

LECTURE 1: INTRO TO WEB PROGRAMMING: LAYERS OF ABSTRACTION

Except where otherwise noted, the contents of this document are Copyright 2012 Marty Stepp, Jessica Miller, Victoria Kirst and Roy McElmurry IV. All rights reserved. Any redistribution, reproduction, transmission, or storage of part or all of the contents in any form is prohibited without the author's expressed written permission. Slides have been modified for Maharishi University of Management Computer Science course CS472 in accordance with instructors agreement with authors.

Maharishi University of Management -Fairfield, Iowa © 2016



All rights reserved. No part of this slide presentation may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage and retrieval system, without permission in writing from Maharishi University of Management.

Wholeness Statement

In this lecture we introduce the basic technologies that make up the Internet, the World Wide Web and the Hyper Text Markup Language (HTML). We will see that many technologies are built on top of other technologies.

The universe is organized in layers. The most basic layer is the unified field of pure intelligence.

The Internet

- A connection of computer networks using the Internet Protocol (IP)
- layers of communication protocols: IP → TCP/UDP → HTTP/FTP/POP/“MTP/“H...
- What's the difference between the Internet and the World Wide Web (WWW)?
- The Web is the collection of web sites and pages around the world; the Internet is larger and also includes other services such as email, chat, online games, etc.

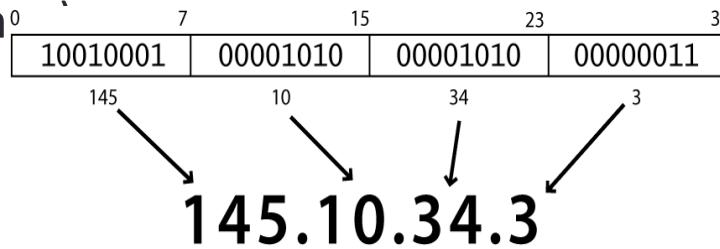


Brief history

- 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; Facebook 2004; Uber 2009
- https://en.wikipedia.org/wiki/History_of_the_Internet

Internet Protocol (IP) IPv4

- The underlying system of communication for all data sent across the Internet.
- Each device has a 32-bit IP address written as four 8-bit numbers (0-255)
- There are two types of IP addresses, servers used to have static IP address while users usually get a dynamic IP address from their ISP.
- Find out your local IP address: in a terminal, type: ipconfig (Windows) or ifconfig (Mac/Lin



- IPv6 addresses are 128-bit IP address written in hexadecimal and separated by colons. An example IPv6 address could be written like this: 3ffe:1900:4545:3:200:f8ff:fe21:67cf

Transmission Control Protocol (TCP)

- Adds multiplexing, guaranteed message delivery on top of IP
- **Multiplexing:** multiple programs using the same IP address
 - **port:** a number given to each program or service
 - port 80: web browser (port 443 for secure browsing)
 - port 25: email
 - port 22: ssh
 - Port 21: File transfer (FTP)
 - port 5190: AOL Instant Messenger
 - [more common ports](#)
- Some programs (games, streaming media programs) use simpler UDP protocol instead of TCP

Web servers and Browsers

- **Web server:** software that listens for web page requests
 - [Apache](#)
 - Microsoft Internet Information Server (IIS)
- **Web browser:** fetches/displays documents from web servers
 - [Mozilla Firefox](#)
 - Microsoft [Internet Explorer](#) (IE)
 - Apple [Safari](#)
 - [Google Chrome](#)
 - [Opera](#)

Domain Name System (DNS)

- A set of servers that map written names to IP addresses
 - Example: www.cs.mum.edu → 69.18.50.54

Uniform Resource Locator (URL)

An identifier for the location of a document on a web site
a basic

URL:`http://www.abc.com/info/index.html`

~~~~ ~~~~~ ~~~~~

protocol      host                  path

- Upon entering this URL into the browser, it would:
- Ask the DNS server for the IP address of `www.abc.com`
- Connect to that IP address at port 80
- Ask the server to GET `/info/index.html`
- Display the result page on the screen

# More advanced URLs

- **Anchor:** jumps to a given section of a web page
  - <http://www.textpad.com/download/index.html#downloads>
  - fetches `index.html` then jumps down to part of the page labeled `downloads`
- **Port:** for web servers on ports other than the default 80
  - <http://www.cs.mum.edu:8080/secret/money.txt>
- **Query string:** a set of parameters passed to a web program
  - <http://www.google.com/search?q=miserable+failure&start=10>
  - parameter `q` is set to "miserable+failure"
  - parameter `start` is set to 10

# Hypertext Transport Protocol (HTTP)

- a set of commands that a computer can send to a server to request files
- Some HTTP commands (your browser sends these internally):
  - **GET**: Requests a specific file or resource from the server
  - **POST**: Submits form information to the server
  - **PUT**: Uploads a file to the server
  - **HEAD**: Requests information about a file from the server, but not the file's entire contents.

# HTTP request and response messages

- Request

GET /index.html HTTP/1.1  
HOST: mumstudents.org

- Response

HTTP/1.1 200 OK

Date: Sun, 30 Aug 2015 16:08:38 GMT

Server: Apache/2.4.7 (Ubuntu)

Last-Modified: Fri, 12 Dec 2014 20:23:01 GMT

Accept-Ranges: bytes

Content-Length: 942

Content-Type: text/html

<!DOCTYPE html>

<html>

...

# Request and Response

The screenshot shows a web browser window with three tabs open: "Gmail: Email from Google", "Gmail: Email from Google", and "CS472: Web Application P". The main content area displays the "CS472: Web Programming" page, which includes the MUM logo and a large title. Below the title is a network performance timeline visualization.

The network timeline shows four requests:

| Name                                           | Method | Status           | Type        | Initiator                  | Size Content      | Time Latency | Timeline |
|------------------------------------------------|--------|------------------|-------------|----------------------------|-------------------|--------------|----------|
| mumstudents.org /cs472                         | GET    | 200 OK           | text/html   | Other                      | 3.03KB<br>58.97KB | 16ms<br>12ms | 4ms      |
| style.css /cs472                               | GET    | 304 Not Modified | text/css    | mumstudents.org:<br>Parser | 210B<br>889B      | 11ms<br>7ms  |          |
| jquery.min.js ajax.googleapis.com/ajax/libs/jq | GET    | 304 Not Modified | text/jav... | mumstudents.org:<br>Parser | 136B<br>90.89KB   | 49ms<br>48ms |          |
| MUM_Logo_Blue_Small.png /cs472/images          | GET    | 304 Not Modified | image/png   | mumstudents.org:<br>Parser | 188B<br>16.44KB   | 29ms<br>3ms  |          |

At the bottom of the timeline, it says "4 requests | 3.55KB transferred | 320ms ( onload: 322ms, DOMContentLoaded: 319ms)".

The bottom navigation bar includes icons for search, refresh, and other developer tools, along with tabs for "All", "Documents", "Stylesheets", "Images", "Scripts", "XHR", "Fonts", "WebSockets", and "Other". There is also a gear icon for settings and a red "x1" icon.

# HTTP status code

- When you request a document from a web server, it sends this document back to you, along with a number called HTTP status code.

| Number                               | Meaning                                     |
|--------------------------------------|---------------------------------------------|
| 200                                  | OK                                          |
| 301-303                              | page has moved (permanently or temporarily) |
| 403                                  | you are forbidden to access this page       |
| 404                                  | page not found                              |
| 500                                  | internal server error                       |
| <a href="#"><u>complete list</u></a> |                                             |

# MIME Types (Multi-Purpose Internet Mail Extension)

- A two-part identifier which many web protocols use to categorize each type of data. broad type/subtype

| MIME Type                             | File extension(s) | Description           |
|---------------------------------------|-------------------|-----------------------|
| application/octet-stream              | .exe              | Executable programs   |
| audio/mpeg                            | .mp3, .mpg        | MPEG or MP3 music     |
| image/gif, image/jpeg,<br>image/png   | .gif, .jpg, .png  | GIF, JPEG, PNG images |
| text/css                              | .css              | Style sheets          |
| text/html                             | .html, .htm, .php | Web pages             |
| <a href="#"><u>MIME Type list</u></a> |                   |                       |

# Web languages / technologies

- Hypertext Markup Language ([HTML](#)): used for writing web pages
- Cascading Style Sheets ([CSS](#)): stylistic info for web pages
- PHP Hypertext Processor ([PHP](#)): dynamically create pages on a web server
- [JavaScript](#): interactive and programmable web pages
- Asynchronous JavaScript and XML ([Ajax](#)): accessing data for web applications
- Extensible Markup Language ([XML](#)): metalanguage for organizing data
- JavaScript Object Notation ([JSON](#)): lightweight data-interchange format that is largely replacing XML in modern web apps

# How browsers display a webpage

1. User machines have IP address on the Internet
2. Server machines have IP address and Domain Name
3. Domain names and IP addresses are registered at global DNS Server
4. When the user opens a browser window and asks for [www.test.com](http://www.test.com)
5. First, the browser will check the local DNS (host file) for the IP address of that domain
6. If not found, it will connect to ISP and ask it for the DNS
7. Once retrieved, the browser will send another request to that server
8. Requests are carried by the IP protocol, sent by the TCP protocol, and represented by HTTP or HTTPS protocol
9. The server will send the browser a response with HTML code.
10. The browser will interpret the HTML code line by line and start building the web page.
11. For every resource not found in the browser cache, the browser will send a new request to the server again asking for that resource and so on.

# Main Point

The internet is a global computer network that uses IP addresses to uniquely identify computers on the network. Through the TCP protocol each IP address can work with multiple services at the same time. One of these services is the HTTP protocol which is used by the World Wide Web to transport HTML pages. These are many layers of the internet.

*During the TM technique, the mind experiences the most fundamental layer of the universe, the field of pure intelligence.*

# Hypertext Markup Language (HTML)

- Describes the content and structure of information on a web page
- Surrounds text content with opening and closing tags
- Each tag's name is called an element
  - Syntax: `<element>content</element>`
  - Example: `<p>This is a paragraph</p>`
- Most whitespace is insignificant in HTML (ignored or collapsed to a single space)
- The newest version is HTML5

# Structure of an HTML5 page

- The **header** describes the page and the **body** contains the page's contents
  - An HTML page is saved into a file ending with extension **.html**
  - **DOCTYPE** tag tells browser the HTML version.
    - In this case, HTML5.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title></title>
  </head>
  <body>
    Page contents
  </body>
</html>
```

# Web Page Metadata

- Data about data - Information describes the page itself

```
<meta charset="utf-8"/>
<meta name="description" content="Learn HTML"/>
<meta name="keywords" content="HTML,CSS"/>
<meta name="viewport" content="width=device-width,
initial-scale=1.0"/>
```

- Placed in the **head** section of your HTML page
- **meta** tags often have both the **name** and **content** attributes
- HTML5 introduced a method to let web designers take control over the viewport (the user's visible area of a web page), through the <meta> tag

# Metadata

| Attribute                         | Value                                                                          | Description                                                                |
|-----------------------------------|--------------------------------------------------------------------------------|----------------------------------------------------------------------------|
| <a href="#"><u>charset</u></a>    | character_set                                                                  | Specifies the character encoding for the HTML document                     |
| <a href="#"><u>name</u></a>       | application-name<br>author<br>description<br>generator<br>keywords<br>viewport | Specifies a name for the metadata                                          |
| <a href="#"><u>content</u></a>    | text                                                                           | Gives the value associated with the http-equiv or name attribute           |
| <a href="#"><u>http-equiv</u></a> | content-type<br>default-style<br>refresh                                       | Provides an HTTP header for the information/value of the content attribute |

# Favorites icon ("favicon")

```
<link href="filename" type="MIME type" rel="relationship"/>  
<link href="yahoo.gif" type="image/gif" rel="shortcut icon"/>
```

- The link tag, placed in the head section, attaches another file to the page
- In this case, an icon to be placed in the browser title bar and bookmarks
- E.g. [https://www.w3schools.com/tags/tryit.asp?filename=tryhtml5\\_link\\_sizes](https://www.w3schools.com/tags/tryit.asp?filename=tryhtml5_link_sizes)



# Relative vs Absolute URL

- **Relative URL**
  - index.html (path relative)
  - graphics/image.png
  - ../about.html (directory relative)
  - ../../stories/stories.html
  - /image.png (root relative)
- **Absolute URL**
  - <http://www.mysite.com>
  - C:\website\images\image.png
- Why won't the last example work when we move our code to production?

# Common Error on Relative URL

- Placing a leading “/” on a relative URL

<http://www.myexample.com/mysite/files/index.html>

<http://www.myexample.com/mysite/files/images/smiley.gif>

<img src=“/images/smiley.gif” alt=“smiley face”/>

Looking for URL:

<http://www.myexample.com/images/smiley.gif>

<img src=“images/smiley.gif” alt=“smiley face”/>



# 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>



# Paragraph: <u>

- Paragraphs of text (block)

<p>

  Lorem quis lorem. Pellentesque ultrices  
  nunc id mauris posuere pulvinar.

</p>

Placed within the body of the page

- [More paragraph examples](#)

# Headings: <h1>, <h2>, ..., <h6>

- Headings to separate major areas of the page (block)

<h1>**Maharishi University**</h1>

<h2>**Department of Computer Science**</h2>

<h3>**WAP Course**</h3>

Maharishi University

Department of Computer Science

WAP Course



# Horizontal Rule: <hr />

- A line to separate sections (block)

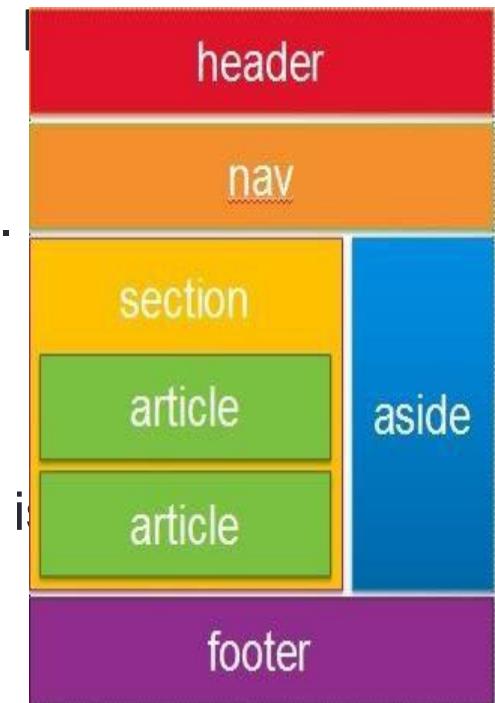
```
<h1>CS472 Web Programming</h1>
<p>Very good, fun course - HTML, CSS, JavaScript,
JSP, Servlets</p>
<hr />
<h1>CS421 Modern Programming Practices</h1>
<p>Profound course - OOAD, Java8 features</p>
```



# New Semantic Elements

They generally have no default outward appearance on the page, instead they give insight into the structure of the page.

- **section**: defines a section in a document.
- **header**: specifies a header for a document or section.
- **footer**: specifies a footer for a document or section.
- **nav**: defines a set of navigation links.
- **aside**: defines some content aside from the content it is in (e.g. sidebar).
- **article**: specifies independent, self-contained content.



[More Semantic Tags](#)

# Main point

The Hyper Text Markup Language uses tags to demarcate different sections of a text. An HTML page always starts with a `<html>` tag, inside of which it has a `<head>` tag to describe the page, and a `<body>` tag of the contents that will actually be displayed. These are the tags you will use for every HTML page.

*The most powerful language is the language of nature, which is the impulses of pure intelligence vibrating within itself.*



# 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>

# Images: <img>

- Inserts a graphical image into the page (inline)
  - Another get request
- The src attribute specifies the image URL
- HTML5 also requires an alt attribute describing the image
- title attribute is an optional tooltip (on ANY element)

```


- Links, or "anchors", to other pages (inline)
- href can be absolute or relative URL
- Anchors are inline elements

```
<p>
  <a href="story1.html">Bruce Wayne, the richest man
in Gotham City, is the alter ego of Batman. Bam!
</a>
</p>
```

- NOTE: In HTML5, you can wrap links around “block-level” elements
- See example: lecture01-examples/link-old.html, lecture01-examples/link-html5.html

# Line break: <br/>

- Forces a line break in the middle of a block element (inline)

<p>Teddy said it was a hat, <br /> So I put it on.</p>

<p>Now Daddy's saying, <br /> Where the heck's the toilet plunger  
gone?</p>

- Warning: Don't over-use br (guideline: >= 2 in a row is bad)
- <br /> should not be used to separate paragraphs or used multiple times in a row to create spacing



# Phrase elements : <em>, <strong>

- em: emphasized text (usually rendered in italic)
- strong: strongly emphasized text (usually rendered in bold)

```
<p>HTML is  
  <em>really</em>, <strong>REALLY</strong>  
    fun!  
</p>
```

HTML is *really*, **REALLY** fun!

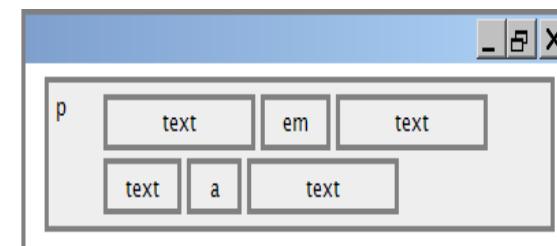
- As usual, the tags must be properly nested for a valid page.

# 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 HTML
- i.e., bad nesting
  - <p> What a <em> lovely </p> day </em>
- Good nesting
  - <p> What a <em> lovely </em> day </p>

# Review: Block elements VS Inline elements

- block elements contain an entire large region of content
  - Examples: paragraphs, lists, table cells
  - The browser places a margin of whitespace between block elements for separation
- inline elements affect a small amount of content
  - Examples: bold text, code fragments, images
  - The browser allows many inline elements to appear on the same line
  - Must be nested inside a block element



# Comments: <!-- -->

- Comments to document your HTML file or "comment out" text
- Many web pages are not thoroughly commented (or at all)
- Useful at the top of page and for disabling code

```
<!-- This is comment -->
```

```
<p>
    WAP courses are<!-- NOT--> a lot
    of fun!
</p>
```



# Unordered list: <ul>, <li>

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

```
<ul>
  <li>No shoes</li>
  <li>No shirt</li>
  <li>No problem!</li>
</ul>
```

- No shoes
- No shirt
- No problem!



# Ordered list: <ol>

- ol represents a numbered list of items (block)
- We can make lists with letters or Roman numerals using CSS (later)

```
<p>RIAA business model:</p>
<ol>
  <li>Sue customers</li>
  <li>Profit!</li>
</ol>
```

RIAA business model:

1. Sue customers
2. Profit!



# Nested Lists

- A list can contain other lists

```
<ul>
  <li>Simpsons:
    <ul>
      <li>Homer</li>
      <li>Marge</li>
    </ul>
  </li>
  <li>Family Guy:
    <ul>
      <li>Peter</li>
      <li>Lois</li>
    </ul>
  </li>
</ul>
```

- **Simpsons**:
  - Homer
  - Marge
- **Family Guy**:
  - Peter
  - Lois



# Definition list: <dl>, <dt>, <dd>

- dl represents a list of definitions of terms (block)
- dt represents each term, and dd its definition

```
<dl>
```

|        |                                  |
|--------|----------------------------------|
| newbie | one who does not have mad skills |
| own    | to soundly defeat                |
| frag   | a kill in a shooting game        |

```
    <dt>newbie</dt>
```

```
    <dd>one who does not have mad skills</dd>
```

```
    <dt>own</dt>
```

```
    <dd>to soundly defeat</dd>
```

```
    <dt>frag</dt>
```

```
    <dd>a kill in a shooting game</dd>
```

```
</dl>
```



# Inline quotations: <q>

- A short quotation (inline)

<p>

Quoth the Raven, <q>Nevermore.</q>

Quoth the Raven, "Nevermore."

Why not just write the following?

<p>Quoth the Raven, "Nevermore."</p>

- Using <q> allows us to apply CSS styles to quotations



# Quotations: <blockquote>

A lengthy quotation (block)

```
<p>As Lincoln said in his famous Gettysburg Address:  
</p>  
<blockquote>  
<p>Fourscore and seven years ago, our  
fathers brought forth on this continent a  
new nation, conceived in liberty, and dedicated  
to the proposition that all men are created  
equal.</p>  
</blockquote>
```

As Lincoln said in his famous Gettysburg Address:

Fourscore and seven years ago, our fathers brought forth on this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal.



# HTML Character Entities

- A way of representing any Unicode character within a web page
- [Complete list of HTML entities](#)
- How would you display the text & on a web page?

| character(s) | entity                    |
|--------------|---------------------------|
| < >          | &lt; &gt;                 |
| é è ñ        | &acute; &egrave; &ntilde; |
| ™ ©          | &trade; &copy;            |
| π δ Δ        | &pi; &delta; &Delta;      |
| И            | &#1048;                   |
| " &          | &quot; &amp;              |



# HTML-encoding text

- To display the link text in a web page, its special characters must be encoded as shown below

&lt;p&gt;

&lt;a href="http://google.com/"&gt; Google&lt;/a&gt;

&lt;/p&gt;

```
<p> <a href="http://google.com/"> Google</a> </p>
```



# Computer code: <code>

- A short section of computer code (usually shown in a fixed-width font) (inline)

```
<p>  
    The <code>ul</code> and  
    <code>ol</code> tags make lists.  
</p>
```

The `ul` and `ol` tags make lists.



# Preformatted text: <pre>

- A large section of pre-formatted text (block)
- a fixed-width font (usually Courier), and it preserves both spaces and line breaks.

```
<pre>
```

**Steve Jobs speaks loudly reality**

```
Steve Jobs speaks loudly reality  
distortion Apple fans bow down
```

**Apple fans bow down**

```
</pre>
```



# Abbreviations: <abbr>

- An abbreviation, acronym, or slang term (inline)

```
<p> Safe divers always remember to check their  
    <abbr title="Self-Contained Underwater Breathing  
        Apparatus">SCUBA</abbr> gear.  
</p>
```

Safe divers always remember to check their SCUBA gear.

Self-Contained Underwater Breathing Apparatus



# HTML tables: <table>, <tr>, <td>

- A 2D table of rows and columns of data (block element)

```
<table>
  <tr>
    <td>1,1</td>
    <td>1,2 okay</td>
  </tr>
  <tr>
    <td>2,1 real wide</td>
    <td>2,2</td>
  </tr>
</table>
```

|               |          |
|---------------|----------|
| 1,1           | 1,2 okay |
| 2,1 real wide | 2,2      |

- **table** defines the overall table, **tr** each row, and **td** each cell's data
- tables are useful for displaying large row/column data sets



# Table headers, captions:

## <th>, <caption>

```
<table>
  <caption>My important data</caption>
  <tr>
    <th>Column 1</th>
    <th>Column 2</th>
  </tr>
  <tr>
    <td>1,1</td>
    <td>1,2 okay</td>
  </tr>
  <tr>
    <td>2,1 real wide</td>
    <td>2,2</td>
  </tr>
</table>
```

| My important data |          |
|-------------------|----------|
| Column 1          | Column 2 |
| 1,1               | 1,2 okay |
| 2,1 real wide     | 2,2      |

- **th** cells in a row are considered headers; by default, they appear bold
- a **caption** at the start of the table labels its meaning



# The **rowspan** and **colspan** attributes

- **colspan** makes a cell occupy multiple columns; rowspan multiple rows

```
<table>
  <tr>
    <th>Column 1</th>
    <th>Column 2</th>
    <th>Column 3</th>
  </tr>
  <tr>
    <td colspan="2">1,1-1,2</td>
    <td rowspan="3">1,3-3,3</td>
  </tr>
  <tr>
    <td>2,1</td>
    <td>2,2</td>
  </tr>
  <tr>
    <td>3,1</td>
    <td>3,2</td>
  </tr>
</table>
```

| Column 1 | Column 2 | Column 3 |
|----------|----------|----------|
| 1,1-1,2  |          |          |
| 2,1      | 2,2      | 1,3-3,3  |
| 3,1      | 3,2      |          |
|          |          |          |

# Don't use tables for layout!

- (borderless) tables appear to be an easy way to achieve grid-like page layouts
- Many "newbie" web pages do this
- But, a table has semantics; it should be used only to represent an actual table of data
- Instead of tables, use divs, widths/margins, floats, etc. to perform layout

# Web Standards

- It is important to write proper HTML code and follow proper syntax.
- Why use valid HTML and web standards?
  - More interoperable across different web browsers
  - More likely that our pages will display correctly in the future



# W3C HTML Validator

- Checks your HTML code to make sure it follows the official HTML syntax
- More picky than the browser, which may render bad HTML correctly

```
<p>
  <a href="http://validator.w3.org/check/referer">
    <img src=http://mumstudents.org/cs472/2013-09/images/w3c-html.png
      alt="Validate" />
  </a>
</p>
```

Note that the validation website expects requests to come from a page that was served by a web server. Will not work with an html page opened from a file. E.g., [mumstudents.org](http://mumstudents.org) web server.



# Main Point

We discussed some of the most common tags. The most important concept is to use tags based on their semantics (meaning), not based on their visual effect (which can easily be changed).

*Through the TM technique, the individual gains access to the language of nature to effortlessly fulfill any desire.*

# CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESSENCE OF KNOWLEDGE

## *The Web Programming Environment*

1. HTML is the basis of Web Programming, every web page is made of HTML.
  2. To be an effective web programmer you also have to understand the deeper underlying realities of HTTP, TCP, and DNS.
- 

3. **Transcendental consciousness** is the experience of the deepest underlying reality, the unified field.
4. **Impulses within the Transcendental field:** the infinite dynamism of the unified field expresses itself into the universe.
5. **Wholeness moving within itself:** In Unity Consciousness, one experiences that this infinite dynamism is the Self.





# CS472 Web Programming

## Lecture 2: CSS

---

Except where otherwise noted, the contents of this document are Copyright 2012 Marty Stepp, Jessica Miller, Victoria Kirst and Roy McElmurry IV. All rights reserved. Any redistribution, reproduction, transmission, or storage of part or all of the contents in any form is prohibited without the author's expressed written permission. Slides have been modified for Maharishi University of Management Computer Science course CS472 in accordance with instructors agreement with authors.

# Maharishi University of Management - Fairfield, Iowa © 2016



All rights reserved. No part of this slide presentation may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage and retrieval system, without permission in writing from Maharishi University of Management.

# Wholeness Statement

---

In this lecture we introduce the basics of CSS as a way to give different visual styles to HTML elements, changing their preset default appearance. *The impulses of pure intelligence, the Veda, give rise to all aspects of the universe.*



# The bad way to produce styles

- ▶ Tags such as **strong**, **em**, **u**, and **font** are discouraged in strict HTML

```
<p>  
    <font face="Arial">Welcome to Greasy Joe's.</font>  
    You will <strong>never</strong>, <em>ever</em>,  
    <u>EVER</u> beat  
    <font size="+4" color="red">OUR</font> prices!  
</p>
```

Welcome to Greasy Joe's. You will **never**, **ever**, **EVER** beat **OUR** prices!

# Content vs. Presentation

---

- ▶ HTML is for content, the information on the page
- ▶ CSS is for presentation, how to display the page
- ▶ Keeping content separate from presentation is a very important web design principle
- ▶ If the HTML contains no styles, its entire appearance can be changed by swapping `.css` files
- ▶ <http://csszengarden.com/>



# Bad Practices, why?

---

## Embedding style sheets

```
<head>
    <style type="text/css">
        p { font-family: sans-serif; color:
red; }
        h2 { background-color: yellow; }
    </style>
</head>
```



## Inline styles: the style attribute



```
<p style="font-family: sans-serif; color: red;">This
is a paragraph</p>
```

Note: It has higher precedence than embedded or linked styles





# Cascading Style Sheets (CSS): <link>

- ▶ CSS describes the appearance and layout of information on a web page
  - ▶ (as opposed to HTML, which describes the content of the page)
- ▶ Can be embedded in HTML or placed into separate .css file (preferred)

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

# Main Point

---

- ▶ Cascading Style Sheets (CSS) provide a way for web developers to specify the appearance of content on the page by using selectors to specify the element(s), and then the values that properties of those elements (like font or color) should have. It is a best practice to keep CSS in a separate file, doing so makes your pages more flexible.

# Basic CSS rule syntax

---

- ▶ A **CSS** file consists of one or more rules
- ▶ A rule's selector specifies HTML element(s) and applies style properties
- ▶ The **\*** selector, selects all elements
- ▶ To add a comment we use: **/\* \* \*/**

```
selector {  
    property: value;  
    property: value; ...  
}  
p {  
    font-family: sans-serif;  
    color: red;  
}
```



# CSS properties for colors

```
p {  
    color: white;  
    background-color: blue;  
}
```

This paragraph uses the style above.

| property         | description                               |
|------------------|-------------------------------------------|
| color            | color of the element's text               |
| background-color | color that will appear behind the element |

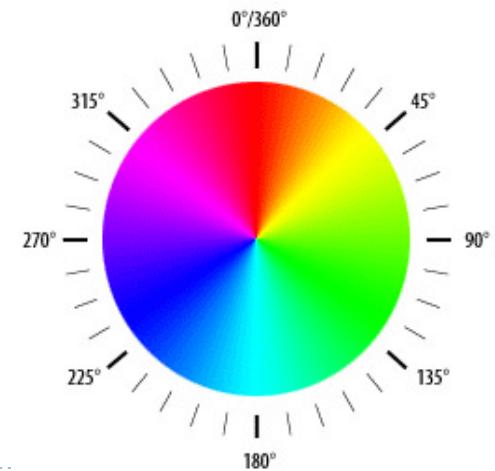
# Specifying colors

- ▶ **Color names:** aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, yellow
- ▶ **RGB & RGBA codes:** red, green, and blue values from 0 to 255
- ▶ **HEX codes:** RGB values in base-16 from 00 (none) to FF (full)
- ▶ **HSL & HSLA codes:** HSL stands for hue, saturation, and lightness
  - ▶ Hue is degree on color wheel (from 0 to 360) - 0 (or 360) is red, 120 is green, 240 is blue.
  - ▶ Hsla demo <https://codepen.io/kman/pen/KwapPZ>
  - ▶ Google css color picker

```
h1 { color: red; }  
h2 { color: rgb(128, 0, 196); }  
h3 { color: rgba(128, 0, 196, 0.5); }
```

```
h4 { color: #FF8800; }
```

```
▶ 12 h5 { color: hsla(120, 60%, 70%, 0.3); }
```



# CSS properties for fonts

| property                                                | description                         | Values                                   |
|---------------------------------------------------------|-------------------------------------|------------------------------------------|
| font-family                                             | which font will be used             | serif or “Courier New”                   |
| font-size                                               | how large the letters will be drawn | A unit value, percentage, or named value |
| font-style                                              | used to enable/disable italic style | normal(default), italic, oblique         |
| font-weight                                             | used to enable/disable bold style   | normal(default), bold, bolder,...        |
| font                                                    | Sets all font properties            | style weigh size family                  |
| <a href="#"><u>Complete list of font properties</u></a> |                                     |                                          |

# CSS properties for fonts

---

```
h1 { /* which font will be used */
      font-family: Georgia;
}
h2 { /* enclose multi-word font names in quotes */
      font-family: "Courier New";
}
h3 { /* can specify multiple fonts from highest to
       lowest priority */
      font-family: Garamond, "Times New Roman", serif;
}
```

- ▶ If the first font is not found on the user's computer, the next is tried.
- ▶ Placing a generic font name at the end of your font-family value ensures that every computer will use a valid font
  - ▶ CSS generic font names: serif, sans-serif, cursive, fantasy, monospace
  - ▶ Serifed fonts easier to read on printed pages, hard to read on computer screens,



## font-size, font-weight, font-style

```
p {  
    /* how large the letters will be drawn */  
    font-size: 14vw;  
    /* used to enable/disable bold style */  
    font-weight: bold;  
    /* used to enable/disable italic style */  
    font-style: italic;  
}
```

***CS472 WAP  
course has a lot  
of fun!***



# Size Units

- ▶ Units: pixels (**px**), point (**pt**), m-size (**em**)
- ▶ **pt** specifies number of points, where a point is 1/72 of an inch on screen
- ▶ **px** specifies a number of pixels on the screen
- ▶ **em** relative to the font-size of the element (2em means 2 times the size of the current font)
  
- ▶ Vague font sizes: xx-small, x-small, small, medium, large, x-large, xx-large, smaller, larger
  
- ▶ <https://webflow.com/blog/how-and-why-to-use-vh-and-vw-in-webflow>
- ▶ [https://www.w3schools.com/cssref/tryit.asp?filename=trycss\\_unit\\_vw](https://www.w3schools.com/cssref/tryit.asp?filename=trycss_unit_vw)

# CSS properties for text

| Property        | Description                          | Values                                                    |
|-----------------|--------------------------------------|-----------------------------------------------------------|
| text-align      | alignment of text within its element | left, center, right, justify                              |
| text-decoration | decorations such as underlining      | underline, overline, line-through, blink, none            |
| text-indent     | Indent first line                    | a size(px, pt, %, em)                                     |
| line-height     | vertical size of each line           | a size(px, pt, %, em)                                     |
| letter-spacing  | Horizontal gap between letters       | a size(px, pt, %, em)                                     |
| word-spacing    | Horizontal gap between words         | a size(px, pt, %, em)                                     |
| text-overflow   | How to handle too-long text          | clip, ellipsis, ellipsis-word                             |
| text-shadow     | A “drop shadow” next to text         | Two distances(px, pt, %, em) plus an optional shaow color |

# CSS properties for text



```
h2 {  
    /* Can also be overline, line-through, blink, or none. Effects can be  
    combined */  
    text-decoration: underline overline;  
    /* Shadow is specified as an X-offset, a Y-offset, and an optional color  
    */  
    text-shadow: -2px 5px gray;  
}  
  
p {  
    /* Alignment of text within its element, can be left, right, center, or  
    justify */  
    text-align: center;  
    /* Space between the lines in two paragraphs */  
    line-height: 30px;  
    /* Space between words in <p> elements should be 30 pixels */  
    word-spacing: 30px;  
    /* Indent the first line of all <p> elements with 50 pixels */  
    text-indent: 50px;  
}
```

# CSS properties for background

| Property                                                                                            | Description                                | Values                                                                           |
|-----------------------------------------------------------------------------------------------------|--------------------------------------------|----------------------------------------------------------------------------------|
| background-color                                                                                    | color to fill background                   | A color                                                                          |
| background-image                                                                                    | image to place in background               | url(image URL)                                                                   |
| background-position                                                                                 | placement of bg image within element       | Two tokens for x/y as top, bottom, left, right, center, or a size(pt, px, %, em) |
| background-repeat                                                                                   | whether/how bg image should be repeated    | repeat(default), repeat-x, repeat-y or no-repeat                                 |
| background-attachment                                                                               | whether bg image scrolls with page         | scroll(default), fixed                                                           |
| background-size  | scaling of bg image                        | a size(pt, px, %, em), cover, or contain                                         |
| background                                                                                          | shorthand to set all background properties |                                                                                  |

# Background



```
body {  
  
    /* image to place in background */  
    background-image: url("../images/draft.jpg");  
  
    /* How bg image should be repeated */  
    /* can be repeat (default), repeat-x, repeat-y, or  
    no-repeat */  
    background-repeat: repeat-x;  
  
    /* placement of bg image within element */ /*  
    value consists of two tokens, can be top, left,  
    right, bottom, center, a percentage, or a length  
    value in px, pt, etc */  
    background-position: 370px 20px;  
}
```

# The `list-style-type` property



- ▶ **none**: No marker
- ▶ **disc**(default), **circle**, **square**
- ▶ **decimal**: 1, 2, 3, etc.
- ▶ **decimal-leading-zero**: 01, 02, 03, etc.
- ▶ **lower-roman** : i, ii, iii, iv, v, etc.
- ▶ **upper-roman** : I, II, III, IV, V, etc.
- ▶ **lower-alpha** : a, b, c, d, e, etc.
- ▶ **upper-alpha** : A, B, C, D, E, etc.
- ▶ **lower-greek** : alpha, beta, gamma, etc.

```
ol {    list-style-type: lower-roman; }
```

i. first item  
ii. second item  
iii. third item

# Main Point

---

We discussed the CSS Properties for color, font, text, background, and lists, which are the basic properties used on almost every page.

<http://www.creativebloq.com/web-design/10-design-concepts-web-developers-need-know-11135255>

# Body styles

---

- ▶ To apply a style to the entire body of your page, write a selector for the body element
- ▶ Saves you from manually applying a style to each element

```
body {  
  font-size: 16px;  
}
```

# Inheriting styles



- ▶ Styles get inherited from containing elements
- ▶ **Not all properties are inherited** (notice link's color below)
  - ▶ E.g., margin

```
body {  
    font-family: sans-serif;  
    background-color: pink;  
}  
  
p {  
    color: green;  
}  
  
a {  
    text-decoration: underline;  
}  
  
h2 {  
    font-weight: bold;  
    text-align: center;  
}
```

## CS472 Web Programming

This [course](#) provides a systematic introduction to programming interactive and dynamic web applications.



# Styles that Conflict

```
/* select multiple elements separated by commas */
p, h1, h2 {
  color: green;
  background-color: grey;
}

/* when two styles set conflicting values for the same
property, the latter style takes precedence */
h2 {
  background-color: blue;
}

<p> This paragraph will use background color grey! </p>
<h2> This heading will use background color blue! </h2>
```

This paragraph will use background color grey!

This heading will use background color blue!

# Chrome Styles Pane

---

- ▶ Invoke the DevTools inspector on element
- ▶ displays styles for selected DOM node
- ▶ gray background are read-only
- ▶ exclamation marks mean that property name and/or value is not understood by the browser,
  - ▶ is ignored
- ▶ A style declaration may contain several properties with the same name. Only the last one takes effect, canceling the preceding ones.
  - ▶ will be struck out, like overridden properties.

# **Cascading style sheets**

---

It's called Cascading Style Sheets because the properties of an element cascade together in this order:

1. Browser's default styles (reference)
  2. External style sheet files (in a <link> tag)
  3. Internal style sheets (in a <style> tag in the page header)
  4. Inline style (the style attribute of an HTML element)
- 
- ▶ Basically, cascading works from top to bottom inside the page (Depends on your order – later styles will always override top ones).
  - ▶ Inheritance is how elements in the HTML markup inherit properties from their parent elements
  - ▶ cascade is how CSS declarations are applied, and how conflicting rules do or don't override each other.



# Override Rules



```
<p class="RedColor BlueColor">  
    Lorem Ipsum  
</p>
```

```
#YellowColor {  
    color: yellow;  
}  
.BlueColor {  
    color: blue;  
}  
.RedColor {  
    color: red;  
}
```

Lo  
rem Ipsum



# Style Specificity

- ▶ When multiple styles apply to an element and have the same origin precedence.
- ▶ The most specific one applies. If they have the same specificity, then the later one will be used.

```
<aside>
  <p><em id="recent" class="awesome">Which awesome color?</em></p>
</aside>
```

```
aside { color: gray; }
p { color: green; }
em { color: yellow; }
.awesome { color: blue; }
em.awesome { color: red; }
#recent { color: black; }
em#recent.awesome { color: orange; }
```

*Which awesome color?*

# Specificity and conflicts

---

- ▶ Specificity- decide which one should win when two or more rules conflict.
- ▶ Rules: each rule's overall selector is given a score based upon approximately the following rules. The rule with the highest score wins if there's a conflict.
  - ▶ Any HTML element mentioned in the rule scores 1 point
  - ▶ Any class mentioned in the rule scores 10 points
  - ▶ Any ID mentioned in the rule scores 100 points
- ▶ Examples:
  - ▶ p.banner - 11
  - ▶ div.box > p - 12
  - ▶ body #logo .box p.banner - 122

# The HTML **class** and **id** attribute

---

- ▶ **id attribute** allows you to give a unique ID to any element on a page
  - ▶ Each ID must be unique; can only be used once in the page
- ▶ **class attribute** is used to group some elements and give a style to only that group
  - ▶ unlike an id, a class can be reused as much as you like on the page

# class vs id examples



```
<p id="mission">Our mission is to provide the most</p>
<p class="special">See our spectacular spatula specials</p>
<p class="special shout">Today only, satisfaction guaranteed</p>
```

```
#mission {
    font-style: italic;
    color: #000000;
}

.special /* any element with class="special" */
background-color: yellow;
font-weight: bold;

}

.p.shout /* only p elements with class="shout" */
color: red;
font-family: cursive;

}
```

*Our mission is to provide the most*

See our spectacular spatula specials

Today only, satisfaction guaranteed

# class naming

---

- ▶ focus on the semantics and meaning of the content vs appearance
- ▶ Bad example: redtext, bigfont
  - ▶ if change style later, it doesn't make sense to be called redtext.
- ▶ Good example:
  - ▶ warningMsg
  - ▶ errorMsg

# CSS pseudo-classes pseudo-elements



- ▶ A **pseudo-class** is used to define a special state of an element
  - ▶ Style an element when a user mouse's over it
  - ▶ Style visited and unvisited links differently
  - ▶ Style an element when it gets focus
  
- ▶ A CSS **pseudo-element** is used to style specified parts of an element
  - ▶ Style the first letter, or line, of an element
    - ▶ ::first-line, ::first-letter
  - ▶ Insert content (pseudo element) before, or after, the content of an element
    - ▶ ::before, ::after

```
selector:pseudo-class { property:value; }
```

```
selector::pseudo-element { property:value; }
```

```
/* double colon notation - ::pseudo-element versus :pseudo-class */
```

# CSS pseudo-classes pseudo-elements

| class                     | description                                           |
|---------------------------|-------------------------------------------------------|
| :active                   | an activated or selected element                      |
| :focus                    | an element that has the keyboard focus                |
| :hover                    | an element that has the mouse over it                 |
| :link                     | a link that has not been visited                      |
| :visited                  | a link that has already been visited                  |
| :nth-child(expr)          | targets specific children of a given element          |
| :first-child, :last-child |                                                       |
| :not(selector)            | all elements that do not match the given CSS selector |
| ::first-line              | the first line of text inside an element              |
| ::first-letter            | the first letter of text inside an element            |

# Examples pseudo-classes



```
/* unvisited link */  
a:link { color: #FF0000; }
```

```
/* visited link */  
a:visited { color: #00FF00; }
```

```
/* mouse over link */  
a:hover { color: #FF00FF; }
```

```
/* click on a link */  
a:active { color: #0000FF; }
```

More info and examples: [Pseudo-classes](#) and [Pseudo-elements](#)

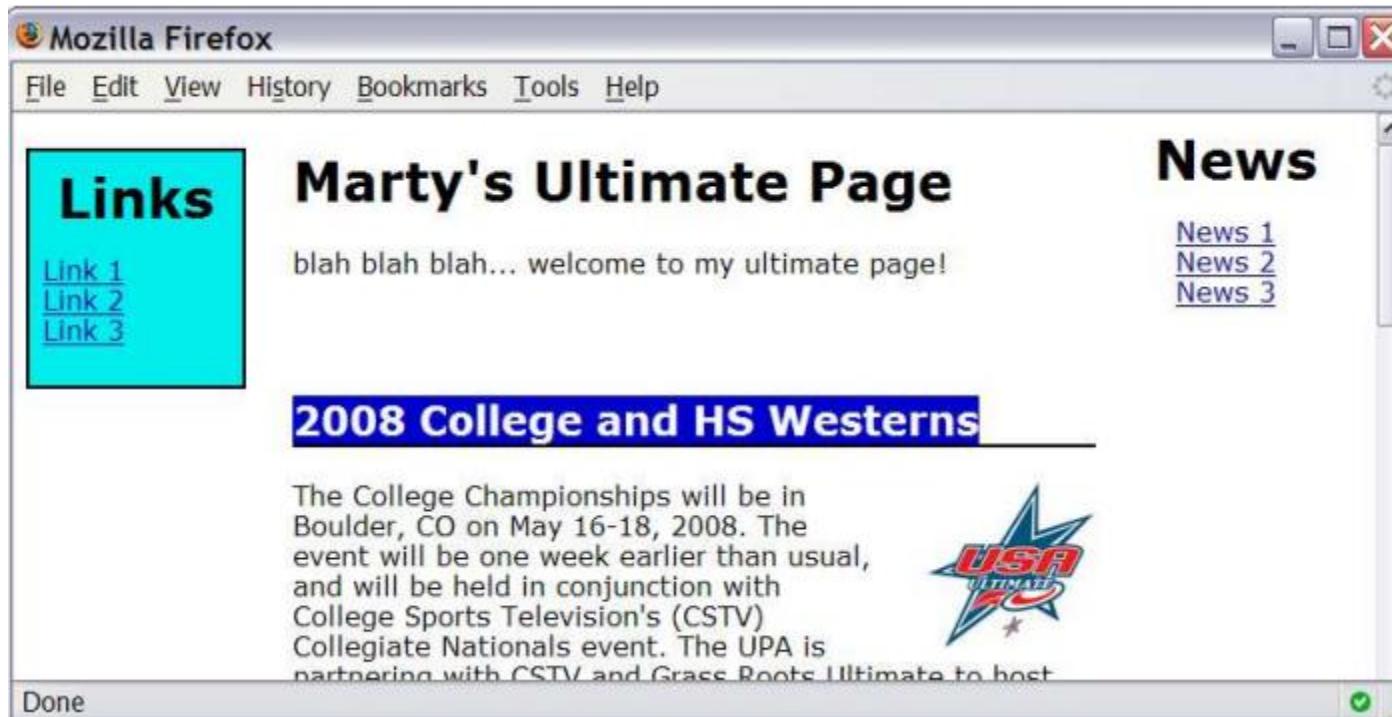
# add *content* to your website using CSS

---

- ▶ CSS has a property called **content**.
- ▶ can only be used with the pseudo elements **::after** and **::before**.
- ▶ The **::after** selector inserts something after the content of each selected element(s).
  - ▶ Use the **content** property to specify the content to insert.
  - ▶ Use the **::before** selector to insert something before the content.
- ▶ [https://www.w3schools.com/cssref/tryit.asp?filename=trycss\\_sel\\_after](https://www.w3schools.com/cssref/tryit.asp?filename=trycss_sel_after)

# Motivation for page sections

- ▶ Want to be able to **style individual elements, groups of elements, sections of text or of the page**
- ▶ Want to create complex page layouts



# Sections of a page: <div> vs <span>

---

- ▶ <div> is a block element
- ▶ <span> is an inline element
- ▶ They have no onscreen appearance, but you can apply styles to them
- ▶ They carry no significant semantic meaning

```
<div class="shout">  
    <h2>Hello</h2>  
    <p class="special">See our specials!</p>  
    <p>We'll beat <span class="shout">all  
        prices!</span></p>  
</div>
```

# Review: HTML5 tags for page sections

- ▶ Serve the same purpose as `div` – more semantic and descriptive than `div`s
  - ▶ Note in example, section can be section of an article or section of document containing articles

```
<div id="header">  
  
<div id="nav">  
  
<div class="article">  
  
<div class="section">  
  
<div id="sidebar">  
  
<div id="footer">
```



# CSS context selectors

---

`selector1 selector2 { properties }`

Applies the given properties to **selector2 only** if it is inside a selector1 on the page

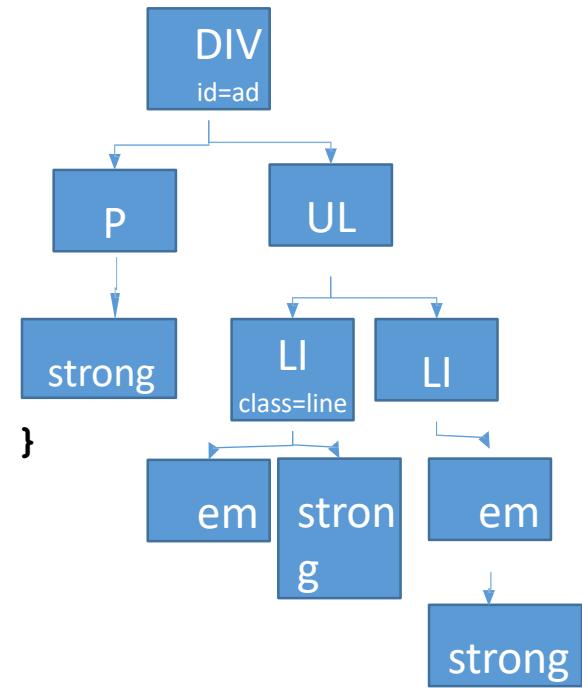
`selector1 > selector2 { properties }`

- ▶ Applies the given properties to **selector2 only** if it is direct child of selector1 (in the DOM)
- ▶ mnemonic: think of > as indicating a direct link (to a child)



# Example

```
<div id="ad">  
  <p>Shop at <strong>Hardwick's Hardware</strong></p>  
  <ul>  
    <li class="line"><em>The </em><strong>best</strong>  
      prices!</li>  
    <li><em><strong>Act while supplies last!</strong></em></li>  
  </ul>  
</div>  
  
ul > li { background-color: blue; }  
li strong { color: red; }  
li > strong { color: green; }  
#ad li.line strong { text-decoration: underline; }
```



See example: lesson3\_examples\contextselector.html,  
contextselectordirect.html

# Main Point

---

The <div> tag provides a generic block level element that can be used for any division or section of your page. The <span> tag provides a generic inline element for specifying any range of text inside a box. By using these tags, combined with CSS context selectors (direct child or descendant) we can write powerful and reusable CSS rules.

# W3C CSS Validator

---

- ▶ Check your CSS to make sure it meets the official CSS specifications
- ▶ More picky than the web browser, which may render malformed CSS correctly

```
<p>
  <a href="http://jigsaw.w3.org/css-validator/check/referer">
    <img src=http://jigsaw.w3.org/css-validator/images/vcss
         alt="Valid CSS!" />
  </a>
</p>
```

# CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

## *Changing Appearances*

1. How a page is displayed is affected by both the HTML and the CSS
  2. Although every HTML tag has a default way of displaying, it can easily be changed with CSS and should never be the basis for using it.  
Instead use HTML tags based on meaning.
- 

3. **Transcendental consciousness** is the field that underlies all differences.
4. **Impulses within the Transcendental field:** the diversity of the universe arises as an expression of the unified field.
5. **Wholeness moving within itself:** In Unity Consciousness, one experiences that this unbounded diversity is the Self.





# CS472 Web Programming

## Lecture 3: Layout

---

Except where otherwise noted, the contents of this document are Copyright 2012 Marty Stepp, Jessica Miller, Victoria Kirst and Roy McElmurry IV. All rights reserved. Any redistribution, reproduction, transmission, or storage of part or all of the contents in any form is prohibited without the author's expressed written permission. Slides have been modified for Maharishi University of Management Computer Science course CS472 in accordance with instructors agreement with authors.

# Maharishi University of Management -Fairfield, Iowa © 2016



All rights reserved. No part of this slide presentation may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage and retrieval system, without permission in writing from Maharishi University of Management.

# Wholeness Statement

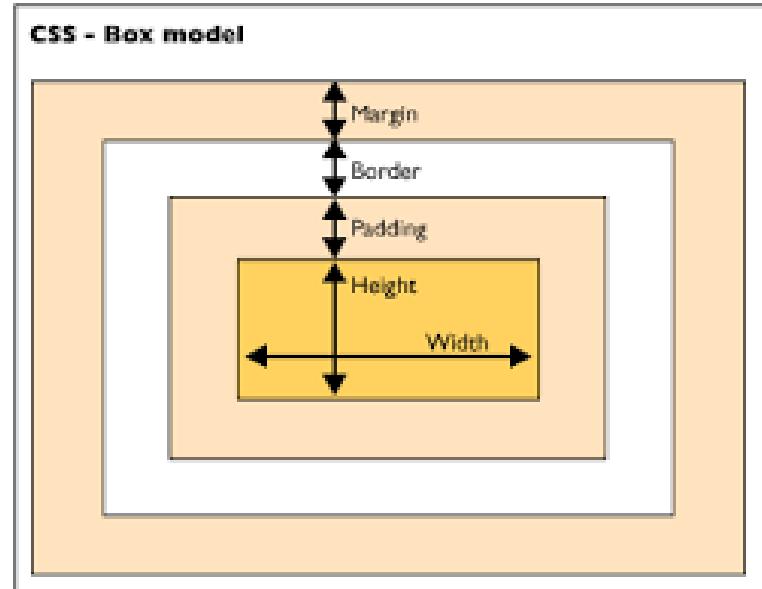
---

In this lecture we will discuss the different tools CSS provides for creating a layout. There are a variety of ways to position an element; most of them are based on taking a block level element and placing it in relation to some other block.

*Through familiarity with the language of nature, the individual gains complete support of nature.*

# The CSS Box Model

- ▶ For layout purposes, every element is composed of:
  - The actual element's **content**
  - A **border** around the element
  - **padding** between the content and the border (inside)
  - A **margin** between the border and other content (outside)
- ▶  $\text{width} = \text{content width} + \text{L/R padding} + \text{L/R border} + \text{L/R margin}$
- ▶  $\text{height} = \text{content height} + \text{T/B padding} + \text{T/B border} + \text{T/B margin}$
- ▶ The standard **width** and **height** properties refer ONLY to the content's width and height.



# CSS properties for borders



```
h2 { border: 5px solid red; }
```

This is the best WAP course!

| Property | Description                                   |
|----------|-----------------------------------------------|
| border   | thickness/style/size of border on all 4 sides |

- ▶ **thickness** (specified in px, pt, em, or thin, medium, thick )
- ▶ **style** (none, hidden, dotted , dashed , double , groove , inset , outset , ridge , solid )
- ▶ **color** (specified as seen previously for text and background colors)

# More border properties

---

| Property                                                                                                                                                                                                                                 | Description                                   |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
| border-color, border-width, border-style                                                                                                                                                                                                 | specific properties of border on all 4 sides  |
| border-bottom, border-left, border-right, border-top                                                                                                                                                                                     | all properties of border on a particular side |
| border-bottom-color, border-bottom-style, border-bottom-width, border-left-color, border-left-style, border-left-width, border-right-color, border-right-style, border-right-width, border-top-color, border-top-style, border-top-width | properties of border on a particular side     |
| <a href="#"><u>Complete list of border properties</u></a>                                                                                                                                                                                |                                               |

# Border example 2

---

```
h2 {  
    border-left: thick dotted #CC0088;  
    border-bottom-color: rgb(0, 128, 128);  
    border-bottom-style: double;  
}
```

This is the best WAP course!

- ▶ each side's border properties can be set individually
- ▶ if you omit some properties, they receive default values (e.g. border-bottom-width above)

# Dimensions

---

- ▶ For **Block elements and img element only**, set how wide or tall this element, or set the max/min size of this element in given dimension.

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

```
p { width: 350px; background-color: yellow; } h2 {  
width: 50%; background-color: aqua; }
```

This paragraph uses the first style above.

An h2 heading

Using **max-width** instead of **width** in this situation will improve the browser's handling of small windows. This is important when making a site usable on mobile. [https://www.w3schools.com/css/tryit.asp?filename=trycss\\_max-width](https://www.w3schools.com/css/tryit.asp?filename=trycss_max-width)

# box-sizing



- ▶ `box-sizing: content-box;` - initial and default value.  
The `width` and `height` properties are measured including only the content, but not the padding, border or margin.
- ▶ `box-sizing: border-box;` The `width` and `height` properties include the content, the padding and border, but not the margin. Note that padding and border will be inside of the box
- ▶ Q: Why is Hooray indented?
- ▶ [responsive design advantage demo](#)

```
.div3 {  
width: 300px;  
height: 100px;  
border: 1px solid blue;  
box-sizing: border-box;  
}  
.div4 {  
width: 300px;  
height: 100px;  
padding: 50px;  
border: 1px solid red;  
box-sizing: border-box;  
}
```

Both divs are the same size now!

Hooray!



## Rounded corners `border-radius`

```
p {  
  border: 3px solid blue;  
  border-radius: 12px;  
}
```



- ▶ Each side's border radius can be set individually, separated by spaces
  - ▶ **Four values:** top-left, top-right, bottom-right, bottom-left
  - ▶ **Three values:** top-left, top-right and bottom-left, bottom-right
  - ▶ **Two values:** top-left and bottom-right, top-right and bottom-left
  - ▶ **One value:** all four corners are rounded equally

# Padding

---

- ▶ The padding shorthand property sets all the padding properties in one declaration. Padding shares the background color of the element. This property can have from one to four values:

```
padding:10px 5px 15px 20px; /* Top, right, bottom, left */  
padding:10px 5px 15px; /* Top, right and left, bottom */  
padding:10px 5px; /* Top and bottom, right and left */  
padding:10px; /* All four paddings are 10px */
```

- ▶ **padding-bottom, padding-left, padding-right, padding-top**

```
h1 {      padding: 20px;  }  
  
h2 {  
      padding-left: 200px;  
      padding-top: 30px;  
}
```

# Margin

---

- ▶ Margins are always transparent. This property can have from one to four values:

```
margin:10px 5px 15px 20px; /* Top, right, bottom, left */  
margin:10px 5px 15px; /* Top, right and left, bottom */  
margin:10px 5px; /* Top and bottom, right and left */  
margin:10px; /* All four margins are 10px */
```

- ▶ margin-bottom, margin-left, margin-right, margin-top

```
h1 {margin: 20px; }  
h2 {  
    margin-left:      200px;  
    margin-top:       30px;  
}
```

- ▶ See example: [lesson3\\_examples/paddingmargin.html](#)
  - ▶ Inspect element in console

# Margin Collapse



- ▶ Vertical margins on different elements that touch each other (thus have no content, padding, or borders separating them) will collapse, forming a single margin that is equal to the greater of the adjoining margins.
  1. Collapsing Margins Between Adjacent Elements
  2. Collapsing Margins Between Parent and Child Elements

```
h1 { margin: 0 0 25px 0; }
```

```
p { margin: 20px 0 0 0; }
```

```
h1 { margin: 0 0 25px 0; }
```

```
p { margin: -20px 0 0 0; }
```

See example:

[lesson3\\_examples/margincollapse.html](#)

[lesson3\\_examples/margincollapse2.html](#)

[lesson3\\_examples/margincollapse3.html](#)



# Centering a block element: auto margins

```
<p> Lorem ipsum dolor sit amet, consectetur  
adipisicing elit, sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua.  
</p>
```

```
p {  
    border: 2px solid black;  
    margin-left: auto;  
    margin-right: auto;  
    width: 33%;  
}
```

- ▶ works best if width is set (otherwise, may occupy entire width of page)
- ▶ to center inline elements within a block element, use text-align: center;

# Details about **block** boxes

---

- ▶ By default **block** elements take the entire width space of the page unless we specify.
- ▶ To align a **block** element at the **center** of a horizontal space you must set a **width** first, and **margin: auto;**
- ▶ **text-align** does not align **block** elements within the page.

# Details about `inline` boxes

---

- ▶ size properties (width, height, min-width, etc.) are ignored for inline boxes
- ▶ margin-top and margin-bottom are ignored, but margin-left and margin-right are not
- ▶ each inline box's vertical-align property aligns it vertically within its block box
- ▶ **text-align** describes how inline content is aligned in its parent block element.
  - ▶ does not control the alignment of block elements, only their inline content
  - ▶ See `lesson3_examples/textalign.html`

# The `vertical-align` property

- ▶ Specifies where an **inline element** should be aligned vertically, with respect to other content on the same line within its block element's box
- ▶ Can be top, middle, bottom, baseline (default), sub, super, text-top, text-bottom, or a length value or % baseline means aligned with bottom of non-hanging letters

```
img {  
    vertical-align: baseline;  
}  
  
img {  
    vertical-align: middle;  
}
```

- ▶ See common error example:  
[lesson3\\_examples/verticalalign.html](#)
- ▶ image is vertically aligned to the baseline of the paragraph, which isn't the same as the bottom



# Main Point

---

The Box Model is a description of how every element has a basic width and height, outside of which it has padding, a border, and margin. For inline elements only the left and right margin and padding affect surrounding elements.

# The float property

## ► Property: float

- ▶ Values: left, right, none (default)
- ▶ `img { float: right; width: 130px; }`
- ▶ Removed from normal block flow of document. Inline content wraps around it as necessary. Block elements ignore it.
- ▶ If you want it to hover on that side of the page with other content wrapping around it, float it.
  - ▶ See example: [lesson3\\_examples/flotrightimage.html](#)

Boris Sadigev (born July 30, 1972) is a fictional Uzbekistan journalist played by British-Jewish comedian Sasha Von Neumann. He is the main character portrayed in the controversial and successful film Boris: Culinary Learnings of America for Make Money to Glorious Nation of Uzbekistan. Boris ...



# Common float bug: missing width

---

- ▶ often floating block elements must have a width property value
  - ▶ if no width is specified, the floating element may occupy 100% of the page width, so no content can wrap around it
  - ▶ See example: [lesson3\\_examples/floatingwithoutwidth.html](#)

# The clear property

```
img.hoveringicon { float: left; margin-right: 1em; }  
h2 { clear: left; background-color: yellow; }  
p { background-color: fuchsia; }
```



Starhome Sprinter is a Flash animated Internet cartoon. It mixes surreal humour with references to 1980s and 1990s pop culture, notably video games, classic television and popular music.

[My Starhome Sprinter Fan Site](#)

See example: `lesson3_examples/clear.html`

| Property | Meaning                                                                        | Values                           |
|----------|--------------------------------------------------------------------------------|----------------------------------|
| clear    | Whether to move this element below any prior floating elements in the document | left, right, both, none(default) |

# Common error: container too short

---

- ▶ If you place a tall floating element inside a block element without much other content, the floating element may hang down past the bottom edge of the block element that contains it.



Starhome Sprinter is a Flash animated Internet cartoon. It mixes surreal humour with references to 1980s and 1990s pop culture, notably video games, classic television and popular music.

- ▶ See example:  
[lesson3\\_examples/commonerrorcontenttooshort.html](lesson3_examples/commonerrorcontenttooshort.html)

# The overflow property

1. Place a final empty element at the bottom, and give suitable clear value.
  - ▶ See example: lesson3\_examples\contenttooshort-emptyelement.html
2. **overflow: hidden**
  - ▶ See example lesson3\_examples/overflow.html

```
.main { border: black 2px solid; overflow: hidden; }
```

| Property | Meaning                                                               | Values                                 |
|----------|-----------------------------------------------------------------------|----------------------------------------|
| overflow | Action to take if element's content is larger than the element itself | visible(default), hidden, scroll, auto |

# Multi-column layouts

---

- ▶ When more than one element floats in the same direction, they stack horizontally

```
div, p { border: 2px solid black; }  
.column { float: right; width: 25%; }
```

```
<div class="column"> Lorem ipsum dolor sit amet,  
consectetuer adipiscing elit. Integer pretium dui  
sit amet felis. </div>  
  
<div class="column"> Integer sit amet diam.  
Phasellus ultrices viverra velit. </div>  
  
<div class="column"> Beware the Jabberwock, my son!  
The jaws that bite, the claws that catch! </div>
```

See example: [lesson3\\_examples/multicolumn-float.html](#)

# Multi-column



```
#columns {  
    column-count: 3;  
    column-gap: 40px;  
    column-rule: 2px dotted gray; }
```

[lesson3\\_examples/multicolumn-count.html](#)

browser splits columns rather than split into smaller divs ourselves.

| Property       | Description                    | Values                   |
|----------------|--------------------------------|--------------------------|
| column-count   | Number of columns to use       | an integer               |
| column-fill    | How to choose columns' size    | balance(default), auto   |
| column-gap     | Space between columns          | a size (px, pt, %, em)   |
| column-rule    | Vertical line between columns  | a width, style and color |
| column-span    | Lets element span many columns | 1 (default) or all       |
| ▶ column-width | width of each column           | a size (px, pt, %, em)   |

# Main Point

---

- ▶ The CSS float property makes its element move to right or left side of the containing box. The clear property moves its element downwards if there is a floating element on the specified side. Float is the easiest way to make something appear on the right or left side and rest wrap around it.



# The position property

| Property                          | Meaning                     | Values                                                                                                                                                                                                                                 |
|-----------------------------------|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| position                          | Location of element on page | <b>static</b> : default position<br><b>relative</b> : offset from its normal static position<br><b>absolute</b> : at a particular offset within its containing element<br><b>fixed</b> : at a fixed location within the browser window |
| top,<br>bottom,<br>left,<br>right | Offsets of element's edges  | A size in px, pt, em or %                                                                                                                                                                                                              |

## **position: static;**

---

- ▶ **static** is the default position value for all elements.
- ▶ An element with **position: static;** is not positioned in any special way.
- ▶ A static element is said to be not positioned and an element with its position set to anything else is said to be positioned.

.static { **position:static;** }

```
<div class="static">
```

```
</div>
```



# position:relative;

- ▶ Set the location of an element to an offset from its normal static position.
- ▶ **relative** behaves the same as **static** unless you add some extra properties. 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. **relative** element stays in its place!

```
<p> This example has <span id="lifted">some text</span> with  
a relative position. </p>
```

```
#lifted {  
  position: relative;  
  left: 2em;  
  top: 1em;  
  border: 2px solid black;  
}
```

This example has some text with a relative position.

See example: lesson3\_examples/position-relative.html



# position: absolute;

- ▶ Elements that are positioned relatively are still considered to be in the normal flow of elements in the document.
- ▶ In contrast, an element that is positioned absolutely is taken out of the flow and thus takes up no space when placing other elements.
- ▶ The absolutely positioned element is positioned relative to *nearest positioned ancestor (non-static)*. If a positioned ancestor doesn't exist, the initial container is used.

```
#menubar{  
width: 100px;  
height: 50px;  
position: absolute;  
top: 20px;  
left: 50px;  
}
```

See example:

[lesson3\\_examples/position-absolute.html](#)

[lesson3\\_examples/position-absoluteblock.html](#)



# position: fixed;

- ▶ Fixed positioning is similar to absolute positioning, with the exception that the element's containing block is the viewport. This is often used to create a floating element that stays in the same position even after scrolling the page.

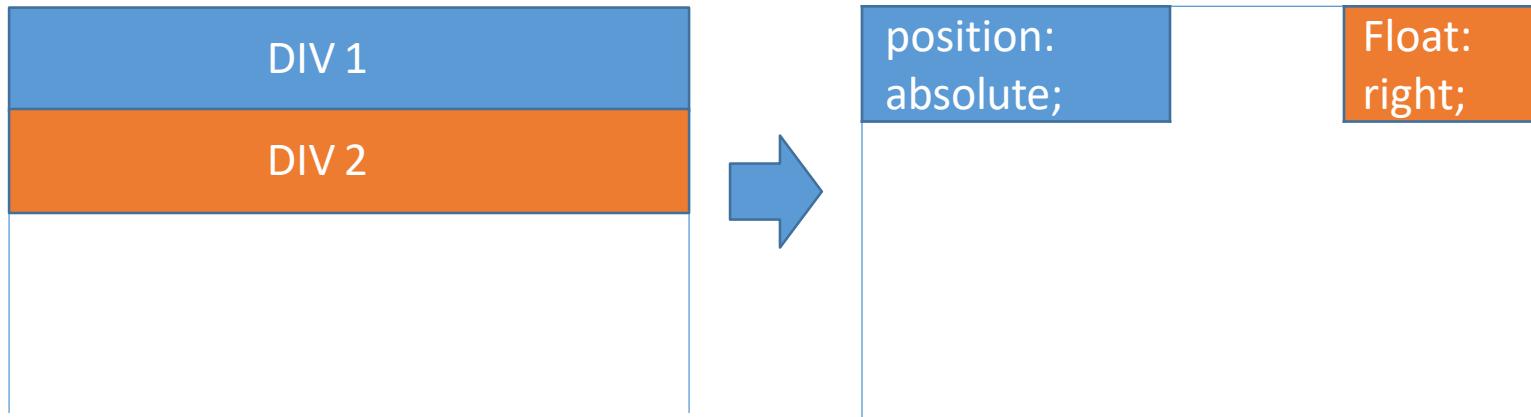
```
#one {  
  position: fixed;  
  top: 80px;  
  left: 10px;  
}
```

See example:  
[lesson3\\_examples/position-fixed.html](#)

- ▶ A fixed element does not leave a gap in the page where it would normally have been located.
- ▶ A fixed element loses its space in the flow
- ▶ A fixed element does not move when you scroll (stays in place)

# Block elements behavior with **position/float**

- ▶ One thing to remember, that once a block element is positioned as **fixed** or **absolute**, it will ONLY occupy the space of its content rather than taking the whole width space.
- ▶ Same thing applies for block elements with **float**.



# Alignment vs. float vs. position

---

1. if possible, lay out an element by aligning its content
  - ▶ horizontal alignment: text-align
  - ▶ set this on a block element; it aligns the content within it (not the block element itself)
  - ▶ E.g., see `lesson3_examples/textalign.html` example
  - ▶ vertical alignment: vertical-align
  - ▶ set this on an inline element, and it aligns it vertically within its containing element
2. if alignment won't work, try floating the element
3. if floating won't work, try positioning the element
  - ▶ absolute/fixed positioning are a last resort and should not be overused

# Main Point

---

Static positioning flows box elements from top to bottom, and inline elements from left to right. Relative positioning keeps the space in the original flow, but displays the element at an offset. Absolute positioning takes the element out of the flow and places it relative to the "containing element". Fixed positioning takes the element out of the flow and places it relative to the viewport.

(review )

## displayproperty: block vs inline

---

- ▶ The default value of display **property** for most elements is usually **block** or **inline**.
- ▶ **Block:** **div** is the standard block-level element. A block-level element starts on a new line and stretches out to the left and right as far as it can. Other common block-level elements are **p** and **form**, and new in HTML5 are **header**, **footer**, **section**, and more.
- ▶ **Inline:** **span** is the standard inline element. An inline element can wrap some text inside a paragraph **<span>** like this **</span>** without disrupting the flow of that paragraph. The **a** element is the most common inline element, since you use them for links.
- ▶ **None:** Another common display value is **none**. Some specialized elements such as **script** use this as their default. It is commonly used with JavaScript to hide and show elements without really deleting and recreating them.

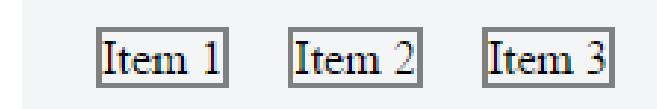


# Displaying **block** elements as **inline**

- ▶ Lists and other **block** elements can be displayed **inline**, flow left-to-right on same line.

*Note: Width will be determined by content (while block elements are 100% of page width)*

```
<ul id="topmenu">  
  <li>Item 1</li>  
  <li>Item 2</li>  
  <li>Item 3</li>  
</ul>
```



```
#topmenu li {  
  display: inline;  
  border: 2px solid gray;  
  margin-right: 1em;  
  list-style-type: none;  
}
```

# Boxes (Photo Gallery)

## The Hard Way (using float)

```
<div class="box">  
  I'm floating!  
</div>  
<div class="box">  
  I'm floating!  
</div>
```

```
<div class="after-box">  
  I'm using clear so I don't float next to the above boxes.  
</div>
```

```
.box {  
  float: left;  
  width: 200px;  
  height: 100px;  
  margin: 1em;  
}  
.after-box {  
  clear: left;  
}
```

## The Easy Way (using inline-block)

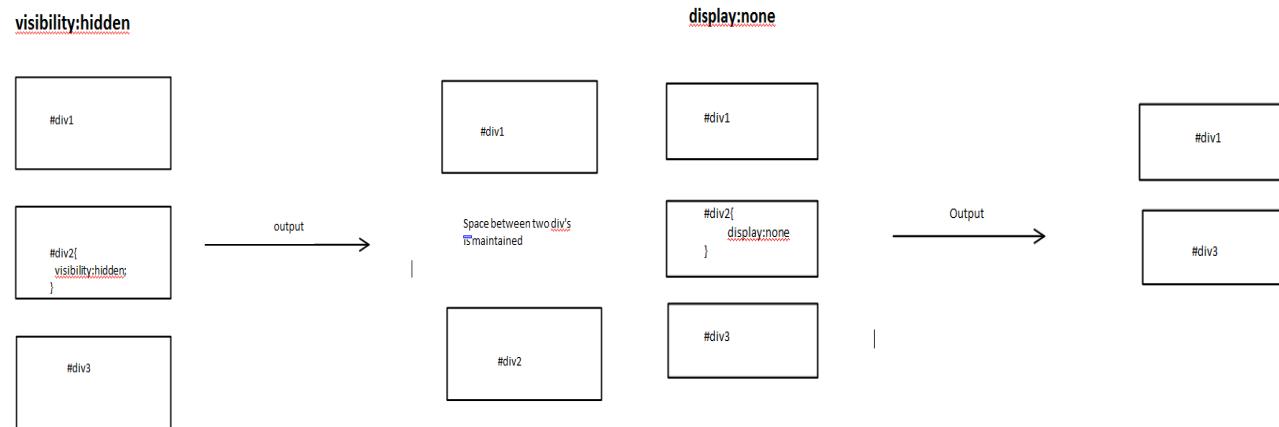
```
<div class="box2">  
  I'm an inline block!  
</div>  
<div class="box2">  
  I'm an inline block!  
</div>
```

```
<div>  
  I don't have to use clear in this case. Nice!  
</div>
```

```
.box2 {  
  display: inline-block;  
  width: 200px;  
  height: 100px;  
  margin: 1em;  
}
```

# Visibility vs Display

- ▶ Setting **display** to **none** will render the page as though the element does not exist.
- ▶ **visibility: hidden;** will hide the element, but the element will still take up the space it would if it was fully visible.



[display: none vs visibility: hidden](#)

# Opacity

---

- ▶ The **opacity** property sets the opacity level for an element.
- ▶ Value ranges from 1.0 (opaque) to 0.0 (transparent)

```
div {  
  opacity: 0.5;  
}
```



Lorem ipsum dolor sit amet, consectetur adipiscing elit.  
Phasellus imperdiet, nulla et dictum interdum, nisi lorem  
egestas odio, vitae scelerisque enim ligula venenatis dolor.

# Browser CSS Reset code

---

- ▶ It's a good practice to reset some body values before we start coding our own CSS rules:

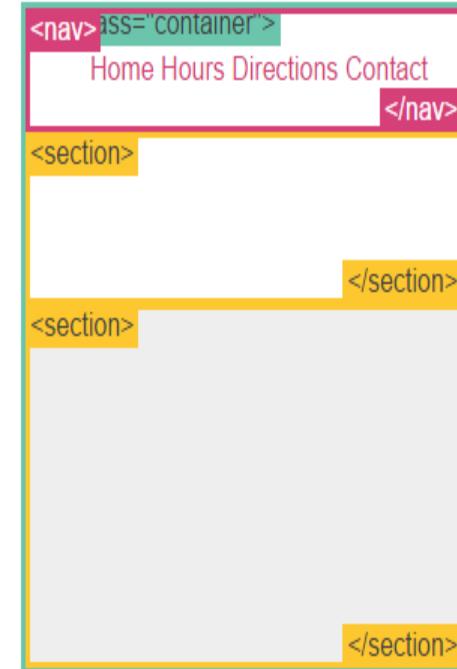
```
body {  
    margin: 0;  
    padding: 0;  
    font-size: 100%;  
    line-height: 1;  
}
```

# mediaqueries



- ▶ **Responsive Design** is the strategy of making a site that responds to the browser and device width.

```
@media screen and (min-width:600px) {  
  nav { float: left; width: 25%; }  
  section { margin-left: 25%; }  
}  
  
@media screen and (max-width:599px) {  
  nav li { display: inline; }  
}
```



- ▶ See `lesson3_examples\mediaquery.html` and `lesson3_examples\mediaqueryBootstrap.html`
- ▶ 42

# Extra Credit – meta viewport

---

- ▶ You can make your layout look even better on mobile using meta viewport.
- ▶ Without a viewport, mobile devices will render the page at a typical desktop screen width, scaled to fit the screen.
- ▶ Pages optimized to display well on mobile devices should include a meta viewport in the head of the document specifying width=device-width, initial-scale=1.

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

# Extra Credit – Viewport (cont)

---

- ▶ Users scroll websites vertically not horizontally!
- ▶ if forced to scroll horizontally, or zoom out it results in a poor user experience.
- ▶ Some additional rules to follow:
  - ▶ 1. Do NOT use large fixed width elements
  - ▶ 2. Do NOT let the content rely on a particular viewport width to render well
  - ▶ 3. Use CSS media queries to apply different styling for small and large screens
  - ▶ [https://www.w3schools.com/css/css\\_rwd\\_viewport.asp](https://www.w3schools.com/css/css_rwd_viewport.asp)
  - ▶ [https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport\\_meta\\_tag](https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport_meta_tag)

# CSS Frameworks

---

Because CSS layout is so tricky, there are CSS frameworks out there to help make it easier. Here are a few if you want to check them out. Using a framework is only a good idea if the framework really does what you need your site to do. They're no replacement for knowing how CSS works.

- [Blueprint](#)
- [Bootstrap](#)
- [Foundation](#)
- [SemanticUI](#)



# CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLESSESS OF KNOWLEDGE

## CSS Positioning: Whole Is Greater than the Sum of the Parts

1. You can use floats and positioning to change where elements are displayed.
  2. The entire visual appearance of a page can be completely altered using different style sheets.
- 
3. **Transcendental consciousness** is the field of unity.
  4. **Impulses within the Transcendental field:** The self-interacting dynamics of the unified field create diversity from unity.
  5. **Wholeness moving within itself:** In Unity Consciousness, one experiences this self interacting dynamics as an aspect of one's personal self.





# CS472 Web Programming

## HTML Forms: Connecting with the Source

---

Except where otherwise noted, the contents of this document are Copyright 2012 Marty Stepp, Jessica Miller, Victoria Kirst and Roy McElmurry IV. All rights reserved. Any redistribution, reproduction, transmission, or storage of part or all of the contents in any form is prohibited without the author's expressed written permission. Slides have been modified for Maharishi University of Management Computer Science course CS472 in accordance with instructors agreement with authors.

# Maharishi University of Management -Fairfield, Iowa © 2016



All rights reserved. No part of this slide presentation may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage and retrieval system, without permission in writing from Maharishi University of Management.

# Wholeness Statement

---

- ▶ In this lecture we will discuss how to generate and process user input. On the client side, we will create HTML forms with different types of widgets that allow the users to submit different types of data, and on the server side we will look at processing these different types of data. *The first expression of the Unified Field is Rishi (knower), Devata (process of knowing), Chandas (known).*

# Query strings and parameters

---

- ▶ **Query string:** a set of parameters passed from a browser to a web server. Often passed by placing name/value pairs at the end of a URL.
- ▶ Below, parameter **username** has value “tina”, and **sid** has value “123456”

<http://mum.edu/login.jsp?username=tina&sid=123456>

<http://www.google.com/search?q=Obama>

<https://www.google.com/?q=HTML+Form>

# Request Command

---

- ▶ There are 3 ways to send a request from a browser tab to the server:
  - ▶ Type url (GET)
  - ▶ Form (GET, POST.. etc)
  - ▶ XHR Request (GET, POST.. etc)
- ▶ Request methods:
  - ▶ GET: only has header (parameter are sent in the header, NO body)
  - ▶ POST: has header and body ([parameters are sent in the body])

# HTTP GET vs. POST requests

---

- ▶ **GET** : asks a server for a page or data
  - ▶ if the request has parameters, they are sent in the URL as a query string (request header)
  - ▶ Some older browsers might limit length of URL
  - ▶ URLs cannot contain special characters without encoding
  - ▶ private data in a URL can be seen or modified by users
- ▶ **POST** : submits data to a web server (to be saved in DB or file or updates state in server application)
  - ▶ parameters are embedded in the HTTP request body, not the URL

# HTML forms

- ▶ Form: a group of UI controls that accepts information from the user and sends the information to a web server
- ▶ The information is sent to the server as a query string
- ▶ JavaScript can be used to create interactive controls (seen later)

The image shows a screenshot of an HTML form. At the top is a small, empty rectangular input field. Below it is a larger text area with a grey border and a placeholder text "Add Comments Here". To the right of the text area is a vertical scroll bar. Below the text area is a horizontal row of five radio buttons, each followed by a value: "Value 1", "Value 2", "Value 3", "Value 4", and "Value 5". The first radio button is checked. Below this row is another horizontal row of five checkboxes, also followed by their respective values: "Value 1", "Value 2", "Value 3", "Value 4", and "Value 5". The second checkbox is checked. At the bottom of the form are two buttons: a blue "Submit" button and a grey "Reset" button.

Value 1 Value 2 Value 3 Value 4  
Value 1 Value 2 Value 3 Value 4 Value 5

Submit Reset

# HTML form: <form>

---

The <form> tag is used to create an HTML form for user input.

The <form> element can contain one or more of the following form elements:

<input>, <textarea>, <button>, <select>, <option>, <optgroup>,  
<fieldset>, <label>, <textarea>, <datalist>, <output>

```
<form action="sales.html" method="get" novalidate  
autocomplete="on">  
    Form controls  
</form>
```

## Form Attributes

- ▶ **action** destination URL
- ▶ **method** get, post
- ▶ **novalidate** (HTML5) specifies that the form should not be validated when submitted
- ▶ **autocomplete** (HTML5) on, off

# Form Example

---

```
<form action="http://www.google.com/search">  
    <div> Let's search Google <input name="q" />  
    <input type="submit" /> </div>  
</form>
```



See example: lecture4\_examples/form.html

# Main Points

---

- ▶ An HTML form allows the user to send data (input parameters) to the server. Forms are created with the `<form>` tag, and can be submitted with either an HTTP GET or POST method.



# Form controls: <input>

```
<input type="text" name="q" value="Colbert Report" />  
<input type="submit" value="Booyah!" />
```

- ▶ input **element** is used to create many UI controls
  - ▶ an inline element that MUST be self-closed
- ▶ name **attribute** specifies name/key of query parameter to pass to server
- ▶ type **can be** button, checkbox, file, hidden, password, radio, reset, submit, text, ...
- ▶ value **attribute** specifies control's initial text



# Text fields: <input>

```
<input type="text" name="username" size="10" maxlength="8" />  
<input type="text" name="password" size="8" />
```

| Attribute    | Value(s)                                                                                          | Description                                                     |
|--------------|---------------------------------------------------------------------------------------------------|-----------------------------------------------------------------|
| value        | text                                                                                              | Initial text to appear in text box                              |
| size         | integer                                                                                           | Visible length of text box, in characters                       |
| maxlength    | integer                                                                                           | Maximum number of chars that may be typed into text box         |
| autocomplete | on, off<br>      | Whether to offer suggestions of text to auto-complete the field |
| autofocus    |  autofocus       | Makes control initially receive keyboard focus                  |
| novalidate   |  novalidate    | Indicates browser should not check value before submitting      |
| placeholder  |  text          | A hint or example of what the user should type;                 |
| pattern      |  regular expr. | A regular expression indicating what input is valid             |
| required     |  required      | Whether browser should display an error if blank                |



# Checkboxes

- ▶ yes/no choices that can be checked and unchecked (**inline**)
- ▶ none, 1, or many checkboxes can be checked at same time
- ▶ Use the **checked** attribute in HTML to initially check the box

```
<input type="checkbox" name="lettuce" /> Lettuce  
<input type="checkbox" name="tomato" checked /> Tomato  
<input type="checkbox" name="pickles" checked /> Pickles
```

Lettuce  Tomato  Pickles



# Radio buttons

- ▶ Sets of mutually exclusive choices (**inline**)
  - ▶ Grouped by **name** attribute (only one can be checked at a time)
  - ▶ Must specify a **value** for each one or else it will be sent as value **on**

```
<input type="radio" name="cc" value="visa" checked /> Visa  
<input type="radio" name="cc" value="mastercard" /> MasterCard  
<input type="radio" name="cc" value="amex" /> American Express
```

Visa  MasterCard  American Express

# <textarea>

---

- ▶ The <textarea> tag defines a multi-line text input control.  
(inline)
- ▶ A textarea can hold an unlimited number of characters, and the text renders in a fixed-width font (usually Courier).
- ▶ The size of a textarea can be specified by the **cols** and **rows** attributes, or even better; through CSS' **height** and **width** properties.

```
<textarea rows="4" cols="20">
```

**Type your comments here .**

```
</textarea>
```

Type your comments  
here

See example: lecture4\_examples/textarea.html

# Text labels: <label>



- ▶ Associates nearby text with control, so you can **click text to activate control**
- ▶ Can be used with **checkboxes** or **radio** buttons
- ▶ **label** element can be targeted by CSS style rules

```
<label> <input type="radio" name="cc" value="visa"  
checked="checked" /> Visa</label>  
  
<label> <input type="radio" name="cc" value="mastercard" />  
MasterCard</label>  
  
<label> <input type="radio" name="cc" value="amex" /> American  
Express</label>
```

Visa  MasterCard  American Express

- ▶ See example: lecture4\_examples/label1.html, lecture4\_examples/label2.html



# Drop-down list <select> and <option>

- ▶ Menus of choices that collapse and expand (inline)
  - ▶ **option** element represents each choice
  - ▶ **select** optional attributes: **disabled**, **multiple**, **size**
  - ▶ optional **selected** attribute sets which one is initially chosen

```
<select name="favoritecharacter">
  <option>Jerry</option>
  <option>George</option>
  <option selected>Kramer</option>
  <option>Elaine</option>
</select>
```

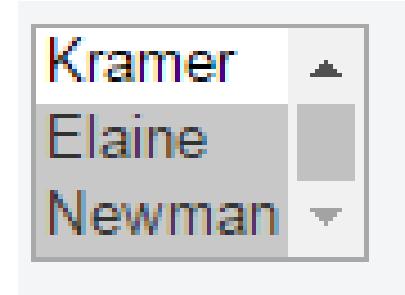




# Using <select> for lists

- ▶ optional **multiple** attribute allows selecting multiple items with shift- or ctrl- click
  - ▶ must declare parameter's name with [] if you allow multiple selections
- ▶ **option** tags can be set to be initially **selected**

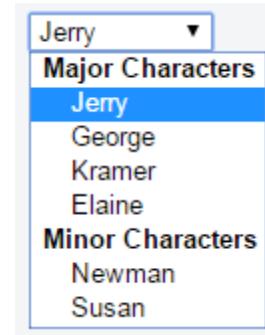
```
<select name="favoritecharacter[]" size="3" multiple>  
  <option>Jerry</option>  
  <option>George</option>  
  <option>Kramer</option>  
  <option>Elaine</option>  
  <option selected>Newman</option>  
</select>
```



# Option groups: <optgroup>



```
<select name="favoritecharacter">  
  <optgroup label="Major Characters">  
    <option>Jerry</option>  
    <option>George</option>  
    <option>Kramer</option>  
    <option>Elaine</option>  
  </optgroup>  
  <optgroup label="Minor Characters">  
    <option>Newman</option>  
    <option>Susan</option>  
  </optgroup>  
</select>
```



# Reset and Submit buttons

---

- ▶ When we click **reset** button, it returns all form controls to their initial values
- ▶ When we click **submit** buttons, it sends all data with the specified **method** (Get/Post) to the **action** page in the form
- ▶ Specify custom text on the button by setting its **value** attribute

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



# Hidden input parameters



- ▶ An invisible parameter that is still passed to the server when form is submitted, it's useful for passing on additional state that isn't modified by the user

```
<input type="text" name="username" /> Name  
<br />  
<input type="text" name="sid" /> SID  
<br />  
<input type="hidden" name="school" value="MUM" />  
<input type="hidden" name="year" value="2048" />
```

|  |      |
|--|------|
|  | Name |
|  | SID  |



# Grouping <fieldset>, <legend>

- Groups of input fields with optional caption (legend)

```
<fieldset>
  <legend>Credit cards:</legend>
  <input type="radio" name="cc" value="visa" checked="checked" />
Visa
  <input type="radio" name="cc" value="mastercard" /> MasterCard
  <input type="radio" name="cc" value="amex" /> American Express
</fieldset>
```

Credit cards:

Visa  MasterCard  American Express

# New Form Controls in HTML5



| Input type | Description                                          |
|------------|------------------------------------------------------|
| color      | A color from a palette of available choices          |
| range      | A slider for selecting values in a given range       |
| date       | A date such as August 29, 2016                       |
| time       | A time of day such as 11:15 PM                       |
| datetime   | A date and time such as 11:15 PM, August 29, 2016    |
| month      | A month of a particular year, such as August, 2016   |
| week       | A week of a particular year, such as August 35, 2016 |



# Styling forms – attribute selector

- ▶ Because most input element are created using input tag, we target each group of elements using this CSS selector:

```
element[attribute="value"] {  
    property: value;  
    property: value;  
    ... property: value;  
}  
  
input[type="text"] {  
    background-color: yellow;  
    font-weight: bold;  
}
```

CS472

# Main Point

---

- ▶ HTML provides many different types of input widgets, including text fields, text areas, check boxes, radio buttons, and dropdown lists, this is also an area HTML 5 is expanding.

# [Bonus] pattern



The **pattern** attribute specifies a regular expression that the **<input>** element's value is checked against. The **pattern** uses the ECMAScript (i.e. JavaScript) flavor of regex.

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

Tip: Use the global **title** attribute to describe the pattern to help the user.

```
<form action="demo_form.jsp">  
    Country code:  
    <input type="text" name="country_code" pattern="[A-Za-z]{3}" title="Three letter country code ">  
    <input type="submit">  
</form>
```

A screenshot of a web browser showing a form. The form has a text input field labeled "Country code:" containing the value "123". To the right of the input is a "Submit" button. A tooltip box is overlaid on the page, containing an exclamation mark icon and the text "Please match the requested format." and "Three letter country code".

# Regular expressions

---

`^ [a-zA-Z_-]+@[([a-zA-Z_-]+)\.]+[a-zA-Z]{2,4}$`

- ▶ Regular expression ("regex"): a description of a pattern of text
- ▶ Can test whether a string matches the expression's pattern
- ▶ Regular expressions are extremely powerful but tough to read
  - ▶ (the above regular expression matches email addresses)
- ▶ Regular expressions are used in all languages:
  - ▶ Java, PHP ,JavaScript, HTML, C#, and other languages
- ▶ Many IDEs allow regexes in search/replace

# Basic regular expressions

---

The simplest regexes simply matches any string that contains that text.

**abc**

above regular expression matches any string containing "abc":

- ▶ YES: "abc", "abcdef", "defabc", ".=.abc.=.", ...
- ▶ NO: " ABC" , " fedcba", "ab c", "PHP", ...
  
- ▶ Note that html5 has implicit anchors ^ and \$, so abc is really ^abc\$
- ▶ Regular expressions are case-sensitive by default.

# Wildcards

---

A dot **.** matches exactly **one-character** except a **\n** line break

.**oo.y**      matches "Doocy", "goofy", "LooNy", ...

# Special characters: |, (), \

---

| means OR

**abc|def|g** matches "abc", "def", or "g"

( ) are for grouping

**(Homer|Marge) Simpson**

matches "Homer Simpson" or "Marge Simpson"

\ starts an escape sequence

many characters must be escaped to match them

literally: / \ \$ . [ ] ( ) ^ \* + ?

**<br\/>** matches lines containing <br /> tags

# Quantifiers: \*, +, ?

---

## \* means 0 or more occurrences

**abc\*** matches "ab", "abc", "abcc", "abccc", ...

**a(bc)\*** matches "a", "abc", "abc<sub>1</sub>c", "abc<sub>2</sub>c<sub>2</sub>", ...

**a.\*a** matches "aa", "aba", "a8qa", "a!xyz  9a", ...

## + means 1 or more occurrences

**a(bc)+** matches "abc", "abc<sub>1</sub>c", "abc<sub>2</sub>c<sub>2</sub>", ...

**Goo+gle** matches "Google", "Goooogle", "Goooogle", ...

...

## ? means 0 or 1 occurrences

**a(bc)?** matches "a" or "abc"

## More quantifiers: {min,max}

---

{min,max} means between min and max occurrences (inclusive)

a(bc){2,4} matches "abc", "abcbc", or  
"abcbcbc"

min or max may be omitted to specify any number

{2,} means 2 or more

{,6} means up to 6

{3} means exactly 3

## Anchors: ^ and \$

---

^ represents the beginning of the string or line;

\$ represents the end

**Jess** matches all strings that contain Jess;

**^Jess** matches all strings that start with Jess;

**Jess\$** matches all strings that end with Jess;

**^Jess\$** matches the exact string "Jess" only

**^Mart.\*Stepp\$** matches "MartStepp", "Marty Stepp", "Martin D Stepp", ... but NOT "Marty Stepp stinks" or "I H8 Martin Stepp"

The html5 spec states that ^ and \$ are implicit

# Character sets: []

---

[ ] group characters into a character set, will match any **single character** from the set

**[bcd] art** matches strings containing "bart",  
"cart", and "dart"

equivalent to **(b|c|d) art** but shorter

inside [], many of the modifier keys act as normal characters

**what[!\*?]\*** matches "what", "what!", "what?\*\*!",  
"what??!", ...

What regular expression matches DNA (strings of A, C, G, or T)?

**[ACGT]+**

# Character ranges: [start-end]

---

inside a character set, specify a range of characters with -

[**a-z**] matches any lowercase letter

[**a-zA-Z0-9**] matches any lower- or uppercase letter or digit

an initial ^ inside a character set negates it

[^**abcd**] matches any character other than a, b, c, or d

inside a character set, - must be escaped to be matched

[+\\-]?[0-9]+ matches an optional + or -, followed by at least one digit

What regular expression matches letter grades such as A, B+, or D- ?

[**ABCDF**] [+\\-]?

# Escape sequences

---

Special escape sequence character sets:

**\d** matches any digit (same as [0-9])

**\D** any non-digit ([^0-9])

**\w** matches any word character (same as [a-zA-Z\_0-9])

**\W** any non-word char

**\s** matches any whitespace character ( , \t, \n, etc.)

**\S** any non-whitespace

What regular expression matches dollar amounts of at least \$100.00 ?

**\\$[1-9]\d{2},\.\d{2}**

# Example - URL



- An <input> element with type="url" that must start with http:// or https:// followed by at least one character:

```
<form action="demo_form.jsp">
```

**Homepage:**

```
  <input type="url" name="website"  
pattern="https?://.+" title="Please enter a URL">  
  <input type="submit">  
</form>
```

A screenshot of a web browser showing a form submission. The form has a label "Homepage:" followed by a text input field containing "dffd". To the right of the input is a "Submit" button. Below the input field is a validation message: a yellow square with a red exclamation mark followed by the text "Please enter a URL." This indicates that the input did not meet the URL pattern requirement defined in the HTML code.

# CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLESSESS OF KNOWLEDGE

## HTML Forms: Connecting with the Source

1. Forms let us submit data to the web server, which can then generate a custom response based on server side information.
  2. GET requests are intended to only retrieve information and should be idempotent. POST requests are intended to submit data and not request a direct response.
- 

**3. Transcendental consciousness** is the experience of the source of thought.

**4. Impulses within the Transcendental field:** duality is created as the boundless interacts with itself, creating the impression of this and that.

**5. Wholeness moving within itself:** In Unity Consciousness, one becomes aware that the Self is nothing but the boundless unity.



# Advanced: Lookaround

---

## Positive lookahead $(?=A)B$

Once a group starts with  $?=$  it means positive lookahead. Find expression A first, if found then expression B follows.

## Negative lookahead $(?!A)B$

Once a group starts with  $?!$  it means negative lookahead. First check if expression A is not found, then check if expression B follows.

# Example - email



- An <input> element with type="email" that must be in the following order: characters@characters.domain (characters followed by an @ sign, followed by more characters, and then a ".")

```
<form action="demo_form.jsp">  
E-mail:  
  <input type="email" name="email" pattern="[a-zA-Z0-9._+\\-]+@[a-zA-Z0-9.\\-]+\\.[a-zA-Z]{2,3}">  
  <input type="submit">  
</form>
```

A screenshot of a web browser showing a form submission. The form has a single input field labeled "E-mail:" containing the value "fdfd". To the right of the input field is a "Submit" button. Below the form, a modal dialog box is displayed with an exclamation mark icon and the text: "Please include an '@' in the email address. 'fdfd' is missing an '@'."

# Examples - password



- An <input> element with type="password" that must contain 8 or more characters that are of at least one number, and one uppercase and lowercase letter:

```
<form action="demo_form.jsp">
```

**Password:**

```
  <input type="password" name="pw" pattern="(?=\d+)(?=.*[a-z]*)(?=.*[A-Z]*).{8,}" title="Must contain at least one number and one uppercase and lowercase letter, and at least 8 or more characters">
```

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

Password: .....

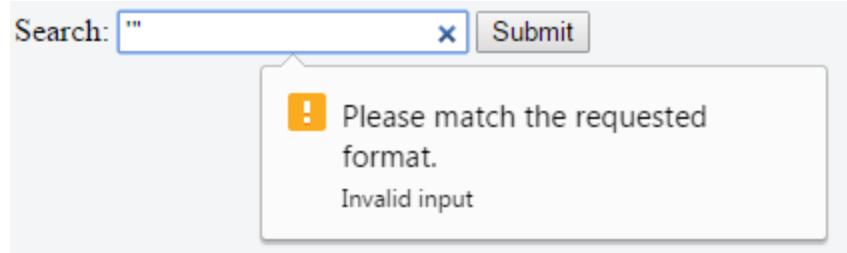
! Please match the requested format.  
Must contain at least one number and one uppercase and lowercase letter, and at least 8 or more characters

# Example - Search



- An <input> element with type="search" that CANNOT contain the following characters: ' or “

```
<form action="demo_form.jsp">  
  Search: <!-- x27 is a single quote and \x22 is a  
  double quote -->  
  <input type="search" name="search"  
        pattern="[^\\x27\\x22]+" title="Invalid input">  
  <input type="submit">  
</form>
```



A screenshot of a web browser showing a search form. The form has a text input field labeled "Search:" containing an empty string. To the right of the input is a "Submit" button. Below the form, a modal dialog box is displayed. The dialog contains an orange exclamation mark icon followed by the text "Please match the requested format." and "Invalid input".



# Lecture 5: JavaScript for Modern Web Apps



---

Except where otherwise noted, the contents of this document are Copyright 2012 Marty Stepp, Jessica Miller, Victoria Kirst and Roy McElmurry IV. All rights reserved. Any redistribution, reproduction, transmission, or storage of part or all of the contents in any form is prohibited without the author's expressed written permission. Slides have been modified for Maharishi University of Management Computer Science course CS472 in accordance with instructors agreement with authors.

# Maharishi University of Management -Fairfield, Iowa © 2016



All rights reserved. No part of this slide presentation may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage and retrieval system, without permission in writing from Maharishi University of Management.

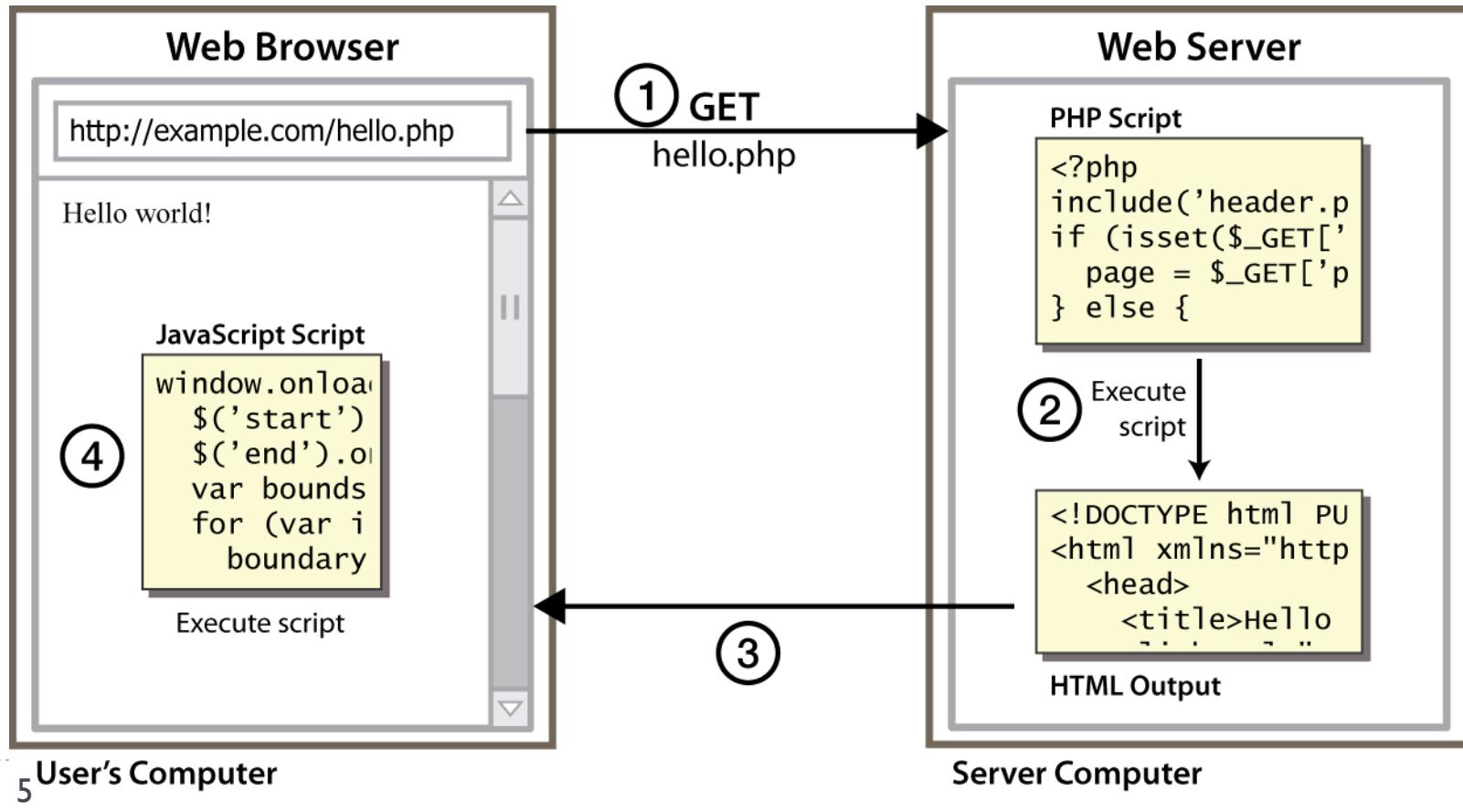
# Key JavaScript Concepts

---

- ▶ Key JavaScript Concepts
- ▶ JavaScript Syntax
- ▶ Program Logic
- ▶ Advanced JavaScript Syntax

# Client-side Scripting

- ▶ client-side script: code runs in browser *after* page is sent back from server
  - ▶ often this code manipulates the page or responds to user actions



# Why use client-side programming?

---

- ▶ **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?

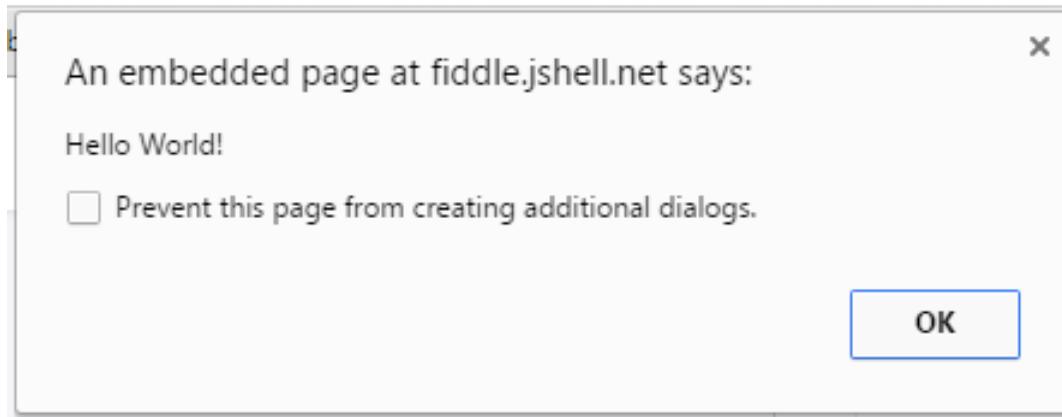
---

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



# A JavaScript statement: alert

```
alert("Hello World!");
```



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

# Variables and types

---

```
var name = expression;
```

```
var age = 32;
```

```
var weight = 127.4;
```

```
var clientName = "Connie Client";
```

- ▶ variables are declared with the var keyword (case sensitive)
  - ▶ Replaced by let and const with JS6
- ▶ types are not specified, but JS does have types ("loosely typed")
  - ▶ Number, Boolean, String, Object (Array , Function) , Null, Undefined
  - ▶ can find out a variable's type by calling typeof



# Block-scoped variable (ES6)

```
let callbacks = []
for (let i = 0; i <= 2; i++) {
  callbacks[i] = function() {
    return i * 2
  }
}
Alert(i);
callbacks[0]() === 0
callbacks[1]() === 2
callbacks[2]() === 4
```

# Constants (ES6)

---

- ▶ Support for constants (also known as "immutable variables"), i.e., variables which cannot be re-assigned new content.
  - ▶ Like `var`, but cannot be reassigned
- ▶ Notice: this only makes the variable itself immutable, not its assigned content (for instance, in case the content is an object, this means the object itself can still be altered).

```
const PI = 3.1415926;
```

```
PI = 10; //error
```

```
const point = {x:1, y: 10};
```

```
point.y = 20; //ok
```

# Number Type

---

```
let enrollment = 99;
```

```
let medianGrade = 2.8;
```

```
let credits = 5 + 4 + (2 * 3);
```

- ▶ 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
  - ▶ What is "2" + 3 ?

# Logical Operators

---

- ▶ `>`, `<`, `>=`, `<=`, `&&`, `||`, `! ==`, `!=`, `====`, `!==`
- ▶ 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
- ▶ Always use *strict equality*

# Comments (same as Java)

---

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

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



# String Type

```
const s = "Connie Client";
let fName = s.substring(0, s.indexOf(" ")); // "Connie"
let len = s.length; // 13
let s2 = 'Melvin Merchant'; // can use "" or ''
```

- ▶ methods: charAt, charCodeAt, fromCharCode, indexOf, lastIndexOf, replace, split, substring, toLowerCase, toUpperCase
  - ▶ charAt returns a one-letter String (there is no char type)
  - ▶ length property (not a method as in Java)
  - ▶ concatenation with + : | + | is 2, but "I" + | is "I |"

# More about String

---

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

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

- ▶ to access characters of a String, use [index] or charAt:

```
const firstLetter = s[0];  
let firstLetter = s.charAt(0);  
let lastLetter = s.charAt(s.length - 1);
```



# Boolean Type

```
const iLikeWebApps = true;  
const ieIsGood = "IE6" > 0; // false  
if ("web dev is great") { /* true */ }  
if (0) { /* false */ }
```

- ▶ any value can be used as a Boolean
  - ▶ "falsey" values: **false**, **0**, **0.0**, **NaN**, empty String(""), **null**, and **undefined**
  - ▶ "truthy" values: anything else, include objects
- ▶ **!!** Idiom – gives boolean value of any variable
  - ▶ const x=5;
  - ▶ console.log(!x);
  - ▶ console.log(x);
  - ▶ console.log (!!x);



# Special values: null and undefined

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

- ▶ **undefined** : has been declared, but no value assigned
  - ▶ e.g., caroline variable above
  - ▶ vars that are "hoisted" to beginning of a function
- ▶ **null** : exists, and was specifically assigned an value of null
- ▶ reference error when try to evaluate a variable that has not been declared
  - ▶ reference error different from undefined
  - ▶ undefined means declared, but no value assigned



# if/else statement (same as Java)

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

- ▶ identical structure to Java's if/else statement
- ▶ JavaScript allows almost anything as a *condition*
  - ▶ *JS idioms*
  - ▶ //initialize a variable if not set yet
    - ▶ if (!a) { a = 10; }
  - ▶ //only use a variable if it has a value
    - ▶ if (b) { console.log(b); }



# for loop (same as Java)

```
for (initialization; condition; update) {  
    statements;  
}
```

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

```
const s1 = "hello";  
let s2 = "";  
for (let i = 0; i < s1.length; i++) {  
    s2 += s1[i] + s1[i];  
} // s2 stores "hheelllloo"
```

# while loops (same as Java)

---

```
while (condition) {
```

```
    statements;
```

```
}
```

```
do {
```

```
    statements;
```

```
} while (condition);
```

- ▶ break and continue keywords also behave as in Java



# Arrays

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

const ducks = ["Huey", "Dewey", "Louie"];
let 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
```

- ▶ two ways to initialize an array – see example for another two ways
- ▶ length property (grows as needed when elements are added)



# Array methods

```
let 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
```

- ▶ 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
  - ▶ unshift and shift add/remove from front
  - ▶ shift and pop return the element that is removed



# Array methods: map, filter, reduce

```
//functional programming: map, filter, reduce can replace many loops

const a = [1,3,5,3,3];

//translate/map all elements in an array to another set of values
const b = a.map(function(elem, i, array) {
  return elem + 3;
})
console.log(b); // [4,6,8,6,6]

//remove unwanted elements based on a condition
const c = a.filter(function(elem, i, array) {
  return elem !== 3;
})
console.log(c); // [1,5]

//find a cumulative or concatenated value based on elements across the array
const d = a.reduce(function(prevVal, elem, i, array) {
  return prevVal + elem;
})
console.log(d); //15
```

# Function Declaration

---

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

```
function square(number) {  
    return number * number;  
}
```

- ▶ declarations are "hoisted" (vs function expressions) – see Lecture07
  - ▶ They can be declared anywhere in a file, and used before the declaration.



# Function Expressions

- ▶ Can be Anonymous function
  - ▶ Widely used in JS with event handlers

```
const square = function(number) { return number *  
number };  
let x = square(4) // x gets the value 16
```

Can also have a name to be used inside the function to refer to itself //NFE (Named Function Expression)

```
const factorial = function fac(n)  
{ return n < 2 ? 1 : n * fac(n - 1) };
```

```
console.log(factorial(3));
```

- ▶ Basically, a function expression is same syntax as a declaration, just used where an expression is expected ('lhs' vs 'rhs')

# Which is better: Function Declaration or Function Expression?



- ▶ Function declarations have two advantages over function expressions:
  - ▶ They are hoisted, so you can call them before they appear in the source code. (some consider this poor style)
  - ▶ They have a name. - the name of a function is useful for debugging; especially anonymous function. (ES6 infers name)
- ▶ Conclusion
  - ▶ Neither of the above are significant
  - ▶ Can use functions declarations if desired, but can always use function expressions to accomplish same thing (except hoisting)
- ▶ `function f1 () {} f1.name // 'f1'`
- ▶ `const f2 = function() {};` `f2.name // '' or 'f2'` in ES6
- ▶ `const f3 = function f3() {};` `f3.name // 'f3'`
  - ▶ Named Function Expression (NFE)

# Anonymous functions

---

- ▶ JavaScript allows you to declare anonymous functions
- ▶ Can be stored as a variable, attached as an event handler, etc.
- ▶ Keeping unnecessary names out of namespace for performance and safety

```
window.onload = function() {  
    alert("Hello World!");  
}
```



# Semicolon ;

- ▶ Semicolons are (technically) ‘optional’
  - ▶ JS implicitly adds them to our code if it makes the parser happy
  - ▶ in certain cases that can cause problems
    - ▶ return, var, break, throw, ...
  - ▶ Best practice to explicitly include them
  - ▶ Include brackets at the end of line versus new line
    - ▶ K&R (Kernighan and Ritchie) versus OTBS (one true brace style) styles

```
function a() {  
    return {  
        a: 1 ;  
    }  
}  
function b()  
{ //OTBS – ok, but not good practice according to some (Crockford, ...)  
    return //semicolon gets inserted here  
    {  
        a: 1 ;  
    }  
}  
console.log(a()); //object  
console.lo(b()); //undefined
```

# Web Storage

---

- ▶ With web storage, web applications can store data locally within the user's browser.
- ▶ Before HTML5, application data had to be stored in cookies, included in every server request. Web storage is more secure, and large amounts of data can be stored locally, without affecting website performance.
- ▶ Unlike cookies, the storage limit is far larger (at least 5MB) and information is never transferred to the server.
- ▶ Web storage is per origin (per domain and protocol). All pages, from one origin, can store and access the same data.

# The *localStorage* Object

---

- ▶ The *localStorage* object stores the data with no expiration date. The data will not be deleted when the browser is closed, and will be available the next day, week, or year.

## Example

```
// Store  
localStorage.setItem("lastname", "Smith");  
  
// Retrieve  
document.getElementById("result").innerHTML  
= localStorage.getItem("lastname");
```

# The *localStorage* Object

---

- ▶ The previous example above could also be written like this:

```
// Store  
localStorage.lastname = "Smith";  
  
// Retrieve  
document.getElementById("result").innerHTML  
= localStorage.lastname;
```

- ▶ The syntax for removing the "lastname" localStorage item is as follows:

```
localStorage.removeItem("lastname");
```

# The `sessionStorage` Object

---

- ▶ The `sessionStorage` object is equal to the `localStorage` object, except that it stores the data for only one session.
- ▶ The data is deleted when the user closes the specific browser tab.
- ▶ The following example counts the number of times a user has clicked a button, in the current session:

```
if (sessionStorage.clickcount) {  
    sessionStorage.clickcount =  
        Number(sessionStorage.clickcount) + 1;  
} else {  
    sessionStorage.clickcount = 1;  
}  
document.getElementById("result").innerHTML =  
    "You have clicked the button " +  
    sessionStorage.clickcount +  
    " time(s) in this session.";
```

# The `sessionStorage` Object

---

- ▶ The following example counts the number of times a user has clicked a button, in the current session:

```
if (sessionStorage.clickcount) {  
    sessionStorage.clickcount =  
        Number(sessionStorage.clickcount) + 1;  
} else {  
    sessionStorage.clickcount = 1;  
}  
  
document.getElementById("result").innerHTML =  
    "You have clicked the button " +  
    sessionStorage.clickcount +  
    " time(s) in this session.";
```

# Main Point

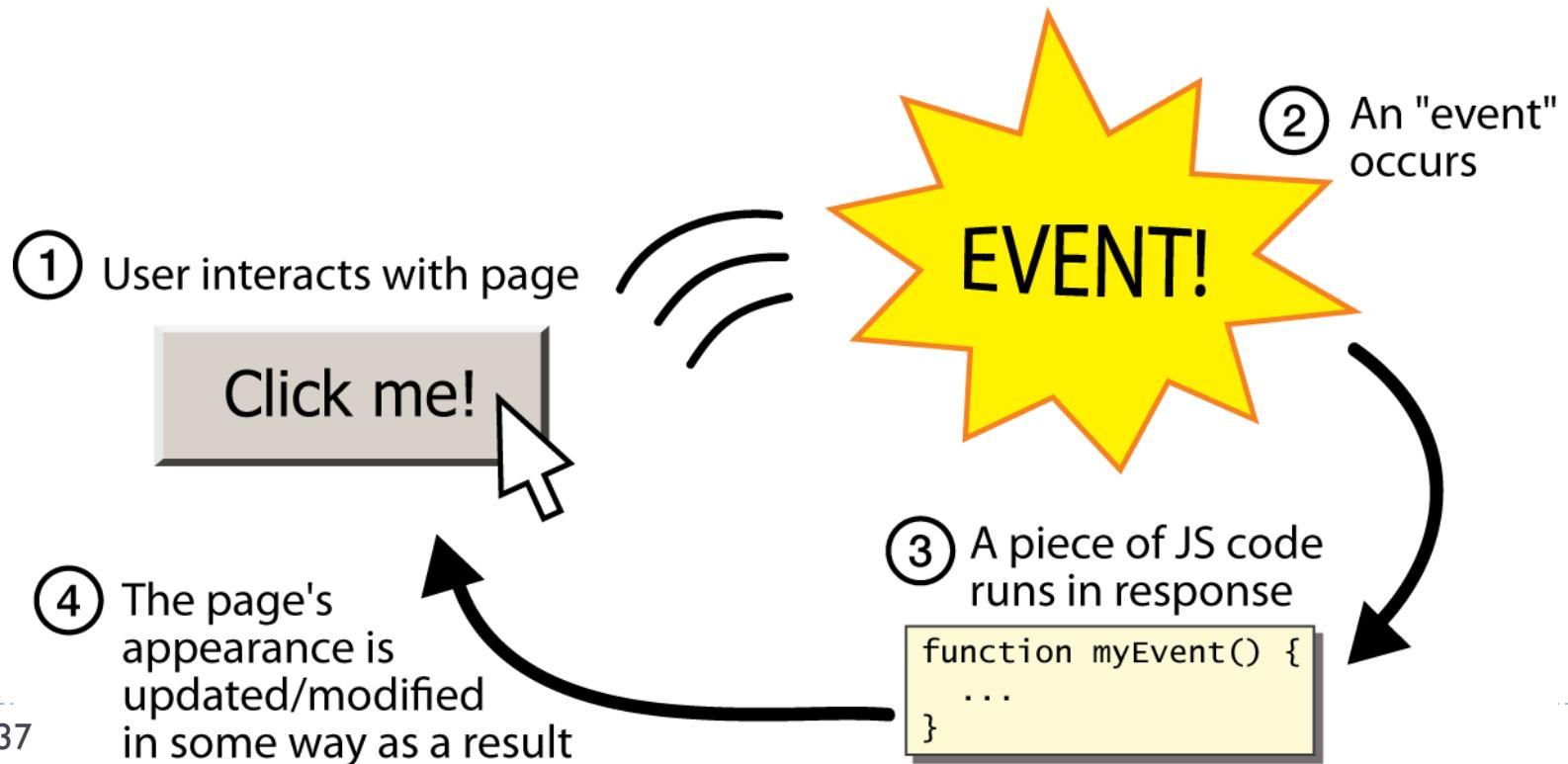
---

JavaScript is a loosely typed language. It has types, but does no compile time type checking. Programmers must be cautious of automatic type conversions, including conversions to Boolean types. It has a flexible and powerful array type as well as distinct types of null and undefined.

**Science of Consciousness:** To be an effective JavaScript programmer one needs to understand the principles and details of the language. If our awareness is established in the source of all the laws of nature then our actions will spontaneously be in accord with the laws of nature for a particular environment.

# Event-driven programming

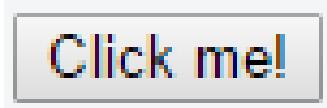
- ▶ JS programs have no main; they respond to user actions called **events**
- ▶ **event-driven programming:** writing programs driven by user events



# Button: <button>

---

<**button**>Click me!</**button**>



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



# Event handlers

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

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

- ▶ 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...

# Main Point

---

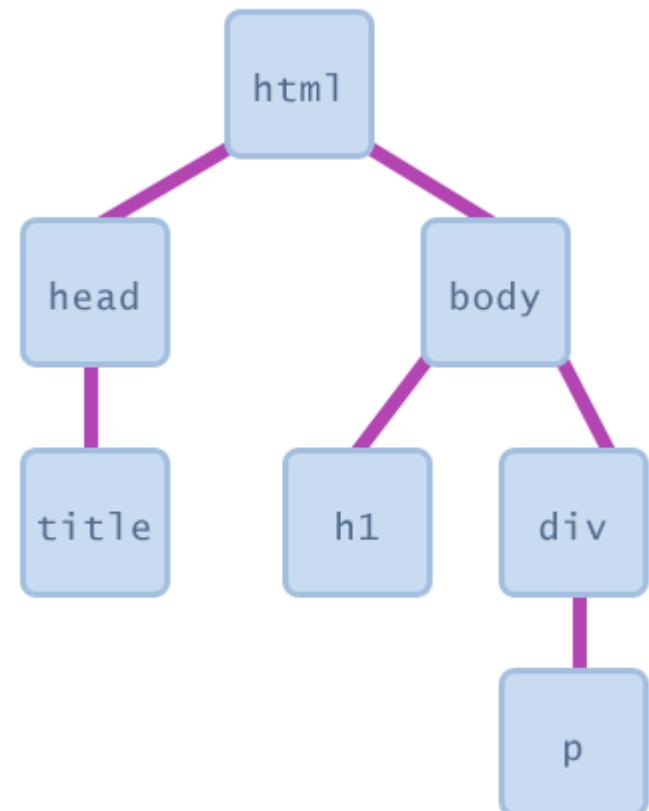
JavaScript programs have no main. They respond to user actions called events.

**Science of Consciousness:** JavaScript was designed as a language that could effectively respond to browser and DOM events. We respond most effectively to events in our environment if our awareness is settled and alert.

# Document Object Model (DOM)

---

- ▶ 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



# DOM element objects

- ▶ every element on the page has a corresponding DOM object
- ▶ access/modify the attributes of the DOM object with *objectName.attributeName*

HTML

```
<p>
  Look at this octopus:
  
  Cute, huh?
</p>
```

DOM Element Object

| Property   | Value         |
|------------|---------------|
| tagName    | "IMG"         |
| <u>src</u> | "octopus.jpg" |
| alt        | "an octopus"  |
| <u>id</u>  | "icon01"      |

JavaScript

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

# Accessing elements: `document.getElementById`



```
const name = document.getElementById("someId");  
  
<button onclick="changeText();">Click me!</button>  
<input id="output" type="text" value="replace me" />  
  
function changeText() {  
  const textbox = document.getElementById("output");  
  textbox.value = "Hello, world!";  
}
```

- ▶ `document.getElementById` returns the DOM object for an element with a given id
- ▶ can change the text in most *form controls* by setting the `value` property
- ▶ Browser automatically updates the screen when any DOM object is changed



# More advanced example

```
<button onclick="swapText()">Click me!</button>
<span id="output2">Hello</span>
<input id="textbox2" type="text" value="Goodbye" />

function swapText() {
  var span = document.getElementById("output2");
  var textBox = document.getElementById("textbox2");
  var temp = span.innerHTML;
  span.innerHTML = textBox.value;
  textBox.value = temp;
}
```

can change the text inside most elements by setting the innerHTML property

See examples: See example: lecture05\_examples/click2.html

lecture05\_examples/click1.html

# Main Point

---

The purpose of most JavaScript code is to manipulate the HTML DOM, which is a set of JavaScript objects that represent each element on an HTML page.

**Science of Consciousness:** The purpose of thought is to produce successful actions and achievements in the world, and more powerful thoughts will produce more successful actions.

# Lecture 6: JavaScript Programming Environment

---

Except where otherwise noted, the contents of this document are Copyright 2012 Marty Stepp, Jessica Miller, Victoria Kirst and Roy McElmurry IV. All rights reserved. Any redistribution, reproduction, transmission, or storage of part or all of the contents in any form is prohibited without the author's expressed written permission. Slides have been modified for Maharishi University of Management Computer Science course CS472 in accordance with instructors agreement with authors.

# Maharishi University of Management -Fairfield, Iowa © 2016



All rights reserved. No part of this slide presentation may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage and retrieval system, without permission in writing from Maharishi University of Management.

# Outline

---

- ▶ Global DOM Objects -- the global context all js programs run in
- ▶ Unobtrusive Javascript -- separation of content (HTML) and behavior (Javascript)
- ▶ DOM Element Objects -- the main operatives of js programs
- ▶ JS Timers -- needed for today's lab, and common tool for js

# The six global DOM objects

- ▶ Every JavaScript program can refer to the following global objects:

| Name      | Description                                        |
|-----------|----------------------------------------------------|
| document  | Current HTML page and its content                  |
| history   | List of pages the user has visited                 |
| location  | URL of the current HTML page                       |
| navigator | Info about the web browser you are using           |
| screen    | Info about the screen area occupied by the browser |
| window    | The browser window                                 |



# The window object

- ▶ *the entire browser window; the top-level object in DOM hierarchy*
  
- ▶ technically, all global code and variables become part of the window object
- ▶ properties:
  - ▶ document, history, location, name
- ▶ methods:
  - ▶ alert, confirm, prompt (popup boxes)
  - ▶ setInterval, setTimeout clearInterval, clearTimeout (timers)
  - ▶ open, close (popping up new browser windows)
  - ▶ blur, focus, moveBy, moveTo, print, resizeBy, resizeTo, scrollBy, scrollTo

# Popup windows with window.open

---

```
window.open ("http://foo.com/bar.html", "My  
Foo Window",  
"width=900,height=600,scrollbars=1");
```

- ▶ window.open pops up a new browser window
- ▶ This method is the cause of all the terrible popups on the web!
- ▶ some popup blocker software will prevent this method from running



# The document object

- ▶ *the current web page and the elements inside it*
- ▶ properties:
  - ▶ anchors, body, cookie, domain, forms, images, links, referrer, title, URL
- ▶ methods:
  - ▶ getElementById
  - ▶ getElementsByName
  - ▶ getElementsByTagName
  - ▶ close, open, write, writeln
- ▶ complete list

# The location object

---

- ▶ *the URL of the current web page*
- ▶ **properties:**
  - ▶ host, hostname, href, pathname, port, protocol, search
- ▶ **methods:**
  - ▶ assign, reload, replace
- ▶ complete list

# The navigator object

---

- ▶ *information about the web browser application*
- ▶ properties:
  - ▶ appName, appVersion, browserLanguage, cookieEnabled, platform, userAgent
  - ▶ complete list
- ▶ Some web programmers examine the navigator object to see what browser is being used, and write browser-specific scripts and hacks:
  - ▶ `if (navigator.appName === "Microsoft Internet Explorer") { ... }`
  - ▶ (this is poor style; you should not need to do this)

# The screen object

---

- ▶ *information about the client's display screen*
- ▶ **properties:**
  - ▶ availHeight, availWidth, colorDepth, height, pixelDepth, width
  - ▶ complete list

# The history object

---

- ▶ *the list of sites the browser has visited in this window*
  
- ▶ properties:
  - ▶ length
- ▶ methods:
  - ▶ back, forward, go
- ▶ complete list
- ▶ sometimes the browser won't let scripts view history properties, for security

# Main Point

---

JavaScript has a set of global DOM objects accessible to every web page. Every JavaScript object runs inside the global window object. The window object has many global functions such as alert and timer methods. At the level of the unified field, an impulse anywhere is an impulse everywhere.

# Unobtrusive JavaScript

---

- ▶ JavaScript event code seen yesterday was *obtrusive*, in the HTML; this is bad style
- ▶ now we'll see how to write unobtrusive JavaScript code
  - ▶ HTML with minimal JavaScript inside
  - ▶ uses the DOM to attach and execute all JavaScript functions
- ▶ allows separation of web site into 3 major categories:
  - ▶ **content** (HTML) - what is it?
  - ▶ **presentation** (CSS) - how does it look?
  - ▶ **behavior** (JavaScript) - how does it respond to user interaction?

# Obtrusive event handlers(bad)

---

```
<button onclick="okayClick();">OK</button>
```

```
function okayClick() {  
    alert("booyah");  
}
```

- ▶ this is bad style (HTML is cluttered with JS code)
- ▶ goal: remove all JavaScript code from the HTML body



# Unobtrusive JavaScript

```
// where element is a DOM element object  
element.onevent = function;
```

```
<button id="ok">OK</button>  
var okButton = document.getElementById("ok");  
okButton.onclick = okayClick;
```

- ▶ it is legal to attach event handlers to elements' DOM objects in your JavaScript code
  - ▶ notice that you do **not** put parentheses after the function's name
- ▶ this is better style than attaching them in the HTML
- ▶ Where should we put the above code?

# Linking to a JavaScript file: script

---

- ▶ JS code can be placed directly in the HTML file's body or head (like CSS)
  - ▶ but this is bad style (should separate content, presentation, and behavior)
- ▶ script tag should be placed in HTML page's head
- ▶ script code should be stored in a separate .js file (like CSS)

```
<script src="example.js" ></script>
```

# When does my code run?

---

```
<html>
  <head>
    <script src="myfile.js"></script> </head>
  <body> ... </body> </html>
```

```
// global code
var x = 3;
function f(n) { return n + 1; }
function g(n) { return n - 1; }
x = f(x);
```

- ▶ your file's JS code runs the moment the browser loads the script tag
  - ▶ any variables are declared immediately
  - ▶ any functions are declared but not called, unless your global code explicitly calls them
- ▶ at this point in time, the browser has not yet read your page's body
  - ▶ none of the DOM objects for tags on the page have been created yet

See example: lecture06\_examples/runjs.html

# A failed attempt at being unobtrusive

---

```
<html>
  <head>
    <script src="myfile.js" type="text/javascript"></script>
  </head>
  <body> ... </body> </html>
<div><button id="ok">OK</button></div>

// code in myfile.js
document.getElementById("ok").onclick = okayClick; //  

  error: cannot set property onclick of null
```

- ▶ problem: myfile.js code runs the moment the script is loaded
- ▶ script in head is processed before page's body has loaded
  - ▶ no elements are available yet or can be accessed yet via the DOM
  - ▶ See lecture06\_examples/failedattempt.html
- ▶ we need a way to attach the handler after the page has loaded...

# The window.onload event

---

```
// this will run once the page has finished loading
function functionName() {
  element.event = functionName;
  element.event = functionName;
  ...
}
window.onload = functionName; // global code
```

- ▶ we want to attach our event handlers right after the page is done loading
  - ▶ there is a global event called `window.onload` event that occurs at that moment
- ▶ in `window.onload` handler we attach all the other handlers to run when events occur

See example: `lecture06_examples/unobtrusivehandler1.html` (does it work?)

# Common unobtrusive JS errors

---

- ▶ many students mistakenly write () when attaching the handler

```
window.onload = pageLoad();
```

```
window.onload = pageLoad;
```

```
okButton.onclick = okayClick();
```

```
okButton.onclick = okayClick;
```

- ▶ **IMPORTANT FUNDAMENTAL CONCEPT !!!**

- ▶ Function reference versus evaluation

- ▶ event names are all lowercase, not capitalized like most variables

```
window.onLoad = pageLoad;
```

```
window.onload = pageLoad;
```

# Anonymous functions

---

```
function (parameters) {  
    statements;  
}
```

- ▶ JavaScript allows you to declare **anonymous functions**
- ▶ creates a function without giving it a name
- ▶ can be stored as a variable, attached as an event handler, etc.
- ▶ Important in JavaScript because of event handling nature and minimizing namespace clutter

# Anonymous function example

---

```
window.onload = function() {  
    var okButton = document.getElementById("ok");  
    okButton.onclick = okayClick;  
};  
  
function okayClick() {  
    alert("booyah");  
}
```

or the following is also legal (though harder to read):

```
window.onload = function() {  
    var okButton = document.getElementById("ok");  
    okButton.onclick = function() {  
        alert("booyah");  
    };  
};
```

See example: lecture06\_examples/anonymous.html

---

# Main Points

---

- ▶ Unobtrusive JavaScript promotes separation of web page content into 3 different concerns: content (HTML), presentation (CSS), and behavior(JS) (ala MVC, knower, known, process of knowing)
- ▶ JavaScript code runs when the page loads it. Event handlers cannot be assigned until the target elements are loaded. Event handlers often use anonymous functions. Creative intelligence proceeds in an orderly sequential manner.

# Recall: DOM element objects

- ▶ every element on the page has a corresponding DOM object
- ▶ access/modify the attributes of the DOM object with *objectName.attributeName*

HTML

```
<p>
  Look at this octopus:
  
  Cute, huh?
</p>
```

DOM Element Object

| Property   | Value         |
|------------|---------------|
| tagName    | "IMG"         |
| <u>src</u> | "octopus.jpg" |
| alt        | "an octopus"  |
| <u>id</u>  | "icon01"      |

JavaScript

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

# DOM object properties

```
<div id="main" class="foo bar">  
  <p>Hello, <em>very</em> happy to see you!</p>  
    
</div>  
  
var mainDiv = document.getElementById("main");  
var icon = document.getElementById("icon");
```

See example: lecture06\_examples/objectproperties.html

| Property  | Description            | Example                               |
|-----------|------------------------|---------------------------------------|
| tagName   | element's HTML tag     | mainDiv.tagName is "DIV"              |
| className | CSS classes of element | mainDiv.className is "foo bar"        |
| innerHTML | Content in element     | mainDiv.innerHTML is "\n <p>Hello..." |
| src       | URL target of an image | icon.src is "greatwall.jpg"           |

# DOM properties for form controls

```
<input id="sid" type="text" size="7" maxlength="7"  
/>  
<input id="frosh" type="checkbox" checked="checked"  
/> Freshman?  
var sid = document.getElementById("sid");  
var frosh = document.getElementById("frosh");
```

| Property | Description                                | Example                      |
|----------|--------------------------------------------|------------------------------|
| value    | the text/value chosen by the user          | sid.value could be "1234567" |
| checked  | whether a box is checked                   | frosh.checked is true        |
| disabled | whether a control is disabled<br>(boolean) | frosh.disabled is false      |
| readOnly | whether a text box is read-only            | sid.readOnly is false        |

# More about from controls

---

```
<select id="captain">  
  <option value="kirk">James T. Kirk</option>  
  <option value="picard">Jean-Luc Picard</option>  
  <option value="cisco">Benjamin Cisco</option>  
</select>  
  
<label>  
  <input id="trekkie" type="checkbox" /> I'm a  
  Trekkie  
</label>
```

- ▶ when talking to a text box or select, you usually want its value
- ▶ when talking to a checkbox or radio button, you probably want to know if it's checked (true/false)

# Abuse of innerHTML

---

```
// bad style!
var paragraph = document.getElementById("welcome");
paragraph.innerHTML = "<p>text and <a
    href='page.html'>link</a>";
```

- ▶ innerHTML can inject arbitrary HTML content into the page
- ▶ however, this is prone to bugs and errors and is considered poor style
  - ▶ innerHTML not part of DOM standard
  - ▶ issues involving DOM rebuilding tree and maintaining node and event handler references
- ▶ Best practice: inject plain text only
  - ▶ Do not use innerHTML to inject HTML tags;

# DOM elements style property

```
<button id="clickme">Color Me</button>  
window.onload = function() {  
    document.getElementById("clickme").onclick =  
        changeColor;  
};  
  
function changeColor() {  
    var clickMe = document.getElementById("clickme");  
    clickMe.style.color = "red";  
}
```

| Property | Description                                        |
|----------|----------------------------------------------------|
| style    | lets you set any CSS style property for an element |

- ▶ contains same properties as in CSS, but with camelCasedNames
  - ▶ examples: backgroundColor, borderLeftWidth, fontFamily

# Common DOM styling errors

---

- ▶ many students forget to write `.style` when setting styles

```
var clickMe = document.getElementById("clickme");
clickMe.color = "red";
clickMe.style.color = "red";
```

- ▶ style properties are capitalized likeThis, not like-this

```
clickMe.style.fontSize = "14pt";
clickMe.style.fontSize = "14pt";
```

- ▶ style properties *must* be set as strings, often with units at the end

```
clickMe.style.width = 200;
clickMe.style.width = "200px";
clickMe.style.padding = "0.5em";
```

- ▶ write exactly the value you would have written in the CSS, but in quotes

# Unobtrusive styling

---

```
function okayClick() {  
    this.style.color = "red";  
    this.className = "highlighted";  
}  
.highlighted { color: red; }
```

- ▶ well-written JavaScript code should contain as little CSS as possible
- ▶ use JS to set CSS classes/IDs on elements
- ▶ define the styles of those classes/IDs in your CSS file
- ▶ What about cssZenGarden?

# Getting/Setting CSS classes

---

```
function highlightField() {  
  // turn text yellow and make it bigger  
  if (!document.getElementById("text").className) {  
    document.getElementById("text").className = "highlight";  
  } else if  
    (document.getElementById("text").className.indexOf("inva  
lid") < 0) {  
    document.getElementById("text").className += "  
highlight";  
  }  
}
```

- ▶ JS DOM's `className` property corresponds to HTML class attribute
- ▶ somewhat clunky when dealing with multiple space-separated classes as one big string

# Problems with reading/changing styles

---

```
<button id="clickme">Click Me</button>  
  
window.onload = function() {  
    document.getElementById("clickme").onclick = biggerFont;  
}  
  
function biggerFont() {  
    var size =  
        parseInt(document.getElementById("clickme").style.fontSize);  
    size += 4;  
    document.getElementById("clickMe").style.fontSize = size  
    + "pt";  
}
```

- ▶ style property lets you set any CSS style for an element
- ▶ problem(??): you cannot (sometimes) read existing styles with it
  - ▶ E.g., older browsers when fontSize is inherited
- ▶ See example: lecture06\_examples/style.html

# Common bug: incorrect usage of existing styles

---

```
document.getElementById("main").style.top =  
document.getElementById("main").style.top + 100 +  
"px"; // bad
```

- ▶ the above example computes e.g. "200px" + 100 + "px" , which would evaluate to "200px100px"
- ▶ a corrected version:

```
document.getElementById("main").style.top =  
parseInt(document.getElementById("main").style.top  
) + 100 + "px"; // correct
```

# Main Points

---

- ▶ The DOM is an API so the JavaScript programmer can conveniently access and manipulate the HTML elements in code.
- ▶ **Science of Consciousness:** The TM and TM-Sidhi programs are techniques that allow anyone to conveniently contact and act at the level of the Unified Field.

# Timer events

---

| method                                                                                    | description                                                  |
|-------------------------------------------------------------------------------------------|--------------------------------------------------------------|
| <code><u>setTimeout</u>(function, delayMS);</code>                                        | arranges to call given function after given delay in ms      |
| <code><u>setInterval</u>(function, delayMS);</code>                                       | arranges to call function repeatedly every <i>delayMS</i> ms |
| <code><u>clearTimeout</u>(timerID);</code><br><code><u>clearInterval</u>(timerID);</code> | stops the given timer so it will not call its function       |

- ▶ both `setTimeout` and `setInterval` return an ID representing the timer
  - ▶ this ID can be passed to `clearTimeout/Interval` later to stop the timer

# Asynchronous & Callbacks

---

- ▶ A callback function is a function you give to another function, to be invoked later when the other function is finished (desired).
- ▶ All callback functions do not execute immediately (even when called), instead they are queued in the browser event queue.
- ▶ How JavaScript deals with being single threaded



# setTimeout Example

```
<button onclick="delayMsg();">Click me!</button>
<span id="output"></span>
```

```
function delayMsg() {
  setTimeout(booyah, 5000);
  document.getElementById("output").innerHTML =
  "Wait for it...";
}

function booyah() {
  // called when the timer goes off
  document.getElementById("output").innerHTML =
  "BOOYAH!";
}
```



# setInterval example

```
timer = null; // stores ID of interval timer
function delayMsg2() {
  if (timer === null) {
    timer = setInterval(rudy, 1000);
  } else {
    clearInterval(timer);
    timer = null;
  }
}

function rudy() { // called each time the timer goes off
  document.getElementById("output").innerHTML += "Rudy!";
}
```



# Passing parameters to timers

```
function delayedMultiply() {  
    // 6 and 7 are passed to multiply when  
    // timer goes off  
    setTimeout(multiply, 4000, 6, 7);  
}  
  
function multiply(a, b) {  
    alert(a * b);  
}
```

- ▶ any parameters after the delay are eventually passed to the timer function
- ▶ why not just write this? `setTimeout(multiply(6, 7), 2000);`

# Common timer errors

---

- ▶ many students mistakenly write () when passing the function

```
setTimeout(booyah(), 2000);
```

```
setTimeout(booyah, 2000);
```

```
setTimeout(multiply(num1 * num2), 2000);
```

```
setTimeout(multiply, 2000, num1, num2);
```

- ▶ what does it actually do if you have the () ?
  - ▶ it calls the function immediately, rather than waiting the 2000ms!
- ▶ **IMPORTANT!!!**

# First-class functions

---

- ▶ Functions can be assigned to variables

```
var myfunc = function(a, x) {  
    return a * b;  
};
```

- ▶ Functions can be passed as parameters

```
function apply(a, b, f) {  
    return f(a, b);  
}  
  
var x = apply(2, 3, myfunc); // 6
```

- ▶ Functions can be return values

```
function getAlert(str) {  
    return function() { alert(str); }  
}  
  
var whatsUpAlert= getAlert("What's up!");  
whatsUpAlert(); // "What's up!"
```

# JavaScript “strict” mode

---

```
"use strict"; your code...
```

```
(function() {  
  "use strict";  
  your code...  
})();
```

- ▶ writing "use strict"; at the very top of your JS file turns on strict syntax checking:
  - ▶ shows an error if you try to assign to an undeclared variable
  - ▶ stops you from overwriting key JS system libraries
  - ▶ forbids some unsafe or error-prone language features
- ▶ You should always turn on strict mode for your code in this class
- ▶ IIFE syntax will be discussed further in next lesson

*Maharishi University of Management  
Department of Computer Science  
CS472 Web Programming*

# Scope, Closures, and Encapsulation in Javascript

## References

- <http://www.cs.washington.edu/education/courses/cse341/10au/lectures/slides/27-scope-closures.pdf>
- <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Closures>
- <http://www.joezimjs.com/javascript/javascript-closures-and-the-module-pattern/>
- <http://www.jblearning.com/catalog/9780763780609/>

# Lexical scope in Java

- In Java, every block ( {} ) defines a scope.

```
public class Scope {  
    public static int x = 10;  
  
    public static void main(String[] args) {  
        System.out.println(x);  
        if (x > 0) {  
            int x = 20;  
            System.out.println(x);  
        }  
        int x = 30;  
        System.out.println(x);  
    }  
}
```

# Function scope in JavaScript

- In JavaScript, there are only two scopes:
  - **global scope**: global environment for functions, vars, etc.
  - **function scope**: every function gets its own inner scope

```
var x = 10;                                // foo.js
function main() {
    var x = 40;
    print(x);
    x = 20;
    if (x > 0) {
        x = 30;
        print(x);
    }
    var f = function(x) { print(x); }
    f(50);
}
// 
x = 70;
```

# Another scope example

```
function f() {  
    var a = 1, b = 20, c;  
    print(a + " " + b + " " + c);           // 1 20 undefined  
    // declares g (but doesn't call immediately!)  
    function g() {  
        var b = 300, c = 4000;  
        print(a + " " + b + " " + c);       // 1 300 4000  
        a = a + b + c;  
        print(a + " " + b + " " + c);       // 4301 300 4000  
    }  
    print(a + " " + b + " " + c);           // 1 20 undefined  
    g();  
    print(a + " " + b + " " + c);           // 4301 20 undefined  
} //run it
```

# Lack of block scope

```
for (var i = 0; i < 10; i++) {  
    print(i);  
}  
print(i); // 10  
if (i > 5) {  
    var j = 3;  
}  
print(j); //run it
```

- any variable declared lives until the end of the function
  - lack of block scope in JS leads to errors for some coders
  - this is a "bad part" of JavaScript (D. Crockford)

# var VS let (ES6)

- var scope – nearest function scope
- let scope – nearest enclosing block

```
function a() {  
  for (var x = 1; x < 10; x++) {  
    console.log(x);  
  }  
  console.log("x: " + x);  
 //10  
}
```

```
function a() {  
  for (let x = 1; x < 10; x++) {  
    console.log(x);  
  }  
  console.log("x: " + x);  
 //ReferenceError: x is not defined  
}
```

- let has block scope
- Use let inside for loops to prevent leaking to Global Scope

# *let* variables in ES

- From ES6, you can use the *let* keyword to declare a variable. The *let* keyword is similar to the *var* keyword. However, variable declared using the *let* keyword is block-scoped, not function-scoped.
- In the following example, we declare the *tmp* variable within a block surrounding by the curly braces {}. The *tmp* variable only exists inside the block, therefore, any reference to it outside of the block will cause a *ReferenceError*.

# *let* example

```
var foo = 20, bar = 10;  
  
{  
    let tmp = foo;  
    foo = bar;  
    bar = tmp;  
}  
  
console.log(tmp); // ReferenceError
```

# *const* in ES6

- The *const* keyword works like the *let* keyword, but the variable that you declare must be initialized immediately with a value, and that value can't be changed afterward.

```
const CODE = 100;  
CODE = 200; // TypeError: CODE is read-only
```

# Hoisting

*Hoisting* is JavaScript's default behavior of moving all declarations to the top of the current function. The following both give the same result:

## Example 1

```
x = 5;  
elem = document.getElementById("demo");  
elem.innerHTML = x;  
var x; // Declare x
```

## Example 2

```
var x; // Declare x  
x = 5;  
elem = document.getElementById("demo");  
elem.innerHTML = x;
```

# Hoisting

JavaScript only hoists declarations, not initializations. The following Examples are equivalent:

## Example 1

```
var x = 5;      // Initialize x
elem = document.getElementById("demo");
elem.innerHTML = x + " " + y;
var y = 7;      // Initialize y
```

## Example 2

```
var x = 5;      // Initialize x
var y;          // Declare y
elem = document.getElementById("demo");
elem.innerHTML = x + " " + y;
y = 7;          // Assign 7 to y
```

# Main Point

JavaScript has global scope and local scope within functions when variables are declared with var, and now has block scope with const and let.

**Science of Consciousness:** The experience of transcending opens our awareness to the expanded vision of unbounded awareness, at the same time that it promotes the ability to focus sharply within any local boundaries.

# First-class functions

- Functions can be assigned to variables

```
var myfunc = function(a, x) {  
    return a * b;  
};
```

- Functions can be passed as parameters

```
function apply(a, b, f) {  
    return f(a, b);  
}  
var x = apply(2, 3, myfunc); // 6
```

- Functions can be return values

```
function getAlert(str) {  
    return function() { alert(str); }  
}  
var whatsUpAlert= getAlert("What's up!");  
whatsUpAlert(); // "What's up!"
```

# Javascript functions

- Function *parameters* are the names listed in the function definition.
- Function *arguments* are the real values passed to (and received by) the function.
- JavaScript function definitions do not specify data types for parameters.
- JavaScript functions do not perform type checking on the passed arguments.
- JavaScript functions do not check the number of arguments received.
- If a function is called with missing arguments (less than declared), the missing values are set to: *undefined*

# arguments Object

*JavaScript functions have a built-in object called the **arguments** object. The **arguments** object contains an array of the arguments used when the function is called (invoked).*

```
function findMax() {  
    var i;  
    var max = -Infinity;  
    for (i = 0; i < arguments.Length; i++) {  
        if (arguments[i] > max) {  
            max = arguments[i];  
        }  
    }  
    return max;  
}
```

```
var x = findMax(1, 123, 500, 115, 44, 88); // 500  
var x = findMax(5, 32, 24); // 32
```

# Arrow functions (ES6)

- Arrow functions are function shorthand using => syntax.
- Syntactically similar to Java 8, lambda expressions
- Two factors influenced the introduction of arrow functions:
  - Shorter functions
  - Non-binding of this (covered later)

# Arrow functions (ES6)

Arrow functions can be a shorthand for an anonymous function.

```
(arguments) => { return statement } // general  
syntax  
argument => { return statement } // one  
parameter  
argument => statement // implicit return  
() => statement // no input
```



```
function multiply (num1, num2) {  
    return num1 * num2;  
}  
var output = multiply(5, 5);  
() => ({ })  
var multiply = (num1, num2)  
    => num1 * num2;  
var output = multiply(5, 5);
```

# Default Parameters (ES6)

```
function log(x=10, y=5) {  
    console.log( x + ", " + y);  
}  
  
log(); // 10, 5  
log(5); // 5, 5  
log(5, 10); // 5, 10
```

# Rest Operator (ES6)

- A **Rest** syntax allows us to represent variable number of arguments as an Array.
  - Its like **varargs** in Java and has same syntax.
  - Rest parameters should be the last parameter in a function.

```
function sum(x,y, ...more){  
    var total = x + y;  
    if(more.length > 0){  
        for (var i=0;  
i<more.length; i++) {  
            total += more[i];  
        }  
    }  
    console.log(total);  
}  
  
sum(4,4); // 8  
sum(4,4,4); // 12
```

# Calling an inner function

```
function init() { //function declaration
    var name = "Mozilla";
    function displayName() {
        alert(name);
    }
    displayName();
}
init();
```

# Returning an inner function

```
function makeFunc() {  
    var name = "Mozilla"; //local to makeFunc  
    function displayName() {  
        alert(name);  
    }  
    return displayName;  
}  
  
var myFunc = makeFunc();  
myFunc(); //is the local variable still accessible by myFunc?  
  
//another reference and demo on inner function scope
```

# Closures

- **closure:** A first-class function that binds to free variables that are defined in its execution environment.
- **free variable:** A variable referred to by a function that is not one of its parameters or local variables.
  - **bound variable:** A free variable that is given a fixed value when "closed over" by a function's environment.
- A *closure* occurs when a(n inner) function is defined and it attaches itself to the free variables from the surrounding environment to "close" up those stray references.

# Closures in JS

```
var x = 1;
function f() {
    var y = 2;
    var summ= function() {
        var z = 3;
        print(x + y + z);
    };
    y = 10; return summ;
}
var g = f();
g(); // 1+10+3 is 14 -- run it
```

- inner function closes over free variables as it is declared
  - grabs references to the names, not values (sees updates)

# Common closure bug

```
var funcs = [];
for (var i = 0; i < 5; i++) {
    funcs[i] = function() { return i; };
}
> funcs[0]();
5
> funcs[1]();
5
```

- Closures that bind a loop variable often have this bug.
  - Why do all of the functions return 5?

# Common closure bug with fix (ES6)

```
//buggy version with var  
var funcs = [];  
for (var i = 0; i < 5; i++) {  
  funcs[i] = function() {  
    return i;  
  };  
}
```

```
//ES6 solution: let vs var  
const funcs = [];  
for (let i = 0; i < 5; i++) {  
  funcs[i] = function() {  
    return i;  
  };  
}
```

```
console.log(funcs[0]());  
console.log(funcs[1]());  
console.log(funcs[2]());  
console.log(funcs[3]());  
console.log(funcs[4]());
```

# Practical uses of closures

- A closure lets you associate some data (the environment) with a function—parallel to properties and methods in OOP.
- Consequently, you can use a closure anywhere you might use an object with a single method.
- Situations like this are common on the web.
  - an event handlers is a single function executed in response to an event.
    - e.g., DOM and timer event handlers
  - closures also very useful in Javascript for encapsulation and namespace protection

# Function factory with closures

- example of closures being helpful with event handling

```
body { font-family: Helvetica, Arial, sans-serif; font-size: 12px; }
h1 { font-size: 1.5em; }
h2 { font-size: 1.2em; }
```

```
<p>Some paragraph text</p>
  <h1>some heading 1 text</h1>
  <h2>some heading 2 text</h2>
  <a href="#" id="size-12">12</a>
  <a href="#" id="size-14">14</a>
  <a href="#" id="size-16">16</a>
```

```
function makeSizer(size) {
  return function() {
    document.body.style.fontSize = size + 'px';
  };
}
document.getElementById('size-12').onclick = makeSizer(12);
document.getElementById('size-14').onclick = makeSizer(16);
document.getElementById('size-16').onclick = makeSizer(20);
```

- <http://jsfiddle.net/vnkuZ> //jsfiddle link
- why does makeSizer need to return a function?
- what is the free variable and why is it needed?

# Encapsulation and namespace protection with closures

- Languages such as Java provide private methods
  - can only be called by other methods in the same class.
- JavaScript does not provide this, but possible to emulate private closures.
- also provide powerful way of managing global namespace,
- Here's how to define some public functions that can access private functions and variables, using closures which is also known as the [module pattern](#):
- “Every real JavaScript programmer should know this if he or she wants to become great” Joe Zim

# Module pattern

```
(function(params) {  
    statements;  
})(params);
```

- declares and immediately calls an anonymous function
  - parens around function are a special syntax that means this is a function expression that will be immediately invoked
    - “immediately invoked function”
  - used to create a new **scope** and **closure** around it
  - can help to avoid declaring global variables/functions
  - used by JavaScript libraries to keep global namespace clean

# Module example

```
// old: 3 globals  
  
var count = 0;  
function incr(n) {  
    count += n;  
}  
function reset() {  
    count = 0;  
}  
incr(4);  incr(2);  
document.write(count);
```

```
// new: 0 globals!  
(function() {  
    var count = 0;  
    function incr(n) {  
        count += n;  
    }  
    function reset() {  
        count = 0;  
    }  
    incr(4);  incr(2);  
    document.write (count);  
})(); //run it
```

- declare-and-call protects your code and avoids globals
  - avoids common problem with namespace/name collisions

# Implied globals

```
name = value;  
  
function foo() {  
    x = 4;  
    print(x);  
} // oops, x is still alive now (global)
```

- if you assign a value to a variable without `var`, JS assumes you want a new *global* variable with that name
  - hard to distinguish
  - this is a "bad part" of JavaScript (D.Crockford)

# Main Point

2. Closures are created whenever an inner function is defined and it closes over its free variables. Closures provide encapsulation of methods and data. Encapsulation promotes self-sufficiency, stability, and re-usability.

# Javascript Objects

# How about classes and objects?

- small programs are easily written without objects
- JavaScript treats functions as *first-class citizens*
- larger programs become cluttered with disorganized functions
- objects group *related data and behavior*
  - helps manage size and complexity, promotes code reuse
- You have already *used* many types of JavaScript objects
  - Strings, arrays, HTML / XML DOM nodes
  - global DOM objects
  - The jQuery object (following lessons)

# Javascript objects

- objects in Javascript are more like associative arrays
- the keys can be any string
- you do not need quotes if the key is a valid javascript identifier
- values can be anything, including functions
- you can add keys dynamically using associative array or the . syntax
- ```
var x = {
  'a': 97, 'b': 98, 'c': 99, 'd': 199,
  'mult': function(a, b) {
    return a * b; }
};
```

# Common examples of using object literals

```
$.ajax("http://example.com/app.php", {  
  'method': "post",      // an object with a field named method (String)  
  'timeout': 2000        // and a field name timeout  
});
```

```
$("<div>",  
  {'css': {  
    'color': red  
  },  
  'id': 'myid',          // an id field  
  'click': myClickHandler // and a method called click  
});
```

- the parameters in {} passed to jQuery functions and methods are object literals
- object literals are the basis of JSON

# Objects that have behavior

```
var name = {  
    ...  
    methodName: function(parameters) {  
        statements;  
    }  
};  
  
var pt = {  
    x: 4, y: 3,  
    distanceFromOrigin: function() {  
        return Math.sqrt(this.x * this.x + this.y * this.y);  
    }  
};  
  
alert(pt.distanceFromOrigin()); // 5
```

- like in Java, objects' methods run "inside" that object
  - inside an object's method, the object refers to itself as this
  - unlike in Java, the this keyword is mandatory in JS

# The global object

- technically *no* JavaScript code is "static" in the Java sense
  - *all* code lives inside of some object
  - there is *always* a `this` reference that refers to that object
- all code is executed inside of a **global object**
  - in browsers, it is also called `window`; in Rhino: `global()`
  - global variables/functions you declare become part of it
    - they use the global object as `this` when you call them
- "*JavaScript's global object [...] is far and away the worst part of JavaScript's many bad parts.*" -- D. Crockford

# Global object and this keyword

```
function printMe() {  
    print("I am " + this.name);  
}  
> var name = “Alfred E. Newman”;  
> var teacher = {  
        name: “Prof. Tyler Durden”  
        department: “CS”  
};  
> teacher.print = printMe;  
> teacher.print();  
I am Prof. Tyler Durden  
> printMe();  
I am Alfred E. Newman
```

# for each over object literal

```
var things = {'a': 97, 'b': 98, 'c': 99 };
for (key in things) {
  console.log(key + ', ' + things[key]);
}
```

a, 97  
b, 98  
c, 99

# Emulating private methods with closures (module pattern)

```
var counter = (function() { //the parens surrounding the function are JS syntax for immediate evaluation
    var privateCounter = 0; //private data
    function changeBy(val) { //private inner function
        privateCounter += val;
    }
    return {
        increment: function() { // three public functions are closures that share the same environment.
            changeBy(1);
        },
        decrement: function() {
            changeBy(-1);
        },
        value: function() {
            return privateCounter;
        }
    }
})();

alert(counter.value()); /* Alerts 0 */
counter.increment();
counter.increment();
alert(counter.value()); /* Alerts 2 */
counter.decrement();
alert(counter.value()); /* Alerts 1 */

//additional reference and demo on closures and the module pattern
```

# Emulating private methods with closures

- We could store this function in a separate variable and use it to create several counters.

```
var makeCounter = function() {
  var privateCounter = 0;
  function changeBy(val) {
    privateCounter += val;
  }
  return {
    increment: function() {
      changeBy(1);
    },
    decrement: function() {
      changeBy(-1);
    },
    value: function() {
      return privateCounter;
    }
  };
};
var counter1 = makeCounter();
var counter2 = makeCounter();
alert(counter1.value()); /* Alerts 0 */
counter1.increment();
counter1.increment();
alert(counter1.value()); /* Alerts 2 */
counter1.decrement();
alert(counter1.value()); /* Alerts 1 */
alert(counter2.value()); /* Alerts 0 */
```

- Could the immediate evaluation syntax be used here also?

PROGRAMMING WITH

# JavaScript

*Algorithms and Applications for Desktop  
and Mobile Browsers*

Author: Ray Toal

Ray Toal



## Chapter 5

### *Functions*

# Functions as properties of an object

- Grouping related functions as properties of a single object help organize large programs
- Keep the global namespace clean
  - Important for performance
  - Critical for avoiding name clashes between scripts

```
var Geometry = {  
    circleArea: function (radius) {  
        return Math.PI * radius * radius; },  
  
    circleCircumference: function (radius) {  
        return 2 * Math.PI * radius; },  
  
    sphereSurfaceArea: function (radius) {  
        return 4 * Math.PI * radius * radius; },  
  
    boxVolume: function (length, width, depth) {  
        return length * width * depth; }  
};
```

# Use Object.create: A circle datatype

```
/* A prototypical circle, designed to be the prototype for all circles
created with the Circle function below.
*/
var protoCircle = {
    radius: 1,
    area: function () {return Math.PI * this.radius * this.radius;},
    circumference: function () {return 2 * Math.PI * this.radius;}
};

var circle = function (r) {
    var circ = Object.create(protoCircle); //protoCircle assigned to prototype prop
    circ.radius = r;
    return circ;
};

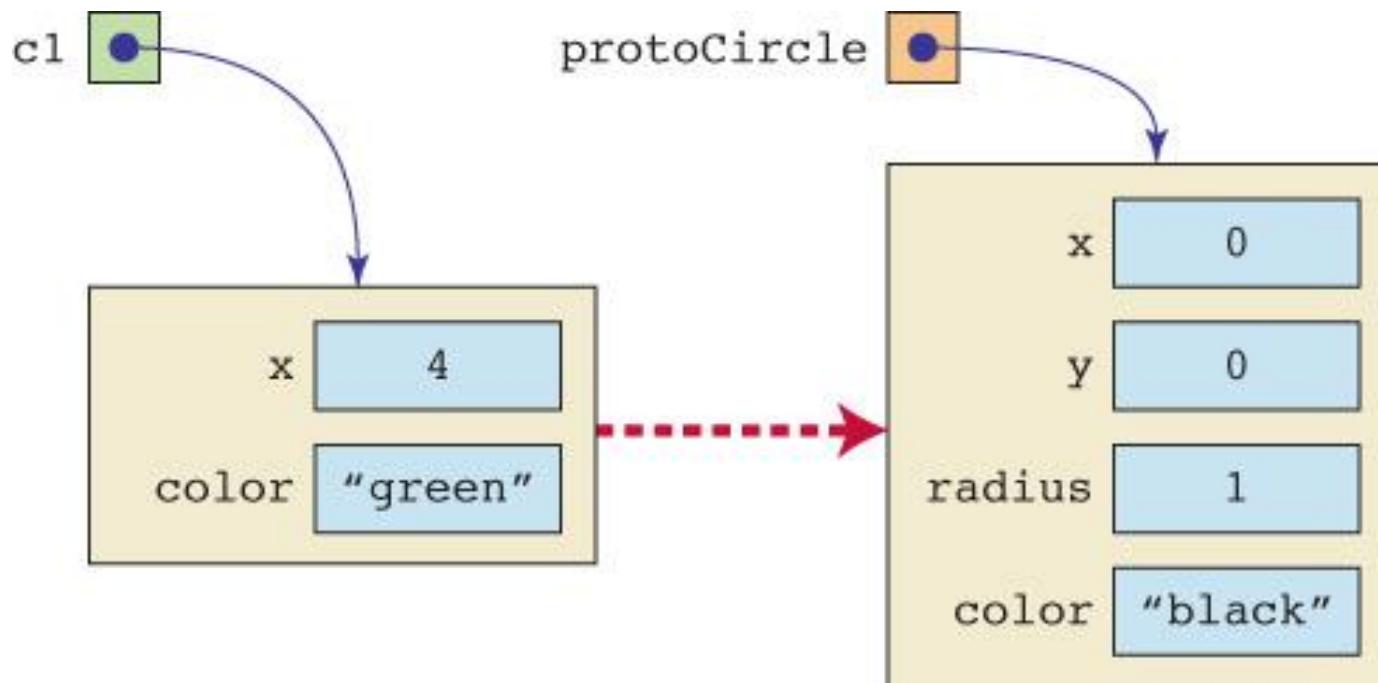
/* Creates a circle with a given radius. */
var c = circle(5);
c.radius => 5
c.area() => 25pi
c.circumference() => 10pi
```

# Object Prototypes

- Every object has a hidden link to another object, called its *prototype*
- JavaScript looks at prototype if can't find a property
  - If still not there, looks to prototype's prototype
  - ... and so on until there is no prototype
- Objects therefore have two kinds of properties:
  - *Own* properties
  - *Inherited* properties
- Prototypes are attached to objects at creation time with `Object.create()`

# Object Prototype Example 1

```
var protoCircle = {x: 0, y: 0, radius: 1, color: "black"};
var c1 = Object.create(protoCircle);
c1.x = 4;
c1.color = "green"; // Now c1.y === 0 and c1.radius === 1
```



# Object Prototype Example 2

Define a prototype guitar and three guitars based on that prototype as follows:

- **prototype**
  - six-stringed, steel-string, right-handed, acoustic, mahogany.
- **first guitar**
  - six-stringed, steel-string, left-handed, electric, mahogany, Fender.
- **second guitar**
  - twelve-stringed, steel-string, right-handed, electric, mahogany, 1953 Les Paul signed by Billie Joe Armstrong.
- **third guitar**
  - six-string, nylon-string, right-handed, acoustic, 1976 Gibson Explorer Limited Edition of unknown composition, owned by The Edge.

# Object Prototype Example 2 (Cont'd)

```
// Prototype: six-stringed, steel-
// string, right-handed, acoustic,
// mahogany
var plainGuitar = {
  strings: 6,
  stringType: "steel",
  hand: "right",
  sound: "acoustic",
  construction: "mahogany"
};
```

```
var g1 = Object.create(plainGuitar);
g1.hand = "left";
g1.sound = "electric";
g1.make = "Fender";
```

```
var g2 = Object.create(plainGuitar);
g2.strings = 12;
g2.sound = "electric";
g2.year = 1953;
g2.make = "Les Paul";
g2.signedBy = "Billie Joe Armstrong";
```

```
var g3 = Object.create(plainGuitar);
g3.stringType = "nylon";
g3.year = 1976;
g3.make = "Gibson";
g3.model = "Explorer Ltd Edition";
g3.construction = undefined;
g3.owner = "The Edge";
```

# Object Prototype Example 3

```
var p = {  
  b: 3,  
  c: 4  
};  
var r = Object.create(p);  
r.a = 1;  
r.b = 2;  
  
console.log(r.a);  
/* Is there an 'a' own property on r? Yes, and its value is 1. */  
  
console.log(r.b);  
/* Is there a 'b' own property on r? Yes, and its value is 2. The  
prototype also has a 'b' property, but it's not visited. This is  
called "property shadowing". */  
  
console.log(r.c);  
/* Is there a 'c' own property on r? No, check its prototype. Is  
there a 'c' own property on p? Yes, its value is 4. */
```

# Keyword *this* in Object Prototypes

```
var o = {  
  a: 2,  
  m: function(b){  
    return this.a + 1;  
  }  
};  
  
console.log(o.m()); // 3  
// When calling o.m in this case, 'this' refers to o  
  
var p = Object.create(o);  
// p is an object that inherits from o  
  
p.a = 4; // creates an own property 'a' on p  
console.log(p.m()); // 5  
// when p.m is called, 'this' refers to p.  
// So when p inherits the function m of o,  
// 'this.a' means p.a, the own property 'a' of p
```

# Creating Objects with “new”

```
function Person(name, age, sex) {  
    this.name = name;  
    this.age = age;  
    this.sex = sex;  
}
```

```
var george = new Person("George Smith", 33, "M");  
var ken = new Person("Ken Jones", 39, "M");
```

- New operator creates a new object and calls the constructor function to initialize the fields.
- The keyword *this* inside the constructor function points to the newly created object.

# Creating Objects with “new”

```
function Car(make, model, year, owner) {  
    this.make = make;  
    this.model = model;  
    this.year = year;  
    this.owner = owner;  
}  
var car1 = new Car("Toyota", "Camry", 1996, george);  
var car2 = new Car("Nissan", "Altima", 2016, ken);
```

- Instead of passing a literal string or integer value when creating the new objects, the above statements pass the objects *george* and *ken* as the parameters for the owners.
- To find out the name of the owner of *car2*, access the following property:

*car2.owner.name*

# Main Point

3. Objects are another widely used encapsulation mechanism in JavaScript. They are easily created with object literals. They can dynamically add new properties; behave like associative arrays; must use 'this' to refer to properties; and have a prototype property that provides class-like functionality. **Science of Consciousness:** All objects in the universe are excitations of the field of pure consciousness.