*, the statement is executed
- Then the *increment expression* is evaluated
- The previous three steps are repeated as long as the *control expression* remains *true*
- When the *control expression* becomes *false*, the loop finishes.

4.6 `do/while` Statement Semantics

do statement or compound statement
  while (control expression)

- The *statement* is executed.
- The *control expression* is evaluated.
- If the *control expression* is *true*, the previous steps are repeated.
- This continues until the *control expression* becomes *false.*
- At that point, the statement execution is finished.

4.6.4 date.html Example

- Displays the components of a Date object
  - getDate(),getDay(),getMonth(),getTime()
- Uses Date objects to time a calculation
  - start and end object
- Illustrates an example of *loop*

Exercise

- Input: Three numbers, using prompt to get each
- Output: The largest of the three numbers
- Hint: Use the predefined function Math.max()

4.7 Object Creation and Modification (pp.158)

- The *new* expression is used to create an object.
  - This includes a call to a *constructor.*
  - The *constructor* creates and initializes all properties of the object.
- Properties of an object are accessed using a dot notation: *object.property*.
- Properties are not variables, so they are not declared.
- The number of properties of an object may vary dynamically in JavaScript.

4.7 Dynamic Properties

- Create my_car and add some properties

```
// Create an Object my_car
var my_car = new Object();
// Create and initialize the make property
my_car.make = "Ford";
// Create and initialize model
my_car.model = "Contour SVT";
```

- The *delete* operator can be used to delete a property from an object.
  - delete my_car.model

4.7 The for-in Loop

- Syntax

```
for (identifier in object)
    statement or compound statement
```

- The loop lets the *identifier* take on each property in turn in the object.
- Printing the properties in my_car:

```
for (var prop in my_car)
  document.write("Name: ", prop, "; Value: ",
    my_car[prop], "<br />");
```

- Result:

```
Name: make; Value: Ford
Name: model; Value: Contour SVT
```


4.8 Arrays

- Arrays are lists of elements indexed by a numerical value.
- Array indexes in JavaScript begin with 0.
- Arrays can be modified in size even after they have been created.

4.8.1 `Array` Object Creation

- Arrays can be created using the *new Array*() method.
  - *new Array* with one parameter creates an empty array of the specified number of elements.
    - new Array(10)
  - new Array with two or more parameters creates an array with the specified parameters as elements.
    - new Array(10, 20)
- Literal arrays can be specified using square brackets to include a list of elements.
  - var alist = [1, “ii”, “gamma”, “4”];
- Elements of an array do not have to be of the same type.

4.8.2 Characteristics of `Array` Objects

- The length of an array can be set at any time.

```
my_List.length=1000;
```

- Assignment to an index greater than or equal to the current length simply increases the length of the array.

```
my_List[1000]=5;
```

Only assigned elements of an array occupy space.

- Suppose an array were created using new Array(200)
- Suppose only elements 150 through 174 were assigned values
- Only the 25 assigned elements would be allocated storage, the other 175 would not be allocated storage

4.8.2 Example insert_names.html

- This example shows the dynamic nature of arrays in JavaScript.

4.8.3 `Array` Methods

- *join* converts all the elements into a string

```
var names = new Array["Mary", "Murray", "Murphy", "Max"];
var name_string = names.join(" : ");
```

- *reverse* does what your expect
- *sort* coerces the elements in the array to strings and sort them

```
names.sort();
```

- *concat* catenates its actual parameters to the array

```
var names = new Array["Mary", "Murray", "Murphy"];
var new_names = names.concat(“Moo”, “Meow”);
```


4.8.3 `Array` Methods

- *slice* fetches a subsequence of the array

```
var list = [2, 4, 6, 8, 10];
var list2 = list.slice(1,3);

var list = ["Bill", "Will", "Jill", "dill"];
var listette = list.slice(2);
```


4.8.3 Two-dimensional Arrays

- A two-dimensional array in JavaScript is an array of arrays.
  - This need not even be rectangular shaped: different rows could have different length.
- Example nested_arrays.html illustrates two-dimensional arrays.

4.9 Function Fundamentals

- Function definition syntax
  - A function definition consists of a header followed by a compound statement.
  - A function header:
    - *function function-name*(*optional-formal-parameters*)
- *return* statements
  - A *return* statement causes a function to cease execution and control to pass to the caller.
  - A *return* statement may include a value sent to caller.
  - A *return* statement without a value implicitly returns *undefined.*

4.9 Function Fundamentals

- Function call syntax
  - Function name followed by parentheses and any actual parameters.
  - Function call may be used as an expression or part of an expression.
- Functions must be defined before use in the page header.

4.9 Functions are Objects

- Functions are objects in JavaScript
- Functions may, therefore, be assigned to variables and to object properties.
- Example

```
function fun() {
document.write("This surely is fun! <br/>");}

ref_fun = fun; // Now, ref_fun refers to the fun object
fun(); // A call to fun
ref_fun(); // Also a call to fun
```

4.9.2 Local Variables

- "The *scope* of a variable is the range of statements over which it is visible"
- A variable not declared using *var* has *global scope*, visible throughout the page, even if implicitly used inside a function definition
- A variable declared with *var* outside a function definition has *global scope*
- A variable declared with *var* inside a function definition has *local scope*, visible only inside the function definition
  - If a global variable has the same name, it is hidden inside the function definition

4.9.3 Parameters

- Parameters named in a function header are called *formal parameters*
- Parameters used in a function call are called *actual parameters*

4.9.3 Parameters

- JavaScript checks *neither* the type *nor* number of parameters in a function call
  - Formal parameters have no type specified
  - Extra actual parameters are ignored
  - If there are fewer actual parameters than formal parameters, the extra formal parameters remain undefined
- *This is typical of scripting languages*
- A property array named *arguments* holds all of the actual parameters, whether or not there are more of them than there are formal parameters
- Example para.html

4.11 Constructors

- Constructors are functions that create and initialize properties for new objects
- A constructor uses the keyword *this* in the body to reference the object being initialized
- Object methods are properties that refer to functions
  - A function to be used as a method may use the keyword *this* to refer to the object for which it is acting
- Example car_constructor.html

Homework

- pp.187
- 2, 11, 13

Part 5

JavaScript and
HTML Documents

Contents

5.1 JavaScript Execution Environment

- JavaScript is executing in a browser
- The *Window* object represents the window displaying a document
  - All properties are visible to all scripts
  - There can be more than one Window object
- The *Document* object represents the document displayed
  - It is refererenced by the *document* property of Window

5.1 JavaScript Execution Environment

- Example first.html
- *window* and *document* property must be lowercase
- *window* has the global scope, and you need not explicitly spell out the *window* property
- *write* is a method of *document*.

Contents

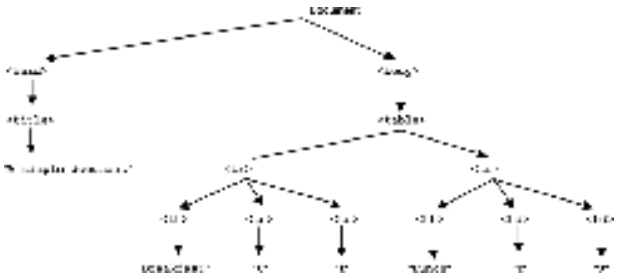
5.2 Document Object Model

- DOM specifications describe an abstract model of a document
  - *Interfaces* describe *methods* and *properties*
  - The interfaces describe a tree structure
  - Different languages will *bind* the interfaces to specific implementations
    - In JavaScript, data are represented as *properties* and operations as *methods*
- Temrs: DOM

5.2 Document Object Model

- The HTML document on page 192 is shown as a conceptual tree

Figure 5.1 The DOM structure example



5.2 Document Object Model

- Nodes of the tree will be JavaScript objects
- *Attributes* of elements become named *properties* of objects
  - <input type=“text” name=“address”>
  - The object representing this node will have two properties
    - *type* property will have value “text”
    - *name* property will have value “address”

Contents

5.3 Element Access in JavaScript

- *Elements* in HTML document correspond to *objects* in JavaScript
- Objects can be addressed in four ways:
  - *forms* and elements array of the Document object
    - Individual elements are specified by index
  - Using the *name* attributes for the form and its elements
    - *name* attribute is required
  - Using *getElementById* with *id* attributes
    - *id* attribute value must be unique for an element
  - Implicit arrays

5.3 Using forms array[1]

- Consider this simple form:

```
<form action = "">
        <input type = "button"  name = "pushMe">
 </form>
```

- The input element can be referenced as
  document.forms[0].elements[0]
- The drawback is that the addressing can be changed due to the addition or deletion of elements.

5.3 Using name Attributes[2]

- All elements from the *reference element* up to, but not including, the *body* must have a name attribute

- Example

```
<form name = "myForm"  action = "">
  <input type = "button"  name = "pushMe">
</form>
```

- Referencing the input

  document.myForm.pushMe

5.3 Using id Attribute[3]

- Set the *id* attribute of the input element

```
<form action = "">
  <input type="button"  id="turnItOn">
</form>
```

- Then use *getElementById* function

  document.getElementById("turnItOn")

- Example dom.html

5.3 Using implicit arrays[4]

- For *checkbox* and *radio button* group, each group has an array, which has the same name as the group name.
- The array is the property of the form.

```
<form id = "vehicleGroup">
  <input type = "checkbox"  name = "vehicles">
      value = "car"  />  Car
  <input type = "checkbox"  name = "vehicles">
      value = "truck"  />  Truck
  <input type = "checkbox"  name = "vehicles">
      value = "bike"  />  Bike
</form>
```


5.3 Using implicit arrays[4]

- The following code is to detect how many checkboxes are checked.

```
var numChecked = 0;
var dom = document.getElementById("vehicleGroup");
for (index = 0; index < dom.vehicles.length; index+
  if (dom.vehicles[index].checked)
    numChenked++;
```


Exercise

- Suppose an HTML has a text box and a button.
- When you have input some words in the text box, show the message with the input words once the button is pressed.

```
<form>
  <input type="text" id="mytext">
  <input type="button" id="mybutton" onclick="foo()">
</form>
```

- *Hint:*

  using *forms*(*elements*)array or *getElementById* function; using the value property to get the input words.

Contents

5.4 Events and Event Handling

- *Event-driven programming* is a style of programming in which pieces of code, *event handlers*, are written to be activated when certain *events* occur
- *Events* represent activity in the environment including, especially, user actions such as moving the mouse or typing on the keyboard, etc.
- An *event handler* is a program segment designed to execute when a certain event occurs

5.4 Events and Event Handling

- *Registration* is the activity of connecting the event handler to a type of event

  (1) In HTML, assign an event *attribute* an event handler

  (2) In JavaScript, assign a DOM node an event handler
  The example will be given later.

5.4 Events, Attributes and Tags

- Particular *events* are associated to certain *attributes* of HTML tags
- The attribute for one kind of event may appear on different tags allowing the program to react to events affecting different components

5.4 Events, Attributes and Tags (pp.197)

Event	Tag Attribute
`blur`	`onblur`
`change`	`onchange`
`click`	`onclick`
`focus`	`onfocus`
`load`	`onload`
`mousedown`	`onmousedown`
`mousemove`	`onmousemove`
`mouseout`	`onmouseout`
`mouseover`	`onmouseover`
`mouseup`	`onmouseup`
`select`	`onselect`
`submit`	`onsubmit`
`unload`	`onunload`

Table 5.2 Event attributes and their tags (pp.198)

5.4 Registering a Handler

- Assigning the event handler script to an event tag attribute:

```
<input type="button" name="myButton"
    onclick=
    "alert('You clicked the button!')"/>
```

- A function call can be used if the handler is longer than a single statement

```
<input type="button" name="myButton"
    onclick="myHandler()"/>
```

5.5 Handling Events from Body Elements

- See the load.html example(pp.200)
- This example illustrates when the page is loaded into main memory.
- The unload event is probably more useful

Figure 5.2 Display of load.html

5.6 Handling Events from Button Elements

An event can be registered for this tag in two ways

<input type="button" name="freeOffer" id="freeButton"/>

(1)Using an event attribute in HTML

<input type="button" name="freeOffer"
id="freeButton" onclick="freebuttonHandler()"/>

(2)Assigning function names to a property of the element node in JavaScript

document.getElementById("freeButton").onclick =
freeButtonHandler;

- Note that the function name, a reference to the function, is assigned

5.6 Checkboxes and Radio Buttons

- The following examples show two different methods for registration.
- Example *radio_click.html* (pp.202)illustrates a script that displays an alert when a radio button is clicked
  - Note: A parameter is passed to the handler function.
- In example radio_*click2.html* (pp.205), a reference to the handler function is assigned to the onclick property of each element node in JavaScript.
  - Note: no parameters are passed to the function when called by the Javascript. The handler code must identify the element that caused the call.

Exercise

- Develop an HTML document that has checkboxes for apple(59 cents each), orange(49 cents each), and banana(39 cents each), along with a ***Submit*** button.
- Each of the checkboxes should have its own *onclick* event handler. These handler add the corresponding cost of each fruit to a total cost.
- When the submit button is pressed, the message "Your total cost is $xxx" is shown. (***hint***: using ***alert***)

5.6 Comparing Registration Methods

- Assigning to an attribute of HTML element is more flexible, allowing passing parameters without having to create an anonymous function
- Assigning to a node property helps separate HTML and the JavaScript code.

5.7 Handling Events from Text Box and Password Elements

- Text boxes correspond to four different events: *blur*, *focus*, *change* and select.
- By manipulating the *focus* event, the user is prevented from changing the amount in a text input box.
  - Example *nochange.html* illustrates 'blurring' a field whenever it gains focus.

5.7 Validating Form Input

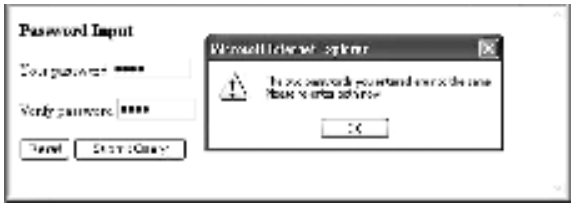
- Checking the format and completeness of input is a common application of JavaScript.
- *Advantages:*
  - Validating data using JavaScript provides quicker interaction for the user.
  - In contrast, validity checking on the server requires a round-trip for the server to check the data and then to respond with an appropriate error page.

5.7 Validating Form Input

- Example pswd_chk.html illustrates validity checking

Figure 5.5 Display of `pswd_chk.html` after it has been filled out

Figure 5.6 Display of `pswd_chk.html` after *Submit Query* has been clicked



Contents

5.9 The `navigator` Object

- Properties of the *navigator* object allow the script to determine characteristics of the browser in which the script is executing.
- The *appName* property gives the name of the browser.
- The *appVersion* gives the browser version.
- Example navigate.html

5.9 Output From `navigate.html`

- Note that the browser is actually FireFox and the version is 2.0.0.4

Summay

- 1 Environment
- 2 DOM model
- 3 How to access elements
- 4 Event handling
- 5 navigator object

Homework

- Develop an HTML document that collects the following information from the user:last name, first name,middle intial, age(restricted to be greater than 17), and weight (restricted to the range of 80-300). You must have event handler for the form elements.
- Message in *alert* window must be produced when errors are detected.

Part 6

Introduction to PHP

Contents

12.1 Origin and Uses of PHP

- PHP is a server-side scripting language, embedded in HTML pages.
- PHP has a good support for *form* processing.
- PHP can interact with a wide variety of databases.
- PHP is now developed, distributed and supported as an open source product.

12.2 Overview of PHP

- Woring principle
  - When a PHP document (with suffix .php) is requested of a server, the server sends the document first to a PHP processor.
  - The result of the processing is the response to the request.

12.2 Overview of PHP

- Two modes of operation
  - *Copy mode* in which plain HTML is copied to the output
  - *Interpret mode* in which PHP code is interpreted and the output from that code is sent to the output
  - The client *never* sees PHP code, only the output produced by the code.

12.2 Overview of PHP

- PHP has typical scripting language characteristics.
  - Dynamic typing, untyped variables
  - Associative arrays
  - Pattern matching
  - Extensive libraries

12.3 General Syntactic Characteristics

- Code inclusion
- Variable name
- Comment style
- Statements

11.3 General Syntactic Characteristics

- Code inclusion style
  - PHP code is contained between the tags <*?php* and ?>
  - PHP code can also be included with the PHP include
    include("table2.inc");
    - When a file is included, the PHP interpreter reverts to copy mode
    - Notice: Code in an included file must be in <*?php* and ?> tags

12.3 General Syntactic Characteristics

- Variable name
  a) All variable names in PHP begin with *$* and continue as usual for variables.
  b) Variable names are case sensitive.
  - *However,* keywords and function names are *not* case sensitive.
- Comment styles
  - *One line comments* can begin with *//* and continue to the end of the line.
  - *Multi-line comments* can begin with */** and end with **/*.

12.3 General Syntactic Characteristics

- Statements
  - PHP statements are terminated with semicolons.
  - Braces are used to create compound statements.
  - Variables cannot be defined in a compound statement unless it is the body of a function.

12.4 Primitives, Operations, Expressions

- Four scalar types: boolean, integer, double, string.
- Two compound types: array, object.
- Two special types: resource and NULL
- Note: object and special types are not covered in this text

11.4 Primitives, Operations, Expressions

- Contents
  - Variables
  - Integer type
  - Double type
  - String type
  - Boolean type
  - Arithmetic operations and expressions
  - Sring operations
  - Scalar type conversions

12.4.1 Variables

- Variables in PHP have no type declarations.
- The type of a variable is set every time it is assigned a value.

```
<?php
$txt = "Hello World!";
$number = 16;
?>
```


12.4.1 Variables

- A variable that has not been assigned a value is called an *unbounded variable,* and has the value *NULL*, which is the only value of the *NULL* type.
- If an *unbounded variable* is used in an expression, *NULL* is coerced to a value according to the context.
  a) NULL is coerced to 0 if a number is needed,
  b) NULL is coerced to the empty string if a string is needed.

12.4.1 Variables

- We can use the *IsSet*() function to test whether a variable has a value, which returns *TRUE* or *FALSE*.

  Example: IsSet($fruit)

12.4.(2-3) Integer Type and Double Type

- PHP distinguishes between integer and floating point numeric types.
- *Integer* is equivalent to long in C, that is, usually 32 bits
- *Double* type values are stored internally as floating point values.
  - Double literals can include a decimal point, an exponent, or both.
  - For example, 0.345E-3.

12.4.4 String Type

- String literals are enclosed in single or double quotes.
  - Single quoted strings have neither escape sequence interpretation nor variable interpolation

    Example: *$sum=5;*

    'The sum is: $sum'
  - Double quoted strings have escape sequences interpreted and variables interpolated

    Example: *"The sum is: $sum"*

    • A literal *$* sign in a double quoted string must be escaped with a backslash \
  - Characters in PHP are one byte string.

12.4.5 Boolean Type

- The boolean type has two values :*TRUE* and *FALSE.*
- Non-boolean type values are coerced as needed by the context.
  a) For numeric values, it evaluates to *FALSE* if it is zero; otherwise, *TRUE.*
  b) For strings, the empty string and the literal string "0" all count as *FALSE.*
  c) *NULL* counts as *FALSE.*

12.4.6 Arithmetic Operators and Expressions

- PHP supports the usual collection of arithmetic operators:

  +, -, *, /, %, ++, --

- A variety of numeric functions is available: `floor, ceil, round, rand, abs, min, max` (pp.481, table 12.2).

12.4.7 String Operations

- *String catenation* is indicated with a period.

  Example strTest.php

- *Characters* are accessed in a string with a subscript enclosed in *curly braces.*

  Example strTest.php

12.4.7 String Operations

- Many useful string functions are provided
  - `strlen` gives the length of a string.
  - `strcmp` compares two strings and return a number.
  - *chop* removes whitespace from the end of a string.
  - See table 12.3 (pp.482)

12.4.8 Scalar Type Conversions

- Types can be determined in two different ways.

  a) use the `gettype` function, which takes a variable as its parameter and return a string of the type name

  b) use `is_int (is_double, is_bool, is_string)` function

12.4.8 Scalar Type Conversions

- Implicit type conversion
- Explicit type conversion

12.4.8 Scalar Type Conversions

- Implicit type conversions are demanded by the context in which an expression appears.

  For example:

  - A string which has only a sign followed by digits is converted to an integer if a numeric value is required.
  - A string which is a valid double literal (including either a period or e or E) converts to a double if a numeric value is required.

12.4.8 Scalar Type Conversions

- Explicit type conversions can be forced in three ways. Suppose that $sum=4.777

  a) `(int)$sum` in the C style

  b) Using several conversion functions such as `intval($sum)` (*intval, doubleval or strval*)

  c) `settype($sum, "integer")`

12.4.9 Assignment Operators

- PHP has the same set of assignment operators as C
- PHP also includes the compound assignments such `+=` and `-=`

12.5 Output

- All ouput of PHP must be in the form of HTML
- Three ways to create output: *echo*, *print* and *printf*
- Each of the functions can be called with or without parentheses around its parameters.
  a) If parentheses are included, only a single string parameter is acceptable.
  b) Otherwise, any number of parameters can appear, which are connected by the *comma* .

12.5 Output

- *echo* function is *most commonly* used to produce the output.

  Example:

  echo "Apples are red
","Oranges are orange";

  Note:

  echo function does not return a value.

12.5 Output

- The *`print`* function is used to send data to output
  - *`print`* takes string parameters, PHP coerces it as necessary.

    Example:

    print("You are welcome");
    print "You are welcome";
    print(47);

    Note:

    a) It return a value (1 if succeeded, 0 if it failed) to indicate whether the operation is completed.
    b) It only accepts one parameter.

12.5 Output

- The *printf* function is also available
  - Format:

    printf(*fat,arg1…*)

    a) The first argument *fat* is a format code with interspersed format codes.
    b) The remaining parameters (such as arg1) are to be formatted.

12.5 Output

- A format code begins with *%* followed by a *field width* and a *type specifier*
  - *Field width* is a single integer to specify the number of characters (minimum) used to display the value or two integers separated by a period to indicate *field width* and *decimal places* for double values.
  - *Type specifiers* are *s* for string, *d* for integer and *f* for double.

12.5 Output

- Example printf.php

  printf("%s world. Day number %5d",$str,$number);

  Note:

  Displays *$number* as an integer and *$str* as a string

12.5 Output

- The example `today.php` uses the *date* function to dynamically generate a page with the current date.
- *date* function, whose first parameter is a string that specifies the part of the date you want to see.
  - *l* requires the day of the week
  - *F* requires the month
  - *j* represents the day of the month, and *S* suffix next to the *j* gets the correct suffix of the day (*st,nd,rd* or *th*)

Class exercise

- Write a php document with the following features
  - There are three variables *a*, *b* and *c*. The value of *a* is "*You are* ". The value of *b* is "*welcome!*". The value of *c* is a double *45.6*.
  - The concatenation of *a* and *b* is displayed with *echo* function.
  - The value of *c* is displayed with *print* function.
  - Hint:
    - Variable name
    - *echo* without parentheses
    - Switch to next line

12.6 Control Statements

- Relational operators
- Boolean operators
- Selection statements
- Loop statements

12.6.1 Relational Operators

- PHP has the usual comparison operators (>, <, <=, >=, ==, and !=) of JavaScript.
- PHP also has the identity operator ===
  - This operator does not force coercion.
  - It produces *TRUE* only if both operands are of the same type and have the same value.
  - Its opposite operator is !==

12.6.2 Boolean Operators

- PHP supports &&, || and ! as in C/C++/Java
- The higher precedence version `and` and `or` are provided
- The `xor` operator is also provided
- Note: All boolean operators are evaluated as short-circuit operators

12.6.3 Selection Statements

- PHP provides an *if* with almost the same syntax as C/C++/Java
  - The only difference is the *elseif*
- The *switch* statement is provided with syntax and semantics similar to C/C++/Java
  - The *case* expressions are coerced before comparing with the control expression
  - `break` is necessary to prevent execution from flowing from one case to the next
  - *default* case can be included

12.6 Loop Statements

- PHP provides the *while, for* and *do-while* as in JavaScript
- The *for* loop is illustrated in the example `powers.php`
- This example also illustrates a number of mathematical functions such as *sqrt*(),*pow*() available in PHP

Class exercise

- Write a php script to disply the weather of fives days from Monday to Friday, using *while* statement
- The weather informtion is a centigrade degree ranging between (25-40). Hint: using rand(a,b) pp.481
- Once the centigrade degree is greater than 35, show a message “It’s very hot”

12.7 Arrays

- Arrays in PHP combine the characteristics of regular arrays and hashes
  a) An array can have elements indexed numerically. These elements are maintained in order.
  b) An array, even the same array, can have elements indexed by string. These are not maintained in any particular order.
- The elements of an array are, conceptually, key/value pairs

12.7 Arrays

- 12.7.1 Array creation
- 12.7.2 Accessing array elements
- 12.7.3 Dealing with arrays
- 12.7.4 Sequential access to array elements
- 12.7.5 Sorting arrays

12.7.1 Array Creation

- Two ways of creating an array

a) Assigning a value to an element of an array

```
$list[0]=17;

$list[]=18
```


12.7.1 Array Creation

b) Using *array* construct

- Create a numerically indexed array
  - $list=array(23, ‘xiv’, “bob”, 777);
  - It creates a traditonal array of four elements, with the keys 0,1,2,3
- Create an array with string indexes
  - $ages=array(“Joe” => 42, “Mary” =>41, “Bif”=>17);

12.7.2 Accessing Array Elements

- Array elements are accessed by using a subscript in square brackets
- The subscripts can be a string or an integer
  - Example

```
print(“Mary is $ages[‘Mary’] years old <br/>”);

print(“Mary is $ages[1] years old <br/>”);
```


12.7.2 Accessing Array Elements

- Multipe elements of an array can be assigned to scalar varaibles in one statement, using the *list* construct
  - Example:
    - $trees=array(“oak”,”pine”,”binary”);
    - list($a,$b,$c)=$trees;

12.7.2 Accessing Array Elements

- The `array_keys` function returns a list of the keys of an array.
- The `array_values` returns a list of values in an array.
  - Example:

```
$highs=array(“Mon”=>74,”Tue”=>70,”Wed”=>67);
$days=array_keys($highs);
$nums=array_values($highs);
```

  - Now the value of *$days* is (“Mon”,”Tue”,”Wed”), and the value of *$nums* is (74,70,67). In both cases, the *keys* are (0,1,2).

12.7.3 Functions with Arrays

- The *unset* function can be used to remove an array or an element of an array
  - Example:
    - $list=array(2,4,6,8);
    - unset($list[2]);
- `is_array(arg)` determines if its argument arg is an array.
- The *in_`array(expr,arg)`* function returns TRUE if the expression *expr* is in the array *arg*

12.7.3 Functions with Arrays

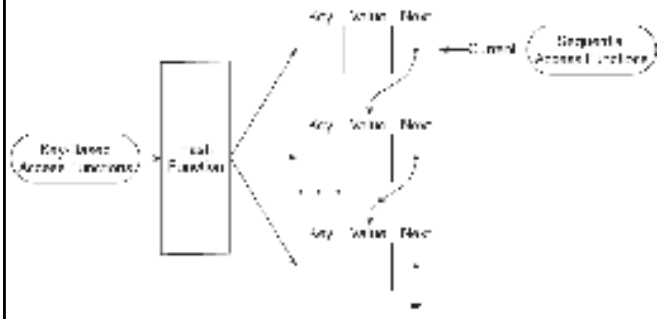
- *implode* converts an array of strings to a single string, separating the parts with a specified string
  - Example:

```
$words=array("Are","you","lonesome","today");
$str=implode(" ",$words);
```

- *explode* converts a string into a list of strings by separating the string at specified characters
  - Example:

```
$str="April  in Paris, Texas is nice";
$words=explode(" ",$str);
```


12.7.4 Sequential Access to Array Elements



12.7.4 Sequential Access to Array Elements

- PHP maintains a marker in each array, called the "*current*" pointer
  - The pointer starts at the first element when the array is created
  - The element being referenced by the pointer can be obtained by the *current* function.
  - Example:

```
$cities=array("London","New York");
$city=current($cities);
print("the first city is $city");
```


12.7.4 Sequential Access to Array Elements

- The *next* function moves the pointer to the next element and returns the value (no key) there
- Example:

```
$city=current($cities);
print("$city<br/>");
while($city=next($cities)) print("$city <br/>");
```


12.7.4 Sequential Access to Array Elements

- The *each* function returns the key/value pair and moves the pointer to the next element
  - The key and value can be accessed using the keys "*key*" and "*value*" on the key/value pair
  - Example:

```
$sal=array("Mike"=>4000,"Tom"=>3000,"Robert"=
  >2000);
while($emp=each($sal))
{   $name=$emp["key"];
    $salary=$emp["value"];
    print("The salary of $name is $salary <br/>");}
```


12.7.4 Sequential Access to Array Elements

- Both functions return false if no more elements are available
- *prev* moves the pointer back towards the beginning of the array by one element
- *reset* moves the pointer to the beginning of the array

12.7.4 Arrays as Stacks

- PHP provides the *array_push* function that appends its arguments to a given array
  - It takes an array as its first parameter
  - After the first parameter, there can be any number of additional parameters
- The function *array_pop* removes the last element of a given array and returns it
  - It takes an array as the single parameter

12.7.4 Iterating Through an Array

- The *foreach* statement has two forms for iterating through an array

  foreach (*array* as *scalar_variable*) *loop body*

  foreach (*array* as *key* => *value*) *loop body*

12.7.4 Iterating Through an Array

- The first version assigns each value in the array to the scalar_variable in turn

```
foreach ($list as $temp)
        print("$temp <br />");
```


12.7.4 Iterating Through an Array

- The second version assigns each key to key and the associated value to value in turn
- In the following example, each day and temperature is printed

```
$lows = array("Mon" => 23, "Tue" =>
 18, "Wed" => 27);
foreach ($lows as $day => $temp)
print("The low temperature on $day
 was $temp <br />");
```


12.7.5 Sorting Arrays

- The *sort()* function sorts the values in an array and makes a numerically subscripted array from the sorted list.
- The function *asort()* sorts the values in an array but keeps the original key/value association.
- The function *ksort()* is similar to asort() but sorts by keys
- The example `sorting.php` illustrates the various sort functions.

12.8 Functions

- Function syntax

  function *name*([*parameters*]) {

  ...

  }
  - The *parameters* are optional, but not the parentheses.
  - Function names are not case sensitive.
  - A return statement causes the function to immediately terminate and return a value, if any.

12.8.2 Parameters

- A *formal parameter*, specified in a function declaration, is simply a variable name.
- If more *actual parameters* are supplied in a call than there are formal parameters, the extra values are ignored.
- If more *formal parameters* are specified than there are *actual parameters* in a call, the extra formal parameters receive no value.

12.8.2 Parameters

- PHP defaults to *pass by value*
  - Putting an ampersand in front of a formal parameter specifies that pass-by-reference.
  - An ampersand can also be added to the actual parameter (which must be a variable name).
  - Example addone.php

12.8.3 The Scope of Variables

- A variable defined in a function is, by default, local to the function.
- A global variable of the same name is not visible in the function
  - Example summer.php
- Declaring a variable in a function with the *global* keyword means that the function uses the global variable of that name which is defined outside the function.
  - Example big_sum.php

12.8.4 Lifetime of Variables

- The default lifetime of a local variable is from the time the function begins to execute to the time the function returns.
- Declaring a variable with the *static* keyword means that the lifetime is from the first use of the variable to the end of the execution of the entire script.
- In this way a function can retain some ‘history’.
  - Example do_it.php

12.10 Form Handling

- The values from forms can be accessed in PHP using the $_POST and $_GET arrays
  - If a form has a text box named “phone” and the form method is *POST*, the value of the element is available in the PHP script as follows:
    - $_POST[“phone”]
- The popcorn3.html and popcorn3.php implement the popcorn ordering using PHP.
  - The printf ()function is used to get two decimal places printed for currency values.

12.11 Opening and Closing Files

- The PHP function `fopen()` is used to create a file handle for accessing a file given by name(the first argument).
  - The second argument of fopen() gives the mode of access
  - The `fopen()` function returns a file handle(a variable).
  - Every open file has a current pointer indicating a point in the file. The input and output operations occur at the current pointer position.

12.11 Opening and Closing Files

- Before calling fopen(), we often use file_exists() function tests if a file, given by name, exists.
- The function fclose() closes a file handle.

Table 12.4 File Use Indicators

Mode	Description
"r"	Read only. The file pointer is initialized to the beginning of the file.
"r+"	Read and write an existing file. The file pointer is initialized to the beginning of the file;
"w"	Write only. Initializes the file pointer to the beginning of the file; creates the file if it does not exist.
"w+"	Read and write. Initializes the file pointer to the beginning of the file; creates the file if it does not exist. Always initializes the file pointer to the beginning of the file before the first write, destroying any existing data.
"a"	Write only. If the file exists, initializes the file pointer to the end of the file; if the file does not exist, creates it and initializes the file pointer to its beginning.
"a+"	Read and write a file, creating the file if necessary; new data is written to the end of the existing data

12.11.2 Reading from a File

- The fread() function reads a given number of bytes from a file given by a file handle, and returns a string of what was read.
  - The entire file can be read by using the filesize(file_name) function to determine the number of bytes in the file.
- The fgets(file_var,len) can read a single line from a file.
  - The first parameter is the file variable, and the second argument is the number of the characters to be read.
  - It reads until it finds a newline character, encounters the end-of-file, or one less than the number.

12.11.2 Reading from a File

- The fgetc(file_var) function reads a single character from a file, whose only argument is the file variable.
- The feof(file_var) function, whose only argument is the file variable, returns true if the last character read was the end of file marker.

12.11.3 Writing to a File

- If a file handle is open for writing or appending, then the `fwrite(file_var,data)` function can be used to write bytes to the file
  - It takes two parameters: a file variable and the string to be written to the file.
  - The return value is the number of bytes written.

12.11.3 Writing to a File

- Example: file.php
  - Exercises 10 and 11 (pp.520)
    - End

Part 7

Database Access Through the Web

Contents

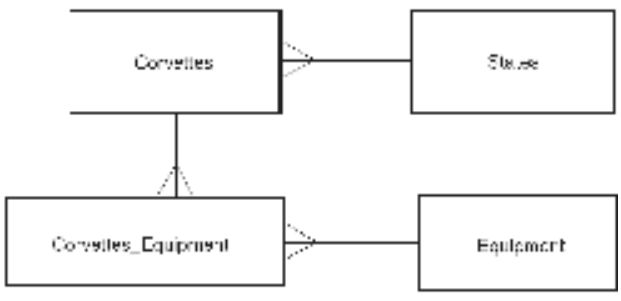
14.1 Relational Databases

- A database stores data in a way allowing
  - Ø Efficient changes(additions, modifications and deletions),
  - Ø Efficient searching.

14.1 Relational Databases

- The relational model is currently the most popular model.
  - Ø Data is stored in many tables.
  - Ø Table columns are named.
  - Ø Each row of a table contains a value for each column, though some values may be missing.
  - Ø Rows are referred to as entities.
  - Ø The *primary key* is one or more columns in a table whose value(s) uniquely identify each row.

14.1 Logical Data Model for Corvettes DB



14.1 Relational Databases

- Example, *Corvettes* table
  - Ø Primary key is an index number.
  - Ø Each row represents a different vehicle.
  - Ø Columns are important characteristics of the vehicles.
  - Ø Figure 14.3

14.1 Relational Databases

- Each state can, potentially, be associated with several cars.
  - Ø Each state could have important data, besides the name.
  - Ø A separate *States* table is created with a primary key.
  - Ø Each entity in the *Corvettes* table refers to the *state* .
  - Ø That way, changes in information about a state would not have to be repeated on each line of the Corvettes table.

14.1 Relational Databases

- Each type of equipment could appear in many cars, each car could have many types of equipment.
  - Ø A table *Equipment* describing equipment is set up.
  - Ø A table *Corvettes_Equipment* giving the *Corvettes* and *Equpment* relation is set up.
    - ü This is specified by pairs of ids: *Corvette-id* and *Equipment-id.*

14.2 Structured Query Language

- SQL is a standardized language for manipulating and querying relational databases.
- Although relational databases support SQL, there may be some minor or significant differences in the implementations for different DBMS such as Oracle, SQL Server, Sybase, DB2.
- SQL is non-procedure language.

14.2 Structured Query Language

- SQL reserved words are not case sensitive
  - Ø However, some systems may treat names such as column names as case sensitive.
- Single quotes ‘ are used for literal strings.

14.2.1 The *CREATE TABLE* Command

- Create a table with specified columns, each column having a specified type of data and satisfying certain constraints.
- Syntax

  CREATE TABLE *table_name*(
  column_name_1 data_type constraints,
  ...
  column_name_n data_type constraints);
- Common types: integer, real, double, char(length)

14.2.1 Create Table Constraints

- The constraint *not null* causes an error to be raised if a row is inserted in which the corresponding column does not have a value.
- The *primary key* constraint causes an error to be raised if a row is inserted in which the corresponding column has a value that equals the value in another row.
  - Ø This can be applied to a group of several columns if the primary key is multi-column.
- The *foreign key* constraints use for the reference to the column in another table.

14.2.1 The CREATE TABLE Example

- create database hjydb;
- use hjydb;
- create table states(
  state_id integer *not null primary key*,
  state char(20)
  );

14.2.1 The CREATE TABLE Example

- create table v(
  v_id integer(4) *not null primary key*,
  body_style char(20),
  miles double,
  year integer,
  state integer,
  foreign key(state)
  references states(state_id));

14.2.2 The *INSERT* Command

- Inserts a new row into a table
- Syntax

  INSERT INTO *table_name*
  (*column_name_1, ..., column_name_n*)
  VALUES (*value_1, ..., value_n*);

  - Ø The values provided will be placed into corresponding columns.
  - Ø Columns not named will receive no value.
    - ü This will cause an error if the column was created with a *not null* constraint

14.2.2 The INSERT Command Example

- insert into states(state_id, state) values(1,'Florida');
- insert into v values(1,'couple',32.3,1976,1);

14.2.3 The *SELECT* Command

- Used to query databases
- The command returns a result, a virtual table.
- *SELECT column-names FROM table-names* [*WHERE condition*];
  - Ø The resultant table has columns as named
  - Ø Rows are derived from the table named
  - Ø The *WHERE* clause is optional
  - Ø The *WHERE* clause specifies constraints on the rows being selected.
  - Ø If * is used for the column names, all columns are selected.

14.2.3 The *SELECT* Command Example

- select v_id, body_style from v;
- select body_style, miles from v where v_id=1;

14.2.3 Joins

- Task: list corvettes that have CD players
- This involves three tables: *Corvettes, Equipment, Corvettes_Equipment*
- A virtual table is constructed with combinations of rows from the two tables Corvettes and Equipment: a *join* of the three tables
- The *WHERE* clause specifies which rows of the join are to be retained in the result

14.2.3 A Query Using a Join

```
SELECT Corvettes.Vette_id,
     Corvettes.Body_style,
     Corvettes.Miles, Corvettes.Year,
     Corvettes.State,
     Equipment.Equip
FROM Corvettes, Equipment
WHERE
     Corvettes.Vette_id =
          Corvettes_Equipment.Vette_id
     AND Corvettes_Equipment.Equip =
          Equipment.Equip_id
     AND Equipment.Equip = 'CD';
```


14.2.4 The *UPDATE* Command

- Changes values in an existing row or some rows
- Syntax

```
UPDATE table_name
SET column_name_1 = value_1,
 ...
 column_name_n = value_n
WHERE column_name = value
```

- The *WHERE* clause identifies the rows to be updated.

14.2.4 The *UPDATE* Command

- update states set state='New York' where state_id=1;

14.2.5 The *DELETE* Command

- Removes one or more rows
- Syntax

```
DELETE FROM table_name
WHERE column_name = value;
```

- The *WHERE* clause determines which rows are deleted

14.2.5 The *DELETE* Command

- Delete from states where state_id=1;

14.2.6 The *DROP* Command

- Remove a table or database from the system
  - ØA database system usually has several databases within it, and collections of tables in each database.
- Syntax

  DROP (TABLE|DATABASE)[*IF EXISTS*] *name*;

  - ØThe *IF EXISTS* clause may be included to prevent an error indication if the table or database doesn't exist.

14.3 Client/Server Database Architecture

- Two-tier architecture
  - ØClient connects to the database to get information.
  - ØServer or client performs computations and user interactions.
- Problems with two-tier
  - ØServers getting smaller so client software getting more complex
  - ØKeeping clients up to date becomes difficult.

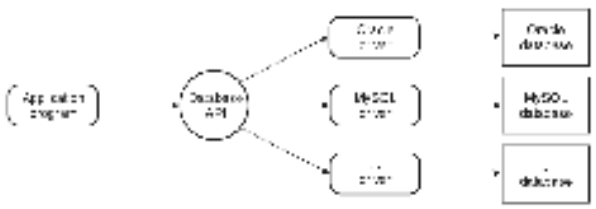
14.3 Client/Server Database Architecture

- Three-tier architecture
  - ØWeb server with applications sits between a browser and the database system.
  - ØThe web server accesses the database, carries out computations and deals with user interactions.

14.3.3 Microsoft Access Architectrue

- Open Database Connectivity (ODBC)
- An application programming interface (API) provides services to interact with a database
- One service is to submit SQL to the database system and to return results
- An ODBC driver manager, on the client, chooses the proper interface for a particular database

14.3 Database Access Architecture



14.3.5 PHP and Database Access

- There are modules available in PHP to access numerous different database systems

14.4 The MySQL Database System

- The first step to use mysql is logging in to the MySQL system, which is done with the following command.
  mysql [-h host] [-u username] [database_name] [-p]
- The second step is to create a new database.
  create database cars;
- The third step is to specify which database to be used.
  use cars;

14.4 The MySQL Database System

- Show how many databases there are on the system
- Syntax:
  show databases;
- Show how many table there are in a specific database
- Syntax:
  show tables;
- Show the schema of table
- Syntax
  desc v;

14.6 Database access with PHP and MySQL

- PHP access to Mysql is done with two documents.
  - Ø One is to collect user request (.html).
  - Ø The other is used to process the request and return the HTML document (.php).

14.6.2 Connecting to MySQL

- The *mysql_connect()* function
  - Ø First parameter is MySQL server host
  - Ø Second parameter is the MySQL username
  - Ø Third parameter is the password
  - Ø Returns false if it fails
- The *mysql_close()* function
- Example: connect.php

14.6.2 Connecting to MySQL

- Selecting a database with *mysql_select_db()*
- For example:
  mysql_select_db('cars')

14.6.3 Requesting MySQL Operations

- The mysql_query function
  - Ø Takes a string parameter of SQL query
  - Ø Returns a result object
- Example:
  $query="select * from v";
  $result=mysql_query($query);

14.6.3 Requesting MySQL Operations

- Functions that apply to the result object
  - `mysql_num_rows` returns number of rows in the result
  - `mysql_num_fields` returns the number of fields (columns) in the result
  - `mysql_fetch_array` returns an array with the next row of results

14.6.3 Requesting MySQL Operations

- Example fetch.php

```
$num_rows=mysql_num_rows($result);
for ($row_num = 0; $row_num < $num_rows; $row_num++)
{
    $row=mysql_fetch_array($result);
    print"<p>Result row number" . $row_num.
        ". State_id:";
    print $row["State_id"];
    print "State";
    print $row["State"];
    print "</p>";
}
```


14.6.3 Requesting MySQL Operations

- Each array with a row from the result contains each field value indexed by position and by column name
  - The *`array_values`* applied to this array has each value twice, one with numeric keys and one with string keys.
  - please refer to pp.589
- Example:

```
$values=array_values($row);
for($index=0;$index<$num_fields;$index++)
  print "$values[2*$index+1]<br/>";
```


14.6.3 Requesting MySQL Operations

- Example:

```
$keys=array_keys($rows);
for($index=0;$index<$num_fields;$index++)
  print "$keys[2*$index+1]<br/>";
```


13.6 PHP/MySQL Example

- The example with `carsdata.html` and `access_cars.php` allows users to submit SQL commands that are executed against the enweb1 database
- The second example demo_2.html and demo_2.php

13.7 JDBC and MySQL(learning by yourself)

- A `DriverManager` must be available for the database system to which connections are being made
  - A driver can be assigned to the property jdbc.drivers
  - The driver class can be referenced in the program

13.7 JDBC and MySQL(learning by yourself)

- The database is specified by a string of the form
  `jdbc:`*`subprotocol_name`*`:`*`more_info`*
  - The subprotocol name is mysql for MySQL
  - other_info might include the database name and a query string providing values such as a username or password
- A connection object is created
  - Using the static `getConnection` method of `DriverManager`
  - Getting a connection from a connection pool

13.7 Metadata

- Metadata refers to information about a database and its tables, including the virtual tables returned from queries
- JDBC supplies methods to retrieve metadata from a database and form a query result
- From a database, table names, column names, column types, for example
- From a result set, column names and the number of columns, for example

13.7 JDBC Example

- The example `JDBCServlet.java` implements a servlet that collects an SQL query from a user, applies it to the Corvettes database and displays the result

Part 8

Introduction to XML

contents

8.1 Introduction

- eXtensible Markup Language (*XML*)
  - Ø A *meta-markup* language
    - ü It is a language for defining markup language

8.1 Introduction

- Motivation for the development of XML
  - Ø Deficiencies of HTML
    - ü HTML is defined to describe the layout of information without considering its meaning.
      - –For example: a list of used cars with color and price can be nested in paragraph element, but cannot be found.
    - ü Lax syntactical rules

8.1 Introduction

- Markup languages defined in XML are known as *applications*
- XML can be written by hand or generated by computer
  - Ø Useful for data exchange
    - ü Data stored in XML documents can be electronically distributed and processed by any number of different processing applications.
    - ü Example first.xml

8.2 The Syntax of XML

- Levels of syntax
  - Ø *Well-formed* documents conform to basic XML rules
  - Ø *Valid documents* are well-formed and also conform to a *schema* which defines details of the allowed content
    - ü The first is document type definitions(DTDs)
    - ü The second is XML schema

8.2 The Syntax of XML

- Well-formed XML documents
  - Ø All *beginning* tags have a matching *ending* tag.
  - Ø If a *beginning* tag is inside an element, the matching *ending* tag is also inside the element.
  - Ø There is one *root* tag that contains all the other tags in a document
  - Ø Attributes must have a value assigned, the value must be quoted.

8.2 The Syntax of XML

- Ø The characters <, >, &, ‘, ” shoule be represented with entity(See next slide)
- Ø Tag name is case-sensitive
- Ø The processing instruction is included in <? and ?>

- Validity is tested against a schema, discussed later

8.2 The Syntax of XML

Tag	entity	meaning
<	<	Less than
>	>	Greater than
&	&	and
‘	'	Single quote
“	"	Double quotes

8.3 XML Document Structure

- Example entity.xml

8.3 XML Document Structure

- An XML document often uses two auxiliary files
  - Ø Schema file: specifying its tag set and structural syntactic rules
    - ü DTD or XML Schema
  - Ø Style file:describing how the content is printed or displayed
    - ü Cascading Style Sheets (CSS)
    - ü XSLT

8.4 Document Type Definitions(DTD)

- A DTD is a set of structural rules called *declarations*, which specify a set of *elements* that can appear in the document as well as *how* and *where* these elements may appear.
  - Ø A set of *declarations*
    - ü Define tags, attributes, entities
    - ü Specify the order and nesting of tags
    - ü Specify which attributes can be used with which tags

8.4 Document Type Definitions(DTD)

- Declaration syntax
  - Ø<!keyword …. >
  - ØFour possible *keywords* can be used:
    - üElEMENT, used to define tags;
    - üATTLIST, used to define tag attributes;
    - üENTITY, used to define entities;
    - üNOTATION, used to define data type notations.
  - ØNote: case sensitive

8.4.1 Declaring Elements

- General syntax
  - Ø<!ELEMENT *element-name* (list of names of child elements)>
  - ØFor exmaple:

    <!ELEMENT memo(from,to,date,re,body)>

8.4.1 Declaring Elements

- Multiplicity: Specifying the number of times that a child element may appear

Modifier	Meaning
+	One or more occurrences
*	Zero or more occurrences
?	Zero or one occurrence

- For example:

  <!ELEMENT person(parent+,age,spouse?,sibling*)>

8.4.1 Declaring Elements

- The leaf node of a DTD specifies the data type of the content of an element.
  - ØIn most cases, the type is *PCDATA*, for parsable character data.
  - ØTwo other content types can be specified with *EMPTY* and *ANY*
    - ü*EMPTY* is to specify that the element has no content.
    - ü*ANY* is to specify that the element may contain literally any content.
  - ØExample:

    <!ELEMENT body(#PCDATA)>

8.4.2 Declaring Attributes

- General syntax

  <!ATTLIST *element-name*
  (*attribute-name attribute-type default-value?*)+ >

  - ØExplanation can be found in pp.307

8.4.2 Declaring Attributes

- Default value

Value	Meaning
#value	The literal value which is used if none is specified in an element.
#FIXED value	The literal value which every element will have and which cannot be changed.
#REQUIRED	No default value is given, but the browser is warned if a default value (the value might be specified in an element.
#IMPLIED	No default value is given, but the browser is warned if a default value (the value might be specified in an element.

8.4.2 Declaring Attributes

- Supose the following specification.

```
<!ATTLIST airplane places CDATA “4”>
<!ATTLIST airplane engine_type CDATA #REQUIRED>
<!ATTLIST airplane price CDATA #IMPLIED>
<!ATTLIST airplane manufacturer CDATA #FIXED “Cessna”>
```

- The following xml is valid for the DTD

  <airplane places =”10” engine_type=”jet”> </airplane>

  Note: for the #FIXED usage

8.4.3 Declaring Entities

- Three commonly used entities: *general entity*, *parameter entity* and *external text entity*.
  - Ø*General entity*

    These entities can be referenced anywhere in the content of an XML document.
  - Ø*Parameter entity*

    These entities can be referenced only in DTD.
  - Ø*External text entity*

    When an entity is longer than a few words, such as a section of a technqical article, its text is defined outside the DTD.

8.4.3 Declaring Entities

- General Syntax

  <!ENTITY [%] *entity-name “entity-value”*>
  - ØWith %: a parameter entity
  - ØWithout %: a general entity
- Parameter entities may only be referenced in the DTD
- External text entity
  - Ø<!ENTITY *entity-name* SYSTEM *“file-location”*>
  - ØThe replacement for the entity is the content of the file

8.4.3 Declaring Entities

- Suppose that a document includes a large number of references to the full name of President Kennedy. We could define an entity as follows.

  <!ENTITY jfk “John Fitzgerald Kennedy”>
- Any XML document can specify the name with just the reference *&jfk*

8.4.4 A sample DTA

- See planes.dtd

8.4.5 Internal and External DTDs

- A DTD can appear inside an XML document or in an external file, as is the case with plances.dtd.
- If the DTD is included in the XML code, it must be introduced with <!DOCTYPE *root-element* [and terminated with]>

8.4.5 Internal and External DTDs

- Internal
  - Ø<!DOCTYPE *root-element* [
    declarations
    ]>
- External file
  - Ø<!DOCTYPE *root-name* SYSTEM *"file-name"*>
  - ØFor example:
    <!DOCTYPE planes_for_sale SYSTEM "planes.dtd">

8.4.5 Internal and External DTDs

- Example 1
  - ØExternal DTD
  - Øplanes.xml and planes.dtd
- Example 2
  - Øplanesdtd.xml

8.7 Displaying Raw XML Documents

- Plain XML documents are generally displayed literally by browsers
  - ØSome of the elements in the display are preceded with dash sign or plus sign, which can be interacted with.
  - ØDemonstrate the planes.xml

8.8 Displaying XML Documents with CSS

- An xml-stylesheet processing instruction can be used to associate a general XML document with a style sheet
  - Ø`<?xml-stylesheet type="text/css" href="planes.css">`
- The property of CSS
  - ØThe style sheet selectors will specify tags that appear in the XML document
  - ØThe *display* property
    - üIt denotes whether an element is to be displayed *inline* or in a separate *block*

8.8 Displaying XML Documents with CSS

- Demonstrate planescss.xml with planes.css

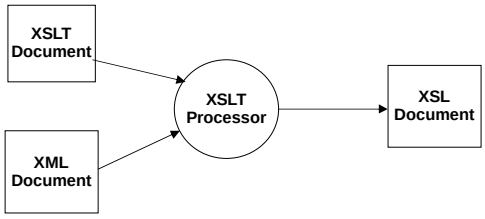
8.9 XSLT Style Sheets(自学)

- eXtensible Stylesheet Language(XSL) is a family of specifications for transforming XML documents
  - ØXSLT: specifies how to transform XML documents into different forms and formats, perhaps using different dtd.
  - ØXPath: specifies how to select parts of a document and compute values
  - ØXSL-FO: specifies a target XML language describing the printed page (not discuss)

8.9 Overview of XSLT

- XSLT describes how to transform XML documents into other XML documents such as XHTML
  - ØXSLT can be used to transform to non-XML documents as well
- A functional style programming language
- Basic syntax is XML
  - ØThere is some similarity to LISP and Scheme
- An XSLT processor takes an XML document as input and produces output based on the specifications of an XSLT document

8.9 XSLT Processing



8.10 XML Processors

- XML processors provide tools in programming languages to read in XML documents, manipulate them and to write them out
- Four purposes
  - ØCheck the basic syntax of the input document
  - ØReplace entities
  - ØInsert default values specified by schemas or DTD's
  - ØIf the parser is able and it is requested, validate the input document against the specified schemas or DTD's

8.10 Purposes of XML Processors

- The basic structure of XML is simple and repetitive, so providing library support is reasonable
- Two different standards/models for processing
  - ØSAX
  - ØDOM

8.10 Parsing

- The process of reading in a document and analyzing its structure is called *parsing*
- The parser provides as output a structured view of the input document

8.10 The SAX Approach

- In the SAX（Simple API for XML） approach, an XML document is read in serially
- As certain conditions, called *events*, are recognized, event handlers are called
- The program using this approach only sees part of the document at a time
- This approach are suitable for processing large XML documents.

8.10 The SAX Approach (Example)

```
public void startElement(String uri, String localName,
String qName, Attributes attributes) throws SAXException
 {
  if(qName.equals(“ad")){System.out.println(“ad”);
  }
---------------------------------------------------------------------
public void endElement(String uri, String localName,
String qName, Attributes attributes) throws SAXException
 {
  if(qName.equals(“ad")){ System.out.println(“ad ends”);
  }
```


8.10 The DOM Approach

- In the DOM(Document Object Model) approach, the parser produces an in-memory representation of the input document
  - Ø Because of the well-formedness rules of XML, the structure is a tree

8.10 The DOM Approach (Example)

```
publicvoid parseXml(String fileName) {
…..
// 获得文档根元素对对象;
Element root = document.getDocumentElement();
// 获得文档根元素下一级子元素所有元素;
 NodeList nodeList = root.getChildNodes();
……
}
```


8.10 The DOM Approach

- Advantages over DOM
  - Ø Parts of the document can be accessed more than once
  - Ø The document can be restructured
  - Ø Access can be made to any part of the document at any time
  - Ø Processing is delayed until the entire document is checked for proper structure and, perhaps, validity
- One major disadvantage is that a very large document may not fit in memory entirely

- End!