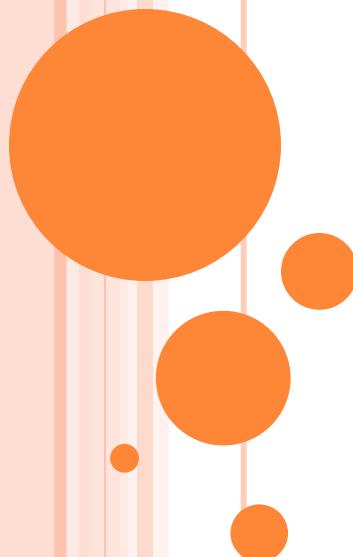


LECTURE 01

INTRODUCTION



THE INTERNET

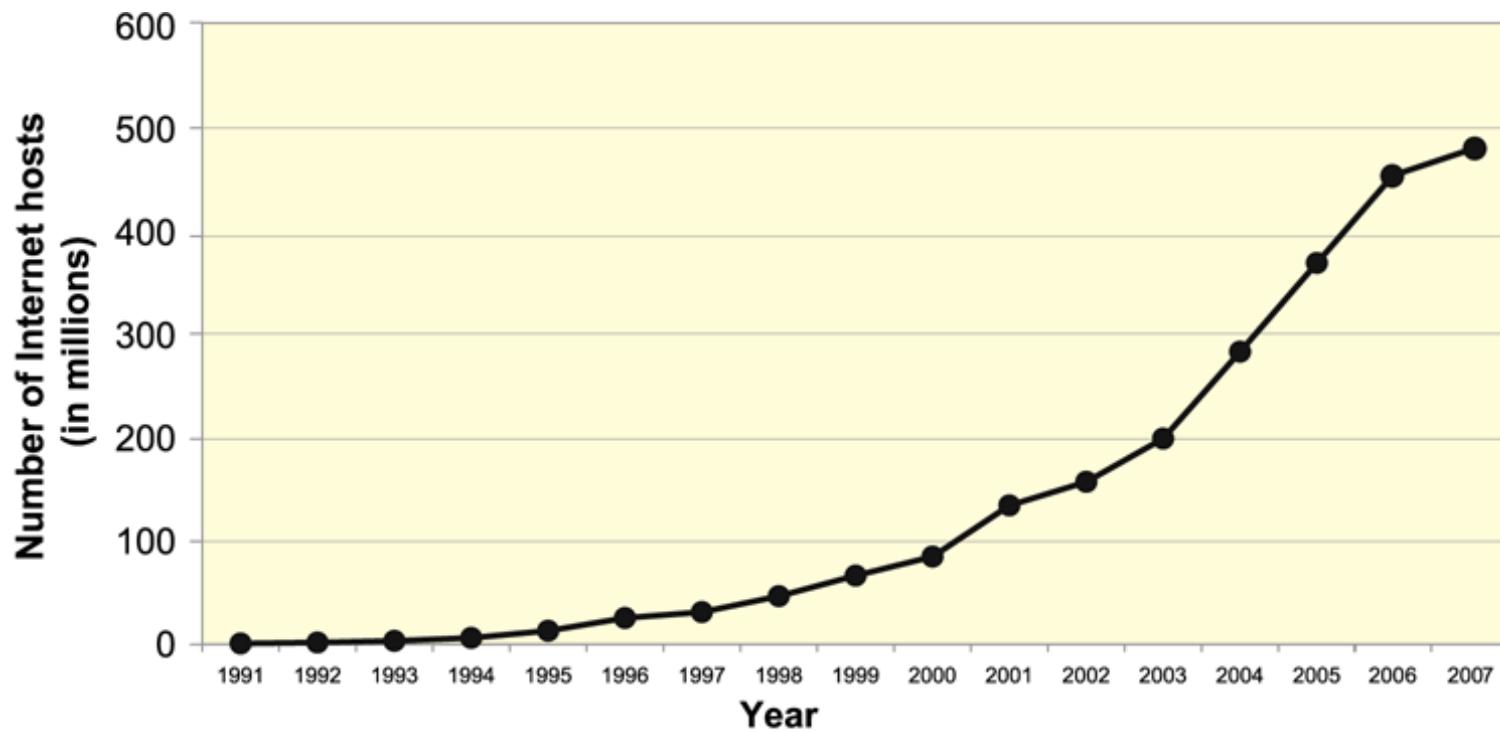
- The Internet is a network of networks that are spread all over the world
- A characteristic Internet is that it is a robust network.
 - If some nodes are removed, data can still be sent between nodes.
 - The Internet also has dynamic routing, where the route of the data is determined at the time of transmission based on current network conditions.



BRIEF HISTORY

- Began as a US Department of Defense network called ARPANET (1960s-70s)
- Initial services: electronic mail, file transfer
- Opened to commercial interests in late 80s
- WWW created in 1989-91 by Tim Berners-Lee
- Popular web browsers released: Netscape 1994, IE 1995
- Amazon.com opens in 1995; Google January 1996





Source: Internet Systems Consortium (<http://www.isc.org/>) and author's estimates

FIGURE 2-1 Growth of the Internet

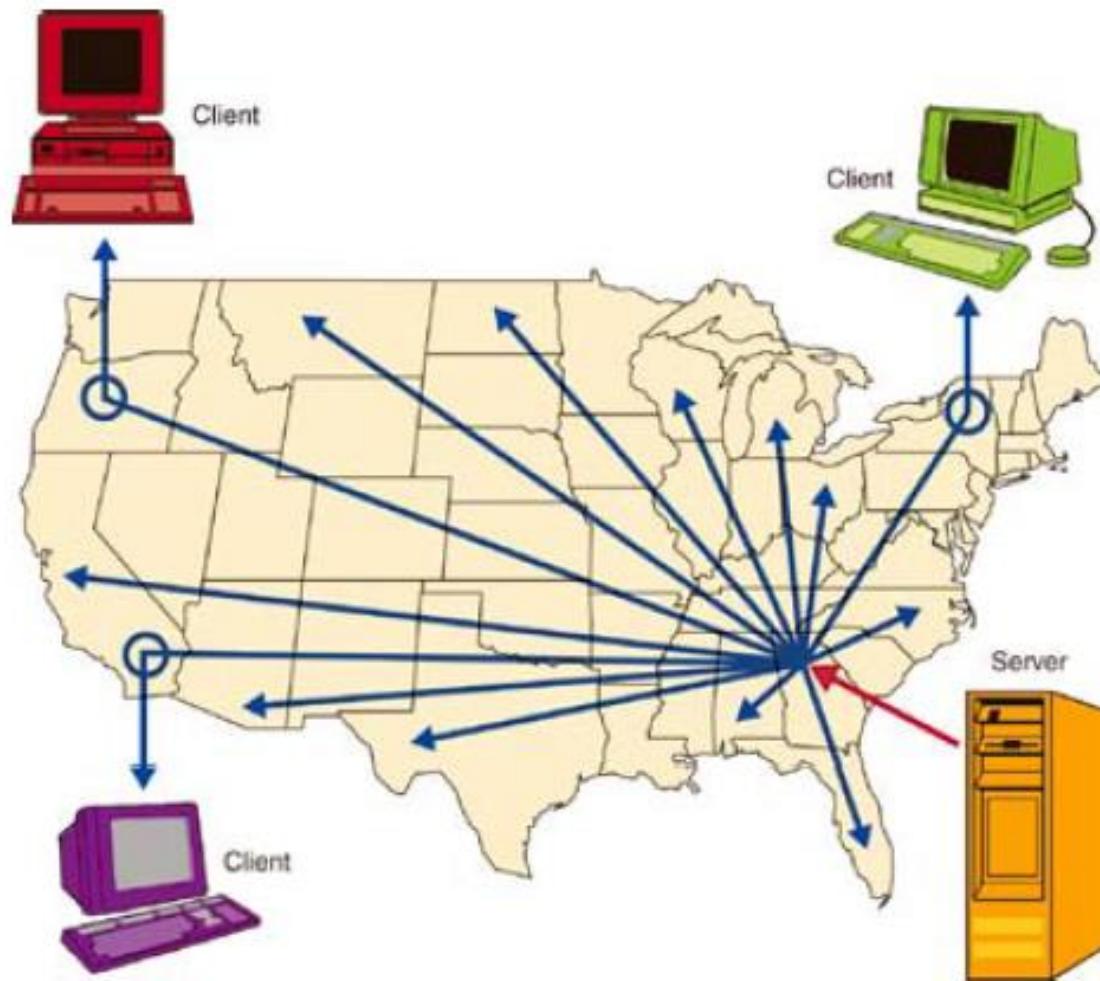
- Estimates
 - More than 140 million Web sites
 - More than 30 billion individual Web pages
- Commercial business Web use increasing

THE CLIENT/SERVER SOFTWARE MODEL

- Clients and servers are host machines
 - A client is the host machine that requests information from the server
 - The server is a resource that provides a service for (many) clients
 - The client/server interaction is the foundation for all Internet communication



THE CLIENT/SERVER SOFTWARE MODEL



THE WORLD WIDE WEB

- The web is a collection of files stored at locations throughout the world.
- These files are written using a special language known as the Hypertext Markup Language (HTML).
 - An HTML file will contain ***text*** which forms the information content of the file, together with ***instructions*** which define how the text is to be displayed.
- A file which is downloaded into a browser is known as a ***web page***.
- The computer that holds web pages is known as a ***web server***.



THE WORLD WIDE WEB

- The user of the World Wide Web employs a program known as a ***browser***.
 - When the user wishes to read a file on the World Wide Web they will inform the browser of its address on the web and the browser will fetch the file.
- Web pages can contain a wide variety of media including audio files, video files, graphics and even programs which can execute while the browser is being viewed.



INTERNET PROTOCOLS

- **Protocol:** collection of network data rules
 - Includes transmission rules
 - Computers must use same protocol

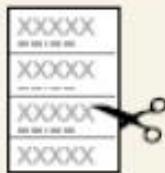


TCP/IP

- Internet protocols
 - **Transmission Control Protocol (TCP)**
 - Controls message, file disassembly into packets before Internet transmission
 - Controls packet reassembly into original formats at destinations
 - **Internet Protocol (IP)**
 - Specifies addressing details for each packet
 - Labels packet with origination and destination addresses
- **TCP/IP** refers to both protocols



MESSAGE TRANSMITTED USING TCP/IP



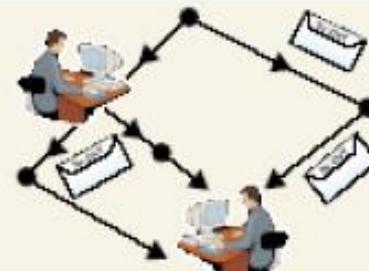
① A message is divided into packets.



② Each packet is addressed to the destination.



④ If a path is congested with heavy traffic or inoperable, packets can be re-routed via other paths.



③ The packets travel to the destination without using a defined path.



⑤ When the packets arrive at the destination, they are reassembled.



IP ADDRESSING

- **IP address** – uniquely identify every computer in a network system

IP address expressed in 32 - bit binary number	Equivalent IP address in decimal representation
10000000 00110100 00110000 00000000	128.52.48.0
11000000 00001001 00000110 00100101	192.9.6.37

IPv4 8 bit 8 bit 8 bit 8 bit **Total = 2^{32}**

e.g. 123 . 23 . 45 . 67

IPv6 16 bit **Total = 2^{128}**

e.g. 1010 : 0 : 0 : 8 : 800:20000:4 : 300

HOST MACHINES AND HOST NAMES

- Each computer on the Internet is a host machine.
- Each computer has a unique Internet Protocol (IP) address, such as 124.110.24.1
 - Some computers have a permanent IP address
 - Some computers borrow an IP address while they are connected to the Internet (Temporary)
- An IP address is not human-friendly



HOST MACHINES AND HOST NAMES

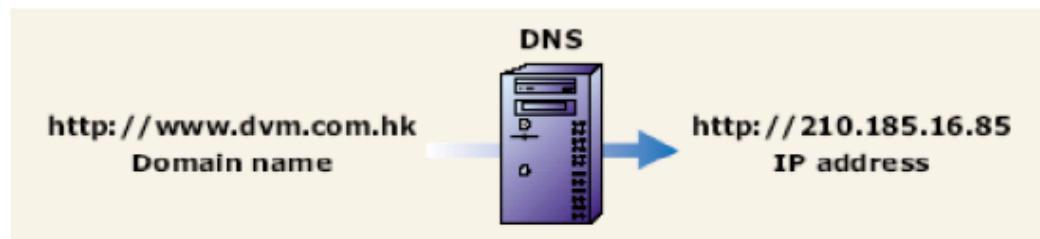
- The IP address for most host machines are mapped to a Domain Name Service (DNS) address in order to be more people-friendly
- The DNS address consists of a host name followed by a domain name
- Example DNS Address: **mail.yahoo.com**
 - Host Name is: mail
 - Domain Name is: yahoo.com
- While each machine has a unique IP address, it can have multiple DNS addresses (called aliases)

DOMAIN NAMES

When you type in a DNS address, a domain name server translates it into an IP address.

IP address in decimal representation	Equivalent Domain names
202.45.176.1	http://www.skqs.com
202.84.12.250	http://www.schoolteam.net

Translation between the IP address and Domain name is done by Domain Name Server (DNS)



DOMAIN NAME SYSTEMS

- Domain names are unique identifiers of computer or network addresses on the web
- Refer to an IP address

- Example:
 - Microsoft.com
 - Un.org
 - Whitehouse.gov



DOMAIN NAME SYSTEMS

Organizational Domains (Generic Top-Level Domains)

.com	Commercial organizations
.edu	Education and academic organizations
.int	International organizations
.mil	U.S. military organizations
.gov	U.S. government organizations
.net	Backbone, regional, and commercial networks
.org	Other organizations such as research and nonprofit



DOMAIN NAME SYSTEMS

Sample Geographic Domains (Country Code Top-Level Domains)		New general TLDs
.au	Australia	.biz Businesses
.fr	France	.coop Cooperatives
.hk	Hong Kong	.info General use
.de	Germany	.pro Professionals
.jp	Japan	.aero Air-transport industries
.kr	Korea (Republic)	
.es	Spain	
.uk	United Kingdom	
.us	United States	



URL

- Refers to a particular Web page or a particular file residing on a Web site.
 - <**scheme**>:<**scheme-dependent-information**>
 - <**scheme**>:<scheme-dependent-information>
- Examples:
 - http (Hyper Text Transfer Protocol)
 - ftp (File Transfer Protocol)



SCHEME DEPENDENT INFORMATION

- <scheme>:<**scheme-dependent-information**>
 - This information is detailed with each scheme
 - Most schemes include the:
 - Machine making the file available
 - "Path"to that file
 - Example:



URL EXAMPLE



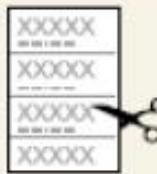
- `http` is the scheme
hyper text transfer protocol
- two slashes (`//`) separate the scheme from the machine/domain name
- `www.7sport.net` is the machine/domain name
- single slash (`/`) separates the name from the path
- Finally `7sport/index.htm` is the path.

PACKET-SWITCHED NETWORKS

- **Packet-switched** network
 - **Packets**
 - Small pieces labeled electronically (origin, sequence, destination address)
 - Travel along interconnected networks
 - Can take different paths
 - May arrive out of order
 - Destination computer
 - Collects packets
 - Reassembles original file or e-mail message



MESSAGE TRANSMITTED USING TCP/IP



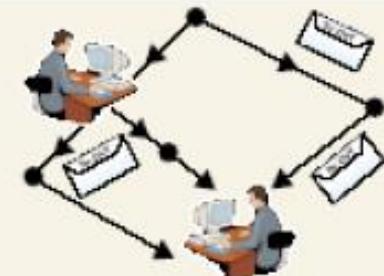
① A message is divided into packets.



② Each packet is addressed to the destination.



④ If a path is congested with heavy traffic or inoperable, packets can be re-routed via other paths.



③ The packets travel to the destination without using a defined path.

⑤ When the packets arrive at the destination, they are reassembled.

ROUTING PACKETS

- **Routing computers**
 - Decide how best to forward each packet
- **Routing algorithms**
 - Programs on router computers
 - Determine best path for packet
 - Routing algorithms applied to routing table information



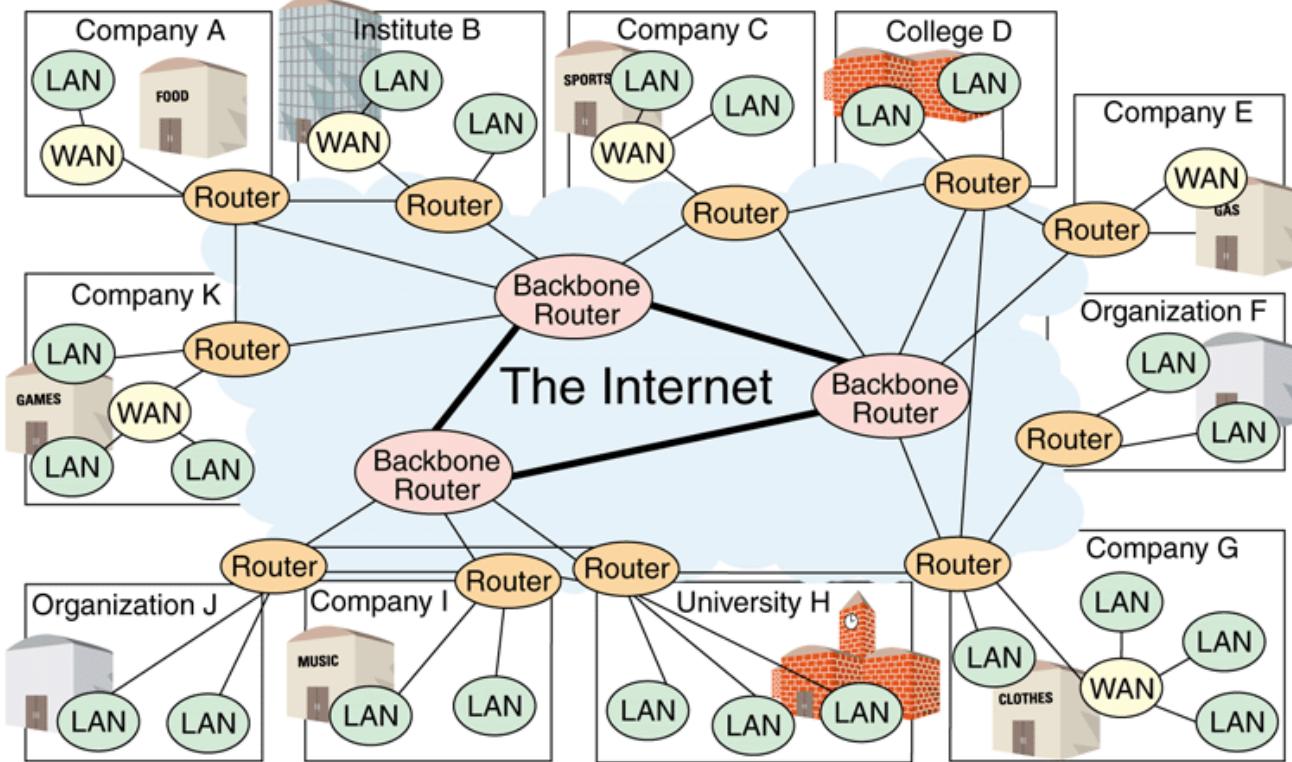


FIGURE 2-3 Router-based architecture of the Internet

○Internet backbone

- Internet routers handle packet traffic
 - Three billion packets per second

WEB CLIENTS AND WEB SERVERS

- Client/server architectures
 - Client requests services from server
- Client computer
 - Uses Web browser software (Web client software)
- Server computer
 - More memory and larger, faster disk drives



VARIOUS MEANINGS OF “SERVER”

- **Server**

- Any computer providing files (programs) to other computers
 - Connected through network
 - Use more capable hardware elements.
 - Can handle number of users.

- **Server software**

- Run Web server software
- Makes files (programs) available to other computers
- Sometimes included with operating system



WEB CLIENT/SERVER COMMUNICATION

- Web browser requests files from Web server
 - *Transportation medium:* the Internet
 - *Request formatted* by browser using HTTP
 - *Request sent* to server computer
 - *Server receives request*
 - Retrieves file containing requested Web page
 - Formats using HTTP
 - Sends back to client over the Internet
 - *Client Web browser*
 - Browser displays information if it is an HTML page
 - Graphics can be slow to appear



CLIENT/SERVER ARCHITECTURE (TWO-TIER)

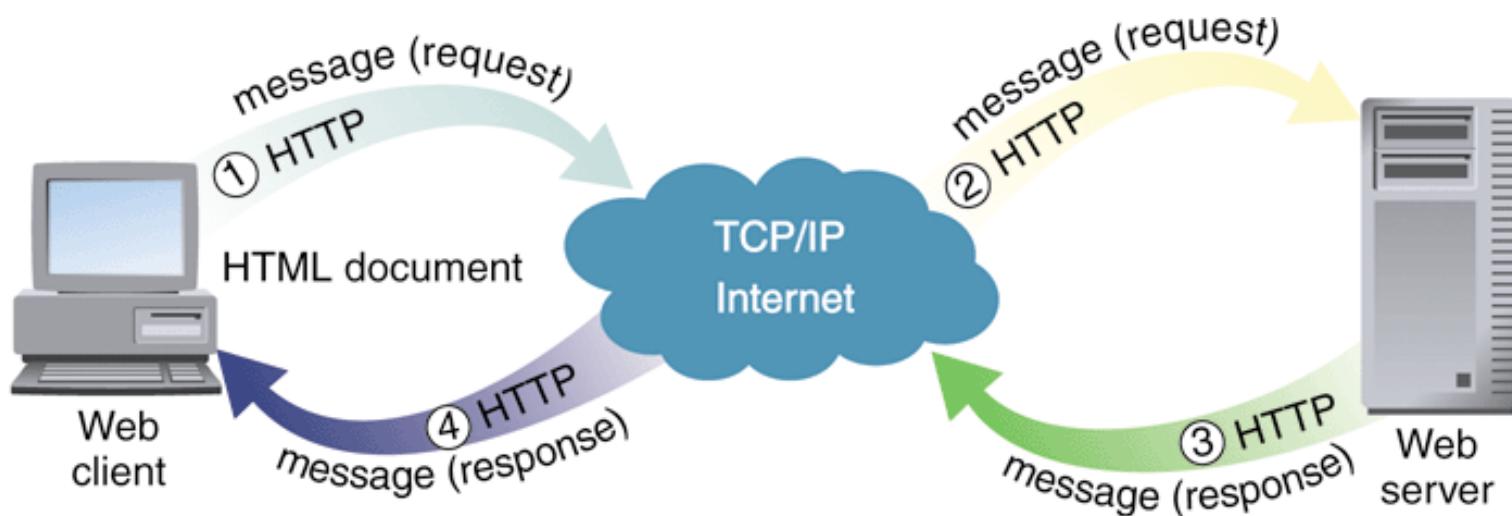


FIGURE 8-3 Message flows in a two-tier client/server network

SOFTWARE FOR WEB SERVERS

- Types of Web server software/programs
 - Operating system software
 - Web server software
 - Other programs
 - Internet utilities
 - E-mail software



OPERATING SYSTEMS FOR WEB SERVERS

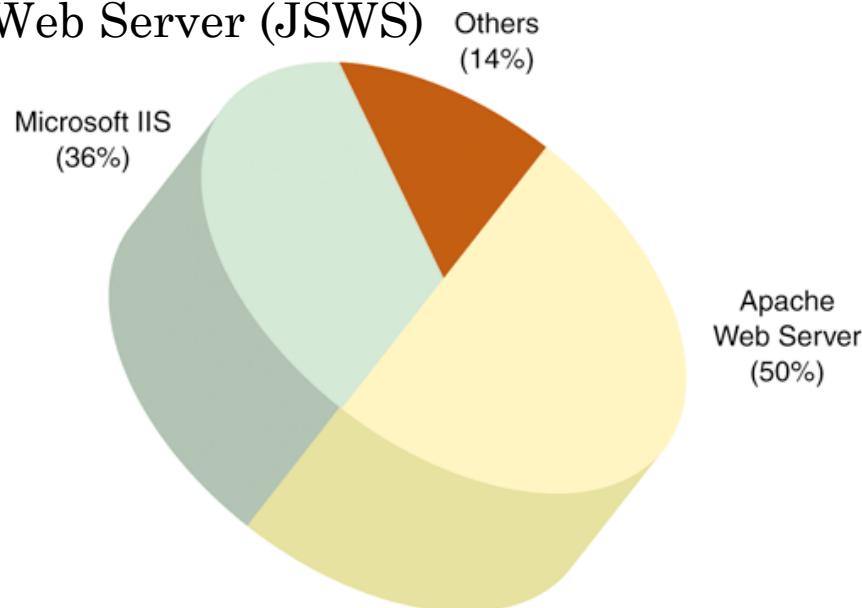
- Operating system tasks

- Running programs, allocating computer resources, providing input and output services
 - *Microsoft Windows Server products*
 - *UNIX-based products*
 - *Linux (open-source operating system)*



WEB SERVER SOFTWARE

- Software that listens for web page requests
- Commonly used Web server programs
 - Apache HTTP Server,
 - Microsoft Internet Information Server (IIS),
 - Sun Java System Web Server (JSWS)



Source: Netcraft Web Surveys, <http://www.netcraft.com>

FIGURE 8-5 Percent of Web active sites that use major Web server software products

WEB BROWSERS

- A Web browser is the software necessary to view information.
- The information in the Web is organized as hypertext, graphics, video, and sound
- A Web page is a document on the Web that you view through your Web browser



WEB BROWSERS (CONT.)

- ***Browsing*** : the act of reading Web pages and clicking on hyperlinks
- Each Web page has a unique address (URL) that you can use to jump directly to it.
- Examples:
 - Chrome
 - Firefox
 - Opera
 - Safari

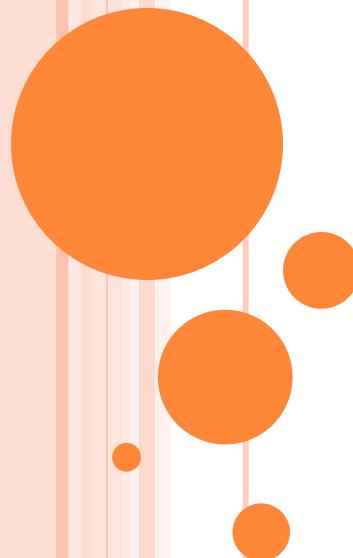


QUESTIONS



LECTURE 02

HTML



WHAT IS HTML?

- HTML stands for **Hypertext Markup Language**, and it is the most widely used language to write Web Pages.
 - **Hypertext** refers to the way in which Web pages (HTML documents) are linked together. Thus, the link available on a webpage is called Hypertext.
 - HTML is a **Markup Language** which means you use HTML to simply "mark-up" a text document with tags that tell a Web browser how to structure it to display.

HTML TAGS

- HTML tags are keywords (tag names) surrounded by angle brackets.

<tagname>content</tagname>

- HTML tags normally come in pairs like <p> and </p>
- The first tag in a pair is the start tag, the second tag is the end tag
- The end tag is written like the start tag, but with a slash before the tag name

EXAMPLE

```
<!DOCTYPE html>
<html>
<head>
    <title>Page Title</title>
    <link rel="shortcut icon" href="logo/1.png"/>
</head>

<body>
    <h1>My First Heading</h1>
    <p>My first paragraph.</p>
    <!-- this is comments -->
</body>
</html>
```

HTML TAGS

- **DOCTYPE:**
 - defines the document type to be HTML
- The text between **<html>** and **</html>**:
 - describes an HTML document
- The text between **<head>** and **</head>**:
 - provides information about the document
- The text between **<title>** and **</title>**:
 - provides a title for the document
 - displayed in the web browser's title bar

HTML TAGS

- The text between **<body>** and **</body>**:
 - describes the visible page content
- The text between **<h1>** and **</h1>**:
 - describes a heading
- The text between **<p>** and **</p>**:
 - describes a paragraph
 - placed within the body of the page

HTML PAGE STRUCTURE

```
<html>

  <head>
    <title>Page title</title>
  </head>

  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
    <p>This is another paragraph.</p>
  </body>

</html>
```

ATTRIBUTES

- Tags can also have attributes, which are extra bits of information.
- Attributes appear inside the opening tag and their values sit inside quotation marks.
- They look something like
`<tag attribute="value">XXXXXX</tag>`

PAGE TITLE: <TITLE>

- Describes the title of the web page
 - Placed within the head of the page
 - Displayed in the web browser's title bar and when bookmarking the page

```
<head>
```

```
    <title>My first web page</title>
```

```
</head>
```

The head element (starts with `<head>` and ends with `</head>`) appears before the body element (starting with `<body>` and ending with `</body>`) and contains information about the page.

<META XXXXXXXX /> (TAG)

- Metadata is data (information) about data.
- The <meta> tag provides metadata about the HTML document.
- Metadata will not be displayed on the page, but will be machine parsable.
- Meta elements are typically used to specify page description, keywords, author of the document, last modified, and other metadata.
- The metadata can be used by browsers (how to display content or reload page), search engines (keywords), or other web services.



EXAMPLE OF USING MEATA

- <head>
- <meta http-equiv="refresh" content="30" />
- <meta name="description" content="Web Development Course" />
- <meta name="keywords" content="PTUK, CSE, Kadoorie" />
- <meta name="author" content="Yazeed Sleet" />
- </head>



PARAGRAPH: <P>

- Paragraphs of text
 - placed within the body of the page

```
<body>
```

```
  <p>My first web page</p>
```

```
  <p>Very Cool</p>
```

```
</body>
```

Web browsers don't usually take any notice of what line your code is on. It also doesn't take any notice of spaces

HEADINGS: <H1>, <H2>, ..., <H6>

- Headings to separate major areas of the page
- placed within the body of the page

<*h1*>*Kadoorie University*</*h1*>

<*h2*>*Computer Engineering*</*h2*>

<*h3*>*Web Development*</*h3*>

Web browsers don't usually take any notice of what line your code is on. It also doesn't take any notice of spaces

LINKS: <A>

- Links, or "anchors", to other pages (inline)
 - uses the href attribute to specify the destination URL

` Kadoorie `

IMAGES:

- Inserts a graphical image into the page (inline)
 - The src attribute tells browser where to find the image
 - Also requires an alt attribute describing the image
 - If placed inside an a anchor, the image will become a link
 - The title attribute specifies an optional tooltip

```

```

NESTING TAGS

- Tags must be correctly nested
- A closing tag must match the most recently opened tag
- The browser may render it correctly anyway, but it is invalid.

HORIZONTAL RULE: <HR>

- A horizontal line to visually separate sections of a page
- Should be immediately closed with />

<*p*>*Kadoorie University*</*p*>

<*hr*/>

<*p*>*Computer Engineering*</*p*>

LINE BREAK:

- Forces a line break in the middle of a block element.
- br should be immediately closed with />

*<p>Kadoorie university is nice,
 and the Computer Engineering is cool</p>*

UNORDERED LIST: ,

- ul represents a bulleted list of items
- li represents a single item within the list

<*ul*>

<*li*> Computer Eng </*li*>

<*li*> Mechanical Eng </*li*>

<*li*> Telecom. Eng </*li*>

</*ul*>

UNORDERED LIST: ,

```
<ul>
  <li> Computer Eng:
    <ul>
      <li> Web Development</li>
      <li> Software Engineering</li>
    </ul>
  </li>
  <li> Mechanical Eng:
    <ul>
      <li> Special Topics I</li>
      <li> Special Topics II</li>
    </ul>
  </li>
  <li> Telecom. Eng:
  </li>
</ul>
```

ORDERED LIST:

- o ol represents a numbered list of items

<*ol*>

<*li*> Computer Eng </*li*>

<*li*> Mechanical Eng </*li*>

<*li*> Telecom. Eng </*li*>

</*ol*>

HTML DESCRIPTION LISTS

- <dl>
 <dt>Coffee</dt>
 <dd>- black hot drink</dd>
 <dt>Milk</dt>
 <dd>- white cold drink</dd>
 </dl>

Coffee

- black hot drink

Milk

- white cold drink

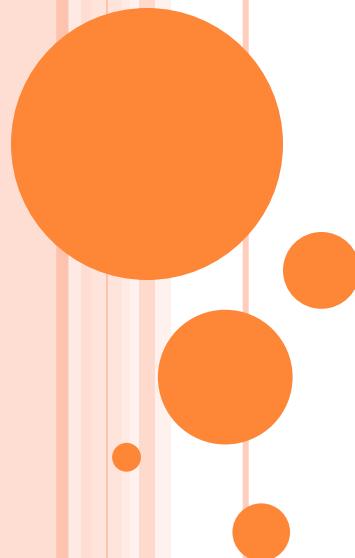


QUESTIONS



LECTURE 03

HTML



HTML TEXT FORMATTING ELEMENTS

- HTML defines special **elements**, for defining text with a special **meaning**.
- Formatting elements were designed to display special **types of text**:
 - Bold text
 - Important text
 - Italic text
 - Emphasized text
 - Marked text
 - Small text
 - Deleted text
 - Inserted text
 - Subscripts
 - Superscripts

BOLD AND STRONG

- The HTML **** element defines **bold** text, without any extra importance.

```
<p>This text is normal.</p>
```

```
<p><b>This text is bold</b>.</p>
```

- The HTML **** element defines **strong** text, with added semantic "strong" importance.

```
<p>This text is normal.</p>
```

```
<p><strong>This text is strong</strong>.</p>
```

ITALIC AND EMPHASIZED

- The HTML **<i>** element defines *italic* text, without any extra importance.

<p>This text is normal.</p>

<p><i>This text is italic</i>.</p>

- The HTML **** element defines *emphasized* text, with added semantic importance.

<p>This text is normal.</p>

<p>This text is emphasized .</p>

UNDERLINE AND PRE-FORMATTED

- The HTML **<u>** element defines *underlined* text.

<p><u>This text is underlined</u>.</p>

- The HTML **<pre>** element defines a block of **pre-formatted** text, with structured spaces and lines.
- To display anything, with right spacing and line-breaks, you must wrap the text in a **<pre>** element:

*<pre>This text is
not formatted</pre>*

SMALL AND MARKED

- The HTML **<small>** element defines **small** text:

```
<h2>HTML <small>Small</small>  
Formatting</h2>
```

- The HTML **<mark>** element defines **marked** or highlighted text:

```
<h2>HTML <mark>Marked</mark>  
Formatting</h2>
```

DELETED AND INSERTED

- The HTML **** element defines **deleted** (removed) of text.

<p>My favorite color is blue red.</p>

- The HTML **<ins>** element defines **inserted** (added) text.

<p>My favorite <ins>color</ins> is red.</p>

SUBSCRIPT AND SUPERSCRIPT

- The HTML **<sub>** element defines **subscripted** text.

<p>This is _{subscripted} text.</p>

- The HTML **<sup>** element defines **superscripted** text.

<p>This is ^{superscripted} text.</p>

HTML STYLES

- Every HTML element has a **default style** (background color is white, text color is black, text-size is 12px ...)
- Changing the default style of an HTML element, can be done with the **style attribute**
- The HTML Style Attribute:
`style="property:value"`

HTML STYLES

- <!DOCTYPE html>
<html>
 <body style="background-color:lightgrey">
 <h1>This is a heading</h1>
 <p>This is a paragraph.</p>
 </body>
</html>

HTML TEXT COLOR

- The **color** property defines the text color to be used for an HTML element
- ```
<!DOCTYPE html>
<html>

<body>
 <h1 style="color:blue">This is a heading</h1>
 <p style="color:red">This is a paragraph.</p>
</body>

</html>
```

# HTML TEXT FONTS

- The **font-family** property defines the font to be used for an HTML element
- ```
<!DOCTYPE html>
<html>
<body>
  <h1 style="font-family:verdana">This is a heading</h1>
  <p style="font-family:courier">This is a paragraph.</p>
</body>
</html>
```

HTML TEXT SIZE

- The **font-size** property defines the text size to be used for an HTML element
- ```
<!DOCTYPE html>
<html>
 <body>
 <h1 style="font-size:300%">This is a heading</h1>
 <p style="font-size:26pt">This is a paragraph.</p>
 </body>
</html>
```

# HTML TEXT ALIGNMENT

- The **text-align** property defines the horizontal text alignment for an HTML element

- <!DOCTYPE html>

- <html>

- <body>

- <h1 style="text-align:center">Centered Heading</h1>

- <p>This is a paragraph.</p>

- </body>

- </html>

# HTML <Q> FOR SHORT QUOTATIONS

- Browsers usually insert **quotation marks** around the <q> element.
- ```
<!DOCTYPE html>
<html>
  <body>
    <p>WWF's goal is to: <q>Build a future where people live
in harmony with nature.</q></p>
  </body>
</html>
```

HTML <abbr> FOR ABBREVIATIONS

- The HTML <**abbr**> element defines an **abbreviation** or an acronym.
- Marking abbreviations can give useful information to browsers, translation systems and search-engines.
- ```
<!DOCTYPE html>
<html>
 <body>
 <p>The <abbr title="World Health Organization">WHO</abbr>
 was founded in 1948.</p>
 </body>
</html>
```

# HTML COMMENT TAGS

- You can add comments to your HTML source by using the following syntax:

*<!-- Write your comments here -->*

- Comments are not displayed by the browser, but they can help document your HTML.

# QUESTIONS



# HTML Lists

- HTML can have:
  - Unordered lists
    - The first item
    - The second item
  - Ordered lists
    - 1. The first item
    - 2. The second item
  - Description lists:
    - The first item**
    - Description of item

# Unordered Lists

- An unordered list starts with the `<ul>` tag.  
Each list item starts with the `<li>` tag.
- The list items will be marked with bullets  
(small black circles).

```

 Coffee
 Tea
 Milk

```

- Coffee
- Tea
- Milk

# The Style Attribute

- A **style** attribute can be added to an **unordered list**, to define the **style** of the marker:

Style	Description
list-style-type:disc	The list items will be marked with bullets (default)
list-style-type:circle	The list items will be marked with circles
list-style-type:square	The list items will be marked with squares
list-style-type:none	The list items will not be marked

# The Style Attribute

```
<!DOCTYPE html>
<html>
<body>
```

```
<h2>Unordered List with Disc Bullets</h2>
```

```
<ul style="list-style-type:square">
 Coffee
 Tea
 Milk

```

```
</body>
</html>
```

- Coffee
- Tea
- Milk

# Ordered Lists

- An ordered list starts with the `<ol>` tag. Each list item starts with the `<li>` tag.
- The list items will be marked with numbers.

```

```

```
 Coffee
```

1. Coffee

```
 Tea
```

2. Tea

```
 Milk
```

3. Milk

```

```

# The Type Attribute

- A **type** attribute can be added to an **ordered list**, to define the type of the marker:

Style	Description
type="1"	The list items will be numbered with numbers (default)
type="A"	The list items will be numbered with uppercase letters
type="a"	The list items will be numbered with lowercase letters
type="I"	The list items will be numbered with uppercase roman numbers
type="i"	The list items will be numbered with lowercase roman numbers

# The Type Attribute

```
<!DOCTYPE html>
<html>
<body>

<h2>Ordered List with Numbers</h2>

<ol type="A">
 Coffee
 Tea
 Milk

</body>
</html>
```

A. Coffee  
B. Tea  
C. Milk

# Description Lists

- A description list, is a list of terms, with a description of each term.
- The **<dl>** tag defines a description list.
  - The **<dt>** tag defines the term (name), and the **<dd>** tag defines the data (description).

```
<dl>
```

```
 <dt>Coffee</dt>
```

```
 <dd>- black hot drink</dd>
```

```
 <dt>Milk</dt>
```

```
 <dd>- white cold drink</dd>
```

```
</dl>
```

Coffee

- black hot drink

Milk

- white cold drink

# Nested HTML Lists

- List can be nested (lists inside lists).

```

 Coffee
 Tea

 Black tea
 Green tea

 Milk

```

- Coffee
- Tea
  - Black tea
  - Green tea
- Milk

# Iframe - Remove the Border

- By default, an iframe has a black border around it.
- To remove the border, add the style attribute and use the CSS border property:

```
<iframe src=" http://www.ptuk.edu.ps "
 style="border:none"></iframe>
```

- Change the size, style and color of the iframe's border:

```
<iframe src=" http://www.ptuk.edu.ps " style="border:5px
 dotted red"></iframe>
```

# Use iframe as a Target for a Link

- An iframe can be used as the target frame for a link.
- The **target** attribute of the link must refer to the **name** attribute of the iframe:

```
<iframe width="100%" height="300px"
src="http://www.ptuk.edu.ps" name="iframe_a
"></iframe>
<p><a href="http://www.goal.com" target="iframe_a
">Goal.com</p>
```

# The HTML <meta> Element

- The <meta> element is used to specify page description, keywords, author, and other metadata.
- Meta data is used by browsers (how to display content), by search engines (keywords), and other web services.
- Define keywords for search engines:

```
<meta name="keywords" content="HTML, CSS, XML,
XHTML, JavaScript">
```

# The HTML <meta> Element

- Define a description of your web page:

```
<meta name="description" content="Free Web tutorials on
HTML and CSS">
```

- Define the author of a page:

```
<meta name="author" content="PTUK">
```

- Refresh document every 30 seconds:

```
<meta http-equiv="refresh" content="30">
```

# HTML FORMS - THE <FORM> ELEMENT

- HTML forms are used to collect user input.
- The **<form>** element defines an HTML form.

*<form>*

.

*form elements*

.

*</form>*

# THE ACTION ATTRIBUTE

- The **action attribute** defines the action to be performed when the form is submitted.
- The common way to submit a form to a server, is by using a submit button.
- Normally, the form is submitted to a web page on a web server.
- If the action attribute is omitted, the action is set to the current page.
- In the example above, a server-side script is specified to handle the submitted form:

```
<form action="action_page.php">
```

# THE METHOD ATTRIBUTE

- The **method attribute** specifies the HTTP method (GET or POST) to be used when submitting the forms:

```
<form action="action_page.php" method="GET">
```

- Or

```
<form action="action_page.php" method="POST">
```

## GET

In GET method, values are visible in the URL.

GET has a limitation on the length of the values, generally 255 characters.

GET performances are better compared to POST because of the simple nature of appending the values in the URL.

This method supports only string data types.

GET results can be bookmarked.

GET request is often cacheable.

GET Parameters remain in web browser history.

## POST

In POST method, values are not visible in the URL.

POST has no limitation on the length of the values since they are submitted via the body of HTTP.

It has lower performance as compared to GET method because of time spent in including POST values in the HTTP body.

This method supports different data types, such as string, numeric, binary, etc.

POST results cannot be bookmarked.

The POST request is hardly cacheable.

Parameters are not saved in web browser history.

# WHEN TO USE GET?

- You can use GET (the default method):
- When you use GET, the form data will be visible in the page address:

*action\_page.php?firstname=Mickey&lastname=Mouse*

# WHEN TO USE POST?

- You should use POST:
  - If the form includes sensitive information (password).
  - POST offers better security because the submitted data is not visible in the page address

# THE <INPUT> ELEMENT

- The <input> element is the most important **form element**.
- The <input> element has many variations, depending on the **type** attribute.
- Main Input Types:

Text	Description
text	Defines normal text input
radio	Defines radio button input (for selecting one of many choices)
submit	Defines a submit button (for submitting the form)

# TEXT INPUT

- **<input type="text">** defines a one-line input field for **text input**:

```
<form>
First name:

<input type="text" name="firstname">

Last name:

<input type="text" name="lastname">
</form>
```

- **Note:** The form itself is not visible. Also note that the default width of a text field is 20 characters.

## INPUT TYPE: PASSWORD

- <input type="password"> defines a password field:

```
<form action="">
User name:

<input type="text" name="userid">

User password:

<input type="password" name="psw">
</form>
```

# RADIO BUTTON INPUT

- **<input type="radio">** defines a **radio button**.
- Radio buttons let a user select ONE of a limited number of choices:

```
<form>
 <input type="radio" name="gender" value="male"
 checked>Male

 <input type="radio" name="gender"
 value="female">Female
 </form>
```

# INPUT TYPE: CHECKBOX

- `<input type="checkbox">` defines a **checkbox**.
- Checkboxes let a user select ZERO or MORE options of a limited number of choices.

```
<form action="action_page.php">
<input type="checkbox" name="vehicle" value="Bike">I have
a bike

<input type="checkbox" name="vehicle" value="Car">I have a
car

<input type="submit">
</form>
```

# THE SUBMIT BUTTON

- **<input type="submit">** defines a button for **submitting** a form to a **form-handler**.
- The form-handler is typically a server page with a script for processing input data.
- The form-handler is specified in the form's **action** attribute:

```
<form action="action_page.php">
First name:

<input type="text" name="firstname" value="Mickey">

Last name:

<input type="text" name="lastname" value="Mouse">

<input type="submit" value="Submit">
</form>
```

# THE NAME ATTRIBUTE

- To be submitted correctly, each input field must have a name attribute.
- This example will only submit the "Last name" input field:

```
<form action="action_page.php">
First name:

<input type="text" value="Mickey">

Last name:

<input type="text" name="lastname" value="Mouse">

<input type="submit" value="Submit">
</form>
```

## <FIELDSET>

- The <fieldset> element groups related data in a form.
- The <legend> element defines a caption for the <fieldset> element

```
<form action="action_page.php">
<fieldset>
<legend>Personal information:</legend>
First name:

<input type="text" name="firstname" value="Mickey">

Last name:

<input type="text" name="lastname" value="Mouse">

<input type="submit" value="Submit"></fieldset>
</form>
```

Personal information:

First name:  
Mickey

Last name:  
Mouse

Submit

# HTML FORM ELEMENTS - <SELECT>

- The **<select>** element defines a **drop-down** list:

```
<select name="cars">
 <option value="volvo">Volvo</option>
 <option value="saab">Saab</option>
 <option value="fiat">Fiat</option>
 <option value="audi">Audi</option>
</select>
```

# HTML FORM ELEMENTS - <SELECT>

- The **<option>** elements defines the options to select.
- The list will normally show the first item as selected.
- You can add a selected attribute to define a predefined option.

```
<select name="cars">
 <option value="volvo">Volvo</option>
 <option value="saab">Saab</option>
 <option value="fiat" selected>Fiat</option>
 <option value="audi">Audi</option>
</select>
```

# HTML <OPTGROUP> TAG

- The <optgroup> is used to group related options in a drop-down list.
- If you have a long list of options, groups of related options are easier to handle for a user.

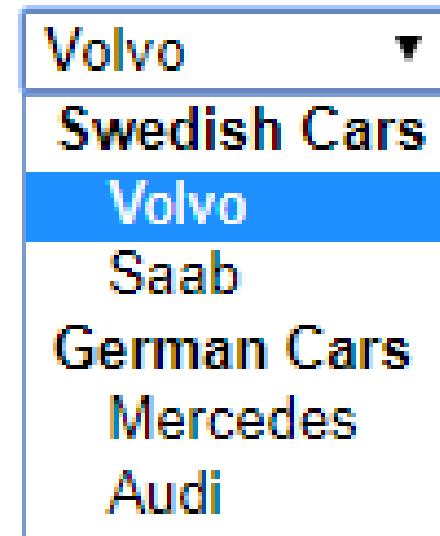
# HTML <OPTGROUP> TAG

```
<select name="cars">

 <optgroup label="Swedish Cars">
 <option
 value="volvo">Volvo</option>
 <option value="saab">Saab</option>
 </optgroup>

 <optgroup label="German Cars">
 <option
 value="mercedes">Mercedes</option>
 <option value="audi">Audi</option>
 </optgroup>

</select>
```



# HTML <OPTGROUP> TAG - DISABLED

- The disabled attribute is a boolean attribute.
- A disabled option-group is unusable and un-clickable.

```
<select name="cars">
 <optgroup label="Swedish Cars">
 <option value="volvo">Volvo</option>
 <option value="saab">Saab</option>
 </optgroup>
 <optgroup label="German Cars" disabled>
 <option value="mercedes">Mercedes</option>
 <option value="audi">Audi</option>
 </optgroup>
</select>
```

# THE <TEXTAREA> ELEMENT

- The **<textarea>** element defines a multi-line input field (**a text area**):

```
<textarea name="message" rows="10" cols="30">
The cat was playing in the garden.
</textarea>
```

# THE <BUTTON> ELEMENT

- The **<button>** element defines a clickable **button**:

```
<button type="button" >Click Me!</button>
```

# HTML <LABEL> TAG

- The **<label>** tag defines a label for an **<input>** element.
- The **for** attribute of the **<label>** tag should be equal to the **id** attribute of the related element to bind them together.

# HTML <LABEL> TAG

```
<form action="action_page.php">
 <label for="male">Male</label>
 <input type="radio" name="gender" id="male"
 value="male">

 <label for="female">Female</label>
 <input type="radio" name="gender" id="female"
 value="female">

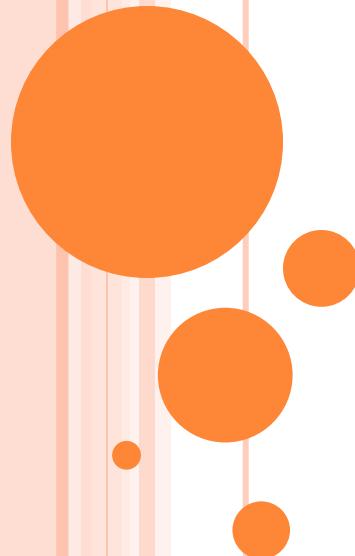
 <input type="submit" value="Submit">
</form>
```



*ALL THE BEST...!*

# LECTURE 08

## HTML INPUT TYPES



Ahmad Rabay'a  
2015  
[rabaya.ahmad@gmail.com](mailto:rabaya.ahmad@gmail.com)

# HTML5 INPUT TYPES

- HTML5 added several new input types:
  - color
  - date
  - datetime-local
  - email
  - month
  - number
  - range
  - time
  - url
  - week

# INPUT TYPE: NUMBER

- The `<input type="number">` is used for input fields that should contain a numeric value.
- You can set restrictions on the numbers(min, max, step and default value “value”).
- Depending on browser support, the restrictions can apply to the input field

```
<form action="action_page.php">
 Quantity (between 1 and 5):
 <input type="number" name="quantity">
 <input type="submit">
</form>
```

## INPUT TYPE: DATE

- The `<input type="date">` is used for input fields that should contain a date.
- Depending on browser support, a date picker can show up in the input field.
- You can use min, max and value.

```
<form action="action_page.php">
 Birthday:
 <input type="date" name="bday">
 <input type="submit">
</form>
```

## INPUT TYPE: COLOR

- The `<input type="color">` is used for input fields that should contain a color.
- Depending on browser support, a color picker can show up in the input field.

```
<form action="action_page.php">
```

*Select your favorite color:*

```
<input type="color" name="favcolor" value="#ff0000">
```

```
<input type="submit">
```

```
</form>
```

# INPUT TYPE: RANGE

- The **<input type="range">** is used for input fields that should contain a value within a range.
- Depending on browser support, the input field can be displayed as a slider control.
- You can use the following attributes to specify restrictions: min, max, step, value.

```
<form action="action_page.php" method="get">
```

*Points:*

```
<input type="range" name="points" min="0" max="10">
```

```
<input type="submit">
```

```
</form>
```

## INPUT TYPE: MONTH

- The **<input type="month">** allows the user to select a month and year.
- Depending on browser support, a date picker can show up in the input field.
- You can use the following attributes to specify restrictions: min, max, value.

```
<form action="action_page.php">
Birthday (month and year):
<input type="month" name="bdaymonth">
<input type="submit">
</form>
```

## INPUT TYPE: WEEK

- The `<input type="week">` allows the user to select a week and year.
- Depending on browser support, a date picker can show up in the input field.

```
<form action="action_page.php">
```

*Select a week:*

```
<input type="week" name="year_week">
```

```
<input type="submit">
```

```
</form>
```

## INPUT TYPE: TIME

- The `<input type="time">` allows the user to select a time (no time zone).
- Depending on browser support, a time picker can show up in the input field.

```
<form action="action_page.php">
```

*Select a time:*

```
<input type="time" name="usr_time">
```

```
<input type="submit">
```

```
</form>
```

## INPUT TYPE: DATETIME-LOCAL

- The `<input type="datetime-local">` allows the user to select a date and time (no time zone).
- Depending on browser support, a date picker can show up in the input field.

```
<form action="action_page.php">
 Birthday (date and time):
 <input type="datetime-local" name="bdaytime">
 <input type="submit" value="Send">
</form>
```

## INPUT TYPE: EMAIL

- The `<input type="email">` is used for input fields that should contain an e-mail address.
- Depending on browser support, the e-mail address can be automatically validated when submitted.

```
<form action="action_page.php">
```

*E-mail:*

```
<input type="email" name="email">
```

```
<input type="submit">
```

```
</form>
```

## INPUT TYPE: URL

- The **<input type="url">** is used for input fields that should contain a URL address.
- Depending on browser support, the url field can be automatically validated when submitted

```
<form action="action_page.php">
```

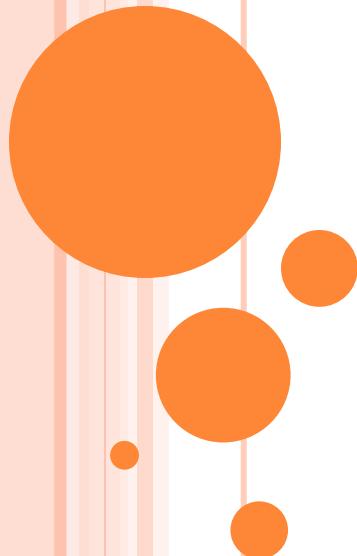
*Add your homepage:*

```
<input type="url" name="homepage">
```

```
<input type="submit">
```

```
</form>
```

# HTML INPUT ATTRIBUTES



# THE VALUE ATTRIBUTE

- The **value** attribute specifies the initial value for an input field.

```
<form action="">
First name:

<input type="text" name="firstname" value="Ali">

Last name:

<input type="text" name="lastname">
</form>
```

# THE READONLY ATTRIBUTE

- The **readonly** attribute specifies that the input field is read only (cannot be changed).

```
<form action="">
First name:

<input type="text" name="firstname" value ="Ali"
 readonly>

Last name:

<input type="text" name="lastname">
</form>
```

# THE DISABLED ATTRIBUTE

- The **disabled** attribute specifies that the input field is disabled.
- A disabled element is un-usuable and un-clickable.
- Disabled elements will not be submitted

```
<form action="">
```

```
First name:

```

```
<input type="text" name="firstname" value ="Ali" disabled>
```

```


```

```
Last name:

```

```
<input type="text" name="lastname">
```

```
</form>
```

# THE SIZE ATTRIBUTE

- The **size** attribute specifies the size (in characters) for the input field:

```
<form action="">
First name:

<input type="text" name="firstname" value ="Ali"
 size="40">

Last name:

<input type="text" name="lastname">
</form>
```

# THE MAXLENGTH ATTRIBUTE

- The **maxlength** attribute specifies the maximum allowed length for the input field.

```
<form action="">
First name:

<input type="text" name="firstname" maxlength="10">

Last name:

<input type="text" name="lastname">
</form>
```

# HTML5 ATTRIBUTES

- HTML5 added the following attributes for <input>:
  - autocomplete
  - autofocus
  - formaction
  - formmethod
  - formtarget
  - height and width
  - list
  - min and max
  - multiple
  - pattern (regexp)
  - placeholder
  - required
  - step
- and the following attributes for <form>:
  - autocomplete
  - novalidate

# THE AUTOCOMPLETE ATTRIBUTE

- The autocomplete attribute specifies whether a form or input field should have autocomplete on or off.
  - When autocomplete is on, the browser automatically complete values based on values that the user has entered before.
- **Tip:** It is possible to have autocomplete "on" for the form, and "off" for specific input fields, or vice versa.

# THE AUTOCOMPLETE ATTRIBUTE

```
<form action="action_page.php" autocomplete="on">
 First name: <input type="text" name="fname">

 Last name: <input type="text" name="lname">

 E-mail: <input type="email" name="email"
 autocomplete="off">

 <input type="submit">
</form>
```

# THE NOVALIDATE ATTRIBUTE

- The novalidate attribute is a <form> attribute
- When present, novalidate specifies that form data should not be validated when submitted

```
<form action="action_page.php" novalidate>
E-mail: <input type="email" name="user_email">
<input type="submit" value="Login">
</form>
```

# THE AUTOFOCUS ATTRIBUTE

- The autofocus attribute is a boolean attribute.
- When present, it specifies that an `<input>` element should automatically get focus when the page loads

```
<form action="action_page.php">
First name:<input type="text" name="fname"
autofocus>

Last name: <input type="text" name="lname">

<input type="submit">
</form>
```

# THE FORMACTION ATTRIBUTE

- The formaction attribute specifies the URL of a file that will process the input control when the form is submitted.
- The formaction attribute overrides the action attribute of the <form> element.

```
<form action="action_page.php">
```

```
First name: <input type="text" name="fname">

```

```
Last name: <input type="text" name="lname">

```

```
<input type="submit" >

```

```
<input type="submit" formaction="demo_admin.asp" >
```

```
</form>
```

# THE FORMMETHOD ATTRIBUTE

- The formmethod attribute defines the HTTP method for sending form-data to the action URL.
- The formmethod attribute overrides the method attribute of the <form> element.

```
<form action="action_page.php" method="get">
 First name: <input type="text" name="fname">

 Last name: <input type="text" name="lname">

 <input type="submit" value="Submit">
 <input type="submit" formmethod="post"
 formaction="demo_post.asp" value="Submit using
 POST">
</form>
```

# THE FORMTARGET ATTRIBUTE

- The formtarget attribute specifies a name or a keyword that indicates where to display the response that is received after submitting the form.
- The formtarget attribute overrides the target attribute of the <form> element.

```
<form action="action_page.php">
 First name: <input type="text" name="fname">

 Last name: <input type="text" name="lname">

 <input type="submit" value="Submit as normal">
 <input type="submit" formtarget="_blank"
 value="Submit to a new window/tab">
</form>
```

# THE HEIGHT AND WIDTH ATTRIBUTES

- The height and width attributes specify the height and width of an `<input>` element.
- The height and width attributes are only used with `<input type="image">`.

```
<form action="action_page.php">
First name: <input type="text" name="fname">

Last name: <input type="text" name="lname">

<input type="image" src="img_submit.gif"
alt="Submit" width="48" height="48">
</form>
```

# THE LIST ATTRIBUTE

- The list attribute refers to a `<datalist>` element that contains pre-defined options for an `<input>` element.

```
<form action="action_page.php" method="get">
<input list="browsers" name="browser">
<datalist id="browsers">
 <option value="Internet Explorer">
 <option value="Firefox">
 <option value="Chrome">
 <option value="Opera">
 <option value="Safari">
</datalist>
<input type="submit">
</form>
```

### **<select> tag**

The user can choose only one option from the given list.

The user has to scan a long list so as to select an option.

The user can be restricted to a list of options.

It doesn't provide the auto-complete feature.

### **<datalist> tag**

The user can choose any option from the given list but can also use its own input.

The user can easily input the option and get the hints and then can be chosen by the user.

The user is not restricted by the list of options.

It provides the auto-complete feature.

# THE MIN AND MAX ATTRIBUTES

- The min and max attributes specify the minimum and maximum value for an <input> element.

*<form action="action\_page.php">*

*Enter a date before 1980-01-01:*

*<input type="date" name="bday" max="1979-12-31"><br>*

*Enter a date after 2000-01-01:*

*<input type="date" name="bday" min="2000-01-02"><br>*

*Quantity (between 1 and 5):*

*<input type="number" name="quantity" min="1" max="5"><br>*

*<input type="submit">*

*</form>*

# THE MULTIPLE ATTRIBUTE

- The multiple attribute is a boolean attribute.
- When present, it specifies that the user is allowed to enter more than one value in the <input> element.
- The multiple attribute works with the following input types: email, file and select.

```
<form action="action_page.php"
 enctype="multipart/form-data">
```

```
 Select images: <input type="file" name="img"
 multiple>
```

```
 <input type="submit">
```

```
</form>
```

# THE PATTERN ATTRIBUTE

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

```
<form action="action_page.php">
```

```
 Country code: <input type="text" name="country_code"
 pattern="[A-Za-z]{3}" title="Three letter country
 code">
```

```
 <input type="submit">
```

```
</form>
```

# THE PLACEHOLDER ATTRIBUTE

- The placeholder attribute specifies a hint that describes the expected value of an input field (a sample value or a short description of the format).
- The hint is displayed in the input field before the user enters a value.

```
<form action="action_page.php">
 <input type="text" name="fname" placeholder="First
 name">

 <input type="text" name="lname" placeholder="Last
 name">

 <input type="submit" value="Submit">
</form>
```

# THE REQUIRED ATTRIBUTE

- The required attribute is a boolean attribute.
- When present, it specifies that an input field must be filled out before submitting the form.

```
<form action="action_page.php">
 Username: <input type="text" name="username"
 required>
 <input type="submit">
</form>
```

# THE STEP ATTRIBUTE

- The step attribute specifies the legal number intervals for an <input> element.
- Example: if step="3", legal numbers could be -3, 0, 3, 6, etc.
- **Tip:** The step attribute can be used together with the max and min attributes to create a range of legal values.

```
<form action="action_page.php">
 <input type="number" name="points" step="3">
 <input type="submit">
</form>
```

# THE STEP ATTRIBUTE

```
<form action="action_page.php">
 Quantity:
 <input type="number" name="points"
 min="0" max="100" step="10" value="30">
 <input type="submit">
</form>
```

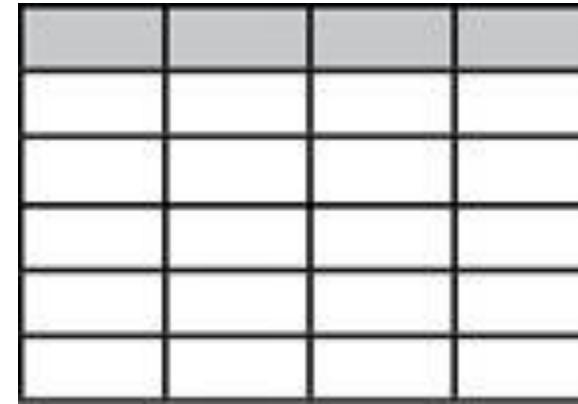


*ALL THE BEST...!*

# Table of Contents

## 1. HTML Tables

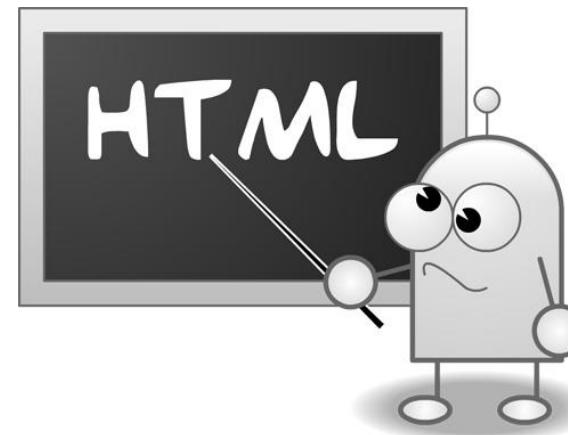
- Simple Tables
- Complete HTML Tables
- Data, Header and Footer Cells



## 2. Nested Tables

## 3. Complex Tables

- Cells Width
- Cell Spacing and Padding
- Column and Row Span



CS time	European date (D/M/Y) & time	Y-M-D date & time	Dollar	Chinese money	IP addresses	Names	Numbers
	29/10/1965	83-03-24		YMB .4	98.176.35.80		26.32 E +03
Fri Mar 22 21:48:49 UTC+0200 1957		1967-08-22 06:07:16 PM		YMB -81.38	162.117.253.34	dyse chidi	
Thu, 14 Feb 2002 04:24:20 UTC	06/07/99 06:46:01 AM	81-02-04 09:09:54 AM		YMB -108.83	122.205.50.6	bochai dychai	-191.45E-05
Monday, May 30, 1994 4:47:31 PM	06/09/05 05:11:16 AM			YMB 33.16		dydy balie	-131.20E+01
09/28/2000	24/11/1957		\$-38.77	YMB 112.42	15.192.151.209		
		97-08-13 00:01:33 AM	\$14.5	YMB -1.75	99.93.147.150	dychai tonchai	-187.28E-05
Mon, 29 Oct 1979 00:44:03 UTC		87-10-16	\$14.66	YMB 61.14		chite malie	- 125.19 E -03
Sat, 9 Jan 1982 05:45:06 UTC	04/06/68	74-10-20	\$20.47		121.169.225.22	dyma bama	138.11E+02
04/05/75		2000-03-20	\$68.84	YMB 88.19	239.133.227.68	made liete	195.44 E +03
Monday, July 15, 2002 1:05:02 AM	01/02/1961 09:40:16 AM		\$97.9	YMB 44.28	223.66.228.116	mava sete	-107
this is footer	row	number	ONE!	adsf	adsf	adsf	adsf
row 10	row	row	row	row	row	row	row
row 11	row	row	row	row	row	row	row

HTML Table Notepad

```
<html>
<head>
<title>How To Create HTML Tables</title>
</head>
<body>
<table border=1 cellspacing=0 cellpadding=0>
<tr>
<td width=110 valign=top>

upper left corner
<td width=110 valign=top>

upper right corner
</td>
</tr>
<tr>
<td width=110 valign=top>

left center cell
<td width=110 valign=top>

right center cell
</td>
</tr>
<tr>
<td width=110 valign=top>

lower left corner
<td width=110 valign=top>

lower right corner
</td>
</tr>
</table>
</body>
</html>
```

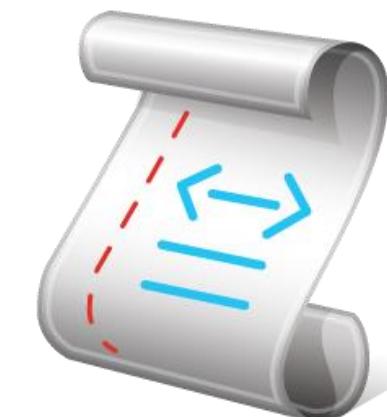
# HTML Tables

Title	Title	Title	Title	Title	Title
Data	Data	Data	Data	Data	Data
Data	Data	Data	Data	Data	Data
Data	Data	Data	Data	Data	Data
Data	Data	Data	Data	Data	Data

# HTML Tables



- Tables represent tabular data
  - A table consists of one or several rows
  - Each row has one or more columns
- Tables comprised of several core tags:
  - **<table></table>**: begin / end the table
  - **<tr></tr>**: create a table row
  - **<td></td>**: create tabular data (cell)
- Tables should not be used for layout
  - Use CSS floats and positioning styles instead



# Simple HTML Tables – Example



```
<table cellspacing="0" cellpadding="5">
```

```
 <tr>
 <td></td>
 <td>Lesson 1</td>
 </tr>
```

```
 <tr>
 <td></td>
 <td>Lesson 2</td>
 </tr>
```

```
 <tr>
 <td></td>
 <td>Lesson 2 - Demos</td>
 </tr>
```

```
</table>
```

-  [Curriculum](#)
-  [Lesson 1](#)
-  [Lesson 1 - Homework](#)
-  [Lesson 2](#)
-  [Lesson 2 - Demos](#)
-  [Lesson 2 - Homework](#)

# Data Cells and Header Cells

- Two kinds of cells in HTML tables
  - Data cells
    - Hold the table data – <td>
  - Header cells
    - Hold the column names – <th>
- Semantically separate data and headers

<b>Id</b>	<b>Name</b>	<b>Email</b>	<b>Investments</b>
231	Albert Master	albert.master@gmail.com	Bonds
210	Alfred Alan	aalan@gmail.com	Stocks
256	Alison Smart	asmart@biztalk.com	Residential Property
211	Ally Emery	allye@easymail.com	Stocks
248	Andrew Phips	andyp@mycorp.com	Stocks
234	Andy Mitchel	andym@hotmail.com	Stocks
226	Angus Robins	arobins@robins.com	Bonds
241	Ann Melan	ann_melan@iinet.com	Residential Property
225	Ben Bessel	benb@hotmail.com	Stocks
235	Bensen Romanolf	benr@albert.net	Bonds

```
<tr> <th>Full Name</th> <th>Grade</th> </tr>
<tr> <td>Mariela Anderson</td> <td>Very Good (5)</td> </tr>
<tr> <td>Georgi Georgiev</td> <td>Excellent (6)</td> </tr>
```

# Complete HTML Tables

- Table rows split into several semantic sections:
  - **<thead>** denotes the table header
    - Contains **<th>** elements, instead of **<td>** cells
  - **<tbody>** denotes collection of table rows holding table data
  - **<tfoot>** denotes table footer
    - It may come before the **<tbody>** elements, but is displayed last

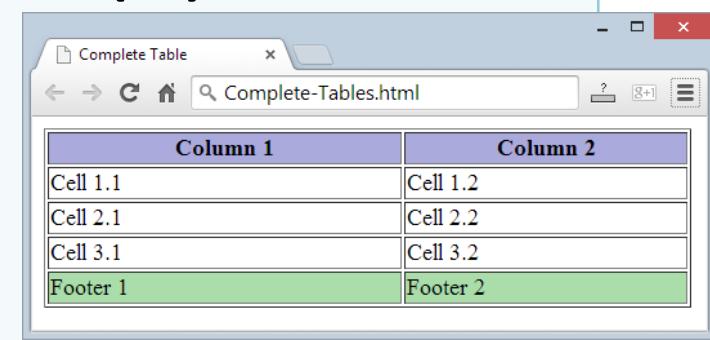
# Complete HTML Table: Example (2)

```
<table>
 <thead>
 <tr><th>Column 1</th><th>Column 2</th></tr>
 </thead>

 <tfoot>
 <tr><td>Footer 1</td><td>Footer 2</td></tr>
 </tfoot>

 <tbody>
 <tr><td>Cell 1.1</td><td>Cell 1.2</td></tr>
 <tr><td>Cell 2.1</td><td>Cell 2.2</td></tr>
 </tbody>

</table>
```



Column 1	Column 2
Cell 1.1	Cell 1.2
Cell 2.1	Cell 2.2
Cell 3.1	Cell 3.2
Footer 1	Footer 2

Although the footer is before the data in the code, it is displayed last

# Nested Tables

Tables in Tables in Tables in Tables...

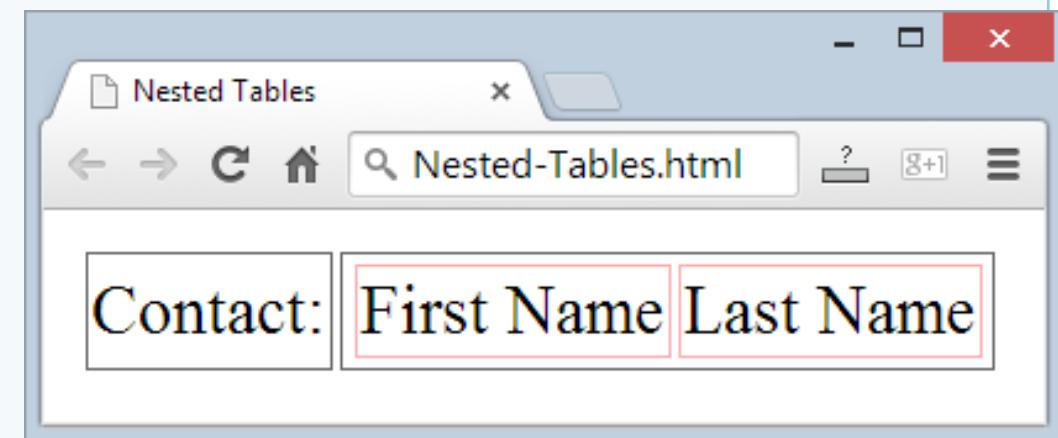


# Nested Tables



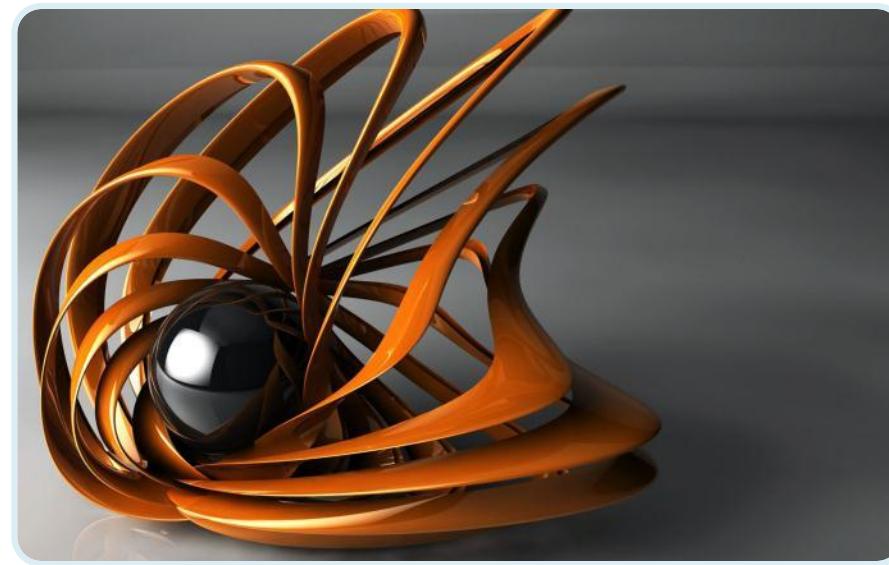
- Table "cells" (`<td>`) can contain nested tables (tables within tables):

```
<table>
 <tr>
 <td>Contact:</td>
 <td>
 <table>
 <tr>
 <td>First Name</td>
 <td>Last Name</td>
 </tr>
 </table>
 </td>
 </tr>
</table>
```



# Complex Tables

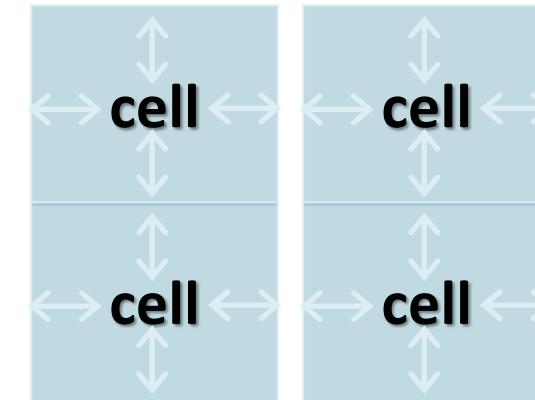
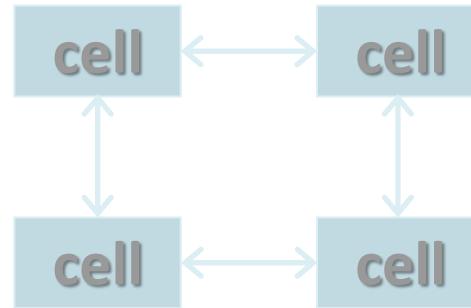
## With Padding, Spacing, Etc.



# Cell Spacing and Padding



- Tables have two attributes related to space
  - **cellspacing**
  - **cellpadding**



- Defines the empty space between cells
- Defines the empty space around the cell content

# Cell Spacing and Padding – Example

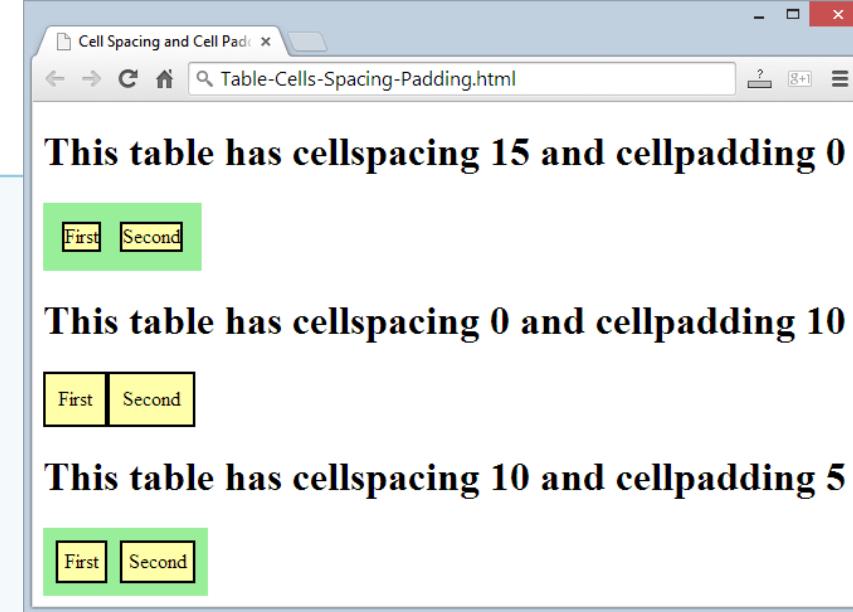
```
<html>
<head><title>Table Cells</title></head>
<body>

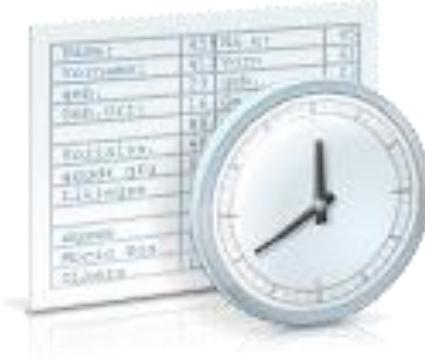
<table cellspacing="15" cellpadding="0">
 <tr><td>First</td>
 <td>Second</td></tr>
</table>

<table cellspacing="0" cellpadding="10">
 <tr><td>First</td><td>Second</td></tr>
</table>

</body>
</html>
```

Deprecated: use CSS instead!





# Row and Column Spans

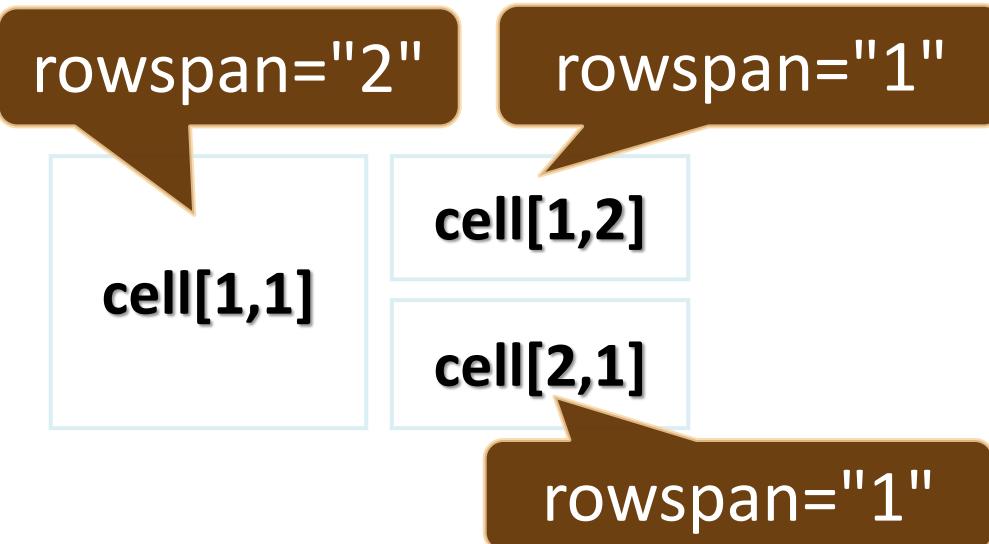
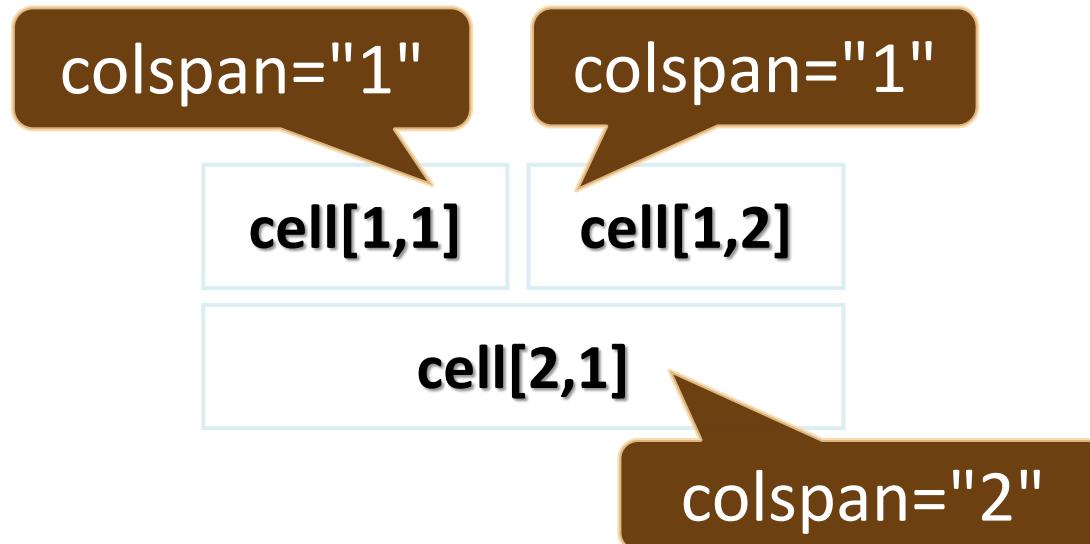
How to Make a Two-Cells Column or Row?



Cell[1,1]	Cell[2,1]	
Cell[1,2]	Cell[3,2]	
Cell[1,3]	Cell[2,2]	Cell[2,3]

# Column and Row Span

- Cells have two attributes related to merging
  - **colspan**
  - **rowspan**



- Defines how many columns the cell occupies

- Defines how many rows the cell occupies

# Column and Row Span – Example



```
<table cellspacing="0">
 <tr>
 <td>Cell[1,1]</td>
 <td colspan="2">Cell[2,1]</td>
 </tr>

 <tr>
 <td>Cell[1,2]</td>
 <td rowspan="2">Cell[2,2]</td>
 <td>Cell[3,2]</td>
 </tr>

 <tr>
 <td>Cell[1,3]</td>
 <td>Cell[2,3]</td>
 </tr>
</table>
```

Cell[1,1]	Cell[2,1]
Cell[1,2]	Cell[2,2]
Cell[1,3]	
Cell[2,3]	

# Row and Column Spans

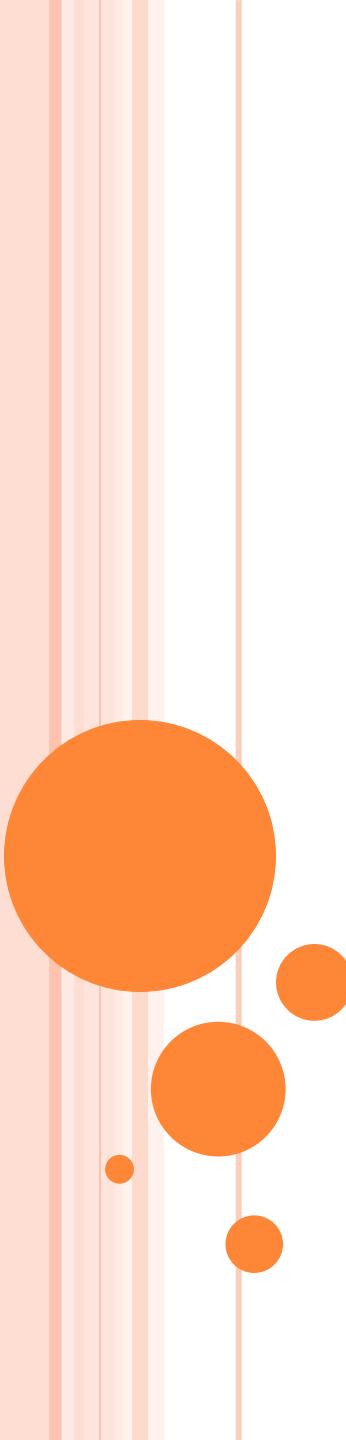
Live Demo

123			
1	2	3	+
4	5	6	-
7	8	9	*
0		.	/

# Summary

- HTML tables
  - Defined by **<table>, <tr>, <td>** tags
  - Semantic tags: **<thead>, <tbody>, <tfoot>**
  - Column / row span: **colspan, rowspan**
- Styling tables:
  - Prefer CSS
  - Old tags: **cellspacing, cellpadding, border**





# CSS

Ahmad Rabay'a  
2016  
[rabaya.ahmad@gmail.com](mailto:rabaya.ahmad@gmail.com)

# STYLING HTML WITH CSS

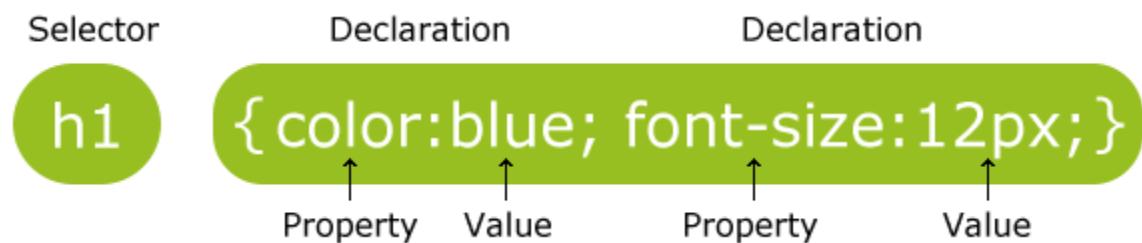
- CSS stands for **Cascading Style Sheets**
- Styling can be added to HTML elements in 3 ways:
  - ***Inline*** - using a **style attribute** in HTML elements.
  - ***Internal*** - using a **<style> element** in the HTML **<head>** section.
  - ***External*** - using one or more **external CSS files**.

# CSS SYNTAX

- CSS styling has the following **syntax**

*element { property:value ; property:value }*

- Multiple styles are separated with semicolon.
- The most common way to add styling, is to keep CSS syntax in separate CSS files.



# INLINE STYLING (INLINE CSS)

- **Inline styling** is useful for applying a unique style to a single HTML element.
  - Inline styling uses the **style attribute**.

```
<!DOCTYPE html>
<html>
<body>
```

```
<h1 style="color:blue">This is a Blue Heading</h1>
```

```
</body>
</html>
```

# INTERNAL STYLING (INTERNAL CSS)

- Internal style used to define a common style for all HTML elements on a page.
- Internal styling is defined in the **<head>** section, using a **<style>** element:

```
<!DOCTYPE html>
<html>

<head>
<style>
 body {background-color:lightgrey}
 h1 {color:blue}
 p {color:green}
</style>
</head>

<body>
 <h1>This is a heading</h1>
 <p>This is a paragraph.</p>
</body>

</html>
```

# EXTERNAL STYLING (EXTERNAL CSS)

- External style sheet is ideal when style is applied to many pages.
- With external style sheets, you can change the look of an entire site by changing one file.
- **External styles** are defined in the **<head>** section of an HTML page, in the **<link>** element.

```
<!DOCTYPE html>
<html>
 <head>
 <link rel="stylesheet" href="styles.css">
 </head>

 <body>
 <h1>This is a heading</h1>
 <p>This is a paragraph.</p>
 </body>

</html>
```

# EXTERNAL STYLING (EXTERNAL CSS)

- The external style sheet should not contain any html tags.
- The style sheet file must be saved with a .css extension.
- An example of a style sheet file called "myStyle.css", is shown below:

```
body {
 background-color: lightblue;
}
h1 {
 color: navy;
 margin-left: 20px;
}
```

# THE ID ATTRIBUTE

- CSS styles define an equal style for all equal elements.
- To define a special style for a special element, you need to add an id attribute to the element.
  - define a different style for the (identified) element:

```
p#p01 {
 color:blue;
}
```

```
<p id="p01">I am different</p>
```

# EXAMPLE OF ID ATTRIBUTE

```
<!DOCTYPE html>
<html>
<head>
<style>
p#p01 {
 color: blue;
}
</style>
</head>
<body>
<p>This is a paragraph.</p>
<p>This is a paragraph.</p>
<p>This is a paragraph.</p>
<p id="p01">I am different.</p>
</body>
</html>
```

# THE CLASS ATTRIBUTE

- To define a style for a special type (class) of elements, add a class attribute to the element.

```
p.error {
 color:red;
}
```

```
<p class="error">I am different</p>
```

# EXAMPLE OF CLASS ATTRIBUTE

```
<!DOCTYPE html>
<html>
<head>
<style>
p.error {
 color:red;
}
</style>
</head>
<body>
<p>This is a paragraph.</p>
<p>This is a paragraph.</p>
<p class="error">I am different.</p>
<p>This is a paragraph.</p>
<p class="error">I am different too.</p>
</body>
</html>
```

# CSS BASIC SELECTORS

- Use ***id*** to address ***single*** elements. Use ***class*** to address ***groups*** of elements.
- CSS selectors are used to "find" (or select) HTML elements based on their id, class, type, attribute, and more.
  - The **element Selector**
  - The **id Selector**
  - The **class Selector**

# THE ELEMENT SELECTOR

- The element selector selects elements based on the element name.

```
<style>
p {
 text-align: center;
 color: red;
}
</style>
</head>
<body>
```

*<p>Every paragraph will be affected by the style.</p>*

*<p id="para1">Me too!</p>*  
*<p>And me!</p>*

*</body>*

# THE ID SELECTOR

- The id selector uses the id attribute of an HTML element to select a specific element.
- An id should be unique within a page, so the id selector is used if you want to select a single, unique element.
- To select an element with a specific id, write a hash character, followed by the id of the element.
- **Do NOT** start an ID name with a number!

# THE ID SELECTOR

```
<style>
p#para1 {
 text-align: center;
 color: red;
}
</style>
</head>
<body>

<p id="para1">Hello World!</p>
<p>This paragraph is not affected by the style.</p>

</body>
```

# THE CLASS SELECTOR

- The class selector selects elements with a specific class attribute.
- To select elements with a specific class, write a period character, followed by the name of the class.
- You can also specify that only specific HTML elements should be affected by a class.

# THE CLASS SELECTOR

```
<style>
 .center {
 text-align: center;
 color: red;
 }
</style>
</head>
<body>

<h1 class="center">Red and center-aligned heading</h1>
<p class="center">Red and center-aligned paragraph.</p>

</body>
```

# GROUPING SELECTORS

- If you have elements with the same style definitions.
- You can group the selectors, to minimize the code.
- To group selectors, separate each selector with a comma.

# GROUPING SELECTORS

```
<style>
h1, h2, p {
 text-align: center;
 color: red;
}
</style>
</head>
<body>

<h1>Hello World!</h1>
<h2>Smaller heading!</h2>
<p>This is a paragraph.</p>

</body>
```

# EXTERNAL STYLE SHEET

- With an external style sheet, you can change the look of an entire website by changing just one file!
- Each page must include a reference to the external style sheet file inside the `<link>` element. The `<link>` element goes inside the head section:

```
<head>
<link rel="stylesheet" type="text/css"
 href="mystyle.css">
</head>
```

# EXTERNAL STYLE SHEET

- The file should not contain any html tags.
- The style sheet file must be saved with a .css extension.
- An example of a style sheet file called "myStyle.css":

```
body {
 background-color: lightblue;
}
```

```
h1 {
 color: navy;
 margin-left: 20px;
}
```

# CSS BACKGROUND

- CSS background properties are used to define the background effects of an element.
- CSS properties used for background effects:
  - background-color
  - background-image
  - background-repeat
  - background-position

# CSS BACKGROUND

```
<style>
body {
 background-color: #b0c4de;
 background-image: url("img_tree.png");
 background-repeat: no-repeat;
 background-position: right top;
}
</style>
</head>
<body>

<h1>Hello World!</h1>
</body>
```

# CSS BACKGROUND

```
<style>
body {
 background: #ffffff url("img_tree.png") no-repeat right
 top;
 margin-right: 200px;
}
</style>
</head>
<body>

<h1>Hello World!</h1>
</body>
```

# CSS TEXT

- Text Color
- Text Alignment
- Text Decoration
- Text Transformation
- Text Indentation

```
<head>
<style>
body {
 color: red; }

h1 {
 color: #00ff00;
 text-align: center;
 text-decoration: overline;
 text-transform: uppercase;}

p.ex {
 color: rgb(0,0,255);
 text-align: right;
 text-decoration: underline;
 text-transform: capitalize;
 text-indent: 50px;}
</style></head>
<body>

<h1>This is heading 1</h1>
<p>This is an ordinary paragraph..</p>
<p class="ex">This is a paragraph with class="ex". This text is
blue.</p>
</body>
```

# CSS FONT

- Font Family
- Font Style
- Font Size

```
<head>
<style>
p.normal {
 font-style: normal;
 font-family: "Times New Roman";
 font-size: 40px
}

p.italic {
 font-style: italic;
 font-family: Arial;
 font-size: 20px
}

</style>
</head>
<body>

<p class="normal">This is a paragraph in normal style.</p>
<p class="italic">This is a paragraph in italic style.</p>

</body>
```

# CSS LINKS

- Links can be styled differently depending on what **state** they are in.
- The four links states are:
  - a:link - a normal, unvisited link
  - a:visited - a link the user has visited
  - a:hover - a link when the user mouses over it
  - a:active - a link the moment it is clicked

```
<head>
<style>
/* unvisited link */
a:link {
 color: #FF0000;
 text-decoration: none;
 background-color: #B2FF99;
}

/* visited link */
a:visited {
 color: #00FF00;
 text-decoration: none;
 background-color: #FFFF85;
}

/* mouse over link */
a:hover {
 color: #FF00FF;
 text-decoration: underline;
 background-color: #FF704D;
}

/* selected link */
a:active {
 color: #0000FF;
 text-decoration: underline;
 background-color: #FF704D;
}
</style>
</head>
<body>

<p>This is PTUK link</p>

</body>
```

# CSS LISTS

- The CSS list properties allow you to:
  - Set different list item markers for ordered lists
  - Set different list item markers for unordered lists
  - Set an image as the list item marker

```
<head>
<style>
ul.a {
 list-style-type: circle;}

ul.b {
 list-style-type: square;}

ol.c {
 list-style-type: upper-roman;}

ol.d {
 list-style-type: lower-alpha;}
</style></head>
<body>

<p>Example of unordered lists:</p>
<ul class="a">
 Coffee
 Tea
 Coca Cola

<ul class="b">
 Coffee
 Tea
 Coca Cola

<p>Example of ordered lists:</p>
<ol class="c">
 Coffee
 Tea
 Coca Cola

<ol class="d">
 Coffee
 Tea
 Coca Cola

</body>
```

```
<head>
<style>
ul {
 list-style-image: url('butpurple.gif');
}
</style>
</head>
<body>

 Coffee
 Tea
 Coca Cola

</body>
```

# CSS Box MODEL

- The CSS box model is a box that wraps around HTML elements, and it consists of: margins, borders, padding, and the content.



```
<head>
<style>
div {
 background-color: lightgrey;
 width: 300px;
 padding: 25px;
 border: 25px solid navy;
 margin: 25px;
}
</style>
</head>
<body>

<div>Palestine Technical University .</div>

</body>
```



*ALL THE BEST...!*

# CSS Lecture 02

# Background

- The CSS background properties are used to define the background effects for elements.
- CSS background properties:
  - background-color
  - background-image
  - background-repeat
  - background-attachment
  - background-position

# Background Color

- The background-color property specifies the background color of an element.
- With CSS, a color is most often specified by:
  - a valid color name - like "red"
  - a HEX value - like "#ff0000"
  - an RGB value - like "rgb(255,0,0)"

```
body {
 background-color: lightblue;
}
```

# Background Image

- The background-image property specifies an image to use as the background of an element.
- By default, the image is repeated so it covers the entire element.

```
body {
 background-image: url("img_tree.png");
 background-repeat: no-repeat;
 background-position: right top;
 background-attachment: fixed;
}
```

# CSS Borders

- The CSS border properties allow you to specify the style, width, and color of an element's border.

I have borders on all sides.

I have a red bottom border.

---

I have rounded borders.

I have a blue left border.

# Border Style

- The border-style property specifies what kind of border to display.
- The following values are allowed:
  - dotted - Defines a dotted border
  - dashed - Defines a dashed border
  - solid - Defines a solid border
  - double - Defines a double border
  - none - Defines no border
  - hidden - Defines a hidden border

```
p.dotted {border-style: dotted;}
p.dashed {border-style: dashed;}
p.solid {border-style: solid;}
p.double {border-style: double;}
```

# Border Width

- The border-width property specifies the width of the four borders.

```
p.one {
 border-style: solid;
 border-width: 5px;
}

p.two {
 border-style: solid;
 border-width: medium;
}

p.three {
 border-style: solid;
 border-width: 2px 10px 4px 20px;
}
```

# Border Color

```
p.one {
 border-style: solid;
 border-color: red;
}

p.two {
 border-style: solid;
 border-color: green;
}

p.three {
 border-style: solid;
 border-color: red green blue yellow;
}
```

# Border - Individual Sides

```
p {
 border-top-style: dotted;
 border-right-style: solid;
 border-bottom-style: dotted;
 border-left-style: solid;
}
```

# Border - Shorthand Property

```
p {
 border: 5px solid red;
}
```

# Example

```
p {
 border-left: 6px solid red;
 background-color: lightgrey;
}
```

Some text

# Rounded Borders

Normal border

Round border

Rounder border

Roundest border

```
p {
 border: 2px solid red;
 border-radius: 5px;
}
```

# CSS Margins

- The CSS margin properties are used to create space around elements, outside of any defined borders.
- With CSS, you have full control over the margins. There are properties for setting the margin for each side of an element (top, right, bottom, and left).

# Margin - Individual Sides

- CSS has properties for specifying the margin for each side of an element:
  - margin-top
  - margin-right
  - margin-bottom
  - margin-left

```
p {
 margin-top: 100px;
 margin-bottom: 100px;
 margin-right: 150px;
 margin-left: 80px;
}
```

# Margin - Shorthand Property

```
p {
 margin: 25px 50px 75px 100px;
}
```

# Margins Example

```
p {
 width:300px;
 margin: auto;
 border: 1px solid red;
}
```

# CSS Padding

- The CSS padding properties are used to generate space around an element's content, inside of any defined borders.
- With CSS, you have full control over the padding. There are properties for setting the padding for each side of an element (top, right, bottom, and left).

# Padding - Individual Sides

- CSS has properties for specifying the padding for each side of an element:
  - padding-top
  - padding-right
  - padding-bottom
  - padding-left

# Padding - Shorthand Property

```
div {
 padding: 25px 50px 75px 100px;
}
```

# CSS Height and Width

- The height and width properties are used to set the height and width of an element.
- The height and width can be set to:
  - auto (this is default. Means that the browser calculates the height and width).
  - or be specified in *length values*, like px, cm.
  - or in percent (%) of the containing block.

# Setting max-width

```
div {
 max-width: 500px;
 height: 100px;
 background-color: powderblue;
}
```

**What is the difference between max-width and width properties?**

# All CSS Dimension Properties

- height
- width
- max-height
- min-height
- max-width
- min-width

# CSS Lecture 03

CSS Tables

# Table Borders

- To specify table borders in CSS, use the border property.

```
table, th, td {
 border: 1px solid black;
}
```

Firstname	Lastname
Peter	Griffin
Lois	Griffin

# Collapse Table Borders

- The border-collapse property sets whether the table borders should be collapsed into a single border:

```
table {
 border-collapse: collapse;
}

table, th, td {
 border: 1px solid black;
}
```

Firstname	Lastname
Peter	Griffin
Lois	Griffin

# Table Width and Height

- Width and height of a table are defined by the width and height properties.

```
table {
 width: 100%;
}

th {
 height: 50px;
}
```

# Horizontal Alignment

- The `text-align` property sets the horizontal alignment (like left, right, or center) of the content in `<th>` or `<td>`.
- By default, the content of
  - `<th>` elements are center-aligned
  - `<td>` elements are left-aligned.

```
th {
 text-align: left;
}
```

# Vertical Alignment

- The vertical-align property sets the vertical alignment (like top, bottom, or middle) of the content in <th> or <td>.
- By default, the vertical alignment of the content in a table is middle (for both <th> and <td> elements).

```
td {
 height: 50px;
 vertical-align: bottom;
}
```

# Table Padding

- To control the space between the border and the content in a table, use the padding property on <td> and <th> elements:

```
th, td {
 padding: 15px;
 text-align: left;
}
```

# Horizontal Dividers

```
th, td {
 border-bottom: 1px solid #ddd;
}
```

First Name	Last Name	Savings
Peter	Griffin	\$100
Lois	Griffin	\$150

# Hoverable Table

- Use the :hover selector on <tr> to highlight table rows on mouse over:

```
tr:hover {background-color: #f5f5f5;}
```

# Striped Tables

- For zebra-striped tables, use the nth-child() selector and add a background-color to all even (or odd) table rows

```
tr:nth-child(even) {background-color: #f2f2f2;}
```

First Name	Last Name	Savings
Peter	Griffin	\$100
Lois	Griffin	\$150
Joe	Swanson	\$300

# Table Color

First Name	Last Name	Savings
Peter	Griffin	\$100
Lois	Griffin	\$150
Joe	Swanson	\$300

```
th {
 background-color: #4CAF50;
 color: white;
}
```

# Responsive Table

- A responsive table will display a horizontal scroll bar if the screen is too small to display the full content
- Add a container element (like <div>) with overflow-x:auto around the <table> element to make it responsive

```
<div style="overflow-x:auto;">

 <table>
 ... table content ...
 </table>

</div>
```

# **CSS Lecture 04**

# CSS Box Model

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

# Width and Height of an Element

- **Important:** When you set the width and height properties of an element with CSS, you just set the width and height of the **content area**.
- To calculate the full size of an element, you must also add **padding**, **borders** and **margins**

```
div {
 width: 320px;
 padding: 10px;
 border: 5px solid gray;
 margin: 0;
}
```

320px (width)  
+ 20px (left + right padding)  
+ 10px (left + right border)  
+ 0px (left + right margin)  
= **350px**

# CSS Text

- color:
- text-align:{left | right | center | justify}
- text-decoration: {line-through | underline | overline | blink | .... }
- text-transform: {uppercase | lowercase | capitalize}
- text-indent: {px | pt | cm | pc | in | mm | em}
- letter-spacing: {px | pt | cm | pc | in | mm | em}
- line-height: {px | pt | cm | pc | in | mm | em}
- direction: {ltr}
- word-spacing:{px | pt | cm | pc | in | mm | em}

# CSS Units

Unit	Description
cm	centimeters
mm	millimeters
in	inches (1in = 96px = 2.54cm)
px *	pixels (1px = 1/96th of 1in)
pt	points (1pt = 1/72 of 1in)
pc	picas (1pc = 12 pt)

# CSS Links

- Links can be styled with any CSS property (e.g. color, font-family, background, etc.).
- In addition, links can be styled differently depending on what **state** they are in.
- The four links states are:
  - a:link - a normal, unvisited link
  - a:visited - a link the user has visited
  - a:hover - a link when the user mouse over it
  - a:active - a link the moment it is clicked

# CSS Lists

```
ul.a {
 list-style-type: circle;
}
```

```
ul.b {
 list-style-type: square;
}
```

```
ol.c {
 list-style-type: upper-roman;
}
```

```
ol.d {
 list-style-type: lower-alpha;
}
```

```
ul {
 list-style-image: url('sqpurple.gif');
}
```

# CSS Layout

- The display property is the most important CSS property for controlling layout.
- The display property specifies if/how an element is displayed.
- Every HTML element has a default display value depending on what type of element it is.
- The default display value for most elements is block or inline.

# Block-level Elements

- A block-level element always starts on a new line and takes up the full width available (stretches out to the left and right as far as it can).
- Examples of block-level elements:
  - <div>
  - <h1> - <h6>
  - <p>
  - <form>
  - <table>

# Inline Elements

- An inline element does not start on a new line and only takes up as much width as necessary.
- Examples of inline elements:
  - <span>
  - <a>
  - <img>

# Display: none

- `display: none;` is commonly used with JavaScript to hide and show elements without deleting and recreating them.

# Override The Default Display Value

- As mentioned, every element has a default display value. However, you can override this.
- Changing an inline element to a block element, or vice versa, can be useful for making the page look a specific way, and still follow the web standards.
- A common example is making inline <li> elements for horizontal menus:

```
li {
 display: inline;
}

a {
 display: block;
}
```

# Hide an Element

- **display : none**
  - The element will be hidden, and the page will be displayed as if the element is not there.
- **visibility : hidden**
  - The element will still take up the same space as before. The element will be hidden, but still affect the layout

# The position Property

- The position property specifies the type of positioning method used for an element.
- There are five different position values:
  - static
  - relative
  - fixed
  - absolute
  - sticky
- Elements are positioned using the top, bottom, left, and right properties.
  - However, these properties will not work unless the position property is set first.

# position: static

- HTML elements are positioned static by default.
- Static positioned elements are not affected by the top, bottom, left, and right properties.
- An element with position: static; is not positioned in any special way; it is always positioned according to the normal flow of the page.

# position: relative

- An element with position: relative; is positioned relative to its normal position.
- Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position.
- Other content will not be adjusted to fit into any gap left by the element.

```
div.relative {
 position: relative;
 left: 30px;
 border: 3px solid #73AD21;
}
```

# position: absolute

- An element with position: absolute; is positioned relative to the nearest **positioned** ancestor.
  - However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

```
div.myDiv1 {
 position: relative;
 width: 400px;
 height: 200px;
 border: 3px solid #73AD21;
}
```

This div element has position: relative;

```
div.myDiv2 {
 position: absolute;
 top: 80px;
 right: 0;
 width: 200px;
 height: 100px;
 border: 3px solid #73AD21;
}
```

This div element has position: absolute;

```
<div class="myDiv1">This div element has position: relative;
 <div class="myDiv2">This div element has position: absolute;</div>
</div>
```

# position: fixed

- An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled.
- The top, right, bottom, and left properties are used to position the element.
- A fixed element does not leave a gap in the page where it would normally have been located.

```
div.fixed {
 position: fixed;
 bottom: 0;
 right: 0;
 width: 300px;
 border: 3px solid #73AD21;
}
```

# position: sticky

- sticky; is positioned based on the user's scroll position.
- A sticky element toggles between **relative** and **fixed**, depending on the scroll position.
- you must specify at least one of top, right, bottom or left for sticky positioning to work.

```
div.sticky {
 position: sticky;
 top: 1cm;
 padding: 5px;
 background-color: #cae8ca;
 border: 2px solid #4CAF50;
}
```

Value	Description
static	Default value. Elements render in order, as they appear in the document flow
absolute	The element is positioned relative to its first positioned (not static) ancestor element
fixed	The element is positioned relative to the browser window
relative	The element is positioned relative to its normal position, so "left:20px" adds 20 pixels to the element's LEFT position
sticky	<p>The element is positioned based on the user's scroll position</p> <p>A sticky element toggles between <code>relative</code> and <code>fixed</code>, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like <code>position:fixed</code>).</p>

# Overlapping Elements

- When elements are positioned, they can overlap other elements.
- The z-index property specifies the stack order of an element (which element should be placed in front of, or behind, the others).
- An element can have a positive or negative stack order

```
img {
 position: absolute;
 left: 0px;
 top: 0px;
 z-index: -1;
}
```

# **CSS Lecture 05**

Value	Description
static	Default value. Elements render in order, as they appear in the document flow
absolute	The element is positioned relative to its first positioned (not static) ancestor element
fixed	The element is positioned relative to the browser window
relative	The element is positioned relative to its normal position, so "left:20px" adds 20 pixels to the element's LEFT position
sticky	<p>The element is positioned based on the user's scroll position</p> <p>A sticky element toggles between <code>relative</code> and <code>fixed</code>, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like <code>position:fixed</code>).</p>

# CSS Overflow

- The **overflow** property specifies whether to clip content or to add scrollbars when the content of an element is too big to fit in a specified area.
- The **overflow** property has the following values:
  - **visible** - Default. The overflow is not clipped. It renders outside the element's box
  - **hidden** - The overflow is clipped, and the rest of the content will be invisible
  - **scroll** - The overflow is clipped, but a scrollbar is added to see the rest of the content
  - **auto** - If overflow is clipped, a scrollbar should be added to see the rest of the content.
- The **overflow** property only works for block elements with a specified height.

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
 background-color: coral;
 width: 200px;
 height: 65px;
 border: 1px solid;
 overflow: visible;
}
</style>
</head>
<body>
```

## Overflow: visible

By default, the overflow is visible, meaning that it is not clipped and it renders outside the element's box:

You can use the overflow property when you want to have better control of the layout. The `overflow` property specifies what happens if content overflows an element's box.

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
 background-color: coral;
 width: 200px;
 height: 65px;
 border: 1px solid black;
 overflow: hidden;
}
</style>
</head>
<body>
```

## Overflow: hidden

With the hidden value, the overflow is clipped, and the rest of the content is hidden:

Try to remove the `overflow` property to understand how it works.

You can use the overflow property when you want to have better control of the layout. The `overflow` property

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
 background-color: coral;
 width: 200px;
 height: 100px;
 border: 1px solid black;
 overflow: scroll;
}
</style>
</head>
<body>
```

## Overflow: scroll

Setting the overflow value to scroll, the overflow is clipped and a scrollbar is added to scroll inside the box. Note that this will add a scrollbar both horizontally and vertically (even if you do not need it):

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
 background-color: coral;
 width: 200px;
 height: 65px;
 border: 1px solid black;
 overflow: auto;
}
</style>
</head>
```

## Overflow: auto

The auto value is similar to scroll, only it adds scrollbars when necessary:

You can use the overflow property when you want to have better control of the layout. The overflow

# All CSS Overflow Properties

- `overflow`
- `overflow-x`
- `overflow-y`

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
 background-color: coral;
 width: 200px;
 height: 65px;
 border: 1px solid black;
 overflow-x: hidden;
 overflow-y: scroll;
}
</style>
</head>
<body>
```

## Overflow-x and overflow-y

You can also change the overflow of content horizontally or vertically.

overflow-x specifies what to do with the left/right edges of the content.

overflow-y specifies what to do with the top/bottom edges of the content.

You can use the overflow property when you want to have better control of the layout. The overflow

# CSS Layout - float

- The float property is used for positioning and layout on web pages.
- The float property can have one of the following values:
  - left - The element floats to the left of its container
  - right- The element floats to the right of its container
  - none - The element does not float (will be displayed just where it occurs in the text). This is default
  - inherit - The element inherits the float value of its parent

# Example

```
img {
 float: right;
}
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac...



# The clear Property

- The clear property specifies what elements can float beside the cleared element and on which side.
- The clear property can have one of the following values:
  - **none**
    - Allows floating elements on both sides. This is default
  - **left**
    - No floating elements allowed on the left side
  - **Right**
    - No floating elements allowed on the right side
  - **both**
    - No floating elements allowed on either the left or the right side
  - **inherit**
    - The element inherits the clear value of its parent

- clear: none;
- clear: left;
- clear: right;
- clear: both;

This  
DIV  
floats  
left

Below is a 200px DIV with 10px padding and a 30px border. If box-sizing is "border-box", the total width of the DIV element is always 200px, but if box-sizing is "content-box", the total width is 200px plus padding and border = 280px.

And  
this  
DIV  
floats  
right

- clear: none;
- clear: left;
- clear: right;
- clear: both;

This  
DIV  
floats  
left

Below is a 200px DIV with 10px padding and a 30px border. If box-sizing is "border-box", the total width of the DIV element is always 200px, but if box-sizing is "content-box", the total width is 200px plus padding and border = 280px.

And  
this  
DIV  
floats  
right

- clear: none;
- clear: left;
- clear: right;
- clear: both;

This  
DIV  
floats  
left

Below is a 200px DIV with 10px padding and a 30px border. If box-sizing is "border-box", the total width of the DIV element is always 200px, but if box-sizing is "content-box", the total width is 200px plus padding and border = 280px.

And  
this  
DIV  
floats  
right

- clear: none;
- clear: left;
- clear: right;
- clear: both;

This  
DIV  
floats  
left

Below is a 200px DIV with 10px padding and a 30px border. If box-sizing is "border-box", the total width of the DIV element is always 200px, but if box-sizing is "content-box", the total width is 200px plus padding and border = 280px.

And  
this  
DIV  
floats  
right

# The clearfix Hack

- If an element is taller than the element containing it, and it is floated, it will overflow outside of its container.

Without Clearfix

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum...



With Clearfix

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum...



- Then we can add **overflow: auto;** to the containing element to fix this problem:

# CSS Layout - inline-block

- Is it possible to create a grid of boxes that fills the browser width and wraps nicely (when the browser is resized)?
- Yes it is possible by using one of the following :
  - float (notice that we also need to specify a clear property for the element after the floating boxes).
  - inline-block value of the display property.

# CSS Layout - inline-block (Con)

```
.floating-box {
 display: inline-block;
 width: 150px;
 height: 75px;
 margin: 10px;
 border: 3px solid #73AD21;
}
```

# CSS Layout - Horizontal & Vertical Align

- To horizontally center a block element (like <div>), use margin: auto.
- Setting the width of the element will prevent it from stretching out to the edges of its container.
- The element will then take up the specified width, and the remaining space will be split equally between the two margins.
- Center aligning has no effect if the width property is not set (or set to 100%).

```
.center {
 margin: auto;
 width: 50%;
 border: 3px solid green;
 padding: 10px;
}
```

# Center Align Text

- To just center the text inside an element, use `text-align: center;`

```
.center {
 text-align: center;
 border: 3px solid green;
}
```

# Center an Image

- To center an image, set left and right margin to auto and make it into a block element.

```
img {
 display: block;
 margin-left: auto;
 margin-right: auto;
 width: 40%;
}
```

# Left and Right Align - Using position

- One method for aligning elements is to use **position: absolute;**

```
.right {
 position: absolute;
 right: 0px;
 width: 300px;
 border: 3px solid #73AD21;
 padding: 10px;
}
```

# Left and Right Align - Using float

- Another method for aligning elements is to use the **float** property:

```
.right {
 float: right;
 width: 300px;
 border: 3px solid #73AD21;
 padding: 10px;
}
```

# CSS Combinators

- A combinator is something that explains the relationship between the selectors.
- There are four different combinators in CSS:
  - descendant selector (space)
  - child selector (>)
  - adjacent sibling selector (+)
  - general sibling selector (~)

# Descendant Selector

- The descendant selector matches all elements that are descendants of a specified element.
- The following example selects all `<p>` elements inside `<div>` elements:

```
div p {
 background-color: yellow;
}
```

```
<div>
 <p>Paragraph 1 in the div.</p>
 <p>Paragraph 2 in the div.</p>
 <p>Paragraph 3 in the div.</p>
</div>

<p>Paragraph 4. Not in a div.</p>
<p>Paragraph 5. Not in a div.</p>
```

# Child Selector

- The child selector selects all elements that are the immediate children of a specified element.
- The following example selects all `<p>` elements that are immediate children of a `<div>` element:

```
div > p {
 background-color: yellow;
}
```

```
<div>
 <p>Paragraph 1 in the div.</p>
 <p>Paragraph 2 in the div.</p>
 <p>Paragraph 3 in the div.</p>
 </div>

<p>Paragraph 4. Not in a div.</p>
<p>Paragraph 5. Not in a div.</p>
```

# Adjacent Sibling Selector

- The adjacent sibling selector selects all elements that are the adjacent siblings of a specified element.
  - Sibling elements must have the same parent element
  - "adjacent" means "immediately following".
- The following example selects all `<p>` elements that are placed immediately after `<div>` elements:

```
div + p {
 background-color: yellow;
}
```

```
<div>
 <p>Paragraph 1 in the div.</p>
 <p>Paragraph 2 in the div.</p>
</div>

<p>Paragraph 3. Not in a div.</p>
<p>Paragraph 4. Not in a div.</p>
```

# General Sibling Selector

- The general sibling selector selects all elements that are siblings of a specified element.
- The following example selects all `<p>` elements that are siblings of `<div>` elements:

```
div ~ p {
 background-color: yellow;
}
```

```
<p>Paragraph 1.</p>

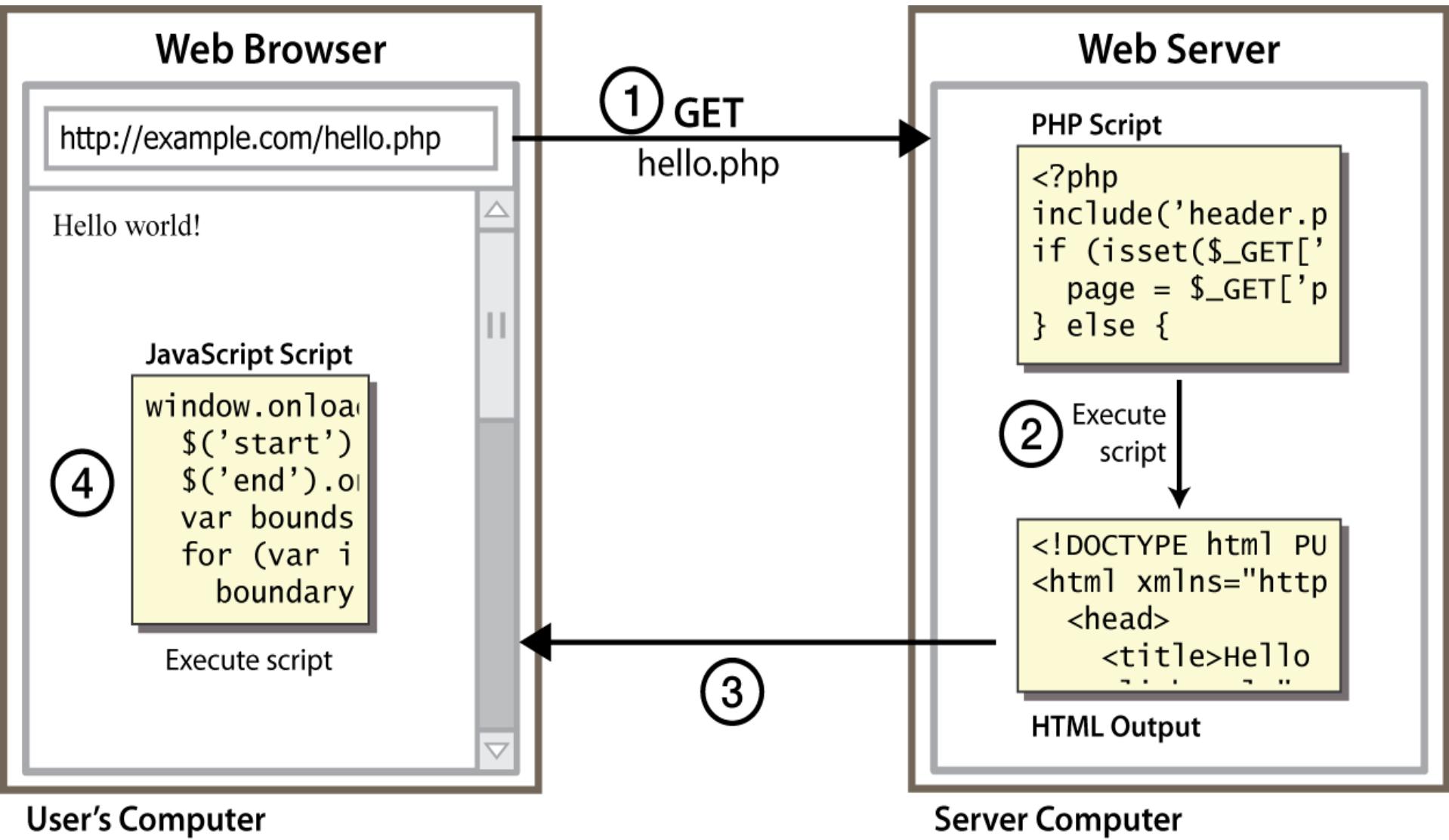
<div>
 <code>Some code.</code>
 <p>Paragraph 2.</p>
</div>

<p>Paragraph 3.</p>
<code>Some code.</code>
<p>Paragraph 4.</p>
```

# Intro to Javascript

# Client Side Scripting

2



# Why use client-side programming?

3

- client-side scripting (JavaScript) benefits:
  - **usability:** can modify a page without having to post back to the server (faster UI)
  - **efficiency:** can make small, quick changes to page without waiting for server
  - **event-driven:** can respond to user actions like clicks and key presses

# What is Javascript?

4

- a lightweight programming language ("scripting language")
  - used to make web pages interactive
  - insert dynamic text into HTML (ex: user name)
  - **react to events** (ex: page load user click)
  - get information about a user's computer (ex: browser type)
  - perform calculations on user's computer (ex: form validation)

# What is Javascript?

5

- a web standard (but not supported identically by all browsers)
- NOT related to Java other than by name and some syntactic similarities

# Javascript

6

- Interpreted, not compiled
- Relaxed syntax and rules
  - fewer and "looser" data types
  - variables don't need to be declared
  - errors often silent (few exceptions)
- contained within a web page and integrates with its HTML/CSS content

# Linking to a JavaScript file: script

7

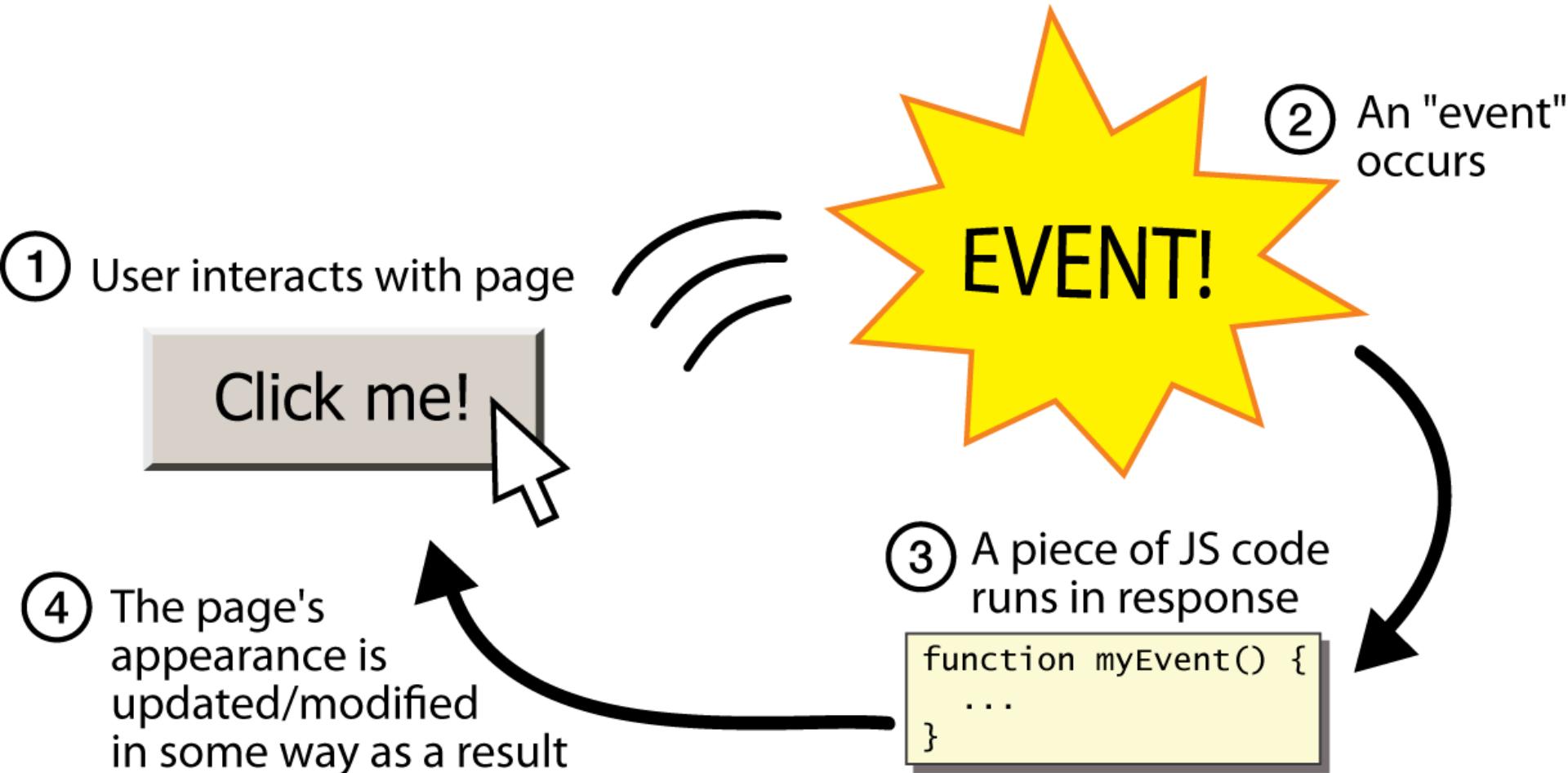
```
<script src="filename" type="text/javascript"></script>
```

HTML

- script tag should be placed in HTML page's head
- script code is stored in a separate .js file

# Event-driven programming

8

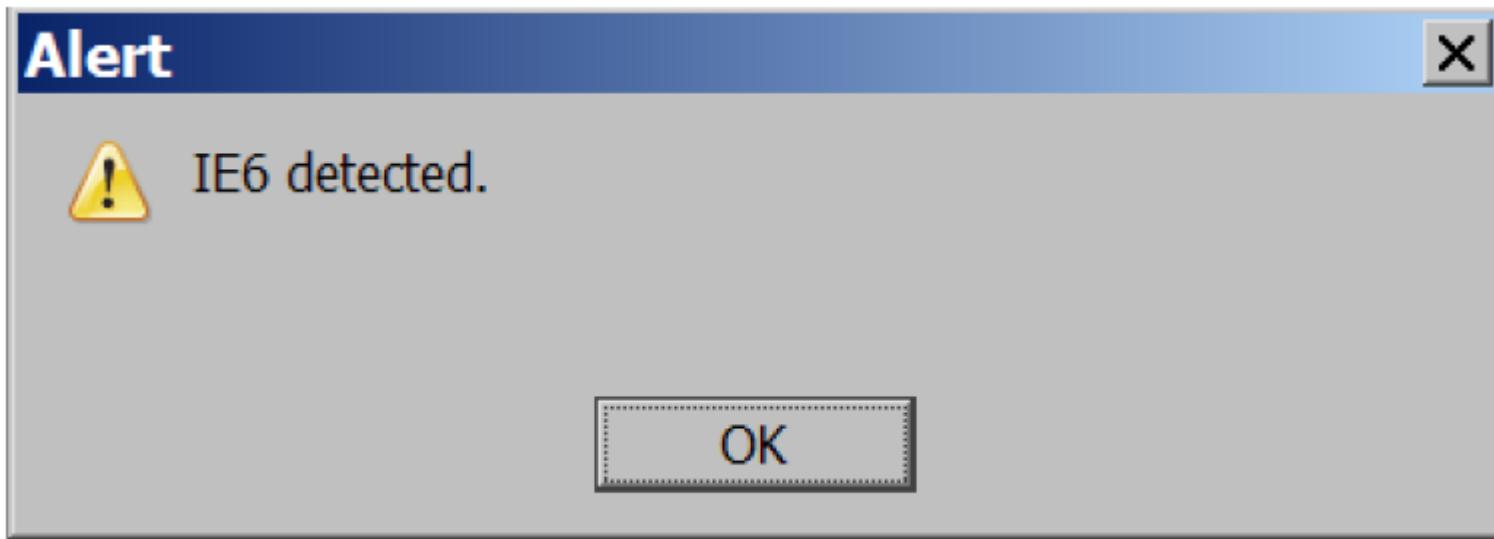


# A JavaScript statement: alert

9

```
alert("IE6 detected..");
```

JS



- a JS command that pops up a dialog box with a message

# Event-driven programming

10

- you are used to programs start with a main method (or implicit main like in PHP)
- JavaScript programs instead wait for user actions called *events* and respond to them
- event-driven programming: writing programs driven by user events
- Let's write a page with a clickable button that pops up a "Hello, World" window...

# Buttons

11

```
<button>Click me!</button>
```

HTML

- button's text appears inside tag; can also contain images
- To make a responsive button or other UI control:
  1. choose the control (e.g. button) and event (e.g. mouse 1. click) of interest
  2. write a JavaScript function to run when the event occurs
  3. attach the function to the event on the control

# JavaScript functions

12

```
function name() {
statement ;
statement ;
...
statement ;
}
```

JS

```
function myFunction() {
 alert("Hello!");
 alert("How are you?");
}
```

JS

- the above could be the contents of example.js linked to our HTML page
- statements placed into functions can be evaluated in response to user events

# Event handlers

13

```
<element attributes onclick="function() ;">...
```

HTML

```
<button onclick="myFunction() ;>Click me!</button>
```

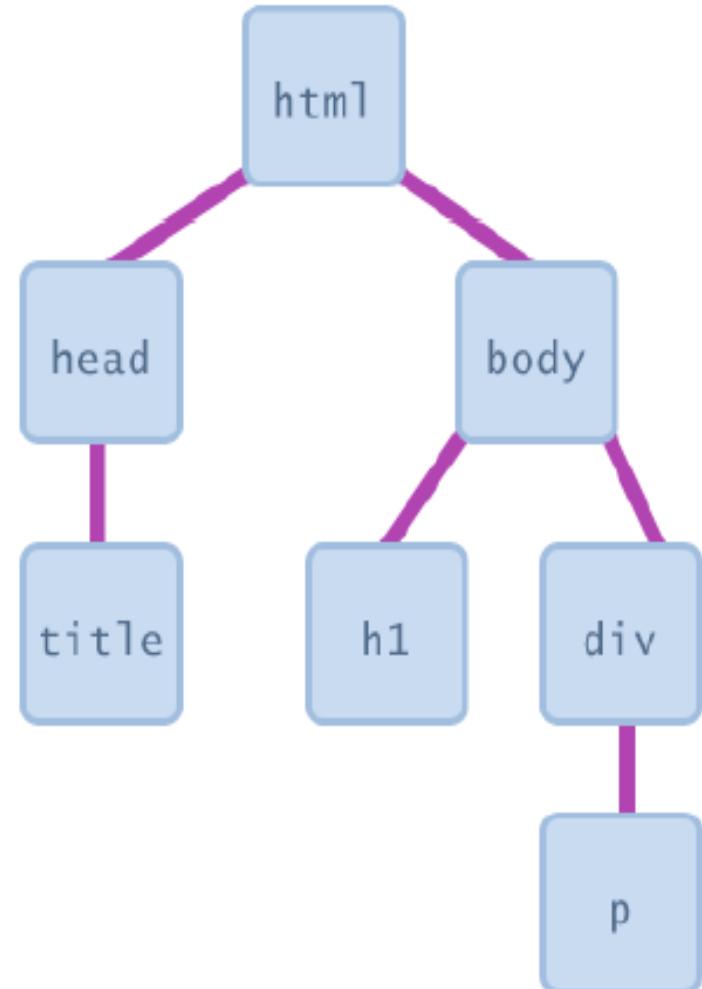
HTML

- JavaScript functions can be set as event handlers
  - when you interact with the element, the function will execute
- onclick is just one of many event HTML attributes we'll use
- but popping up an alert window is disruptive and annoying
  - A better user experience would be to have the message appear on the page...

# Document Object Model (DOM)

14

- most JS code manipulates elements on an HTML page
- we can examine elements' state
  - e.g. see whether a box is checked
- we can change state
  - e.g. insert some new text into a div
- we can change styles
  - e.g. make a paragraph red



# Accessing elements: `document.getElementById`

15

```
var name = document.getElementById("id");
```

JS

```
<button onclick="changeText();>Click me!</button>
replace me
<input id="textbox" type="text" />
```

HTML

```
function changeText() {
 var span = document.getElementById("output");
 var textBox = document.getElementById("textbox");

 span.innerHTML=textBox.value;

 textBox.style.color = "red";
}
```

JS

# DOM element objects

16

## HTML

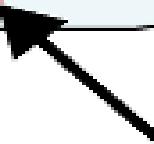
```
<p>
 Look at this octopus:

 Cute, huh?
</p>
```



### DOM Element Object

Property	Value
tagName	"IMG"
src	"octopus.jpg"
alt	"an octopus"
id	"icon01"



## JavaScript

```
var icon = document.getElementById("icon01");
icon.src = "kitty.gif";
```

# Accessing elements:

## document.getElementById

17

- `document.getElementById` returns the DOM object for an element with a given id
- can change the text inside most elements by setting the `innerHTML` property
- can change the text in form controls by setting the `value` property

# Changing element style: element.style

18

Attribute	Property or style object
color	color
padding	padding
background-color	backgroundColor
border-top-width	borderTopWidth
Font size	fontSize
Font famiy	fontFamily

```
function changeText() {
 //grab or initialize text here

 // font styles added by JS:
 text.style.fontSize = "13pt";
 text.style.fontFamily = "Comic Sans MS";
 text.style.color = "red"; // or pink?
}
```

JS

# More Javascript Syntax

# Variables

2

```
var name = expression;
```

JS

```
var clientName = "Connie Client";
var age = 32;
var weight = 127.4;
```

JS

- variables are declared with the var keyword (case sensitive)
- types are not specified, but JS does have types ("loosely typed")
  - ▣ Number, Boolean, String, Array, Object, Function, Null, Undefined
  - ▣ can find out a variable's type by calling typeof

# Number type

3

```
var enrollment = 99;
var medianGrade = 2.8;
var credits = 5 + 4 + (2 * 3);
```

JS

- integers and real numbers are the same type (no int vs. double)
- same operators: + - \* / % ++ -- = += -= \*= /= %=
- similar precedence to Java
- many operators auto-convert types: "2" \* 3 is 6

# Comments (same as Java)

4

```
// single-line comment
/* multi-line comment */
```

JS

- identical to Java's comment syntax
- recall: 4 comment syntaxes
  - HTML: <!-- comment -->
  - CSS/JS/PHP: /\* comment \*/
  - Java/JS/PHP: // comment
  - PHP: # comment

# Math object

5

```
var rand1to10 = Math.floor(Math.random() * 10 + 1);
var three = Math.floor(Math.PI);
```

JS

- **methods:** abs, ceil, cos, floor, log, max, min, pow, random, round, sin, sqrt, tan

# Special values: null and undefined

6

```
var ned = null;
var benson = 9;
// at this point in the code,
// ned is null
// benson's 9
// caroline is undefined
```

JS

- undefined : has not been declared, does not exist
- null : exists, but was specifically assigned an empty or null value

# Logical operators

7

- > , < , >=, <= , && , ||, ! , ==, != , ===, !==
- most logical operators automatically convert types:
  - 5 < "7" is true
  - 42 == 42.0 is true
  - "5.0" == 5 is true
- === and !== are strict equality tests; checks both type and value
  - "5.0" === 5 is false

# if/else statement (same as Java)

8

```
if (condition) {
 statements;
} else if (condition) {
 statements;
} else {
 statements;
}
```

JS

- identical structure to Java's if/else statement
- JavaScript allows almost anything as a condition

# Boolean type

9

```
var iLike190M = true;
var ieIsGood = "IE6" > 0; // false
if ("web devevelopment is great") { /* true */ }
if (0) { /* false */ }
```

JS

- any value can be used as a Boolean
  - "falsey" values: 0, 0.0, "", null, and undefined
  - "truthy" values: anything else
- converting a value into a Boolean explicitly:
  - var boolValue = Boolean(otherValue);
  - var boolValue = !! (otherValue);

# for loop (same as Java)

10

```
var sum = 0;
for (var i = 0; i < 100; i++) {
 sum = sum + i;
}
```

JS

```
var s1 = "hello";
var s2 = "";
for (var i = 0; i < s.length; i++) {
 s2 += s1.charAt(i) + s1.charAt(i);
}
// s2 stores "hheelllloo"
```

JS

# while loops (same as Java)

11

```
while (condition) {
 statements;
}
```

JS

```
do {
 statements;
} while (condition);
```

JS

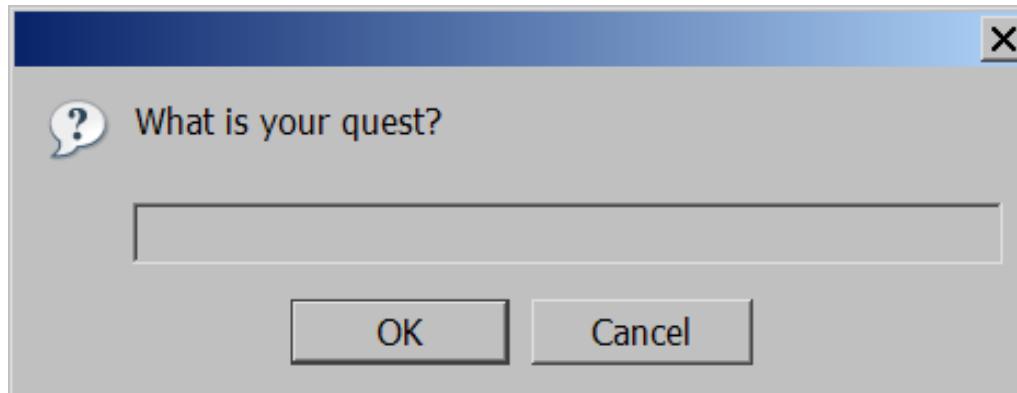
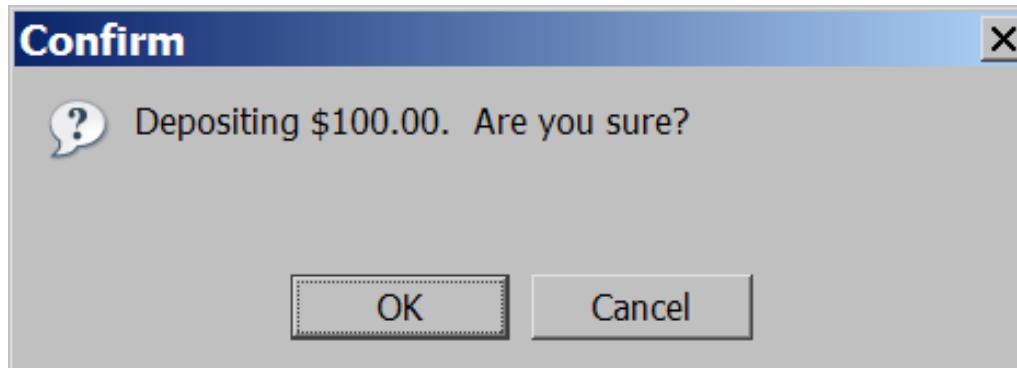
- break and continue keywords also behave as in Java

# Popup boxes

12

```
alert("message"); // message
confirm("message"); // returns true or false
prompt("message"); // returns user input string
```

JS



# Arrays

13

```
var name = [] // empty array
var name = [value, value, ..., value] // pre-filled
name[index] = value // store element
```

JS

```
var ducks = ["Huey", "Dewey", "Louie"];
var stooges = [] // stooges.length is 0
stooges[0] = "Larry"; // stooges.length is 1
stooges[1] = "Moe"; // stooges.length is 2
stooges[4] = "Curly"; // stooges.length is 5
stooges[4] = "Shemp"; // stooges.length is 5
```

JS

# Array methods

14

```
var a = ["Stef", "Jason"]; // Stef, Jason
a.push("Brian"); // Stef, Jason, Brian
a.unshift("Kelly"); // Kelly, Stef, Jason, Brian
a.pop(); // Kelly, Stef, Jason
a.shift(); // Stef, Jason
a.sort(); // Jason, Stef
```

JS

- array serves as many data structures: list, queue, stack, ...
- **methods:** concat, join, pop, push, reverse, shift, slice, sort, splice, toString, unshift
  - push and pop add / remove from back
    - shift and pop return the element that is removed
  - unshift and shift add / remove from front

```
<script language="javascript">
function Test() {
 var y=document.getElementById("I01");
 var x=y.attributes;
 for(var i=0; i<x.length;i++)
 {
 alert(x[i].name+" "+x[i].value);
 }
}
</script>

<button onclick="Test()" > Click</button>
```

```
<script language="javascript">
function Test() {
 var y=document.getElementsByTagName ("img");

 for(var j=0; j<y.length; j++)
 {
 var x=y[j].attributes;

 for(var i=0; i<x.length; i++)
 {
 alert(x[i].name+" "+x[i].value);
 }
 }
}
</script>

<button onclick="Test()" > Click</button>
```

# String type

17

```
var s = "Connie Client";
var fName = s.substring(0, s.indexOf(" ")); // "Connie"
var len = s.length; // 13
var s2 = 'Melvin Merchant';
```

JS

- **methods:** charAt, charCodeAt, fromCharCode, indexOf, lastIndexOf, replace, split, substring, toLowerCase, toUpperCase
  - charAt returns a one-letter String (there is no char type)
- Strings can be specified with “ “ or ‘ ’
- concatenation with + :
  - 1 + 1 is 2, but "1" + 1 is "11"

# More about String

18

- escape sequences behave as in Java: \' \" \& \n \t  
\\
- converting between numbers and Strings:

```
var count = 10;
var s1 = "" + count; // "10"
var s2 = count + " bananas, ah ah ah!"; // "10 bananas, ah
ah ah!"
var n1 = parseInt("42 is the answer"); // 42
var n2 = parseFloat("booyah"); // NaN
```

JS

- accessing the letters of a String:

```
var firstLetter = s[0]; // fails in IE
var firstLetter = s.charAt(0); // does work in IE
var lastLetter = s.charAt(s.length - 1);
```

JS

# Splitting strings: split and join

19

```
var s = "the quick brown fox";
var a = s.split(" "); // ["the", "quick", "brown", "fox"]
a.reverse(); // ["fox", "brown", "quick", "the"]
s = a.join("!"); // "fox!brown!quick!the"
```

JS

- **split** breaks apart a string into an array using a delimiter
  - can also be used with regular expressions (seen later)
- **join** merges an array into a single string, placing a delimiter between them

# JavaScript

- **What?**

Client-side scripting language

- **Why?**

- control document appearance
- control browser
- interact with document content
- interact with user
- read/write cookies

# JavaScript Objects

- **Objects**  
Examples:
  - window
  - document
  - form
  - location
  - history
- **Object: Properties**
  - window.location
  - document.title
- **Object: Methods**
  - document.write
  - window.open
  - form.submit

# Event Attributes

- onLoad / onUnload
- onMouseover / onMouseout
- onClick
- onChange
- onFocus
- onSelect

# JavaScript in HTML

- Document Head or Body  
`<script> ... </script>`
- External Script`<SCRIPT src="url">`  
src attribute to refer to external JavaScript document.
- "Inline" scripts as values of event attributes

This **SCRIPT** element is placed at the end of the HTML document:

```
<SCRIPT type="text/javascript">
 document.write(
 "",
 document.title, // write title
 "",
 "
",
 window.location, // write location
 ""
)
</SCRIPT>
```

The following **SCRIPT** element is placed within the **HEAD** element:

```
<SCRIPT type="text/javascript">
function WriteFooter() {
 document.write(
 "",
 document.title, // write title
 "",
 "
",
 document.URL, // write url
 ""
)
}
</SCRIPT>
```

Then, the function, *WriteFooter*, is called upon at the end of the document:

```
<SCRIPT type="text/javascript">
 WriteFooter();
</SCRIPT>
```

The following `SCRIPT` element is placed within the `HEAD` element:

```
<SCRIPT
 type="text/javascript"
 src="footer.js">
</SCRIPT>
```

The function, `WriteFooter`, is contained within `footer.js`:

```
function WriteFooter() {
 document.write(
 "",
 document.title, // write document title
 "",
 "
",
 document.URL, // write document url
 ""
)
}
```

# Form Validation

Enter your name:

---

```
<SCRIPT type="text/javascript">
function Validate(thisform) {
 if(thisform.YourName.value == null || thisform.YourName.value == "") {
 alert("Please enter your name!");
 thisform.YourName.focus();
 return false;
 }
}
</SCRIPT>
```

---

```
<FORM action="http://www.fas.harvard.edu/cgi-bin/dph/echo.pl"
method="GET" onsubmit="return Validate(this)">
 Enter your name: <INPUT type="text" name="YourName">

 <INPUT type="submit" value="Submit Information">
</FORM>
```

# Image Rollover Example

## Using the *images* object

```
<a href="http://www.courses.harvard.edu/~cscie12/syllabus/"
 onMouseover="document.images[0].src='syllabus2.gif'"
 onMouseout="document.images[0].src='syllabus1.gif'"
>
<IMG src="syllabus1.gif"
 alt="[Syllabus]"
>
```

# **Dynamic HTML (DHTML)**

- HTML
- CSS
- DOM
- JavaScript

# DHTML Example: Visible and Hidden

```

Hide Menu
Show Menu

<div style="visibility: hidden" id="menu">
Harvard University

Division of Continuing Education

Extension School

Fundamentals of Web Site Development
</div>
```

# What is jQuery?

- jQuery is a lightweight, "write less, do more", JavaScript library.
- The purpose of jQuery is to make it much easier to use JavaScript on your website.
- jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code.
- jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation

# Adding jQuery to Your Web Pages

- There are several ways to start using jQuery on your web site. You can:
  - Download the jQuery library from [jQuery.com](http://jQuery.com)
  - Include jQuery from a CDN, like Google

# Downloading jQuery

- jQuery can be downloaded from [jQuery.com](https://jquery.com).
- The jQuery library is a single JavaScript file, and you reference it with the HTML <script> tag (notice that the <script> tag should be inside the <head> section):

```
<head>
<script src="jquery-3.3.1.min.js"></script>
</head>
```

# jQuery CDN

- If you don't want to download and host jQuery yourself, you can include it from a CDN (Content Delivery Network).
- Both Google and Microsoft host jQuery.

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
</head>
```

```
<head>
<script src="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-3.3.1.min.js"></script>
</head>
```

# jQuery Syntax

- Basic syntax is: **`$(selector).action()`**
  - A \$ sign to define/access jQuery
  - A (*selector*) to "query (or find)" HTML elements
  - A jQuery *action*() to be performed on the element(s)
- Examples:
  - `$(this).hide()` - hides the current element.
  - `$("p").hide()` - hides all `<p>` elements.
  - `$(".test").hide()` - hides all elements with class="test".
  - `$("#test").hide()` - hides the element with id="test".

# The Document Ready Event

```
$(document).ready(function(){
 // jQuery methods go here...
});
```

# The Document Ready Event

- All jQuery methods in our examples, are inside a document ready event:
  - This is to prevent any jQuery code from running before the document is finished loading (is ready).
- It is good practice to wait for the document to be fully loaded and ready before working with it.
  - This also allows you to have your JavaScript code before the body of your document, in the head section.
- Here are some examples of actions that can fail if methods are run before the document is fully loaded:
  - Trying to hide an element that is not created yet
  - Trying to get the size of an image that is not loaded yet

# jQuery Selectors

- jQuery selectors allow you to select and manipulate HTML element(s).
- jQuery selectors are used to "find" (or select) HTML elements based on their name, id, classes, types, attributes, values of attributes and much more.
- It's based on the existing CSS Selectors, and in addition, it has some own custom selectors.

# The element Selector

- The jQuery element selector selects elements based on the element name.

```
$document).ready(function(){
 $("button").click(function(){
 $("p").hide();
 });
});
```

# The #id Selector

```
$(document).ready(function(){
 $("button").click(function(){
 $("#test").hide();
 });
});
```

# The .class Selector

```
$(document).ready(function(){
 $("button").click(function(){
 $(".test").hide();
 });
});
```

Syntax	Description
<code>\$("*")</code>	Selects all elements
<code>\$(this)</code>	Selects the current HTML element
<code>\$(".p.intro")</code>	Selects all <code>&lt;p&gt;</code> elements with <code>class="intro"</code>
<code>\$(".p:first")</code>	Selects the first <code>&lt;p&gt;</code> element
<code>\$(".ul li:first")</code>	Selects the first <code>&lt;li&gt;</code> element of the first <code>&lt;ul&gt;</code>
<code>\$(".ul li:first-child")</code>	Selects the first <code>&lt;li&gt;</code> element of every <code>&lt;ul&gt;</code>
<code>\$("[href]")</code>	Selects all elements with an <code>href</code> attribute
<code>\$(".a[target='_blank']")</code>	Selects all <code>&lt;a&gt;</code> elements with a <code>target</code> attribute value equal to <code>"_blank"</code>
<code>\$(".a[target!='_blank'])")</code>	Selects all <code>&lt;a&gt;</code> elements with a <code>target</code> attribute value NOT equal to <code>"_blank"</code>
<code>\$(":button")</code>	Selects all <code>&lt;button&gt;</code> elements and <code>&lt;input&gt;</code> elements of <code>type="button"</code>
<code>\$(".tr:even")</code>	Selects all even <code>&lt;tr&gt;</code> elements
<code>\$(".tr:odd")</code>	Selects all odd <code>&lt;tr&gt;</code> elements

# jQuery Event Methods

- What are Events?
  - All the different visitors' actions that a web page can respond to are called events.
  - An event represents the precise moment when something happens.
  - Examples:
    - moving a mouse over an element
    - selecting a radio button
    - clicking on an element

# jQuery Event Methods

<b>Mouse Events</b>	<b>Keyboard Events</b>	<b>Form Events</b>	<b>Document/Window Events</b>
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

# Commonly Used jQuery Event Methods

- `$(document).ready()`
- `click()`
- `dblclick()`
- `mouseenter()`
- `mouseleave()`
- `mousedown()`
- `mouseup()`
- `focus()`
- `blur()`
- `hover()`
- `on()`

# hover()

```
$("p").hover
(
 function()
 {
 $(this).css("background-color", "yellow");
 },

 function()
 {
 $(this).css("background-color", "pink");
 }
);
```

# on()

```
$("p").on({
 mouseenter: function(){
 $(this).css("background-color", "lightgray");
 },
 mouseleave: function(){
 $(this).css("background-color", "lightblue");
 },
 click: function(){
 $(this).css("background-color", "yellow");
 }
});
```

# jQuery Effects - Hide and Show

```
<head>
<script src="jquery-3.3.1.js"></script>
<script>
$(document).ready(function() {
 $("#hide").click(function() {
 $("p").hide();
 });
 $("#show").click(function() {
 $("p").show();
 });
});
</script>
</head>
<body>

<p>If you click on the "Hide" button, I will disappear.</p>

<button id="hide">Hide</button>
<button id="show">Show</button>
```

# jQuery Effects - Fading

- With jQuery you can fade an element in and out of visibility.
- jQuery has the following fade methods:
  - fadeIn()
  - fadeOut()
  - fadeToggle()
  - fadeTo()

```
$("button").click(function(){
 $("#div1").fadeIn();
 $("#div2").fadeIn("slow");
 $("#div3").fadeIn(3000);
});
```

# Bootstrap

# What is Bootstrap?

- Bootstrap is a free front-end framework for faster and easier web development.
- Bootstrap includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, modals, image carousels and many other, as well as optional JavaScript plugins.
- Bootstrap also gives you the ability to easily create responsive design.

# What is Responsive Web Design?

- Responsive web design is about creating web sites which automatically adjust themselves to look good on all devices, from small phones to large desktops.

# Why Use Bootstrap?

- **Easy to use:**
  - Anybody with just basic knowledge of HTML and CSS can start using Bootstrap.
- **Responsive features:**
  - Bootstrap's responsive CSS adjusts to phones, tablets, and desktops.
- **Mobile-first approach:**
  - In Bootstrap 3, mobile-first styles are part of the core framework.
- **Browser compatibility:**
  - Bootstrap is compatible with all modern browsers (Chrome, Firefox, Internet Explorer, Safari, and Opera)

# Where to Get Bootstrap?

- Download Bootstrap from [getbootstrap.com](http://getbootstrap.com).
- Include Bootstrap from a CDN.

# Bootstrap CDN

```
<!-- Latest compiled and minified CSS -->
<link rel="stylesheet"
 href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

<!-- jQuery library -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<!-- Latest compiled JavaScript -->
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
```

# One advantage of using the Bootstrap CDN:

- Many users already have downloaded Bootstrap from MaxCDN when visiting another site.
  - As a result, it will be loaded from cache when they visit your site, which leads to faster loading time.
- Also, most CDN's will make sure that once a user requests a file from it, it will be served from the server closest to them, which also leads to faster loading time.

# jQuery

- Bootstrap uses jQuery for JavaScript plugins (like modals, tooltips, etc).
- However, if you just use the CSS part of Bootstrap, you don't need jQuery.

```
<!-- jQuery library -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
```

# Create Web Page With Bootstrap

## 1. Add the HTML5 doctype

- Bootstrap uses HTML elements and CSS properties that require the HTML5 doctype. Always include the HTML5 doctype at the beginning of the page, along with the lang attribute and the correct character set:

```
<!DOCTYPE html>
<html lang="en">
 <head>
 <meta charset="utf-8">
 </head>
</html>
```

# Create Web Page With Bootstrap (Con.)

## 2. Bootstrap 3 is mobile-first:

- Bootstrap 3 is designed to be responsive to mobile devices.
- Mobile-first styles are part of the core framework.
- To ensure proper rendering and touch zooming, add the following <meta> tag inside the <head> element:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

# Create Web Page With Bootstrap

## 3. Containers:

- Bootstrap also requires a containing element to wrap site contents.
- There are two container classes to choose from:
  - The `.container` class provides a responsive **fixed width container**.
  - The `.container-fluid` class provides a **full width container**, spanning the entire width of the viewport.
- **Note:** Containers are not nestable (you cannot put a container inside another container).

# **BOOTSTRAP GRIDS**

# Bootstrap Grids

# Grid Classes

- The Bootstrap grid system has four classes:
  - xs (for phones - screens less than 768px wide)
  - sm (for tablets - screens equal to or greater than 768px wide)
  - md (for desktops - screens equal to or greater than 992px wide)
  - lg (for larger desktops - screens equal to or greater than 1200px wide)
- The classes above can be combined to create more dynamic and flexible layouts.
- **Tip:** Each class scales up, so if you wish to set the same widths for xs and sm, you only need to specify xs.

# Grid System Rules

- Rows must be placed within a:
  - .container
  - .container-fluid
- Use rows to create horizontal groups of columns.
- Content should be placed within columns
- Columns create gutters (gaps between column content) via padding.
- Grid columns are created by specifying the number of 12 available columns you wish to span.
  - For example, three equal columns would use three .col-sm-4

# Bootstrap Grid - Stacked-to-horizontal

- The following example shows a simple "stacked-to-horizontal" two-column layout, meaning it will result in a 50%/50% split on all screens, except for extra small screens, which it will automatically stack (100%):

```
<div class="container">
 <div class="row">
 <div class="col-sm-6" style="background-color:yellow;">
 </div>
 <div class="col-sm-6" style="background-color:pink;">
 </div>
 </div>
 </div>
```

# Example

```
<div class="container-fluid">
 <div class="row">
 <div class="col-sm-3 col-md-6" style="background-color:yellow;">
 </div>
 <div class="col-sm-9 col-md-6" style="background-color:pink;">
 </div>
 </div>
 </div>
```

# Example

```
<div class="row">
 <div class="col-sm-3">.col-sm-3</div>
 <div class="col-sm-6">.col-sm-6</div>
 <div class="col-sm-3">.col-sm-3</div>
</div>
```

# Example

```
<div class="row">
 <div class="col-sm-8">
 .col-sm-8
 <div class="row">
 <div class="col-sm-6">.col-sm-6</div>
 <div class="col-sm-6">.col-sm-6</div>
 </div>
 </div>
 <div class="col-sm-4">.col-sm-4</div>
</div>
```

# Example

```
<div class="row">
 <div class="col-xs-9 col-md-7">.col-xs-9 .col-md-7</div>
 <div class="col-xs-3 col-md-5">.col-xs-3 .col-md-5</div>
</div>

<div class="row">
 <div class="col-xs-6 col-md-10">.col-xs-6 .col-md-10</div>
 <div class="col-xs-6 col-md-2">.col-xs-6 .col-md-2</div>
</div>

<div class="row">
 <div class="col-xs-6">.col-xs-6</div>
 <div class="col-xs-6">.col-xs-6</div>
</div>
```

# **CONTEXTUAL COLORS AND BACKGROUNDS**

# Contextual Colors and Backgrounds

- Bootstrap has some contextual classes that can be used to provide "meaning through colors".

- Some of the text color classes are:

- .text-muted
- .text-primary
- .text-success
- .text-info
- .text-warning
- .text-danger

**And more**

This text is muted.

This text is important.

This text indicates success.

This text represents some information.

This text represents a warning.

This text represents danger.

# Contextual Colors and Backgrounds (Con)

- The classes for background colors are:
  - .bg-primary
  - .bg-success,
  - .bg-info,
  - .bg-warning
  - .bg-danger:

This text is important.

This text indicates success.

This text represents some information.

This text represents a warning.

This text represents danger.

# **BOOTSTRAP TABLES**

# Bootstrap Basic Table

```
<table class="table">
```

```

```

```
</table>
```

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

# Striped Rows Table

```
<table class="table table-striped">

</table>
```

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

# Bordered Table

```
<table class="table table-bordered">

</table>
```

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

# Hover Rows

```
<table class="table table-hover">

</table>
```

# Condensed Table

```
<table class="table table-condensed">

</table>
```

The **.table-condensed** class makes a table more compact by cutting cell padding in half:

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

# Contextual Classes

- Contextual classes can be used to color table rows (`<tr>`) or table cells (`<td>`):

Firstname	Lastname	Email
Default	Defaultson	def@somemail.com
Success	Doe	john@example.com
Danger	Moe	mary@example.com
Info	Dooley	july@example.com
Warning	Refs	bo@example.com
Active	Activeson	act@example.com

# Contextual Classes (Con.)

- The contextual classes that can be used are:

Class	Description
.active	Applies the hover color to the table row or table cell
.success	Indicates a successful or positive action
.info	Indicates a neutral informative change or action
.warning	Indicates a warning that might need attention
.danger	Indicates a dangerous or potentially negative action

# Responsive Tables

- The **.table-responsive** class creates a responsive table.
- The table will then scroll horizontally on small devices (under 768px).
- When viewing on anything larger than 768px wide, there is no difference on the layout.

```
<div class="table-responsive">
 <table class="table">
 ...
 </table>
</div>
```

# **BOOTSTRAP IMAGES**

# Bootstrap Image Shapes

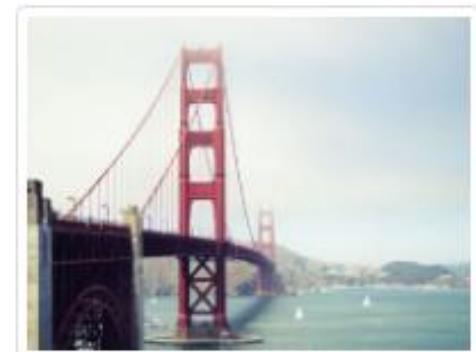
Rounded Corners:



Circle:



Thumbnail:



```



```

# Responsive Images

- Responsive images automatically adjust to fit the size of the screen.
- Create responsive images by adding an **.img-responsive** class to the `<img>` tag. The image will then scale nicely to the parent element.

```

```

# Image Gallery

- You can also use Bootstrap's grid system in conjunction with the **.thumbnail** class to create an image gallery.



*Lorem ipsum donec id elit non mi  
porta gravida at eget metus.*



*Lorem ipsum donec id elit non mi  
porta gravida at eget metus.*



*Lorem ipsum donec id elit non mi  
porta gravida at eget metus.*

# Image Gallery (Con.)

```
<div class="col-md-4">
 <div class="thumbnail">

 <div class="caption">
 <p>Lorem ipsum...</p>
 </div>

 </div>
</div>
```

# well

- The **.well** class adds a rounded border around an element with a gray background color and some padding:

```
<div class="well">Basic Well</div>
```

Basic Well

# Well Size

Small Well

Normal Well

Large Well

```
<div class="well well-sm">Small Well</div>
<div class="well well-lg">Large Well</div>
```

# Alerts

**Success!** This alert box could indicate a successful or positive action.

**Info!** This alert box could indicate a neutral informative change or action.

**Warning!** This alert box could indicate a warning that might need attention.

**Danger!** This alert box could indicate a dangerous or potentially negative action.

# Alerts (Con.)

```
<div class="alert alert-success">
 Success! Indicates a successful or positive action.
</div>

<div class="alert alert-info">
 Info! Indicates a neutral informative change or action.
</div>

<div class="alert alert-warning">
 Warning! Indicates a warning that might need attention.
</div>

<div class="alert alert-danger">
 Danger! Indicates a dangerous or potentially negative action.
</div>
```

# Alert Links

- Add the alert-link class to any links inside the alert box to create "matching colored links"

**Success!** You should [read this message](#).

```
<div class="alert alert-success">
 Success! You should read this message.
</div>
```

# Closing Alerts

**Success!** This alert box could indicate a successful or positive action.



**Info!** This alert box could indicate a neutral informative change or action.



**Warning!** This alert box could indicate a warning that might need attention.



**Danger!** This alert box could indicate a dangerous or potentially negative action.



# Closing Alerts (Con.)

```
<div class="alert alert-success alert-dismissible">
 ×
 Success! Indicates a successful or positive action.
</div>
```

# Animated Alerts

- The .fade and .in classes adds a fading effect when closing the alert message:

```
<div class="alert alert-danger fade in">
```

# Button Styles

Basic

Default

Primary

Success

Info

Warning

Danger

Link

- The button classes can be used on an `<a>`, `<button>`, or `<input type="button">` element:
  - `.btn`
  - `.btn-default`
  - `.btn-primary`
  - `.btn-success`
  - `.btn-info`
  - `.btn-warning`
  - `.btn-danger`
  - `.btn-link`

# Button Styles (Con.)

```
<button type="button" class="btn">Basic</button>
<button type="button" class="btn btn-default">Default</button>
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-info">Info</button>
<button type="button" class="btn btn-warning">Warning</button>
<button type="button" class="btn btn-danger">Danger</button>
<button type="button" class="btn btn-link">Link</button>
```

---

# Button Styles (Con.)

```
Link Button
<button type="button" class="btn btn-info">Button</button>
<input type="button" class="btn btn-info" value="Input Button">
<input type="submit" class="btn btn-info" value="Submit Button">
```

Link Button

Button

Input Button

Submit Button

# Button Sizes

Large

Medium

Small

XSmall

```
<button type="button" class="btn btn-primary btn-lg">Large</button>
<button type="button" class="btn btn-primary btn-md">Medium</button>
<button type="button" class="btn btn-primary btn-sm">Small</button>
<button type="button" class="btn btn-primary btn-xs">XSmall</button>
```

# Block Level Buttons

Button 1

Button 2

```
<button type="button" class="btn btn-primary btn-block">Button 1</button>
<button type="button" class="btn btn-default btn-block">Button 2</button>
```

# Active/Disabled Buttons

Active Primary

Disabled Primary

```
<button type="button" class="btn btn-primary active">Active Primary</button>
<button type="button" class="btn btn-primary disabled">Disabled Primary</button>
```

# Button Groups

```
Apple Samsung Sony
```

```
<div class="btn-group">
 <button type="button" class="btn btn-primary">Apple</button>
 <button type="button" class="btn btn-primary">Samsung</button>
 <button type="button" class="btn btn-primary">Sony</button>
</div>
```

# Button Groups (Con.)

```
<div class="btn-group btn-group-lg">
 <button type="button" class="btn btn-primary">Apple</button>
 <button type="button" class="btn btn-primary">Samsung</button>
 <button type="button" class="btn btn-primary">Sony</button>
</div>
```

# Button Groups (Con.)

```
<div class="btn-group-vertical">
 <button type="button" class="btn btn-primary">Apple</button>
 <button type="button" class="btn btn-primary">Samsung</button>
 <button type="button" class="btn btn-primary">Sony</button>
</div>
```



# Button Groups (Con.)

```
<div class="btn-group">
 <button type="button" class="btn btn-primary">Apple</button>
 <button type="button" class="btn btn-primary">Samsung</button>
 <div class="btn-group">
 <button type="button" class="btn btn-primary dropdown-toggle" data-toggle="dropdown">
 Sony </button>
 <ul class="dropdown-menu" role="menu">
 Tablet
 Smartphone

 </div>
</div>
```



# Justified Button Groups

```
<div class="btn-group btn-group-justified">
 Apple
 Samsung
 Sony
</div>
```

Apple

Samsung

Sony

# Glyphicon

Glyphicon can be used in text, buttons, toolbars, navigation, forms, etc.

Here are some examples of glyphicons:

Envelope glyphicon: 

Print glyphicon: 

Search glyphicon: 

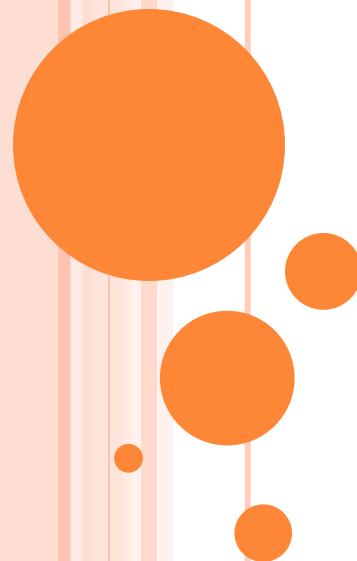
Download glyphicon: 

```

```

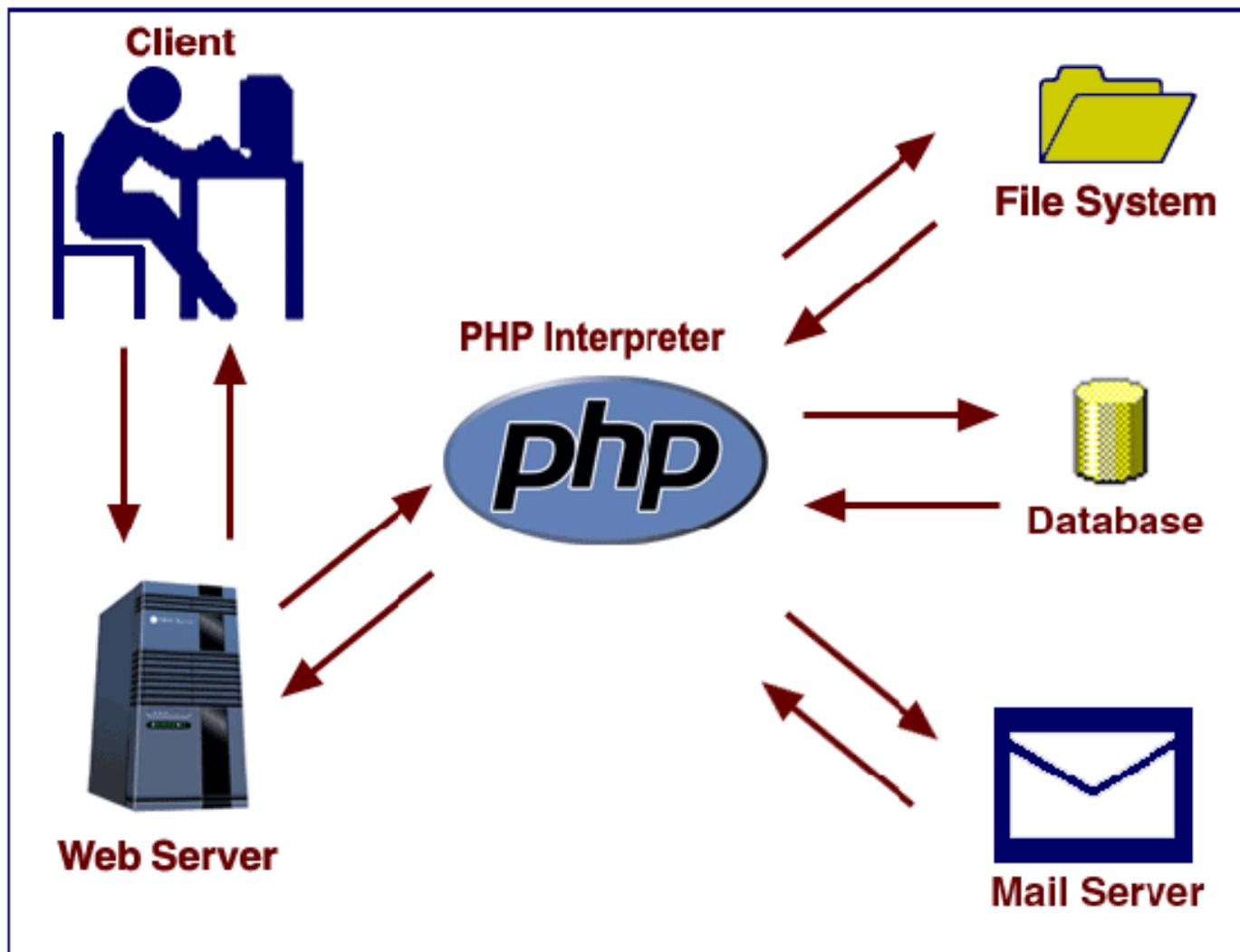
# LECTURE O1

PHP



Ahmad Rabay'a  
2015  
[rabaya.ahmad@gmail.com](mailto:rabaya.ahmad@gmail.com)

# SERVER SIDE SCRIPTING



# SERVER SIDE SCRIPTING

- PHP is a server scripting language, and a powerful tool for making dynamic and interactive Web pages.
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use



# PHP FILE

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code are executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php“



# PHP CAN DO

- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data



# WHY PHP

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free. Download it from the official PHP resource: [www.php.net](http://www.php.net)
- PHP is easy to learn and runs efficiently on the server side

# PHP

- PHP Started in 1994
- W3Techs reports that, as of October 2022, "PHP is used by 74.4% of all the websites whose server-side programming language. PHP version 7.4 is the most used version.
- In the past, we had to install PHP alone, then MySQL, then Apache, and at the end, you connect PHP with MySQL.
- Now, there are packages that contains the three components, and you install them in one step.

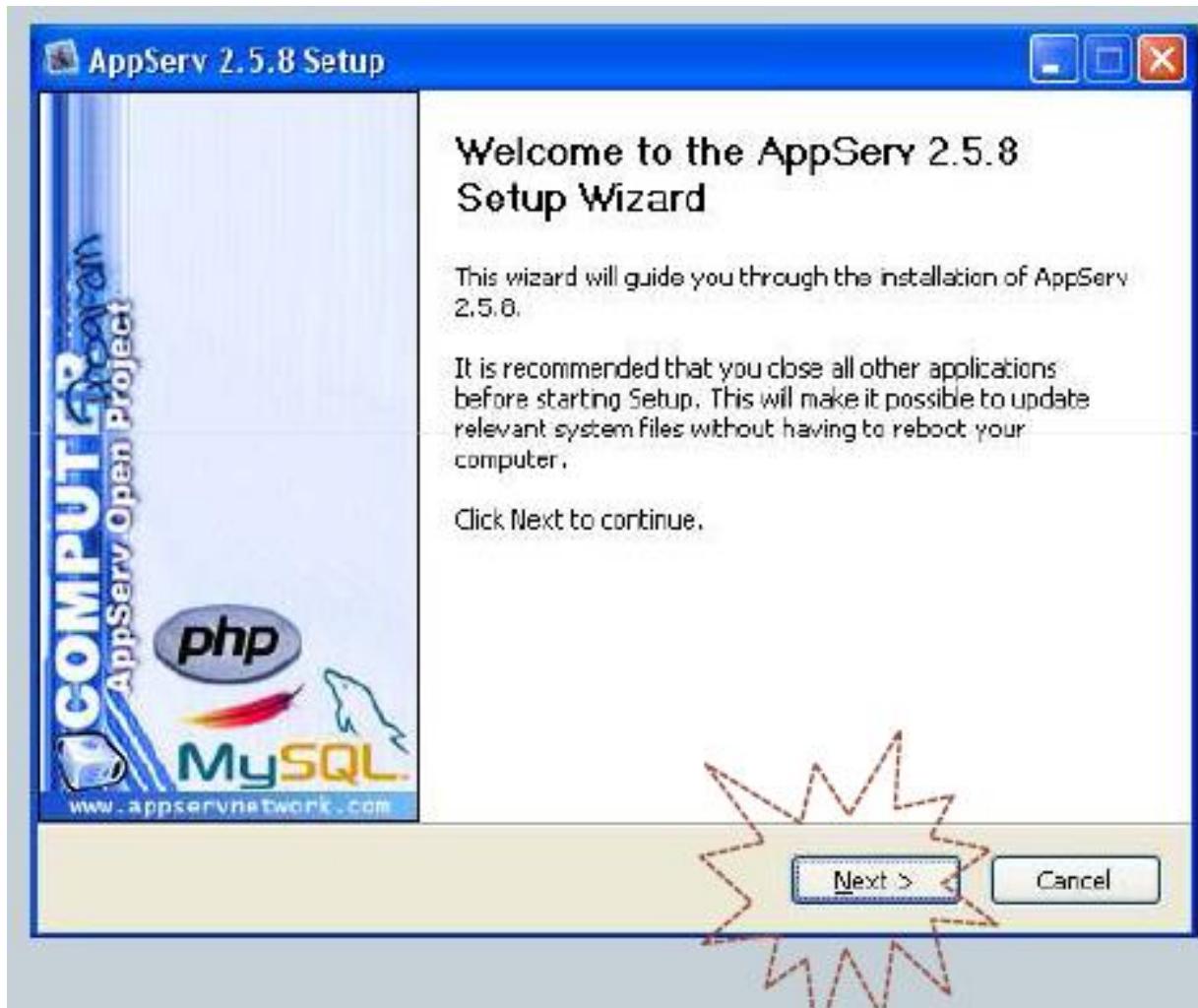


# PHP

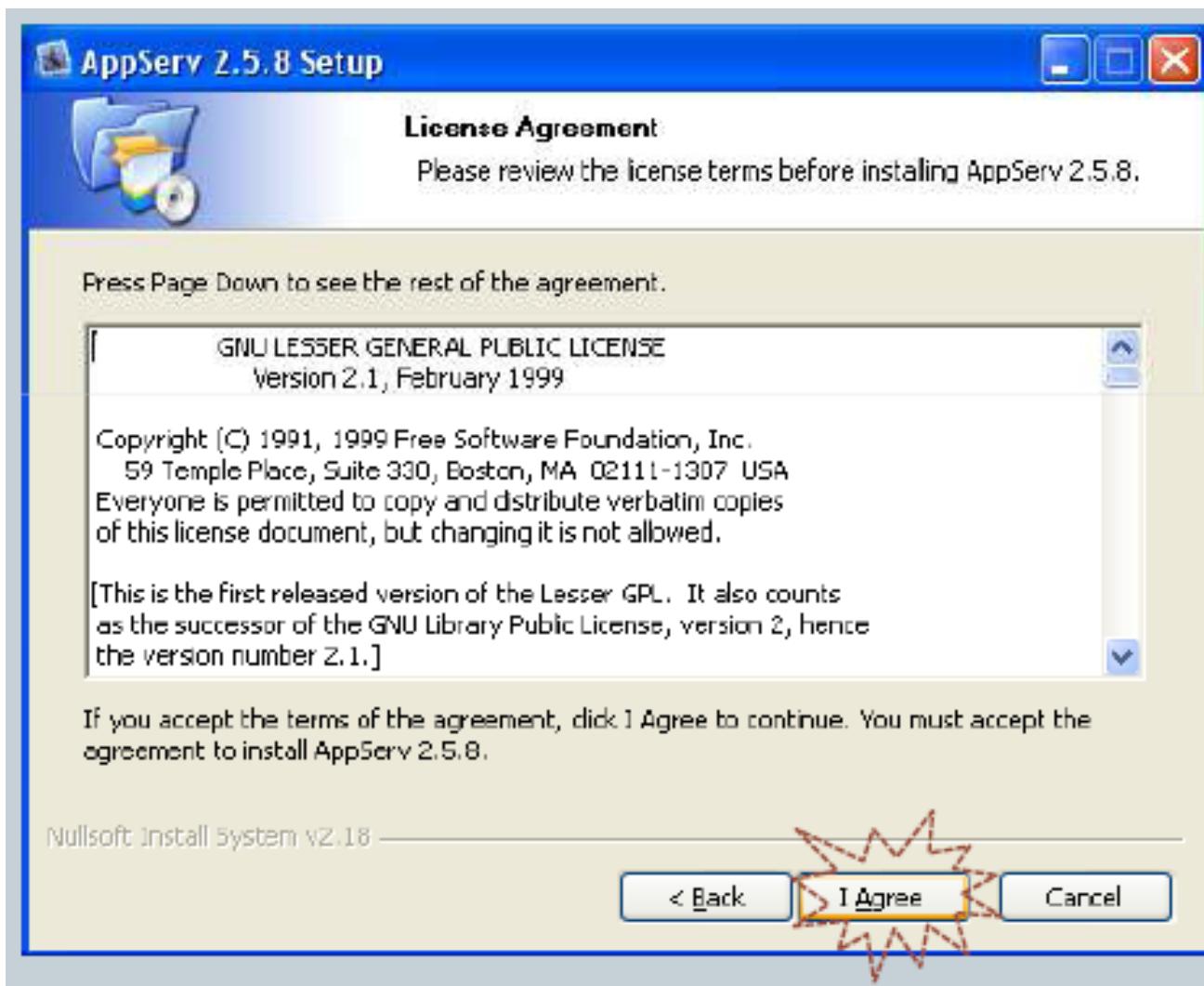
- If your server has activated support for PHP you do not need to do anything.
- Just create some .php files, place them in your web directory, and the server will automatically parse them for you.
- Servers like: WAMP, Appserv
- Editor: Notepad, Notepad++, Dreamweaver



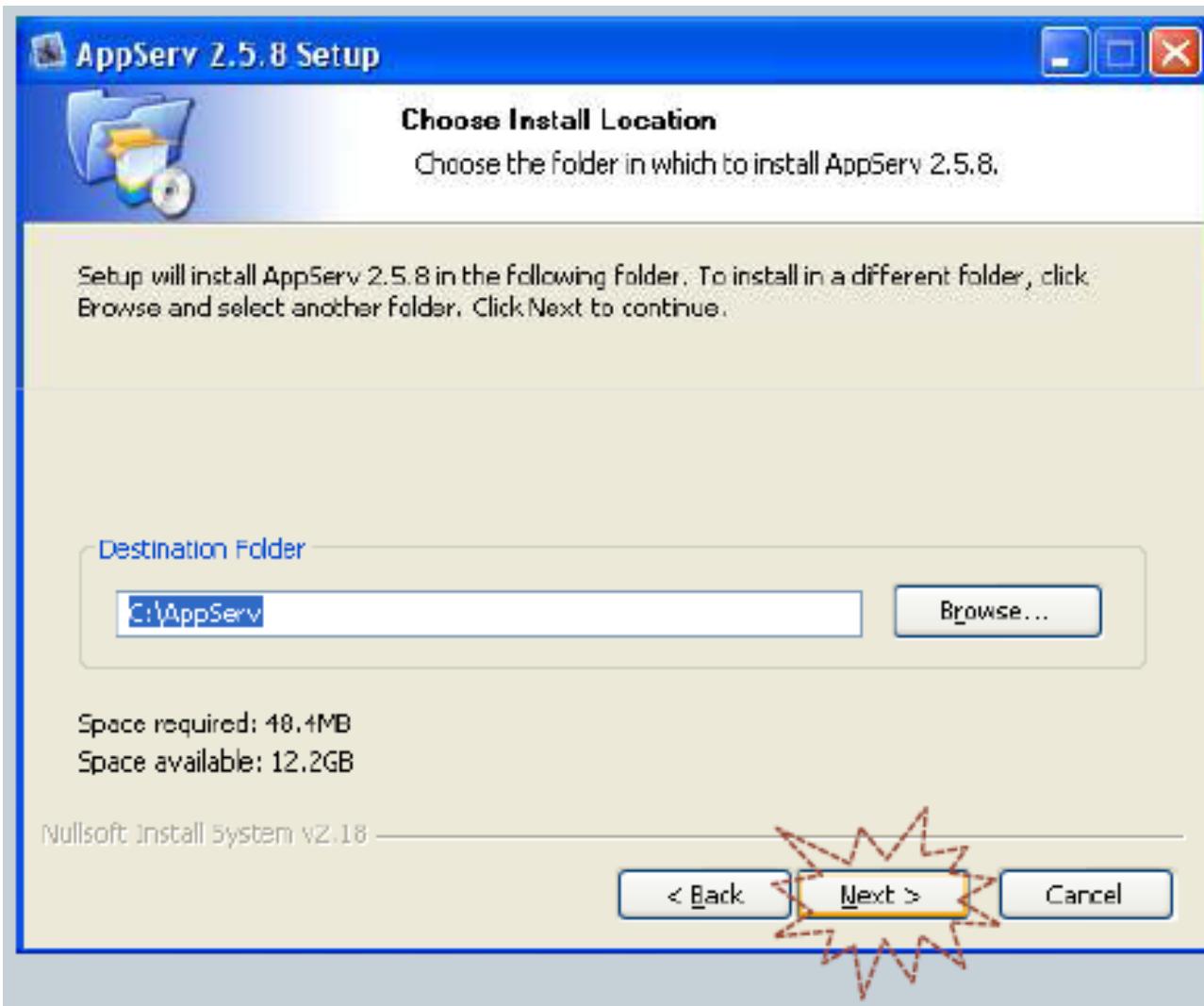
# INSTALLATION...



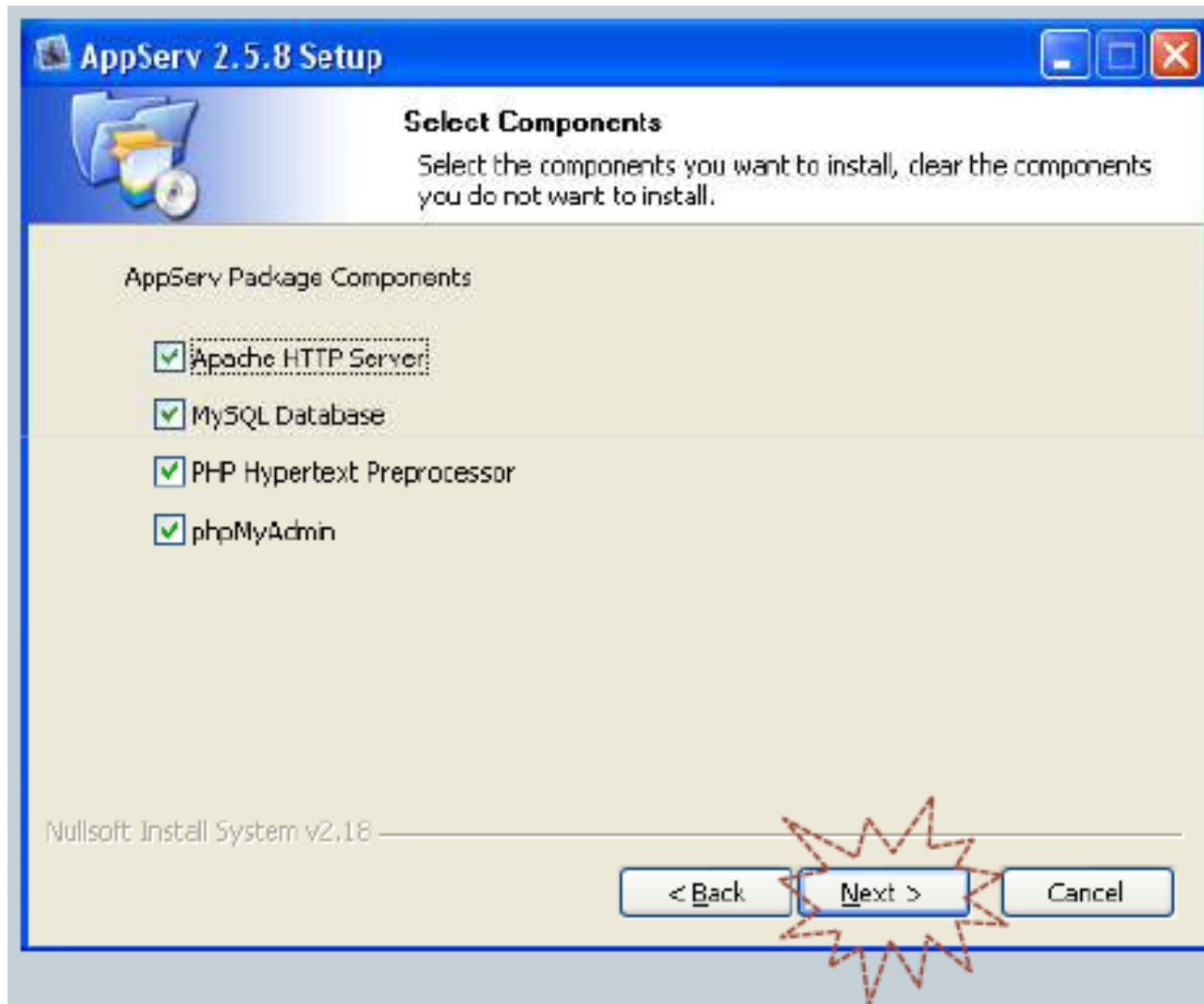
# INSTALLATION...



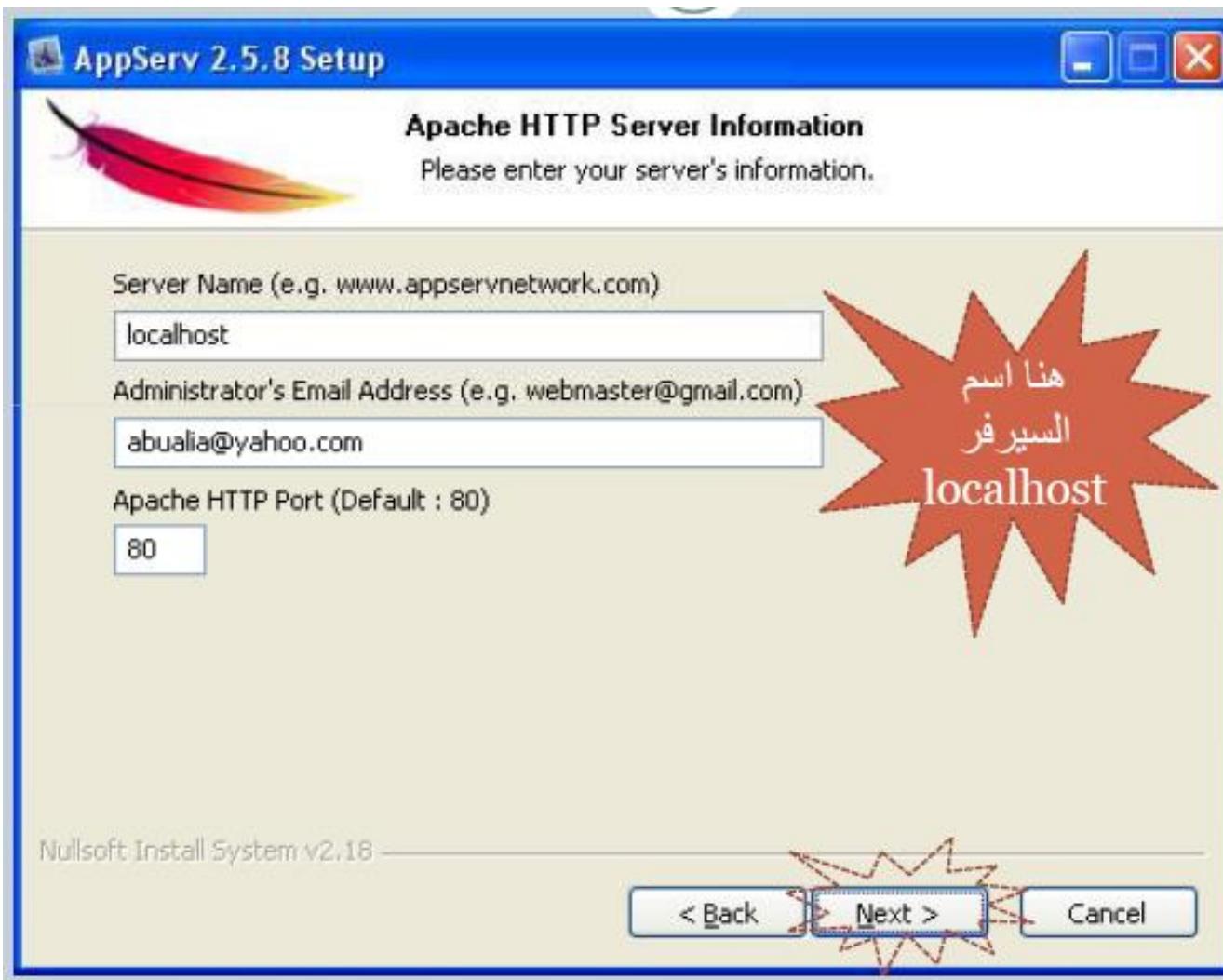
# INSTALLATION...



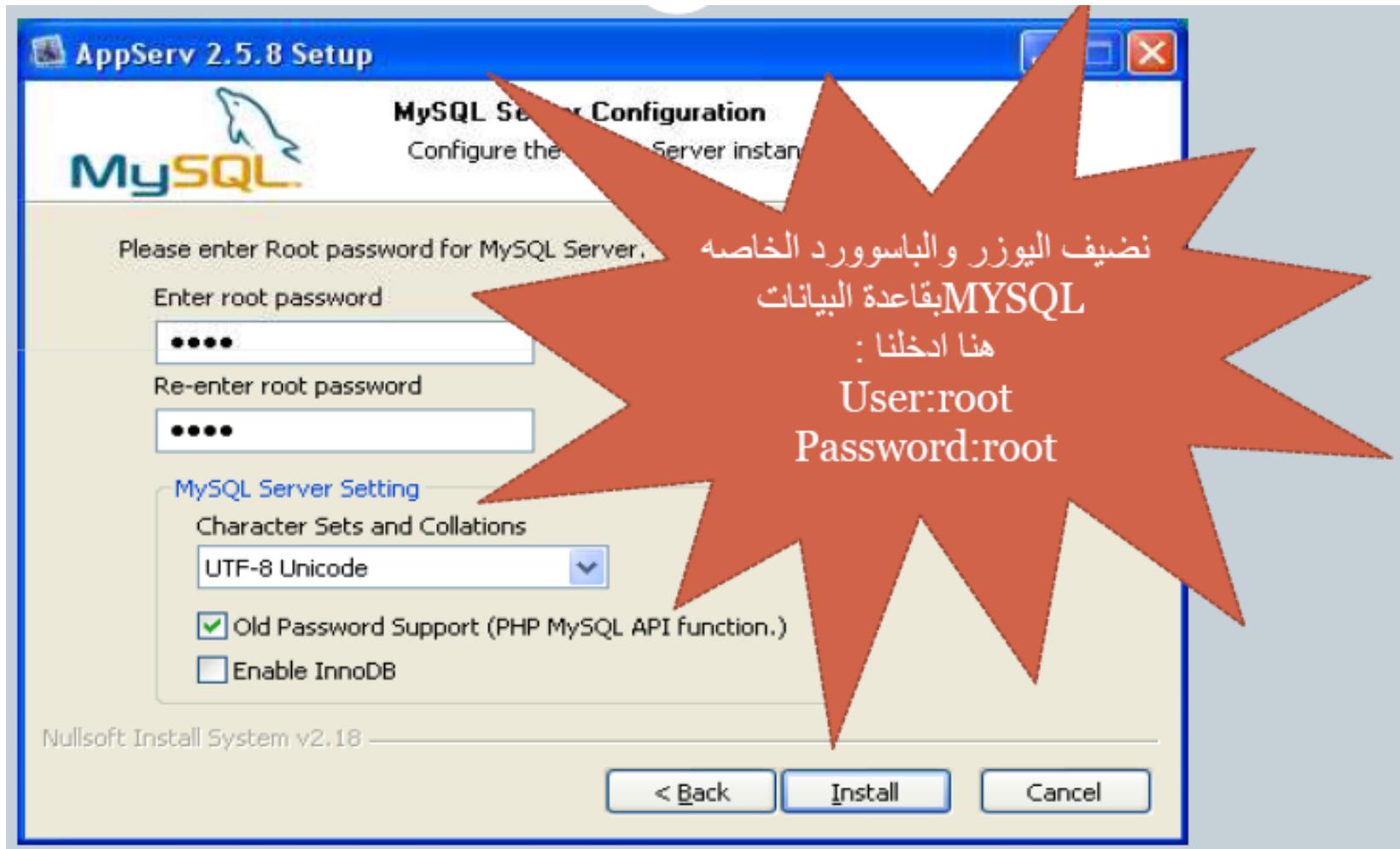
# INSTALLATION...



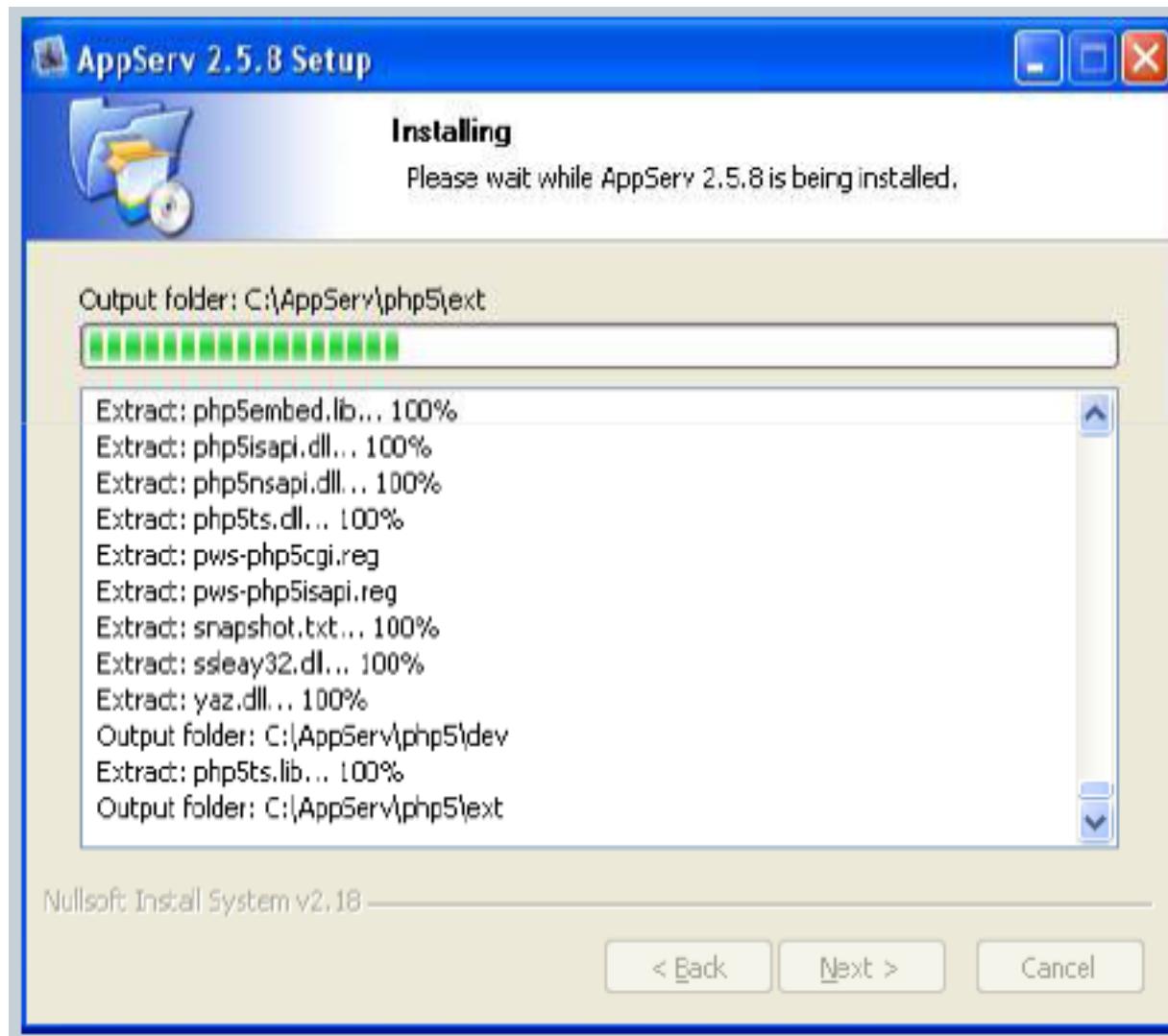
# INSTALLATION...



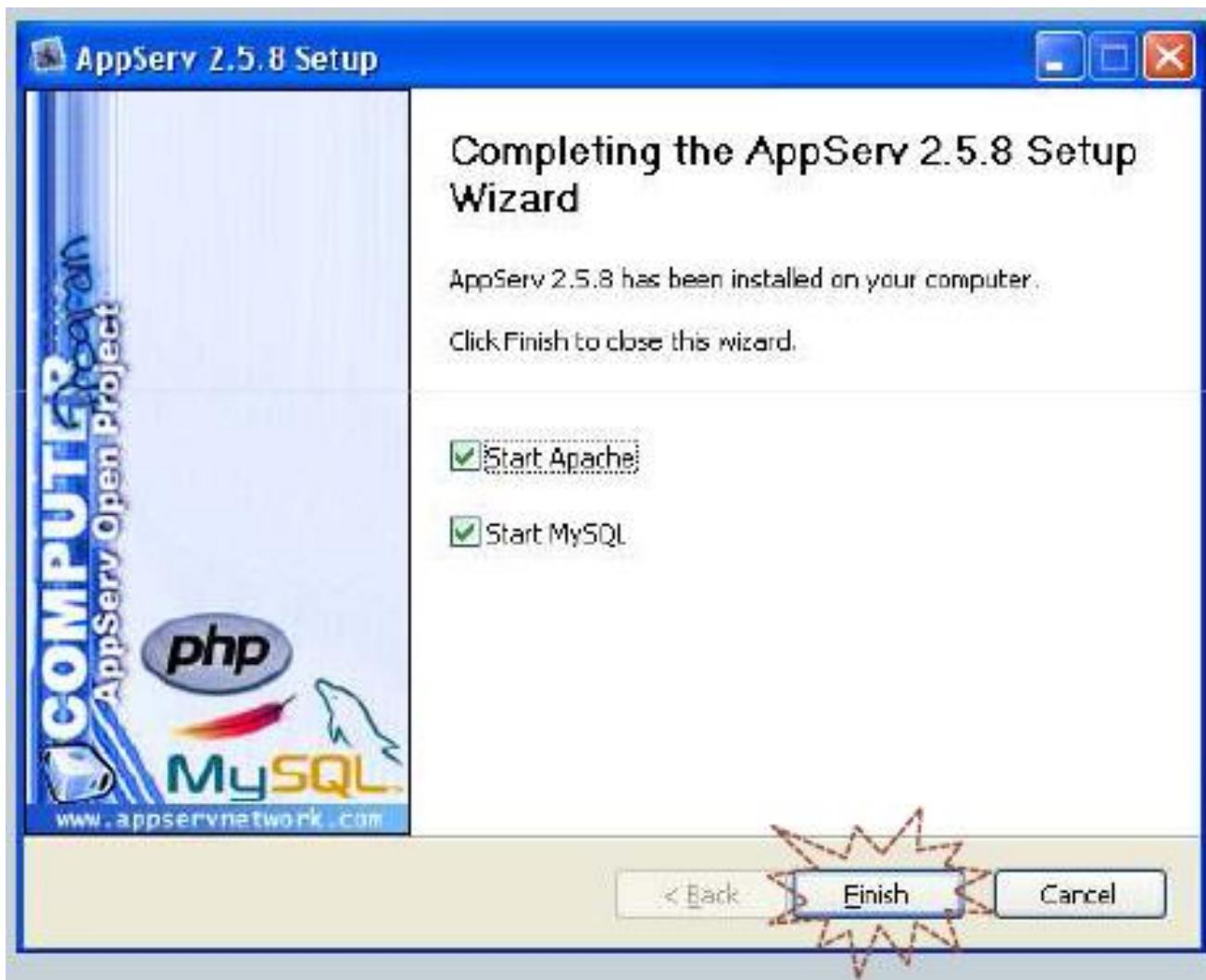
# INSTALLATION...



# INSTALLATION...



# INSTALLATION...



# YOUR SERVER

- As now your computer became a server.
- Not all hard disk is a server, the server is within the www folder inside the AppServ
- To call the server, we write localhost in the web browser.



# PHP PROGRAM

- After writing your php code, save it as *first.php*.
- Save the .php file in the www folder.
- Then call the page, by writing  
<http://localhost/first.php>.



# PHP PROGRAM

- A PHP script starts with `<?php` and ends with `?>`:
- A PHP file normally contains HTML tags

- `<!DOCTYPE html>`  
`<html>`  
`<body>`

`<h1>My first PHP page</h1>`

`<?php`  
`echo "Hello World!";`  
`?>`

`</body>`  
`</html>`



# HERE WE GO...



*ALL THE BEST...!*

# Variables

# Creating (Declaring) PHP Variables

- In PHP, a variable starts with the \$ sign, followed by the name of the variable.
- Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

```
<?php
$txt = "Hello world!";
$x = 5;
$y = 10.5;
?>
```

# PHP Variables

- A variable can have a short name (like x and y) or a more descriptive name (age, car\_name, total\_volume).
- Rules for PHP variables:
  - A variable starts with the \$ sign, followed by the name of the variable
  - A variable name must start with a letter or the underscore character
  - A variable name cannot start with a number
  - A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
  - Variable names are case-sensitive (\$age and \$AGE are two different variables)

# Output Variables

- The PHP echo statement is often used to output data to the screen.
- Examples:

```
<?php
 $txt = "Palestine";
 echo "I love $txt!";
?>
```

```
<?php
 $txt = "Palestine";
 echo "I love ". $txt. "!";
?>
```

```
<?php
 $a=5;
 $b=6;
 echo $a+$b;
?>
```

# PHP is a Loosely Typed Language

- In the example above, notice that we did not have to tell PHP which data type the variable is.
- PHP automatically converts the variable to the correct data type, depending on its value.
- In other languages such as C, C++, and Java, the programmer must declare the name and type of the variable before using it.

# PHP Variables Scope

- In PHP, variables can be declared anywhere in the script.
- The scope of a variable is the part of the script where the variable can be referenced/used.
- PHP has three different variable scopes:
  - local
  - global
  - static

# Global Scope

- A variable declared **outside** a function has a **GLOBAL SCOPE** and can only be accessed outside a function.

```
<?php
$x = 5; // global scope

function myTest() {
 // using x inside this function will generate an error
 echo "<p>Variable x inside function is: $x</p>";
}
myTest();

echo "<p>Variable x outside function is: $x</p>";
?>
```

( ! ) Notice: Undefined variable: x in C:\wamp\www\test\index.php on line 14				
Call Stack				
#	Time	Memory	Function	Location
1	0.0007	240136	{main}()	..\index.php:0
2	0.0007	240272	myTest()	..\index.php:16

Variable x inside function is:

Variable x outside function is: 5

# LOCAL SCOPE

- A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function.

```
<?php
function myTest() {
 $x = 5; // local scope
 echo "<p>Variable x inside function is: $x</p>";
}
myTest();

// using x outside the function will generate an error
echo "<p>Variable x outside function is: $x</p>";
?>
```

# PHP The global Keyword

- The global keyword is used to access a global variable from within a function.
- To do this, use the global keyword before the variables (inside the function):

```
<?php
$x = 5;
$y = 10;

function myTest() {
 global $x, $y;
 $y = $x + $y;
}

myTest();
echo $y; // outputs 15
?>
```

# PHP The global Keyword (Con.)

- PHP also stores all global variables in an array called **`$GLOBALS[index]`**.
- The *index* holds the name of the variable.
- This array is also accessible from within functions and can be used to update global variables directly.
  - See the example in the next slide

# PHP The global Keyword (Con.)

```
<?php
$x = 5;
$y = 10;

function myTest() {
 $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}

myTest();
echo $y; // outputs 15
?>
```

# PHP The static Keyword

- Normally, when a function is completed / executed, all of its variables are deleted.
- However, sometimes we want a local variable NOT to be deleted. We need it for a further job.
- To do this, use the **static** keyword when you first declare the variable:
  - See the example in the next slide:

# PHP The static Keyword (Con.)

```
<?php
function myTest() {
 static $x = 0;
 echo $x;
 $x++;
}

myTest();
myTest();
myTest();
?>
```

# The PHP echo Statement

- The echo statement can be used with or without parentheses: echo or echo().

```
<?php
echo "<h2>PHP is Fun!</h2>";
echo "Hello world!
";
echo "I'm about to learn PHP!
";
echo "This ", "string ", "was ", "made ", "with multiple parameters.";
?>
```

# Display Variables

```
<?php

$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

echo "<h2>" . $txt1 . "</h2>";
echo "Study PHP at " . $txt2 . "
";
echo $x + $y;
?>
```

# The PHP print Statement

- The print statement can be used with or without parentheses: print or print().

```
<?php
print "<h2>PHP is Fun!</h2>";
print "Hello world!
";
print "I'm about to learn PHP!";
?>
```

# Display Variables

```
<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

print "<h2>" . $txt1 . "</h2>";
print "Study PHP at " . $txt2 . "
";
print $x + $y;
?>
```

# PHP echo and print Statements

- echo and print are more or less the same. They are both used to output data to the screen.
- The differences are small:
  - echo has no return value while print has a return value of 1 so it can be used in expressions.
  - echo can take multiple parameters (although such usage is rare) while print can take one argument.
  - echo is marginally faster than print.

# PHP Data Types

- Variables can store data of different types, and different data types can do different things.
- PHP supports the following data types:
  - String
  - Integer
  - Float (floating point numbers - also called double)
  - Boolean
  - Array
  - Object
  - NULL

# PHP String

- A string is a sequence of characters, like "Hello world!".
- A string can be any text inside quotes. You can use single or double quotes:

```
<?php
$x = "Hello world!";
$y = 'Hello world!';

echo $x;
echo "
";
echo $y;
?>
```

# PHP Integer

- An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.
- Rules for integers:
  - An integer must have at least one digit
  - An integer must not have a decimal point
  - An integer can be either positive or negative
  - Integers can be specified in three formats:
    - decimal (10-based),
    - hexadecimal (16-based - prefixed with 0x)
    - or octal (8-based - prefixed with 0)

# PHP Float

- A float (floating point number) is a number with a decimal point.

```
<?php
$x = 10.365;
var_dump($x);
?>
```

# PHP Boolean

- A Boolean represents two possible states: TRUE or FALSE.

```
$x = true;
```

```
$y = false;
```

# PHP Array

- An array stores multiple values in one single variable.
- In the following example \$cars is an array. The PHP var\_dump() function returns the data type and value:

```
<body>
<?php

$var=array('ahmad','ali',35);
var_dump($var);
print_r($var);
?>
```

```
array (size=3)
 0 => string 'ahmad' (length=5)
 1 => string 'ali' (length=3)
 2 => int 35
```

```
Array ([0] => ahmad [1] => ali [2] => 35)
```

# PHP Object

- An object is a data type which stores data and information on how to process that data.
- In PHP, an object must be explicitly declared.
- First we must declare a class of object.
  - For this, we use the class keyword.
  - A class is a structure that can contain properties and methods:

See the example in the next slide

# PHP Object (Con.)

```
<?php
class Car {
 function Car() {
 $this->model = "VW";
 }
}

// create an object
$herbie = new Car();

// show object properties
echo $herbie->model;
?>
```

# PHP NULL Value

- Null is a special data type which can have only one value: NULL.
- A variable of data type NULL is a variable that has no value assigned to it.
- If a variable is created without a value, it is automatically assigned a value of NULL.
- Variables can also be emptied by setting the value to NULL:

# PHP NULL Value (Con.)

```
$x="Hello World";
var_dump($x);
echo "
";
$x=null;
var_dump($x);
```

```
string(11) "Hello World"
NULL
```

# PHP STRING FUNCTIONS

# Get The Length of a String

- The PHP `strlen()` function returns the length of a string.

```
<?php
echo strlen("Hello world!"); // outputs 12
?>
```

# Count The Number of Words in a String

```
<?php
echo str_word_count("Hello world!"); // outputs 2
?>
```

# Reverse a String

```
<?php
echo strrev("Hello world!"); // outputs !dlrow olleH
?>
```

# Search For a Specific Text Within a String

```
<?php
echo strpos("Hello world!", "world"); // outputs 6
?>
```

# Replace Text Within a String

```
<?php
echo str_replace("world", "Dolly", "Hello world!"); // outputs Hello Dolly!
?>
```

# PHP CONSTANTS

# PHP Constants

- A constant is an identifier (name) for a simple value.
- The value cannot be changed during the script.
- A valid constant name starts with a letter or underscore (no \$ sign before the constant name).
- **Note:** Unlike variables, constants are automatically global across the entire script.

# Create a PHP Constant

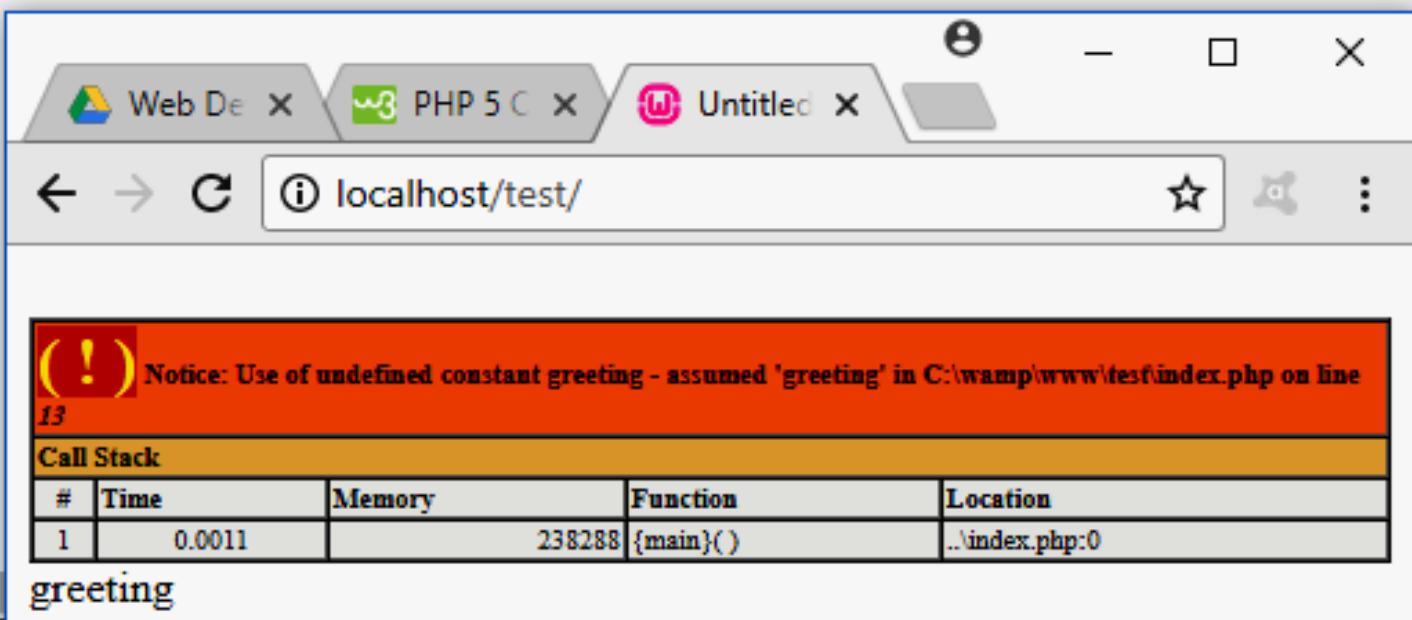
- `define(name, value, case-insensitive)`
  - *name*: Specifies the name of the constant
  - *value*: Specifies the value of the constant
  - *case-insensitive*: Specifies whether the constant name should be case-insensitive. Default is false

```
<?php
define("GREETING", "Welcome to W3Schools.com!");
echo GREETING;
?>
```

# Create a PHP Constant (Con.)

```
<?php
define("GREETING", "Welcome to W3Schools.com!", false);
echo greeting;
?>
```

```
</body>
</html>
```



# Constants are Global

```
<?php
define("GREETING", "Welcome to W3Schools.com!");

function myTest() {
 echo GREETING;
}

myTest();
?>
```

# PHP OPERATORS

# PHP Operators

- Operators are used to perform operations on variables and values.
  - Arithmetic operators
  - Assignment operators
  - Comparison operators
  - Increment/Decrement operators
  - Logical operators
  - String operators
  - Array operators

# PHP Arithmetic Operators

Operator	Name	Example	Result
+	Addition	<code>\$x + \$y</code>	Sum of <code>\$x</code> and <code>\$y</code>
-	Subtraction	<code>\$x - \$y</code>	Difference of <code>\$x</code> and <code>\$y</code>
*	Multiplication	<code>\$x * \$y</code>	Product of <code>\$x</code> and <code>\$y</code>
/	Division	<code>\$x / \$y</code>	Quotient of <code>\$x</code> and <code>\$y</code>
%	Modulus	<code>\$x % \$y</code>	Remainder of <code>\$x</code> divided by <code>\$y</code>
**	Exponentiation	<code>\$x ** \$y</code>	Result of raising <code>\$x</code> to the <code>\$y</code> 'th power (Introduced in PHP 5.6)

# PHP Assignment Operators

Assignment	Same as...	Description
<code>x = y</code>	<code>x = y</code>	The left operand gets set to the value of the expression on the right
<code>x += y</code>	<code>x = x + y</code>	Addition
<code>x -= y</code>	<code>x = x - y</code>	Subtraction
<code>x *= y</code>	<code>x = x * y</code>	Multiplication
<code>x /= y</code>	<code>x = x / y</code>	Division
<code>x %= y</code>	<code>x = x % y</code>	Modulus

# PHP Comparison Operators

Operator	Name	Example	Result
<code>==</code>	Equal	<code>\$x == \$y</code>	Returns true if <code>\$x</code> is equal to <code>\$y</code>
<code>===</code>	Identical	<code>\$x === \$y</code>	Returns true if <code>\$x</code> is equal to <code>\$y</code> , and they are of the same type
<code>!=</code>	Not equal	<code>\$x != \$y</code>	Returns true if <code>\$x</code> is not equal to <code>\$y</code>
<code>&lt;&gt;</code>	Not equal	<code>\$x &lt;&gt; \$y</code>	Returns true if <code>\$x</code> is not equal to <code>\$y</code>
<code>!==</code>	Not identical	<code>\$x !== \$y</code>	Returns true if <code>\$x</code> is not equal to <code>\$y</code> , or they are not of the same type

# PHP Comparison Operators (Con.)

>	Greater than	<code>\$x &gt; \$y</code>	Returns true if \$x is greater than \$y
<	Less than	<code>\$x &lt; \$y</code>	Returns true if \$x is less than \$y
<code>&gt;=</code>	Greater than or equal to	<code>\$x &gt;= \$y</code>	Returns true if \$x is greater than or equal to \$y
<code>&lt;=</code>	Less than or equal to	<code>\$x &lt;= \$y</code>	Returns true if \$x is less than or equal to \$y

# PHP Increment / Decrement Operators

<b>Operator</b>	<b>Name</b>	<b>Description</b>
<code>++\$x</code>	Pre-increment	Increments \$x by one, then returns \$x
<code>\$x++</code>	Post-increment	Returns \$x, then increments \$x by one
<code>--\$x</code>	Pre-decrement	Decrements \$x by one, then returns \$x
<code>\$x--</code>	Post-decrement	Returns \$x, then decrements \$x by one

# PHP Logical Operators

Operator	Name	Example	Result
and	And	<code>\$x and \$y</code>	True if both <code>\$x</code> and <code>\$y</code> are true
or	Or	<code>\$x or \$y</code>	True if either <code>\$x</code> or <code>\$y</code> is true
xor	Xor	<code>\$x xor \$y</code>	True if either <code>\$x</code> or <code>\$y</code> is true, but not both
&&	And	<code>\$x &amp;&amp; \$y</code>	True if both <code>\$x</code> and <code>\$y</code> are true
	Or	<code>\$x    \$y</code>	True if either <code>\$x</code> or <code>\$y</code> is true
!	Not	<code>!\$x</code>	True if <code>\$x</code> is not true

# PHP String Operators

<b>Operator</b>	<b>Name</b>	<b>Example</b>	<b>Result</b>
.	Concatenation	<code>\$txt1 . \$txt2</code>	Concatenation of \$txt1 and \$txt2
.=	Concatenation assignment	<code>\$txt1 .= \$txt2</code>	Appends \$txt2 to \$txt1

# PHP CONDITIONAL STATEMENTS

# PHP Conditional Statements

- In PHP we have the following conditional statements:
  - **if statement** - executes some code if one condition is true
  - **if...else statement** - executes some code if a condition is true and another code if that condition is false
  - **if...elseif....else statement** - executes different codes for more than two conditions
  - **switch statement** - selects one of many blocks of code to be executed

# Examples

```
<?php
$t = date("H");

if ($t < "20") {
 echo "Have a good day!";
}
?>
```

```
<?php
$t = date("H");

if ($t < "20") {
 echo "Have a good day!";
} else {
 echo "Have a good night!";
}
?>
```

```
<?php
$t = date("H");

if ($t < "10") {
 echo "Have a good morning!";
} elseif ($t < "20") {
 echo "Have a good day!";
} else {
 echo "Have a good night!";
}
?>
```

# Example

```
$favcolor = "red";

switch ($favcolor) {
 case "red":
 echo "Your favorite color is red!";
 break;
 case "blue":
 echo "Your favorite color is blue!";
 break;
 case "green":
 echo "Your favorite color is green!";
 break;
 default:
 echo "Your favorite color is neither red, blue, nor green!";
}
```

# PHP Loops

```
<?php
$x = 1;

while($x <= 5) {
 echo "The number is: $x
";
 $x++;
}
?>
```

```
<?php
$x = 1;

do {
 echo "The number is: $x
";
 $x++;
} while ($x <= 5);
?>
```

# PHP Loops (Con.)

```
<?php
for ($x = 0; $x <= 10; $x++) {
 echo "The number is: $x
";
}
?>
```

# The PHP foreach Loop

- The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

```
<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $value) {
 echo "$value
";
}
?>
```

# PHP FUNCTIONS

# PHP User Defined Functions

- Besides the built-in PHP functions, we can create our own functions.
  - A function is a block of statements that can be used repeatedly in a program.
  - A function will not execute immediately when a page loads.
  - A function will be executed by a call to the function.

# PHP User Defined Functions

A user defined function declaration starts with the word "function":

## Syntax

```
function functionName() {
 code to be executed;
}
```

**Note:** A function name can start with a letter or underscore (not a number).

**Tip:** Give the function a name that reflects what the function does!

Function names are NOT case-sensitive.

# Example

```
<?php
function writeMsg() {
 echo "Hello world!";
}

writeMsg(); // call the function
?>
```

# Example

```
<?php
function familyName($fname, $year) {
 echo "$fname Born in $year
";
}

familyName("Hege", "1975");
familyName("Stale", "1978");
familyName("Kai Jim", "1983");
?>
```

# Example

```
<?php
function setHeight($minheight = 50) {
 echo "The height is : $minheight
";
}

setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

# Example

```
<?php
function sum($x, $y) {
 $z = $x + $y;
 return $z;
}

echo "5 + 10 = " . sum(5, 10) . "
";
echo "7 + 13 = " . sum(7, 13) . "
";
echo "2 + 4 = " . sum(2, 4);
?>
```

# PHP ARRAYS

# Create an Array in PHP

- In PHP, there are three types of arrays:
  - **Indexed arrays** - Arrays with a numeric index
  - **Associative arrays** - Arrays with named keys
  - **Multidimensional arrays** - Arrays containing one or more arrays

# PHP Indexed Arrays

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0), like this:

```
$cars = array("Volvo", "BMW", "Toyota");
```

or the index can be assigned manually:

```
$cars[0] = "Volvo";
$cars[1] = "BMW";
$cars[2] = "Toyota";
```

# The count() Function

- The count() function is used to return the length (the number of elements) of an array:

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo count($cars);
?>
```

# Loop Through an Indexed Array

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
$arrlength = count($cars);

for($x = 0; $x < $arrlength; $x++) {
 echo $cars[$x];
 echo "
";
}
?>
```

# PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.

There are two ways to create an associative array:

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
```

or:

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

# Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
?>
```

# Loop Through an Associative Array

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach($age as $x => $x_value) {
 echo "Key=" . $x . ", Value=" . $x_value;
 echo "
";
}
?>
```

# PHP - Sort Functions For Arrays

- `sort()` - sort arrays in ascending order
- `rsort()` - sort arrays in descending order
- `asort()` - sort associative arrays in ascending order, according to the value
- `arsort()` - sort associative arrays in descending order, according to the value
- `ksort()` - sort associative arrays in ascending order, according to the key
- `krsort()` - sort associative arrays in descending order, according to the key

# PHP Array Operators

Operator	Name	Example	Result
+	Union	<code>\$x + \$y</code>	Union of <code>\$x</code> and <code>\$y</code>
<code>==</code>	Equality	<code>\$x == \$y</code>	Returns true if <code>\$x</code> and <code>\$y</code> have the same key/value pairs
<code>===</code>	Identity	<code>\$x === \$y</code>	Returns true if <code>\$x</code> and <code>\$y</code> have the same key/value pairs in the same order and of the same types
<code>!=</code>	Inequality	<code>\$x != \$y</code>	Returns true if <code>\$x</code> is not equal to <code>\$y</code>
<code>&lt;&gt;</code>	Inequality	<code>\$x &lt;&gt; \$y</code>	Returns true if <code>\$x</code> is not equal to <code>\$y</code>
<code>!==</code>	Non-identity	<code>\$x !== \$y</code>	Returns true if <code>\$x</code> is not identical to <code>\$y</code>

# PHP - Multidimensional Arrays

```
$cars = array
(
 array("Volvo",22,18),
 array("BMW",15,13),
 array("Saab",5,2),
 array("Land Rover",17,15)
);
```

# PHP - Multidimensional Arrays (Con.)

```
<?php
$cars = array
(
 array("Volvo",22,18),
 array("BMW",15,13),
 array("Saab",5,2),
 array("Land Rover",17,15)
);

for ($row = 0; $row < 4; $row++) {
 echo "<p>Row number $row</p>";
 echo "";
 for ($col = 0; $col < 3; $col++) {
 echo "".$cars[$row][$col]."";
 }
 echo "";
}
?>
```

## Row number 0

- Volvo
- 22
- 18

## Row number 1

- BMW
- 15
- 13

## Row number 2

- Saab
- 5
- 2

## Row number 3

- Land Rover
- 17
- 15

# **Object Oriented Programming in PHP**

# Class

- This is a programmer-defined data type.
- Includes local functions as well as local data.
- You can think of a class as a template for making many instances of the same kind (or class) of object.

# Object

- An individual instance of the data structure defined by a class.
- You define a class once and then make many objects that belong to it.
- Objects are also known as instance.

# Member Variables

- These are the variables defined inside a class.
- This data -normally- will be invisible to the outside of the class and can be accessed via member functions.
- These variables are called attribute of the object once an object is created.

# Member Functions

- These are the function defined inside a class and are used to access object data.

# Con.

- **Inheritance:**
  - When a class is defined by inheriting existing function of a parent class then it is called inheritance.
- **Parent class :**
  - A class that is inherited by another class.
  - This is also called a base class or super class.
- **Child Class :**
  - A class that inherits from another class.
  - This is also called a subclass or derived class.

# Polymorphism

- This is an object oriented concept where same function can be used for different purposes.
  - For example; having two functions with the same function name and parameters (i.e., *function signature*). One of the functions is in the parent class and the other is in the child class. (overriding)

# Con.

- **Data Abstraction :**
  - Any representation of data in which the implementation details are hidden (abstracted).
- **Encapsulation :**
  - Refers to a concept where we encapsulate all the data and member functions together to form an object.

# Con.

- **Constructor :**
  - Refers to a special type of function which will be called automatically whenever there is an object formation from a class.
- **Destructor :**
  - Refers to a special type of function which will be called automatically whenever an object is deleted or goes out of scope.

# Defining PHP Classes

```
<?php
 class phpClass {
 var $var1;
 var $var2 = "constant string";

 function myfunc ($arg1, $arg2) {
 [..]
 }
 [..]
 }
?>
```

# Example

```
<?php
 class Books {
 /* Member variables */
 var $price;
 var $title;

 /* Member functions */
 function setPrice($par){
 $this->price = $par;
 }

 function getPrice(){
 echo $this->price ."
";
 }

 function setTitle($par){
 $this->title = $par;
 }

 function getTitle(){
 echo $this->title ."
";
 }
 }
?>
```

# Creating Objects in PHP

```
$physics = new Books;
$maths = new Books;
$chemistry = new Books;
```

# Calling Member Functions

```
$physics->setTitle("Physics for High School");
$chemistry->setTitle("Advanced Chemistry");
$maths->setTitle("Algebra");

$physics->setPrice(10);
$chemistry->setPrice(15);
$maths->setPrice(7);

$physics->getTitle();
$chemistry->getTitle();
$maths->getTitle();
$physics->getPrice();
$chemistry->getPrice();
$maths->getPrice();
```

# Constructor Functions

```
class Book
{
 var $price;
 var $title;

 function Book($t, $p)
 {
 $this->price=$p;
 $this->title=$t;
 }

 function show()
 {
 return "the price = $this->price, $this->title";
 }
}

$physics = new Book ("Physics for High School", 10);
$maths = new Book ("Advanced Chemistry", 15);
$chemistry = new Book ("Algebra", 7);
```

# Inheritance

- The effect of inheritance is that the child class (or subclass or derived class) has the following characteristics:
  - Automatically has all the non-private member variable declarations of the parent class.
  - Automatically has all the same non-private member functions as the parent, which (by default) will work the same way as those functions do in the parent.

# Example

```
class Novel extends Books {
 var $publisher;

 function setPublisher($par){
 $this->publisher = $par;
 }

 function getPublisher(){
 echo $this->publisher. "
";
 }
}
```

# Function Overriding

- Function definitions in child classes override definitions with the same name in parent classes.
- In a child class, we can modify the definition of a function inherited from parent class.

# Public Members

- Unless you specify otherwise, properties and methods of a class are public.
  - That is to say, they may be accessed in three possible situations:
    - From outside the class in which it is declared
    - From within the class in which it is declared
    - From within another class that implements the class in which it is declared

# Private members

- By designating a member private:
  - you limit its accessibility to the class in which it is declared.

```
class Book
{
 private $price;
 private $title;

 function Book($t, $p)
 {
 $this->price=$p;
 $this->title=$t;
 }

 function show()
 {
 return "the price = $this->price, $this->title";
 }

 private function test()
 {
 //this is a private function.
 }
}
```

# Protected members

- A protected property or method is accessible
  - in the class in which it is declared.
  - in classes that extend that class.
- A class member can be made protected by using protected keyword in front of the member.

# Con.

```
class Book
{
 protected $price;
 protected $title;

 function Book($t, $p)
 {
 $this->price=$p;
 $this->title=$t;
 }

 function show()
 {
 return "the price = $this->price, $this->title";
 }

 protected function test()
 {
 //this is a protected function.
 }
}
```

# Interfaces

- Interfaces are defined to provide a common function names to the implementers.
- Different implementers can implement those interfaces according to their requirements.
- You can say, interfaces are skeletons which are implemented by developers.

# Interface (Con.)

```
interface Mail {
 public function sendMail();
}
```

```
class Report implements Mail {
 // sendMail() Definition goes here
}
```

# Static

- Declaring class members or methods as static makes them accessible without needing an instantiation of the class.
  - A **member variable** declared as static **can not** be accessed with an instantiated class object.

# Static (Con.)

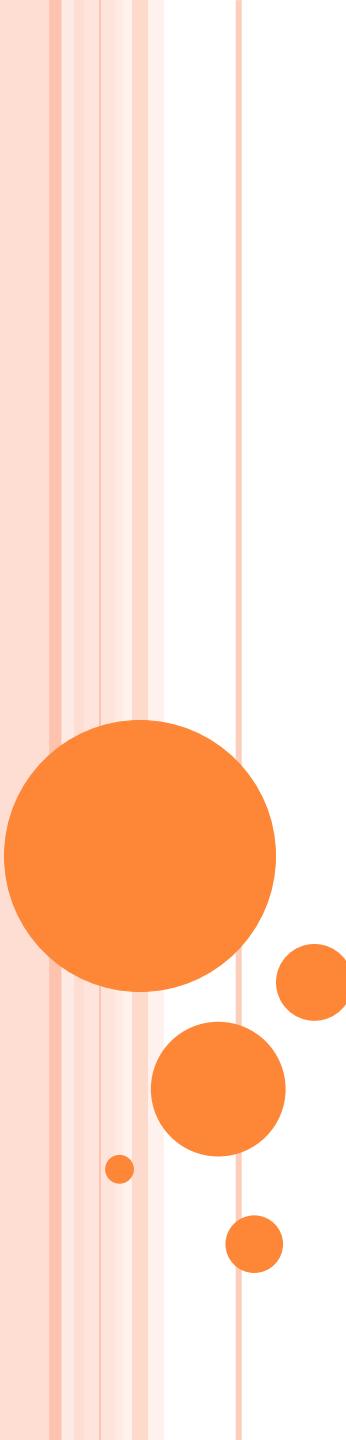
```
class Book
{
 var $price;
 var $title;
 static $x=0;

 function book()
 {
 self::$x++;
 }

 static function hello()
 {
 echo "
hello word";
 }
}
```

```
echo Book::$x;
$java=new Book();
echo "
".Book::$x;
$java=new Book();
echo "
".Book::$x;
Book::hello();
$java->hello();

// $java::x; Error
// $java->x; Error
```



# PHP

## DATE AND TIME

Ahmad Rabay'a  
2015  
[rabaya.ahmad@gmail.com](mailto:rabaya.ahmad@gmail.com)

# THE PHP DATE() FUNCTION

- The PHP date() function is used to format a date and/or a time.
- The PHP date() function formats a timestamp to a more readable date and time.
- Syntax

*date(format,timestamp)*

- Format: Required. Specifies the format of the timestamp.
- Timestamp: Optional. Specifies a timestamp. Default is the current date and time



## SIMPLE DATE

- The required *format* parameter of the `date()` function specifies how to format the date (or time).
- Some characters that are commonly used for dates
  - d - Represents the day of the month (01 to 31)
  - m - Represents a month (01 to 12)
  - Y - Represents a year (in four digits)
  - l (lowercase 'L') - Represents the day of the week

## EXAMPLE

```
<!DOCTYPE html>
<html>
<body>

<?php
 echo "Today is ". date("Y/m/d") . "
"; // Today is 2022/12/16
 echo "Today is ". date("Y.m.d") . "
"; // Today is 2022.12.16
 echo "Today is ". date("Y-m-d") . "
"; // Today is 2022-12-16
 echo "Today is ". date("l"); // Today is Friday
?>

</body>
</html>
```



# SIMPLE TIME

- Some characters that are commonly used for times:
  - h - 12-hour format of an hour with leading zeros (01 to 12)
  - i - Minutes with leading zeros (00 to 59)
  - s - Seconds with leading zeros (00 to 59)
  - a - Lowercase Ante meridiem and Post meridiem (am or pm)

```
<?php
echo "The time is ". date("h:i:sa"); // The time is 04:10:37pm
?>
```



# CREATE A DATE WITH PHP MKTIME()

- The *timestamp* parameter in the date() function specifies a timestamp.
  - If you do not specify a timestamp, the current date and time will be used.
- The mktimed() function returns the timestamp for a date.
- Syntax
  - mktimed(hour,minute,second,month,day,year)*



# EXAMPLE

```
<!DOCTYPE html>
<html>
<body>

<?php
$d=mktime(11, 14, 54, 8, 12, 2014);
echo "Created date is ". date("Y-m-d h:i:s a", $d);
// Created date is 2014-08-12 11:14:54 am
?>

</body>
</html>
```



# CREATE A DATE FROM A STRING With PHP

## STRTOTIME()

- The PHP strtotime() function is used to convert a human readable string to a time.
- PHP is quite clever about converting a string to a date, so you can put in various values.
- Syntax  
*strtotime(time,now)*



# EXAMPLE

```
$d=strtotime("today");
echo "Today ". date("Y-m-d (l)", $d) . "
<hr>";
```

```
$d=strtotime("-3 Months");
echo date("Y-m-d (l)", $d) . "
";
```

```
$d=strtotime("previous sunday");
echo date("Y-m-d (l)", $d) . "
";
```

```
$d=strtotime("yesterday");
echo date("Y-m-d (l)", $d) . "
";
```

```
$d=strtotime("tomorrow");
echo date("Y-m-d (l)", $d) . "
";
```

```
$d=strtotime("next Monday");
echo date("Y-m-d (l)", $d) . "
";
```

```
$d=strtotime("+3 Months");
echo date("Y-m-d (l)", $d) . "
";
```

Today 2022-12-16 (Friday)

2022-09-16 (Friday)

2022-12-11 (Sunday)

2022-12-15 (Thursday)

2022-12-17 (Saturday)

2022-12-19 (Monday)

2023-03-16 (Thursday)

- The example below outputs the dates for the next six Saturdays:

```
<?php
$startdate=strtotime("saturday");
$enddate=strtotime("+6 weeks", $startdate);

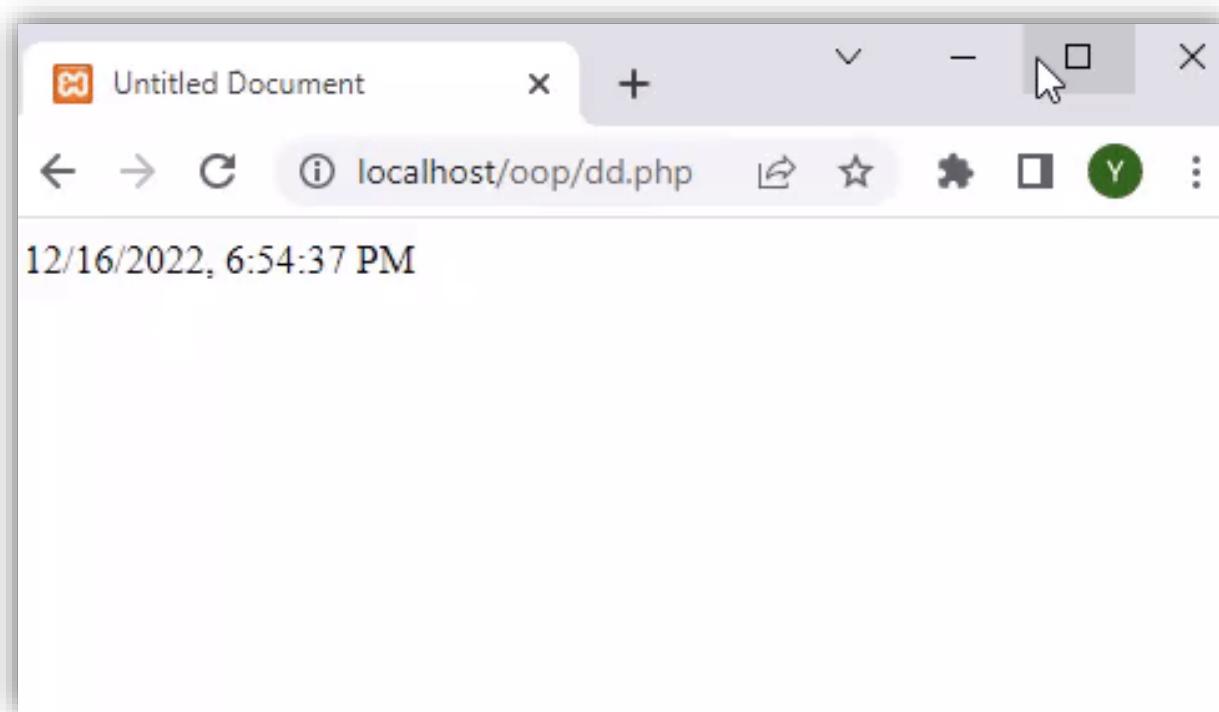
while ($startdate < $enddate) {
 echo date("d/M/Y", $startdate), "
";
 $startdate = strtotime("+1 week", $startdate);
}
?>
```

17/Dec/2022  
24/Dec/2022  
31/Dec/2022  
07/Jan/2023  
14/Jan/2023  
21/Jan/2023

```


<script>
var x=setInterval(myFunction,1000);

function myFunction()
{
 var d=new Date();
 document.getElementById('d').innerHTML=d.toLocaleString();
}
</script>
```





*ALL THE BEST...!*

# PHP Form Handling

# PHP Form Handling

```
<html>
<body>
```

```
<form action="welcome.php" method="post">
Name: <input type="text" name="name">

E-mail: <input type="text" name="email">

<input type="submit">
</form>
```

```
</body>
</html>
```

- The PHP superglobals `$_GET` and `$_POST` are used to collect form-data.
- When the user fills out the form on the right and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php".
- The form data is sent with the HTTP POST method.

# To display the submitted data

```
<html>
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="name">

E-mail: <input type="text" name="email">

<input type="submit">
</form>

</body>
</html>
```

```
<html>
<body>
<div style="border: 1px dashed black; padding: 10px; margin-top: 10px;">
Welcome <?php echo $_POST["name"]; ?>

Your email address is: <?php echo $_POST["email"]; ?>

```

# GET vs. POST

- Both GET and POST create an array  
(e.g. `array( key => value, key2 => value2, key3 => value3, ...)`).  
This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.
- Both GET and POST are treated as `$_GET` and `$_POST`.
- GET and POST are superglobals:
  - they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

## GET vs. POST (Con.)

- `$_GET` is an array of variables passed to the current script via the URL parameters.
- `$_POST` is an array of variables passed to the current script via the HTTP POST method.

# When to use GET?

- Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL).
- GET also has limits on the amount of information to send. The limitation is about 2000 characters.
- However, because the variables are displayed in the URL, it is possible to bookmark the page.
  - This can be useful in some cases.
- GET may be used for sending non-sensitive data.
- **Note:** GET should NEVER be used for sending passwords or other sensitive information!

# When to use POST?

- Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request)
- POST has **no limits** on the amount of information to send.
- POST supports advanced functionality such as support for multi-part binary input while uploading files to server.
- However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

# Trim()

- trim() will strip the following characters from the beginning and the end of string:
  - " " (ASCII 32 (0x20)), an ordinary space.
  - "\t" (ASCII 9 (0x09)), a tab.
  - "\n" (ASCII 10 (0x0A)), a new line (line feed).
  - "\r" (ASCII 13 (0x0D)), a carriage return.
- Example :
  - \$text = "\t\tThese are a few words :) ...";
  - \$trimmed = trim(\$text);

# stripslashes()

- The stripslashes() function removes backslashes added.

## Syntax

```
stripslashes(string)
```

### Parameter

### Description

*string*

Required. Specifies the string to check

## Technical Details

### Return Value:

Returns a string with backslashes stripped off

# htmlspecialchars()

- The htmlspecialchars() function is used to converts special characters to HTML entities.
- Examples :
  - Special character -> HTML entity:
    - & (ampersand) ->&amp
    - ' (single quote) -> &#039
    - < (less than), -> &lt
    - > (greater than), -> &gt

# Validate Form Data With PHP

- The first thing we will do is to pass all variables through PHP's `htmlspecialchars()` function.
  - If a user tries to submit the following in a text field:

```
<script>location.replace('http://www.ptuk.edu.ps')</script>
```

this would not be executed, because it would be saved as HTML escaped code, like this:

```
<script>location.replace('http://www.hacked.com')</script>
```

The code is now safe to be displayed on a page or inside an e-mail.

# Validate Form Data With PHP (Con.)

- Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)
- Remove backslashes (\) from the user input data (with the PHP stripslashes() function)

# Validate Form Data Example

```
<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
 $name = test_input($_POST["name"]);
 $email = test_input($_POST["email"]);
 $website = test_input($_POST["website"]);
 $comment = test_input($_POST["comment"]);
 $gender = test_input($_POST["gender"]);
}

function test_input($data) {
 $data = htmlspecialchars($data);
 $data = trim($data);
 $data = stripslashes($data);
 return $data;
}
?>
```

# Keep The Values in The Form

Name: <input type="text" name="name" value="<?php echo \$name;?>">

E-mail: <input type="text" name="email" value="<?php echo \$email;?>">

Website: <input type="text" name="website" value="<?php echo \$website;?>">

Comment: <textarea name="comment" rows="5" cols="40"><?php echo \$comment;?></textarea>

Gender:

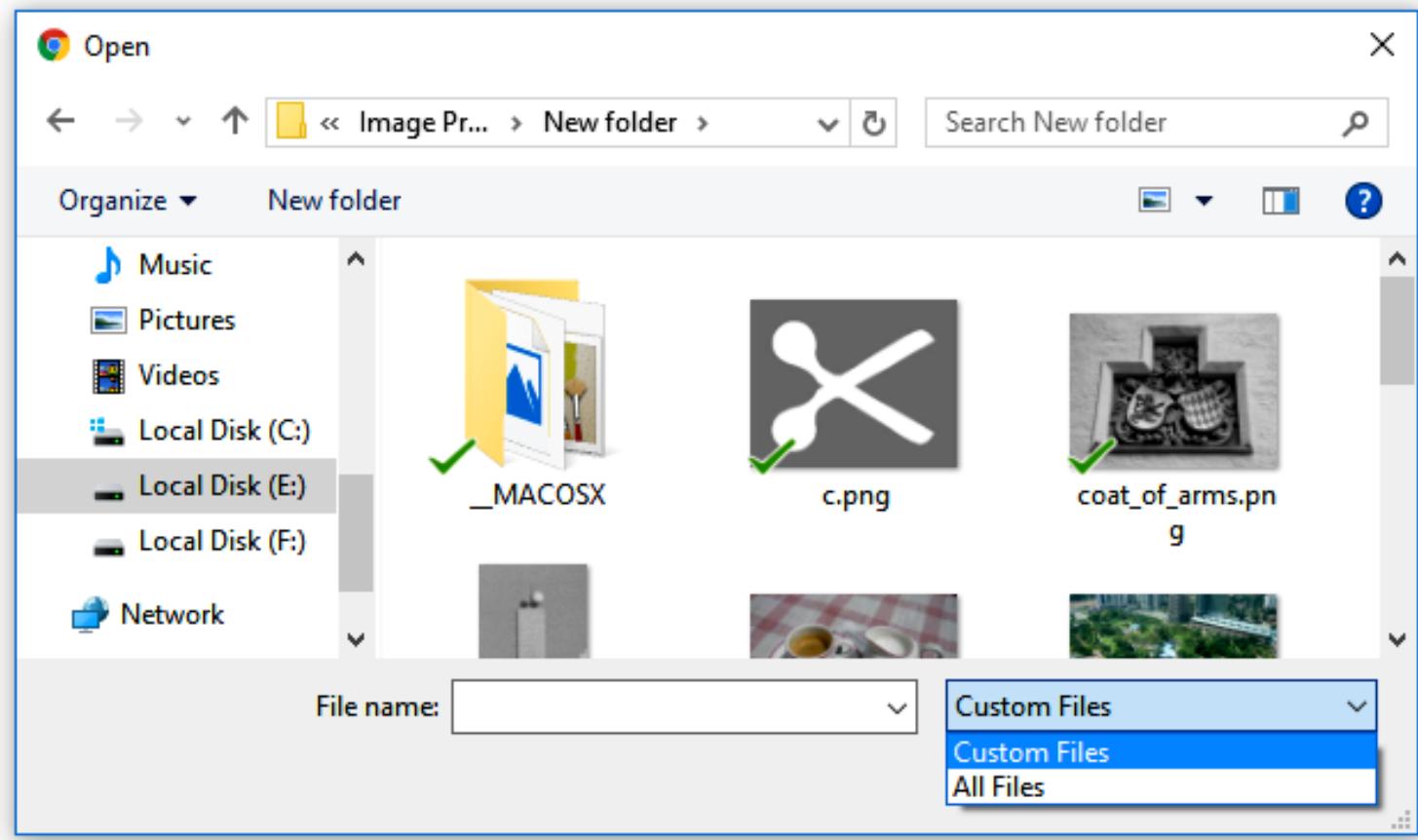
```
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="female") echo "checked";?>
value="female">Female
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="male") echo "checked";?>
value="male">Male
```

# File Upload and Download with PHP

```
<form enctype="multipart/form-data">
First Name: <input type="text" name="fname" /><hr />
Photo : <input type="file" name="photo" accept=".png, .jpg , .jpeg" />
</form>
```

First Name:

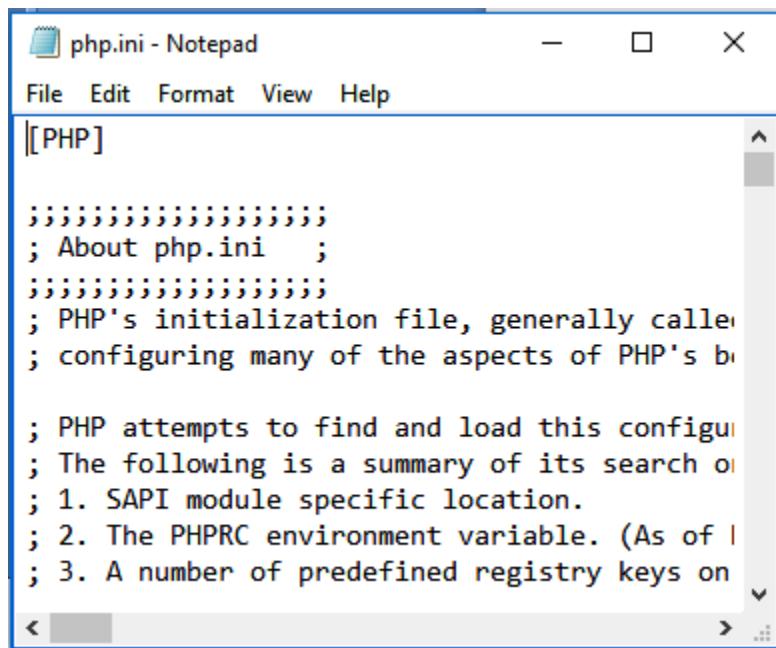
Photo :  No file chosen



# File Size Upload Limit

- For file uploading and PHP script execution there is default configuration in **PHP.ini**.
- However almost hosting providers give chance to developer to customize this default configuration by override **php.ini** or **.htaccess** .

# File Size Upload Limit (Con.)



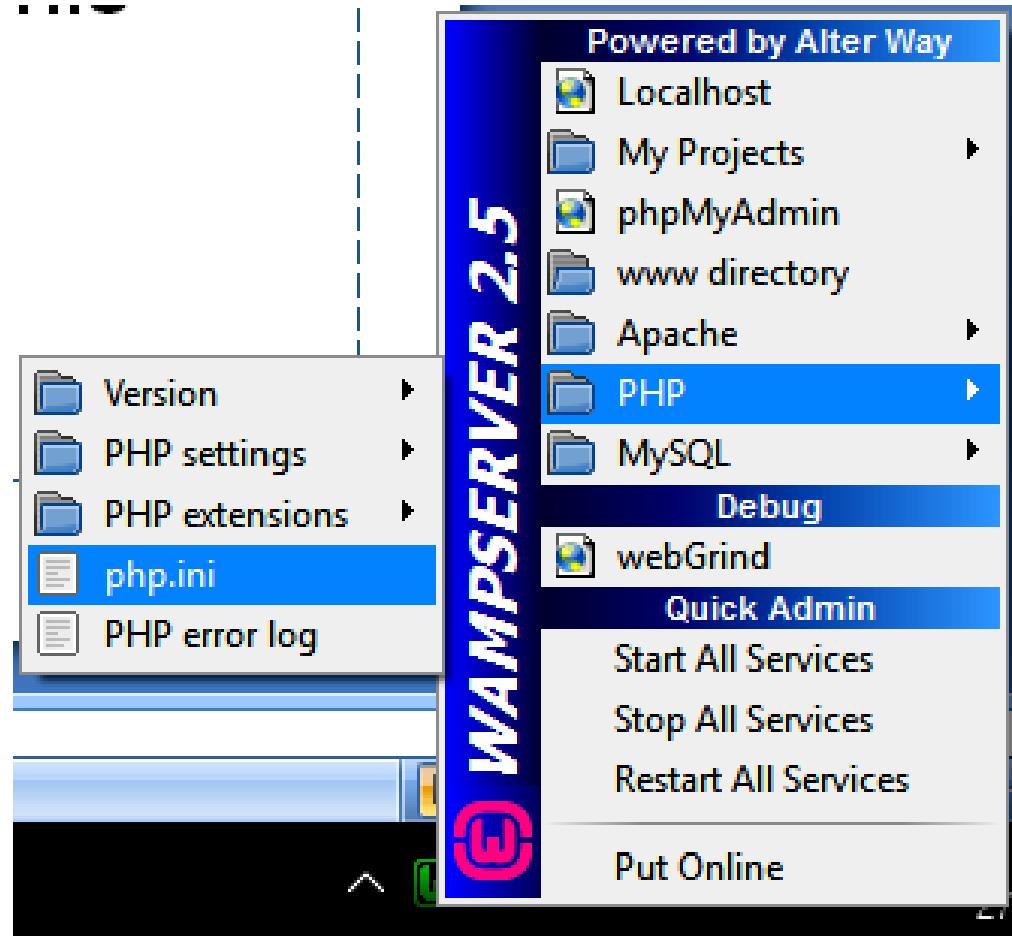
php.ini - Notepad

File Edit Format View Help

[[PHP]]

```
;;;;;;;
; About php.ini ;
;;;;;;;
; PHP's initialization file, generally called
; configuring many of the aspects of PHP's behav

; PHP attempts to find and load this configuration
; The following is a summary of its search order:
; 1. SAPI module specific location.
; 2. The PHPRC environment variable. (As of 4.2.0)
; 3. A number of predefined registry keys on Win32
;
```



# File Size Upload Limit (Con.)

```
;;;;;;;;;;;
; File Uploads ;
;;;;;;;;;;;

; Whether to allow HTTP file uploads.
; http://php.net/file-uploads
file_uploads = On

; Temporary directory for HTTP uploaded files (will use system default if not
; specified).
; http://php.net/upload-tmp-dir
upload_tmp_dir = "c:/wamp/tmp"

; Maximum allowed size for uploaded files.
; http://php.net/upload-max-filesize
upload_max_filesize = 64M

; Maximum number of files that can be uploaded via a single request
max_file_uploads = 20
```

- All uploaded file info is available through the superglobal **\$\_FILES**:

```
Photo : <input type="file" name="photo" accept=".png, .jpg , .jpeg" />
```

- **\$\_FILES** [ 'photo' ] [ 'error' ]
- **\$\_FILES** [ 'photo' ] [ 'name' ]
- **\$\_FILES** [ 'photo' ] [ 'size' ]
- **\$\_FILES** [ 'photo' ] [ 'tmp\_name' ]
- **\$\_FILES** [ 'photo' ] [ 'type' ]

[name] => MyFile.jpg

[type] => image/jpeg

[tmp\_name] => php6hst32

[error] => UPLOAD\_ERR\_OK

[size] => 98174

```
<?php

if(isset($_FILES['image'])) {
 $errors= array();
 $file_name = $_FILES['image']['name'];
 $file_size = $_FILES['image']['size'];
 $file_tmp = $_FILES['image']['tmp_name'];
 $file_type = $_FILES['image']['type'];
 $file_ext=strtolower(end(explode('.',$_FILES['image']['name'])));
}

$expensions= array("jpeg","jpg","png");

if(in_array($file_ext,$expensions)=== false) {
 $errors[]="extension not allowed, please choose a JPEG or PNG file.";
}

if($file_size > 2097152) {
 $errors[]='File size must be less than or equal to 2 MB';
}

if(file_exists("images/".$file_name))
{
 $errors[]='There is a file with the same name';
}

if(empty($errors)==true) {
 move_uploaded_file($file_tmp,"images/".$file_name);
 echo "Success";
} else{
 print_r($errors);
}
}
```

# Some observations

- Always-set form method to POST
- Always-set form enctype to multipart/form-data
- Check files type on client side and server side also.
- Increase the script time limit and memory limit to upload large file.
- Don't use web method (this method) to upload larger than 500mb, instead use ftp upload interface.

# **PHP MySQL Database**

# PHP Connect to MySQL

- PHP 5 and later can work with a MySQL database using:
  - **MySQLi extension** (the "i" stands for improved)
  - **PDO (PHP Data Objects)**

# Should I Use MySQLi or PDO?

- Both MySQLi and PDO have their advantages.
- PDO will work on 12 different database systems, whereas MySQLi will only work with MySQL databases.
  - So, if you have to switch your project to use another database, PDO makes the process easy. You only have to change the connection string and a few queries. With MySQLi, you will need to rewrite the entire code - queries included.
- Both are object-oriented.
- Both support Prepared Statements. Prepared Statements protect from SQL injection, and are very important for web application security.

# Open a Connection to MySQL

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);

// Check connection
if ($conn->connect_error) {
 die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
```

# Open a Connection to MySQL

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
 $conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);
 // set the PDO error mode to exception
 $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
 echo "Connected successfully";
}
catch(PDOException $e)
{
 echo "Connection failed: " . $e->getMessage();
}
?>
```

```
try {
 $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
 // set the PDO error mode to exception
 $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
 $sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";
 // use exec() because no results are returned
 $conn->exec($sql);
 echo "New record created successfully";
}
catch(PDOException $e)
{
 echo $sql . "
" . $e->getMessage();
}

$conn = null;
```

# PHP Get ID of Last Inserted Record

```
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";
// use exec() because no results are returned
$conn->exec($sql);
$last_id = $conn->lastInsertId();
```

# PHP Insert Multiple Records

```
$conn->beginTransaction();
// our SQL statements
$conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')");
$conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com')");
$conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')");

// commit the transaction
$conn->commit();
echo "New records created successfully";
```

# Prepared Statements and Bound Parameters

- A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.
- Prepared statements basically work like this:
  - **Prepare:**
    - An SQL statement template is created and sent to the database. Certain values are left unspecified.
  - The database parses, **compiles**, and **performs query optimization** on the SQL statement template, and stores the result without executing it.
  - **Execute:**
    - At a later time, the application binds the values to the parameters, and the database executes the statement.
    - The application may execute the statement as many times as it wants with different values

# **prepared statements have three main advantages**

- Prepared statements reduces parsing time as the preparation on the query is done only once (although the statement is executed multiple times).
- Bound parameters minimize bandwidth to the server as you need send only the parameters each time, and not the whole query.
- Prepared statements are very useful against SQL injections.

# Prepared Statement Example

```
$conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
// set the PDO error mode to exception
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

```
// prepare sql and bind parameters
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email)
VALUES (:firstname, :lastname, :email)");
$stmt->bindParam(':firstname', $firstname);
$stmt->bindParam(':lastname', $lastname);
$stmt->bindParam(':email', $email);
```

```
// insert a row
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";
$stmt->execute();
```

```
// insert another row
$firstname = "Mary";
$lastname = "Moe";
$email = "mary@example.com";
$stmt->execute();
```

```
<?php
require_once("Class.Tools.php");

class Faculty
{
 static function getAllFaculties()
 {
 $pdo=Database::connect();
 $query=$pdo->prepare("select * from faculty");
 $query->execute();
 return $query;
 }
}
?>

require_once('Class.Faculty.php');
$data=Faculty::getAllFaculties();
$row=$data->fetch(2);
$fields=array_keys($row);

echo "<table border='1' cellspacing='0'><tr>";

for ($i=0;$i<count($fields);$i++)
echo "<th>$fields[$i]</th>";
echo "</tr>";

do
{
 echo "<tr>";
 foreach($row as $key=>$value)
 echo "<td>$value</td>";
 echo "</tr>";
}
while ($row=$data->fetch(2));

echo "</table>";
```

# PHP Sessions

# xHTML - a 'stateless' environment

## **stateless**

(adj.) Having no information about what occurred previously.

- Each request for a new web page is processed without any knowledge of previous pages requested or processed.

# Maintain state

Most modern applications *maintain* state, which means that they remember what you were doing last time you ran the application, and they remember all your configuration settings. This is extremely useful because it means you can mould the application to your working habits.

For example:

A user ‘logs in’ to a web page. Once logged in, the user can browse the site while maintaining their logged in **state**.

# Is PHP stateless?

- Variables are destroyed as soon as the page script finishes executing.
- It is possible to add data to a database/text file to add persistent data, although this is not connected with a particular user...

# Is PHP Stateless... No!

- The usual way to maintain state in PHP pages is via the use of ***Sessions***.
- Another way to maintain state in PHP is via the use of **Cookies**.

# Sessions

- A session is a way to store information (in variables) to be used across multiple pages.
  - (e.g. username, favorite color, etc).
- By default, session variables last until the user closes the browser.
- So; Session variables hold information about one single user, and are available to all pages in one application.

Client



Server



**USER1: First Request**

**start\_session():**

- Create new session with a unique identifier number e.g. **\$1**.
- Create a new file with a name equal to the session id.
- Put some data as we like in the session.

**USER1: First Response**

**Set\_Cookie: PHPSESSID=\$1**

Browser save PHPSESSID in a cookie.

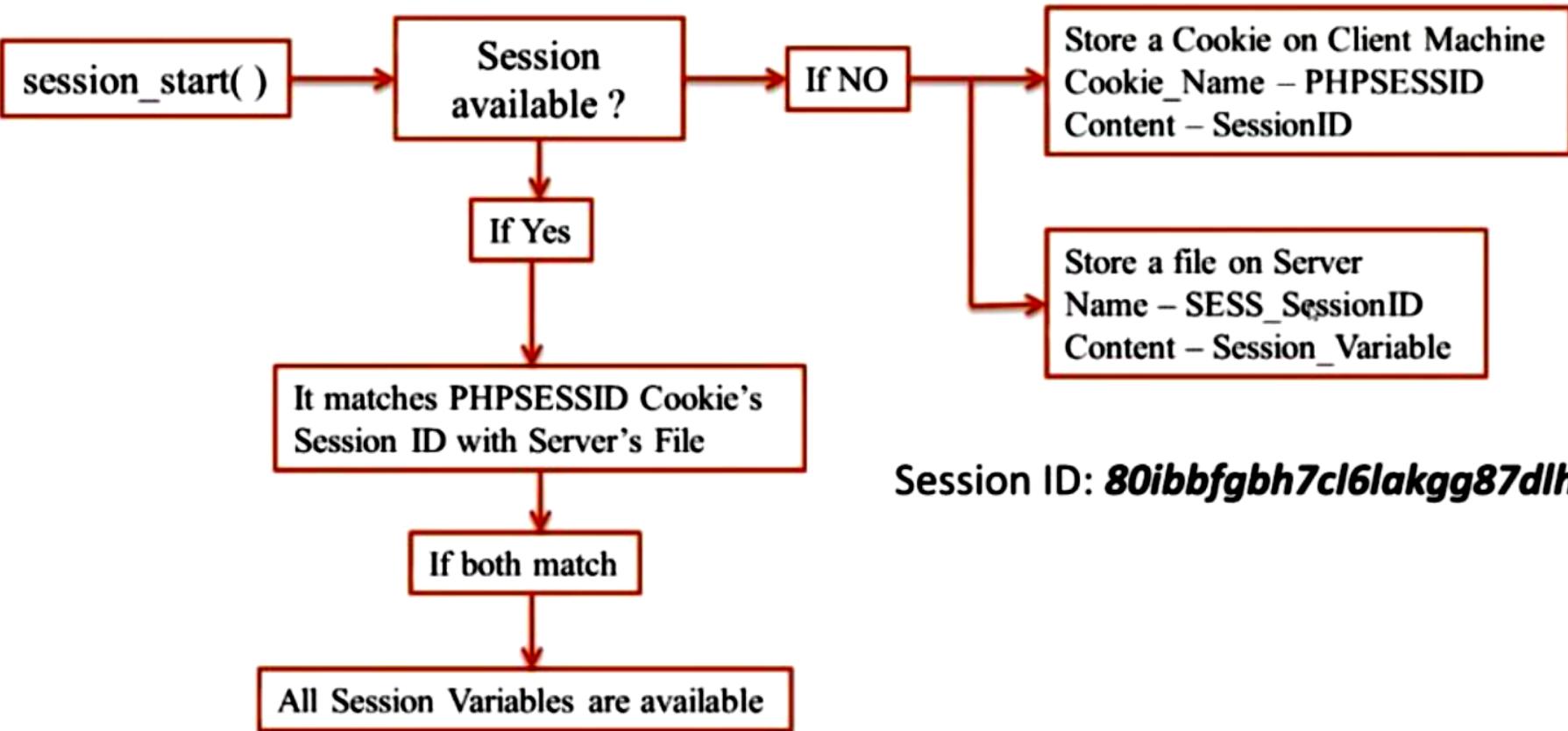
When the user send a new request, the browser put the PHPSESSID cookie in the HTTP request.

**USER1: Second Request; CookiePHPSESSID=\$1**

**start\_session():**

- Find the file that is associated to the session id received by the cookie and retrieve its content to be available in the page.

# How does a PHP session work?



Session ID: **80ibbfgh7cl6lakgg87dlhv7v**

# How does a PHP session work?

1. PHP creates a 16-byte long unique identifier number (stored as a string of 32 hexadecimal characters, e.g a86b10aeb5cd56434f8691799b1d9360) for an individual session.
2. PHPSESSID cookie passes that unique identification number to users' browser to save that number.

# How does a PHP session work?

3. A new file is created on the server with the same name of unique identification number with sess\_ prefix (ie sess\_a86b10aeb5cd56434f8691799b1d9360.)
4. The browser sends that cookie to the server with each request.

# How does a PHP session work?

5. If PHP gets that unique identification number from PHPSESSID cookie (on each request).
  - Then PHP searches in the temporary directory and compares that number to the file name.
    - If both are the same, then it retrieves the existing session.

# Destroy the Session

- A session gets destroyed when the user:
  - Closes the browser
  - or leaves the site.
  - Or the server terminates the session after the predetermined period of session time expires.

# Create a Session

Start a PHP Session:

- A session is started with the `session_start()` function.
- Session variables are set with the PHP global variable: `$_SESSION`.

```
<?php
// Start the session
session_start();
?
<!DOCTYPE html>
<html>
<body>
```

```
<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>
```

# Get Session Variable Values

- From any page in the current session, we can access the session information we set on ant other page.

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".
";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>
```

# Modify a PHP Session Variable

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
```

# Destroy a PHP Session

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>
```

# PHP Cookies

# What is a Cookie?

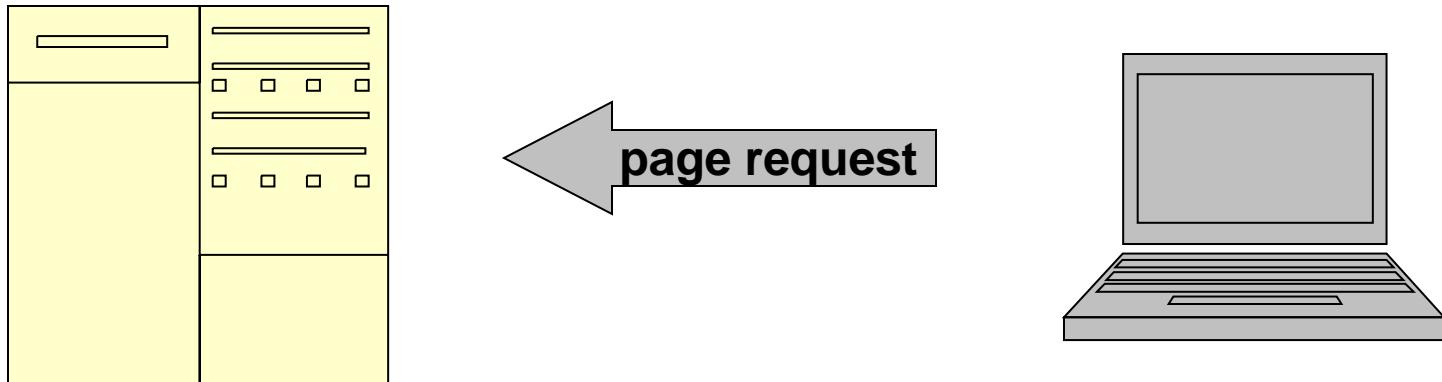
- A cookie is a small text file that is stored on a user's computer.
- Each cookie on the user's computer is connected to a particular domain.

Browser	Cookie count limit per domain	Total size of cookies
Chrome	180	4096
Firefox	150	4097
Opera	60	4096
Safari	600	4093

Feb 3, 2022

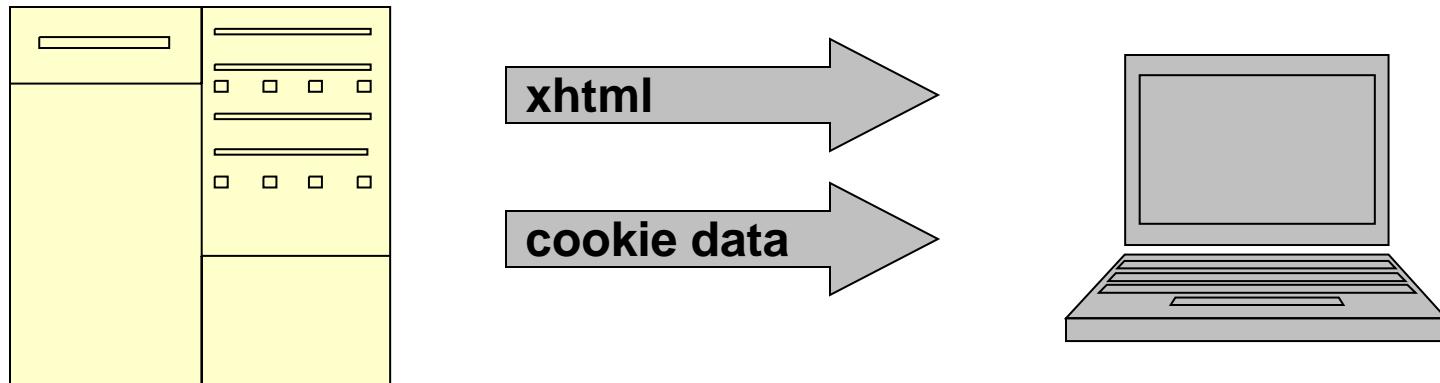
# Example (1)

1. User sends a request for page at [www.example.com](http://www.example.com) for the *first* time.



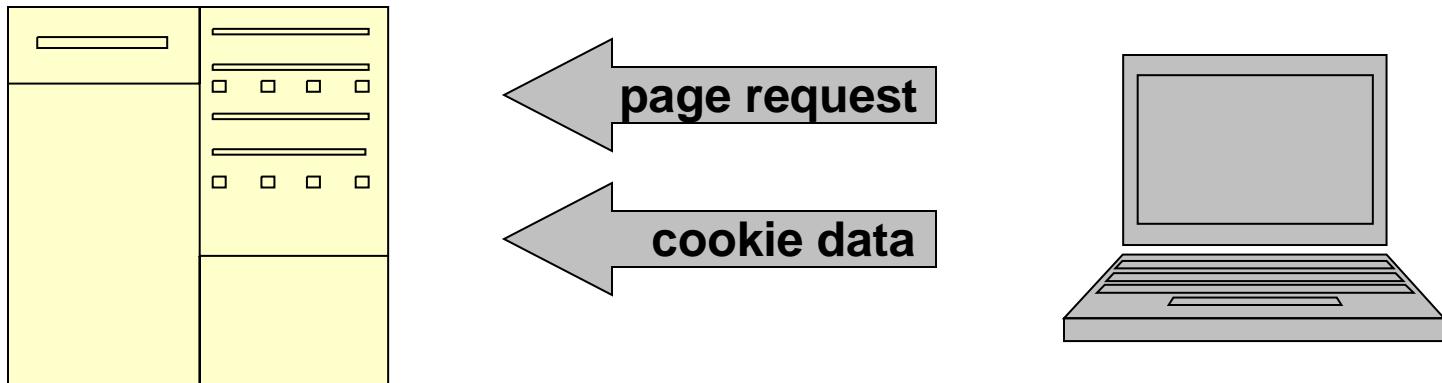
# Example (1) [con.]

2. Server sends back the page `xhtml` to the browser AND stores some data in a cookie on the user's PC.



# Example (1) [con.]

- At the next page request for domain [www.example.com](http://www.example.com), all cookie data associated with this domain is sent too.



# Set a cookie

**setcookie(name ,value ,expire ,path ,domain ,secure)**

**name** = cookie name

**value** = data to store (string)

**expire** = timestamp when the cookie expires.  
Default is that cookie expires when browser  
is closed.

**path** = Path on the server within and below  
which the cookie is available on.

**domain** = Domain at which the cookie is  
available for.

**secure** = If cookie should be sent over HTTPS  
connection only. Default false.

# Set a cookie - example

```
setcookie('name' , 'Ahmad')
```

- This command will set the cookie called name on the user's PC containing the data Ahmad.
- It will expire and be deleted when the browser is closed (default **expire**).
- It will be available to all pages in the same directory or subdirectory of the page that set it (the default **path** and **domain**).

# Set a cookie - examples

```
setcookie('gender' , 'male' , 0 , '/')
```

- This command will set the cookie called gender on the user's PC containing the data male.
- It will expire and be deleted when the browser is closed.
- It will be available to all pages in the same directory or subdirectory of the page that set it.
- It will be available within the entire domain that set it.

# Set a cookie - example

```
setcookie('age','20',time() + 60*60*24*30)
```

- This command will set the cookie called age on the user's PC containing the data 20.
- It will expire and be deleted after 30 days.
- It will be available to all pages in the same directory or subdirectory of the page that set it (the default **path** and **domain**).

# Read cookie data

- All cookie data is available through the superglobal **\$\_COOKIE**:

```
$variable = $_COOKIE['cookie_name']
```

or

```
$variable = $HTTP_COOKIE_VARS['cookie_name'];
```

e.g.

```
$age = $_COOKIE['age']
```

# Storing an array using cookie

- Only **strings** can be stored in Cookie files.
  - To store an array in a cookie:
    - Convert it to a string by using the **serialize()** PHP function.
  - The array can be reconstructed using the `unserialize()` function once it had been read back in.
- Remember cookie size is limited!

```
<body>
<?php
$test=array('name'=>'Ahmad','age'=>'25');
$s=serialize($test);
var_dump($test);
var_dump($s);
$u=unserialize($s);
var_dump($u);
?>
</body>
```

```
array (size=2)
 'name' => string 'Ahmad' (Length=5)
 'age' => string '25' (Length=2)

string 'a:2:{s:4:"name";s:5:"Ahmad";s:3:"age";s:2:"25";}' (Length=48)

array (size=2)
 'name' => string 'Ahmad' (length=5)
 'age' => string '25' (Length=2)
```

# Delete a cookie

- To remove a cookie, simply overwrite the cookie with a new one with an expiry time in the past...

```
setcookie('cookie_name', '', time() - 6000)
```

- Note that theoretically any number taken away from the `time()` function should do, but due to variations in local computer times, it is advisable to use a day or two.

# To be first.. HEADER REQUESTS

- As the **setcookie** command involves sending a HTTP header request, it must be executed **before any xhtml is echoed to the browser, including whitespace.**

```
1 <?
2 setcookie('name','Robert');
3 ?>
4 <html>
5 <head>
```

correct!

```
1 <?
2 setcookie('name','Robert');
3 ?>
4 <html>
5 <head>
6 <head>
```

echoed  
whitespace  
before  
**setcookie**

incorrect.

# PHP File Handling

# PHP Manipulating Files

- PHP has several functions for creating, reading, uploading, and editing files.
- When you are manipulating files you must be very careful.
  - You can do a lot of damage if you do something wrong.
    - Common errors are:
      - Editing the wrong file.
      - Filling a hard-drive with garbage data.
      - Deleting the content of a file by accident.

# PHP readfile() Function

- The readfile() function reads a file and writes it to the output buffer.
- Assume we have a text file called “test.txt”, stored on the server, that looks like :

```
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language
```

- ```
<?php  
echo readfile("test.txt");  
?>
```

```
AJAX = Asynchronous JavaScript and XML CSS = Cascading Style Sheets  
HTML = Hyper Text Markup Language PHP = PHP Hypertext Preprocessor SQL  
= Structured Query Language SVG = Scalable Vector Graphics XML =  
EXtensible Markup Language236
```

PHP Open File - fopen()

- A better method to open files is with the `fopen()` function.
 - This function gives you more options than the `readfile()` function.
- ```
<?php
$myfile = fopen("test.txt", "r") or die("Unable to open file!");
?>
```

# Open mode

- The file may be opened in one of the following modes:

Modes	Description
r	<b>Open a file for read only.</b> File pointer starts at the beginning of the file
w	<b>Open a file for write only.</b> Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a	<b>Open a file for write only.</b> The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x	<b>Creates a new file for write only.</b> Returns FALSE and an error if file already exists
r+	<b>Open a file for read/write.</b> File pointer starts at the beginning of the file
w+	<b>Open a file for read/write.</b> Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a+	<b>Open a file for read/write.</b> The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x+	<b>Creates a new file for read/write.</b> Returns FALSE and an error if file already exists

# PHP Read File - fread()

```
<?php
$myfile = fopen("text.txt", "r") or die("Unable to open file!");
echo fread($myfile,filesize("test.txt"));
?>
```

- The fread() function reads from an open file.
- The first parameter of fread() contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

# PHP Close File - fclose()

- The fclose() function is used to close an open file.
- It's a good programming practice to close all files after you have finished with them.
  - You don't want an open file running around on your server taking up resources!

```
<?php
$myfile = fopen("test.txt", "r");
// some code to be executed....
fclose($myfile);
?>
```

# PHP Read Single Line - fgets()

- The fgets() function is used to read a single line from a file.
  - After a call to the fgets() function, the file pointer has moved to the next line.
  - The example below outputs the first line of the “test.txt” file:

```
<?php
myfile = fopen("test.txt", "r") or die("Unable to open file!");
echo fgets($myfile);
fclose($myfile);
?>
```

# PHP Check End-Of-File - feof()

- The feof() function checks if the "end-of-file" (EOF) has been reached.
  - The feof() function is useful for looping through data of unknown length.
  - The example below reads the “test.txt” file line by line, until end-of-file is reached:

```
<?php
$myfile = fopen("testy.txt", "r") or die("Unable to open");
while(!feof($myfile)) {
 echo fgets($myfile) . "
";
}
fclose($myfile);
?>
```

# PHP Read Single Character - fgetc()

- The fgetc() function is used to read a single character from a file.
  - After a call to the fgetc() function, the file pointer moves to the next character.
  - The example below reads the “test.txt” file character by character, until end-of-file is reached:

```
-<?php
myfile = fopen("test.txt", "r") or die("Unable to open");
while(!feof($myfile)) {
 echo fgetc($myfile);
}
fclose($myfile);
?>
```

# PHP File Create/Write

- The fopen() function is also used to create a file.
  - If you use fopen() on a file that does not exist, it will create it, given that the file is opened for writing (w) or appending (a).
  - The example below creates a new file called "testfile.txt". The file will be created in the same directory where the PHP code resides:
- `$myfile = fopen ("testfile.txt", "w")`

# PHP Write to File - fwrite()

- The fwrite() function is used to write to a file.
  - The first parameter of fwrite() contains the name of the file to write to and the second parameter is the string to be written.
  - The example below writes a couple of lines into a new file called "newfile.txt":
- ```
<?php
myfile =
fopen("newfile.txt", "w") or die("Error");
fwrite($myfile, "PTUK\n");
fwrite($myfile, "Faculty of Engineering");
fclose($myfile);
?>
```

PHP Overwriting

- When we open an existing file for writing. All the existing data will be ERASED and we start with an empty file.

JSON

JSON Introduction

- **JSON:** JavaScript Object Notation.
- **JSON** is a syntax for storing and exchanging data.
- **JSON** is text, written with JavaScript object notation.

Exchanging Data

- When exchanging data between a browser and a server, the data can only be text.
- JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.
- We can also convert any JSON received from the server into JavaScript objects.

JSON simple example

```
<!DOCTYPE html>
<html>
<body>

<div id="test">
</div>

<script>
var myObj = { name: "John", age: 31, city: "New York" };
var myJSON = JSON.stringify(myObj);
document.getElementById("test").innerHTML=myJSON;
</script>

</body>
</html>
```

← → ⌂ ⓘ File | C:/Users/admin/Desktop/Untitled-1.html

{"name": "John", "age": 31, "city": "New York"}

Sending \ Receiving Data

```
<script>
var myObj = { name: "John", age: 31, city: "New York" };
var myJSON = JSON.stringify(myObj);
window.location="test.html?x="+myJSON;
</script>
```

esktop/test.html?x={"name":"John","age":31,"city":"New%20York"}

```
<script>

data= window.location.href;
var data=data.split("=");
data=data[1];
data=data.split("%22").join('"');
data=data.split("%20").join(' ');
var obj=JSON.parse(data);
document.getElementById("demo").innerHTML="Name : "+obj.name;
</script>
```

JSON Data Types

- In JSON, values must be one of the following data types:
 - String
 - Number
 - Object (JSON object)
 - Array
 - Boolean
 - null

PHP and JSON

- PHP has some built-in functions to handle JSON.
 - json_encode()
 - json_decode()

PHP - json_encode()

```
<?php  
$age = array("Peter"=>35, "Ben"=>37, "Joe"=>43);  
  
echo json_encode($age);  
?>
```

```
{"Peter":35,"Ben":37,"Joe":43}
```

```
<?php  
$cars = array("Volvo", "BMW", "Toyota");  
  
echo json_encode($cars);  
?>
```

```
["Volvo","BMW","Toyota"]
```

PHP - json_encode()

```
<?php  
$myObj->name = "John";  
$myObj->age = 30;  
$myObj->city = "New York";  
  
$myJSON = json_encode($myObj);  
  
echo $myJSON;  
?>
```

```
{"name":"John","age":30,"city":"New York"}
```

PHP - json_decode()

```
<?php  
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';  
  
$obj = json_decode($jsonobj);  
  
echo $obj->Peter;  
echo $obj->Ben;  
echo $obj->Joe;  
?>
```

353743

PHP - json_decode()

```
<?php  
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';  
  
$obj = json_decode($jsonobj);  
  
foreach($obj as $key => $value) {  
    echo $key . " => " . $value . "<br>";  
}  
?>
```

Peter => 35
Ben => 37
Joe => 43

SERIALIZE VS JSON IN PHP

When we use serialize?

- Use serialize when your application is specific to PHP.
 - This is useful for storing or passing PHP values around without losing their type and structure.
 - Which means it can represent PHP types, including instances of your own classes and you'll get your objects back, still instances of your classes, when unserializing your data.

When we use JSON?

- JSON is not PHP specific.
 - Use JSON when your application is not specific to PHP, i.e it need to exchange data between different platforms.
 - Almost every and each languages have libraries to read/write JSON data.

```
<?php
class A{
    private $Name;
    static $count1=0;
    function A($N) {
        $this->Name=$N;
        self::$count1++;
    }

    function printInfo () {
        echo "My name is ".$this->Name."<br>";
    }
}
$a=new A("Ali");
$ja=json_encode($a);
$sa=serialize($a);
$ja2=json_decode($ja);
$sa2=unserialize($sa);
// $ja2->printInfo(); Error
echo "<br>";
$sa2->printInfo();
```

My name is Ali