ศูนย์ฝึกอบรมเทคโนโลยีสารสนเทศและการสื่อสารทหาร

เอกสารประกอบการเรียนวิชา

การสร้างเว็บไซต์ให้รองรับทุกหน้าจอด้วย Meteor Framework

กองฝึกอบรม
กรมการสื่อสารทหาร
กองบัญชาการกองทัพไทย

http://comlec.rtarf.mi.th/micttc/index.php

การพัฒนาเว็บไซต์ให้รองรับทุกหน้าจอด้วย

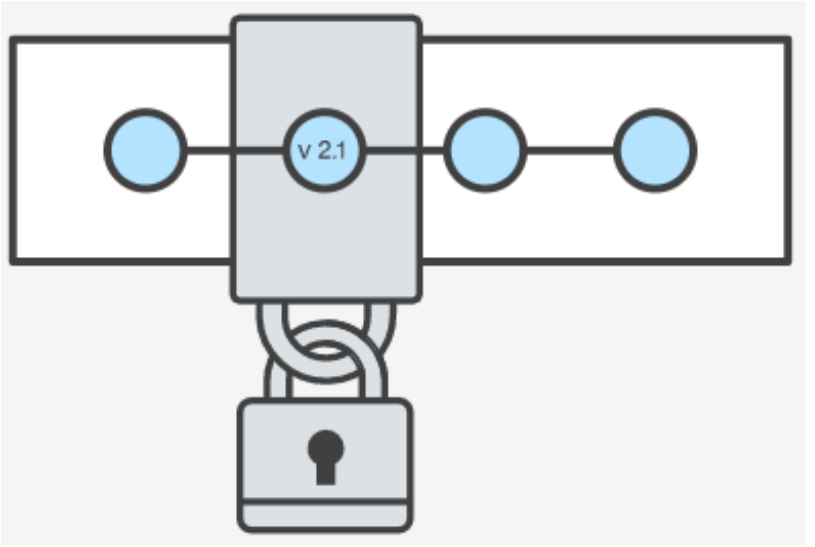
Meteor Framework

November 17-30, 2017

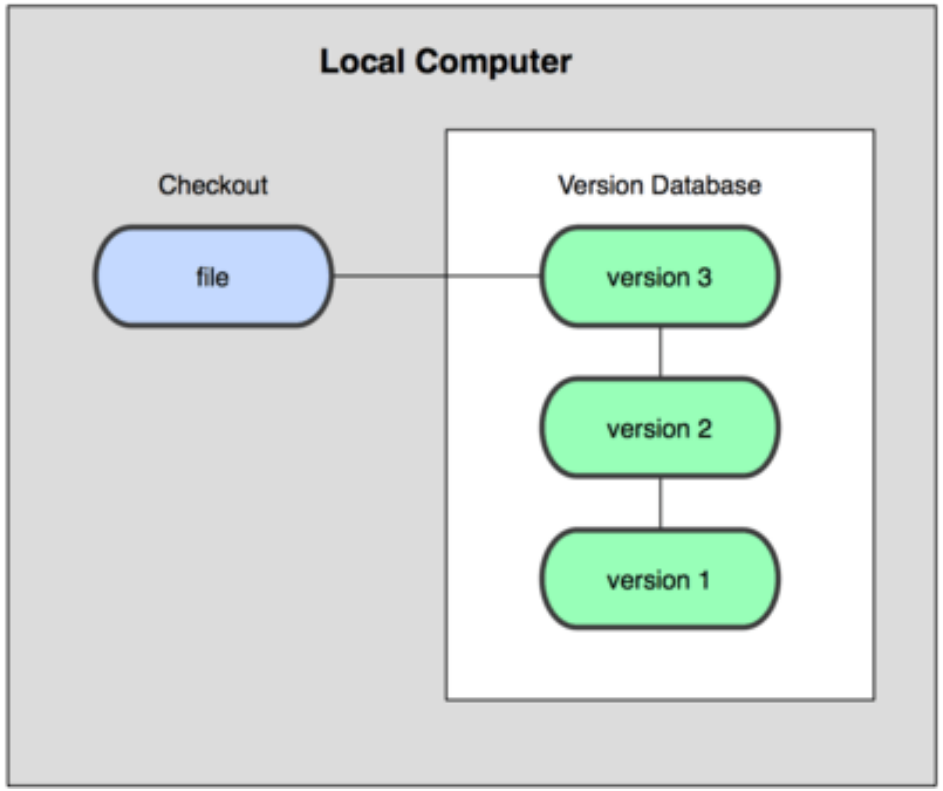
Tools && Framework

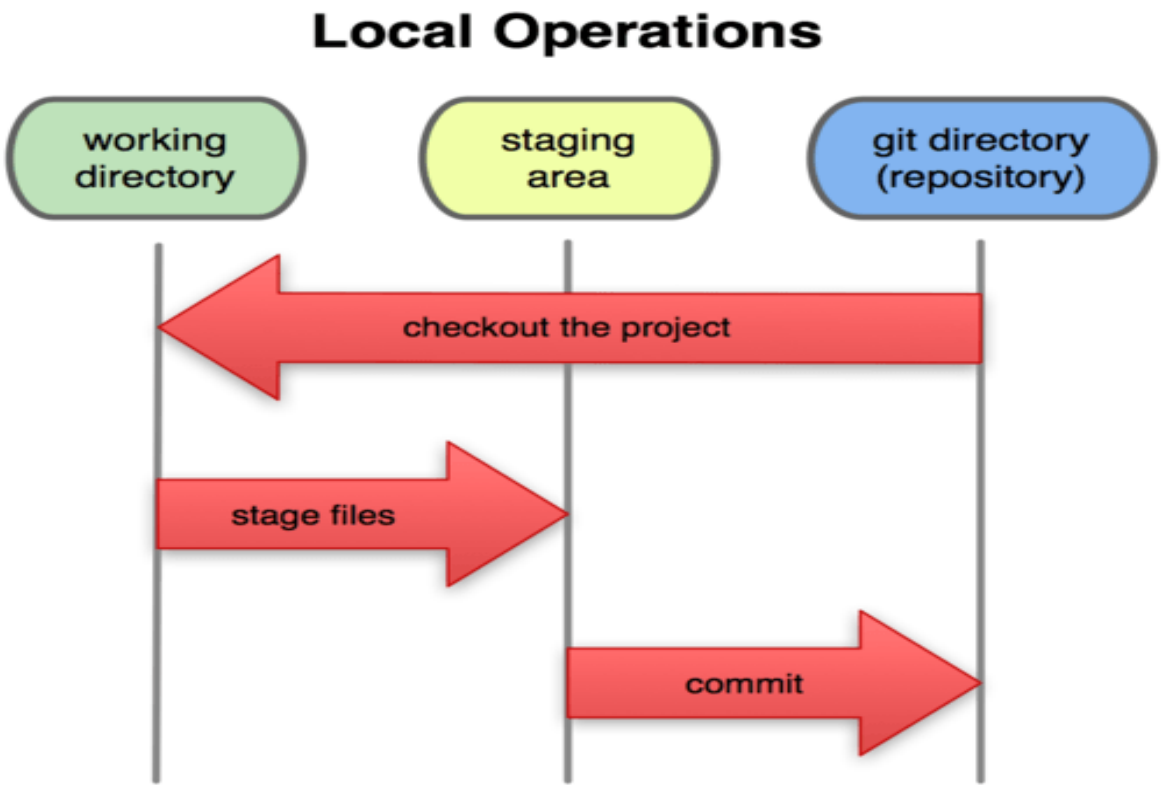
git

Is Version Control.

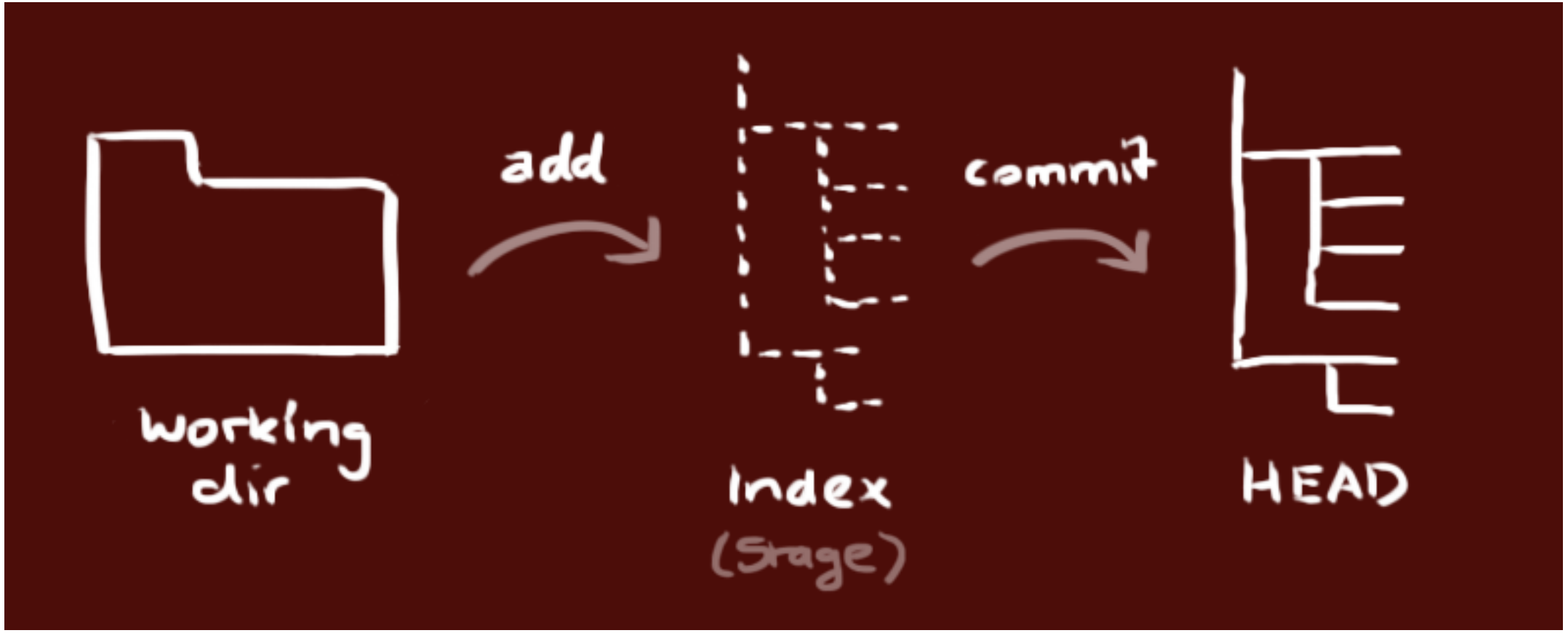


Version Control Systems แบบ Local





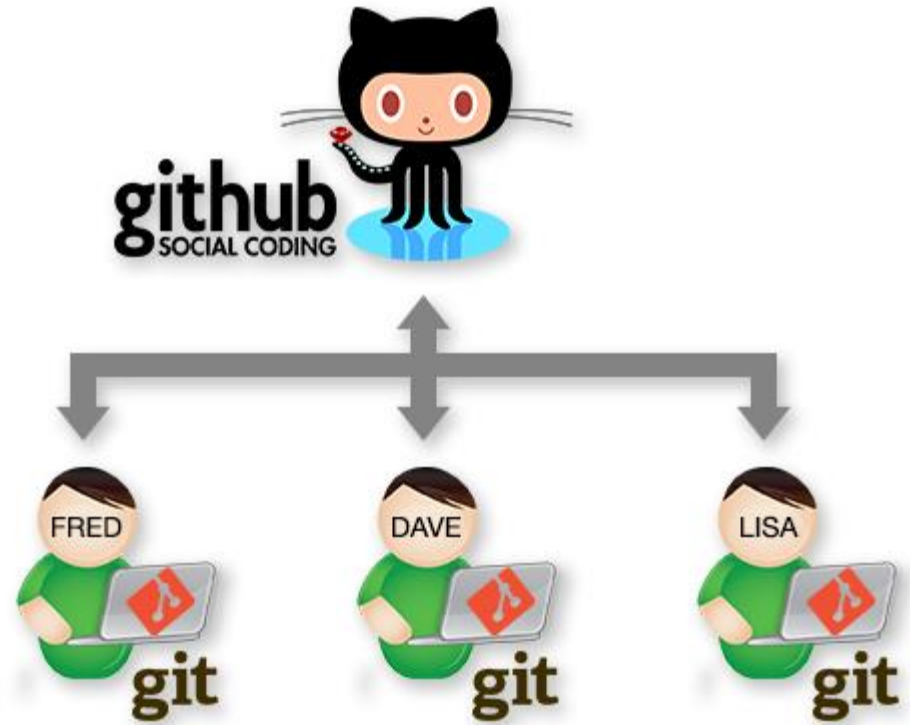
workflow

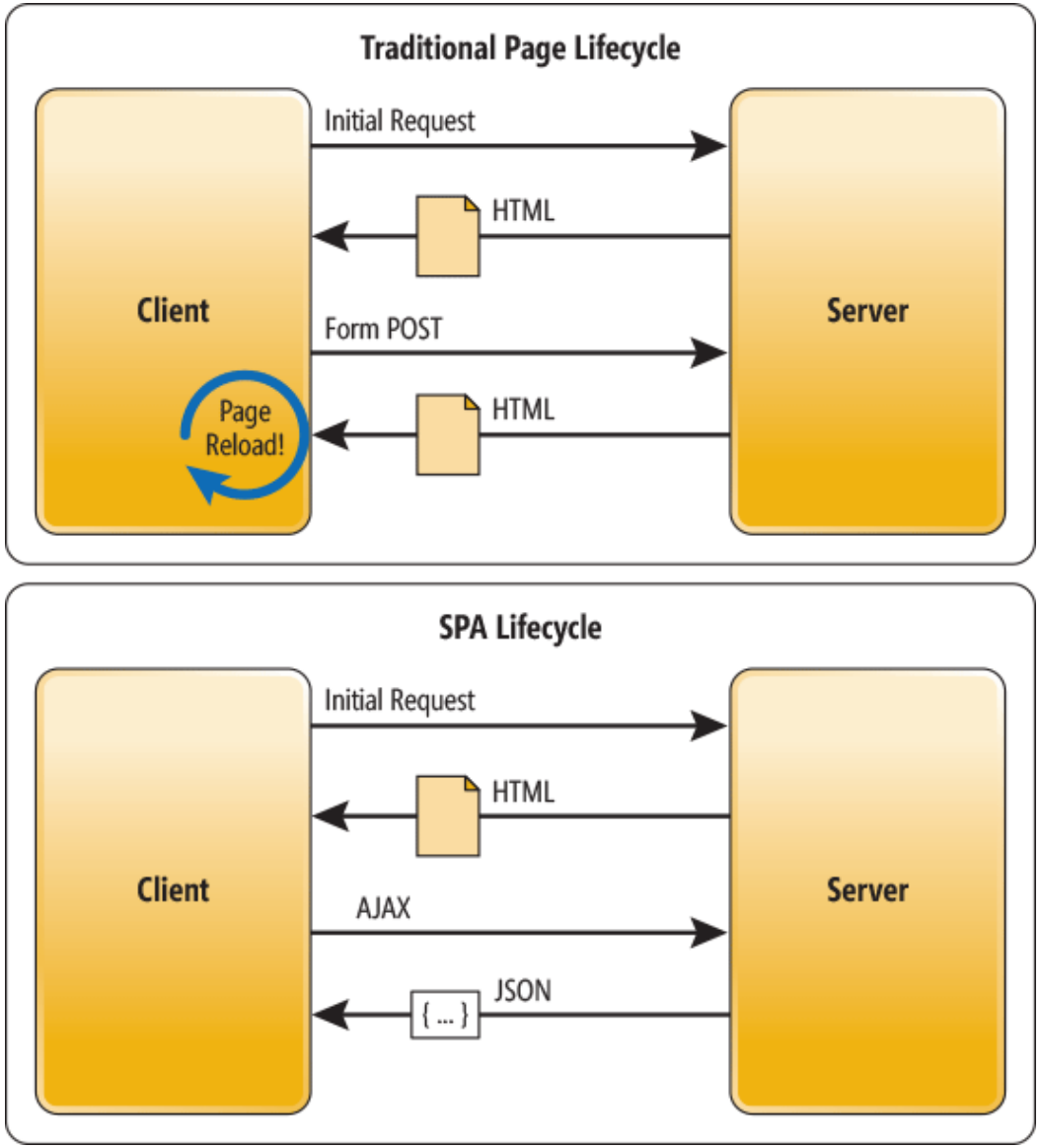


GitHub

is a Web-based Git Version Control.


git
github
SOCIAL CODING


github
SOCIAL CODING
FRED
DAVE
LISA
git
git
git


Traditional Page Lifecycle
Client
Server
Initial Request
HTML
Form POST
HTML
Page Reload!
SPA Lifecycle
Client
Server
Initial Request
HTML
AJAX
JSON
{ ... }

Web Evolution

Web 1.0 Readable

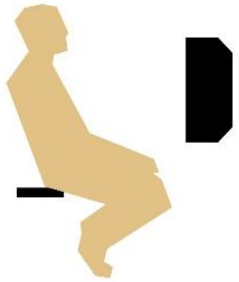
Web 2.0 Writable

Web 3.0 Semantics

Web 4.0 Symbiotic

Web 5.0 Emotional

MEDIA	MODUS	INVOLVEMENT
1.0	LEAN BACK	



WEB OF CONTENT
ATTENTION

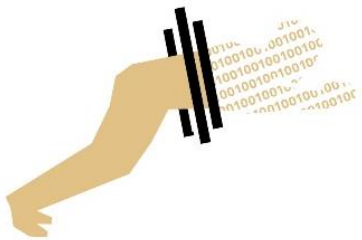
- INTERNET
- TV
- PRINT
- OUTDOOR
- E-MAIL
- SEARCH
- RADIO

MEDIA	MODUS	INVOLVEMENT
2.0	MOVE FORWARD	

WEB OF COMMUNICATION
EMPOWERMENT

- WEB 2.0
- UGC / VIDEOS
- WEBLOG
- INTERACTIVE OOH
- INSTANT MESSAGING
- CROWDSOURCING
- PODCAST
- SOCIAL NETWORKS
- WIKI
- VIRALS
- SOCIAL COMMERCE
- AUCTIONS
- WIDGETS

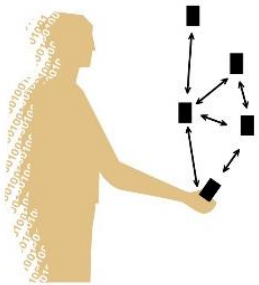
MEDIA	MODUS	INVOLVEMENT
3.0	JUMP IN	...TEMPORARY...



WEB OF CONTEXT
IMMERSION

- WEB 3D
- CINEMATIC GAMES
- SEMANTIC WEB
- HOLO ENTERTAINMENT
- 3D CHAT
- SMART SEARCH
- HOLO TV & GAMING
- VIRTUAL WORLDS
- SMART AV EXPLORATION
- MMORPG
- VIRTUAL SHOPPING
- SMART ADVERTISING
- CONSOLE
- GAMING
- GESTURE CONTROL

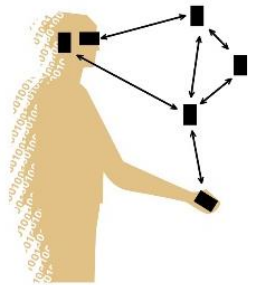
MEDIA	MODUS	INVOLVEMENT
4.0	ALWAYS ON	...PERMANENT...



WEB OF THINGS
CONNECTIVITY

- OUTERNET
- AUGMENTED REALITY
- PRINT PLUS
- OUTDOOR INWORLD
- HYPERLOCALITY
- GEOSPATIAL WEB
- CONNECTED SPACE
- AUGMENTED GOOGLES
- AI AGENTS
- ULTRA LOCAL NETWORKS
- CLOUD COMPUTING
- VOICE PROCESSING
- WEARABLE TECHNOLOGY
- MOBILE COMMUNITY
- TACTILE INTERFACE
- FACE RECOGNITION
- OBJECT RECOGNITION
- SHY TECH

MEDIA	MODUS	INVOLVEMENT
5.0	PLUG IN	...IMPLANTED...



WEB OF THOUGHTS
EXTENSION

- BRAIN COMPUTER INTERFACE
- TOTAL IMMERSION
- ACTIVE CONTACT LENSE
- AUGMENTED VISION
- TELEPATHY
- NEURO WEB INTERFACES
- 5 SENSE IMMERSION
- CLAYTRONICS
- ARTIFICAL BRAIN
- REAL WORLD AVATARS
- NEUROBOTS
- NATURAL LANGUAGE PROCESSING
- NEURO ENHANCEMENT
- EYECHIP
- IMPLANTS
- HUMAN TECHNOLOGY CONVERGENCE
- FULLBODY PROSTHESIS
- VOICELESS COMMUNICATION
- BRAINWAVE CONTROL
- HUMAN 2.0

Request

Verb URI HTTP Version

```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

Request Line

Request Headers

Request Message Header

A blank line separates header & body

Request Message Body

HTTP Version Status Code

```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

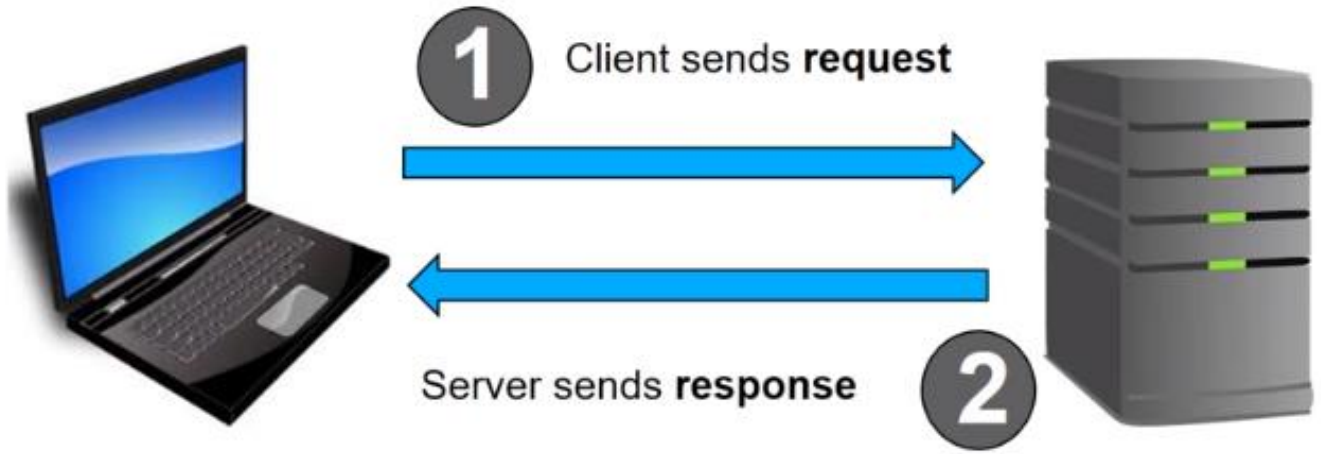
Status Line

Response Headers

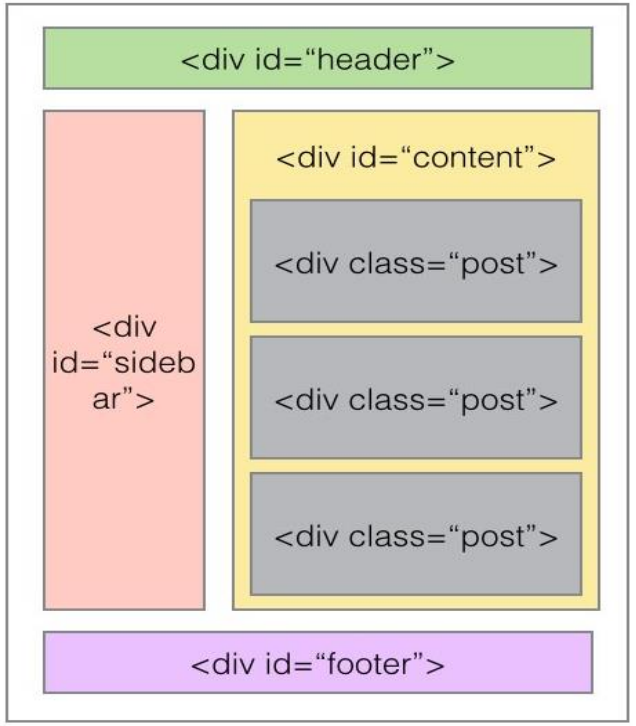
Response Message Header

A blank line separates header & body

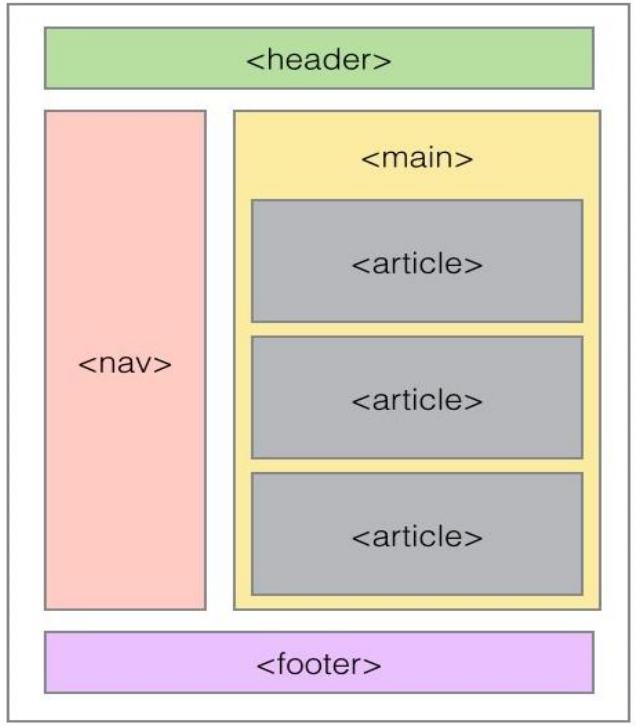
Response Message Body

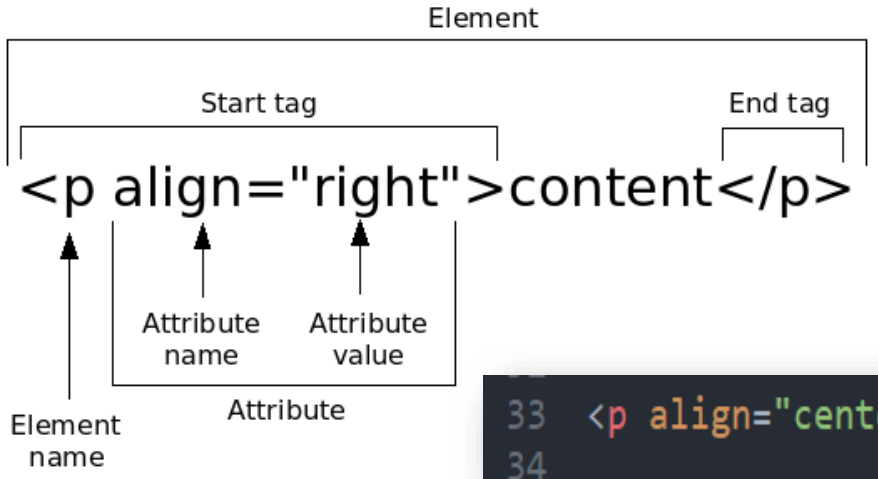


HTML4: Lots of Classes/IDs



HTML5: Semantic Tags/Sections






```
33  <p align="center"> Text Position Testing  </p>
34
```

"Hyper Text Markup Language"

The standard markup language for creating Web pages.

```
<!DOCTYPE html>

<html>
    <head>
        <title>
            My First Web Page
        </title>
    </head>

    <body>
        Hello, World!
    </body>
</html>
```

Important HTML Tags: <head>

- Contains information about the document, not content
- Common elements included within <head></head>:
  - <title> - contains page title, displayed in browser's title bar
  - <link> – used to add CSS stylesheets and icons to page
  - <meta> – used to specify metadata like page descriptions and keywords
  - <script> – used to add JavaScript code to the page

```
<!DOCTYPE>
<html>
<head>
  <title>My Web Page 2.0</title>
  <link rel="stylesheet" type="text/css" href="/style.css">
  <meta name="description" content="Learning about HTML.">
  <meta name="keywords" content="html, web development">
  <script src="code.js"></script>
</head>
</html>
```

Important HTML Tags: <body>

- Appears directly beneath the head element
- Contains all web page content (images, text, etc.)
- Most web pages have one single body element

```
<!DOCTYPE html>
<html>
<head>
  . . .
</head>

<body>
  Hello world!
</body>

</html>
```

Bootstrap

Build responsive, mobile-first projects on the web with the world's most popular front-end component library.

```
1 <!DOCTYPE html>

  <html>

    <head>
3   <meta charset="utf-8">   4
    <title>Title here</title>   5
    </head>
2
    <body>
6   Page content goes here.
    </body>

  </html>
```

HTML Content

- Remember, the HTML specifies the **structure** but not how the content will be **displayed**
- It is up to the browser to decide how to display the content

```
<p>
This is some text.

This is some more text.
And here's a little bit more.
</p>
```

This is some text. This is some more text. And here's a little bit more.

Review: HTML Tags

- **<!DOCTYPE>** specifies the version of HTML
- **<html>** root of the entire document
- **<head>** section that provides the page title, meta information, includes other files, etc.
- **<body>** the actual content

Important HTML Tags: <p>

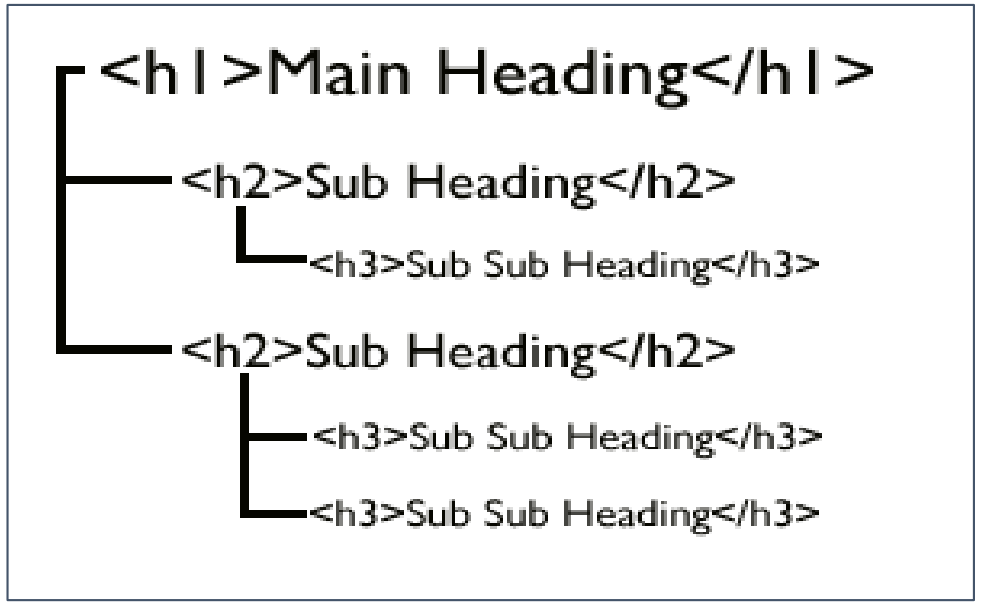
- Appears anywhere within the body to represent a paragraph of text expressing a single thought
- Usually displayed with vertical space before and after paragraph

```
<!DOCTYPE html>
<head> . . . </head>

<body>
  (all web content goes here!)
  <p> This is a paragraph within the p tag that expresses a single thought or
      idea. The paragraph should be surrounded with a vertical white space
      buffer both before and after the paragraph. </p>
</body>

</html>
```

Important HTML Tags: <h#>

- Referred to as section heading tags
- HTML supports 6 heading tags
  - <h1> used for most important titles (ex: title of entire web page)
  - <h2> used for next important subheader
  - <h6> used for least important subheader
- Magnitude of importance affects font size that will be displayed
- Can be used to divide content into readable subsections
- Browser determines font and size depending on header magnitude

h1
h2
h3
h4
h5
h6




```
<body>
  <h1>This is the most important header</h1>
  <p>This is a paragraph supporting the most important header.</p>
  <h3>This is a subheader</h3>
  <p>This is a paragraph supporting the subheader. Notice that the font
     size of the subheader is smaller than the size of the most important
     header. </p>
</body>
```

This is the most important header

This is a paragraph supporting the most important header.

This is a subheader

This is a paragraph supporting the subheader. Notice that the font size of the subheader is smaller than the size of the most important header.

Important HTML Tags: <b> and <i>

- **<b>** indicates that the text should be **bold**
- **<i>** indicates that the text should be *italicized*

- Similar tags are **** and ****, respectively, which are meant to demonstrate that the text is "important"

```
<!DOCTYPE html>
<html>
<body>
<p>He named his car <b>ทดสอบตัวหนังสือหนา (Bold)</b>, because it was very fast.</p>
</body>
</html>
```

He named his car **ทดสอบตัวหนังสือหนา (Bold)**, because it was very fast.

```
<!DOCTYPE html>
<html>
<body>
<p>He named his car <i>ทดสอบตัวหนังสือเอียง (Italic)</i>, because it was very fast.</p>
</body>
</html>
```

He named his car *ทดสอบตัวหนังสือเอียง (Italic)*, because it was very fast.

Important HTML Tags: <hr> and

- Both tags used to mark a break in content
  - **<hr>** Represents a more serious shift in content, visually separates content by inserting a visible line between preceding and subsequent content
  - **
** Represents a single line break, inserts a blank line

```
<p> This is a paragraph about
how your browser works. </p>
<hr>
<p> This is a paragraph about
how to set up your environment
to begin creating HTML
documents. </p>
<br>
<p> This is a paragraph about
  important HTML tags that
  you should know. </p>
```

This is a paragraph about how your browser works.

This is a paragraph about how to set up your environment to begin creating HTML documents.

This is a paragraph about important HTML tags that you should know.

Important HTML Tags:

- Inline container for organized content
- Similar to div but different in the following ways;
  - Block-level elements (div) are designed to contain larger chunks of content designed to stand alone as a unit; always starts with a new line
  - Inline elements (span) designed to contain smaller pieces of content, usually within a larger block of content; does not start with new line

```
<p>Today we've learned about
  <span>DOCTYPE declarations</span>,
  <span>head tags</span>,
  <span> body tags</span>,
  <span>heading tags</span>,
  <span>paragraph tags</span>, and
  <span>Inline vs. Block-level
        containers</span>.
</p>
```

Important HTML Tags: <!-- *Comments* -->

- Text in the HTML that will not be rendered in the browser
- Often used for:
  - Explaining the HTML to or leaving notes for other programmers
  - Temporarily removing HTML content

```
<!-- This is a listing of some important people -->
<br>Eliana
<br>Swapneel
<!-- <br>Chris -->
<br>Lydia
```

Important HTML Tags:

- Provides additional structure to web page
- Block-level container for organized content
- Often used for:
  - Page headers/footers
  - Menu or Navigation bar
  - Photo galleries
  - Ads or outside media

```
<div>
  <h3> Thought of the Day </h3>
  <p>Today's thought of the day: "Don't
     make excuses, make improvements"
     -Tyra Banks</p>
  <p>Additional supporting text</p>
</div>
```

Putting It All Together

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page 2.0</title>
     <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
     <meta name="description" content= "Learning about the most important web tags in html.">
     <meta name="keywords" content= "html, web development">
  </head>
  <body>
     <h1>This is the most important header</h1>
     <p>This is a paragraph supporting the most important header.</p>
     <h3>This is a subheader</h3>
     <p>This is a paragraph supporting the subheader. Notice that the font size of the subheader
      is smaller than the size of the most important header. </p>
     <hr>
     <div>
       <h3> Thought of the Day </h3>
       <p>Today's thought of the day: <em>"Don't make excuses, make improvements"></em>
       --Tyra Banks</p>
       <p>Additional supporting text</p>
     </div>
     <hr>
    <p>Today we've learned about <span>DOCTYPE declarations</span>, <span>head tags</span>, <span>
         body tags</span>, <span>heading tags</span>, <span>paragraph tags</span>, &amp;amp; <span>
       Inline vs. Block-level containers</span>.
     </p>
  </body>
</html>
```

This is the most important header

This is a paragraph supporting the most important header.

This is a subheader

This is a paragraph supporting the subheader. Notice that the font size of the subheader is smaller than the size of the most important header.

Thought of the Day

Today's thought of the day: *"Don't make excuses, make improvements"*> --Tyra Banks

Additional supporting text

Today we've learned about DOCTYPE declarations, head tags, body tags, heading tags, paragraph tags, & Inline vs. Block-level containers.

HTML Symbols & Special Characters

HTML Entity	Appearance
` `	Non-breaking space; allows for extra white space between words
`<`	<
`>`	>
`&`	&
`&copy;`	©
`&reg;`	®

HTML Attributes

- Purpose of an attribute: provide additional information about a particular HTML element
- Always included within element's start tag
- Usually comes in name/value pair as follows: ***name="value"***
  - *name* – usually specifies the property of the element for which additional information is being provided
  - *value* –this is selected from set of possible values for given property

```
<p name="value"> This is a paragraph with a defined attribute </p>
```

Review: HTML tags

- **<p>** paragraph
- **<h#>** header
- **<b>** bold (also ****)
- **<i>** italics (also ****)
- **<hr>** horizontal rule
- **
** break
- ** ** block of text
- **** small section of text

Core Attributes: `title`

- Provides a suggested title for an element
- When user hovers over the element, a “tooltip” will appear at the cursor with the suggested title

```
<h1 title="Welcome!"> Test Title </h1>
```

Core Attributes: `style`

- Used to change the visual presentation of an element
- Value string will be include multiple *key:value* pairs as well (separated by semicolons)

```
<h1>this is a heading</h1>
```


```
<h1 style="color:red; text-transform:capitalize">this is a heading</h1>
```

```
<p style="color:blue">1 is an odd number</p>
<p style="color:white; background-color:blue">2 is an even number</p>
<p style="color:blue">3 is an odd number</p>
<p style="color:white; background-color:blue">4 is an even number</p>
<p style="color:blue">5 is an odd number</p>
<p style="color:white; background-color:blue">6 is an even number</p>
```

1 is an odd number

2 is an even number

3 is an odd number

4 is an even number

5 is an odd number

6 is an even number

Other properties of "style" attribute

Property	Description	Sample values
background-color	The color that appears behind the text	red, yellow, #012345
font-family	Font used to render the text	verdana, courier
font-size	Size of text, possibly relative to default size	12px, 4in, 200%
text-align	Horizontal text alignment	center, left, right

```
Hello <span style="color:white; background-color:green;
font-family:verdana; font-size:200%">world!</span>
```

Hello world!

Core Attributes: `id` and `class`

- Used to uniquely identify elements within an HTML document

- **`id`**
  - Provide ability to refer to specific element; id must be unique
  - Examples: header, footer

- **`class`**
  - Provide ability to refer to subgroups of elements within html document; does not have to be unique
  - Examples: comment, warning

Summary

- You can use a tag's **attributes** to give it properties

- The **`style`** attribute allows you to change the appearance of the text within that element

- The **`class`** attribute allows you to group elements so that they can easily have the same style applied

```
<p class="odd">1 is an odd number</p>

<p class="even">2 is an even number</p>

<p class="odd">3 is an odd number</p>

<p class="even">4 is an even number</p>

<p class="odd">5 is an odd number</p>

<p class="even">6 is an even number</p>
```

1 is an odd number

2 is an even number

3 is an odd number

4 is an even number

5 is an odd number

6 is an even number

```
<!DOCTYPE html>
<html>

<head>
<style>
.odd {
    color: blue;
}
.even {
    color: white;
    background-color: blue;
}
</style>
</head>

<body>
<p class="odd">1 is an odd number</p>
<p class="even">2 is an even number</p>
<p class="odd">3 is an odd number</p>
<p class="even">4 is an even number</p>
<p class="odd">5 is an odd number</p>
<p class="even">6 is an even number</p>
</body>

</html>
```

Lists in HTML: <ol> and

- Lists can either be ordered or unordered

**Ordered List: **

1. January
2. February
3. March

```
<ol>
   <li> January </li>
   <li> February </li>
   <li> March </li>
</ol>
```

**Unordered List: **

- Eliana
- Chris
- Swapneel

```
<ul>
   <li> Eliana </li>
   <li> Chris </li>
   <li> Swapneel </li>
</ul>
```

Unordered Lists - Style

- Use CSS style properties to determine the style of bullet point used within the list as follows (inline example)

```
<ul style="list-style-type:circle">
   <li> Eliana </li>
   <li> Chris </li>
   <li> Swapneel </li>
</ul>
```

Unordered List

- Eliana
- Chris
- Swapneel

list-style-type:value	Bullet View
disc	• Disc is the default value
circle	○ This is the circle view
square	▪ This is the square view
none	No bullets, just list items

Nested Lists

- List items can themselves include lists as well to produce a nested list effect

```
<ul style="list-style-type:circle">
   <li> Eliana </li>
   <li> Chris </li>
      <ol type="A">
         <li> SD2x </li>
         <li> SD4x </li>
      </ol>
   <li> Swapneel </li>
</ul>
```

- Eliana
- Chris
  A. SD2x
  B. SD4x
- Swapneel

Ordered Lists - Style

- Use the *type* attribute of <ol> tag to select numbering of each item

```
<ol type="A" >
   <li> January</li>
   <li> February</li>
   <li> March</li>
</ol>
```

Ordered List

A. January
B. February
C. March

type	View
type="1"	1. This is the default setting 2. for ordered lists
type="A"	A. Items ordered with B. Uppercase letters
type="a"	a. Items ordered with b. Lowercase letters
type="I"	I. Items ordered with II. Uppercase Roman Numerals
type="i"	i. Items ordered with ii. Lowercase Roman Numerals

Forms

- Forms are used to retrieve information from user of a Web page
- Enclose all form fields within **<form>** elements

Forms: <input type="value">

```
<form>
  Full name:<br>
   <input type="text" name="fullname" value="Jane Doe">
  <br>
  Email Address:<br>
   <input type="email" name="email"
    value="jdoe@example.com">
  <br><br>
  <input type="submit" value="Submit">
</form>
```

Full name:
Jane Doe
Email Address:
jdoe@example.com
Submit

Forms – Text Inputs

Full Name:
Email Address:
Password:
Date of Birth: mm/dd/yyyy

```
<br> Full Name:
<input type="text" name="username">

<br> Email Address:
<input type="email" name="email">

<br> Password:
<input type="password" name="password">

<br> Date of Birth:
<input type="date" name="dob">
```

Forms – Checkboxes

Race: African American Asian White American Indian Pacific Islander Other

```
<br> Race:

<input type="checkbox" name="race" value="afrAmer"> African American

<input type="checkbox" name="race" value="asian"> Asian

<input type="checkbox" name="race" value="white"> White

<input type="checkbox" name="race" value="amIndian"> American Indian

<input type="checkbox" name="race" value="pacIsl"> Pacific Islander

<input type="checkbox" name="race" value="other"> Other
```

Forms – Radio Buttons

Gender: Male Female Undisclosed

```
<br> Gender:

<input type="radio"
       name="genderOption"
       value="male"> Male

<input type="radio"
       name="genderOption"
       value="female"> Female

<input type="radio"
       name="genderOption"
       value="undisclosed"> Undisclosed
```

Forms – Buttons

Submit Reset

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

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

Application Form

Full Name:
Email Address:
Password:
Date of Birth: mm/dd/yyyy
Graduation Year:
Phone Number:

Demographics

Gender: Male Female Undisclosed

Race: African American Asian White American Indian Pacific Islander Other

Personal Preferences:

Favorite Color:

Resume Submission: Choose File No file chosen

Submit Reset

Forms – Slider/Range Input

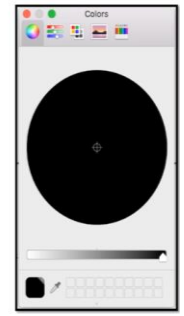
Graduation Year:

```
<br> Graduation Year:
<input type="range" name="gradYr" min="1950" max="2020">
```

Forms – Color Chooser

Favorite Color:

```
<br> Favorite Color:
<input type="color" name="favcolor">
```



Forms – File Uploads

Resume Submission: Choose File No file chosen

```
<br> Resume Submission:
<input type="file" name="resume">
```

Forms: Input Types

Type	Form Field
button	Clickable button
checkbox	Checkbox options
color	Color picker
date	Date picker (year, month, day)
email	Email address field
file	File select/browse for file uploading
hidden	Hidden input field
image	Allow image to serve as submit button
month	Month and year control
time	Time control

Type	Form Field
number	Number format entry
password	Masked characters for password entry
radio	Radio button options
range	Slider control to enter number as a range
reset	Reset all form values to default values (button)
search	Text field for searching
submit	Submit button
tel	Phone number input
text	Plaintext field
url	Field for URL

Summary

- HTML **lists** allow us to organize information in an HTML page and specify the appearance
- HTML **forms** allow us to accept input from the user

What is CSS?

- **Cascading Style Sheets** (CSS) are a formatting language used to describe the appearance of content in an HTML file

- CSS has a standardized specification defined by the World Wide Web Consortium (W3C)

- You can use an HTML tag's **attributes** to give it properties

- The **`style`** attribute allows you to change the appearance of the text within that element

- The **`class`** attribute allows you to group elements so that they can easily have the same style applied

Plain HTML

Here are some memorable quotes from movies!

You can find more at the Internet Movie Database (IMDb).

You killed my father. Prepare to die.

Inigo Montoya in *The Princess Bride*

I've never been to this part of the castle. Well, not awake. I sleepwalk, you see. That's why I wear shoes to bed.

Luna Lovegood in *Harry Potter and the Half-Blood Prince*

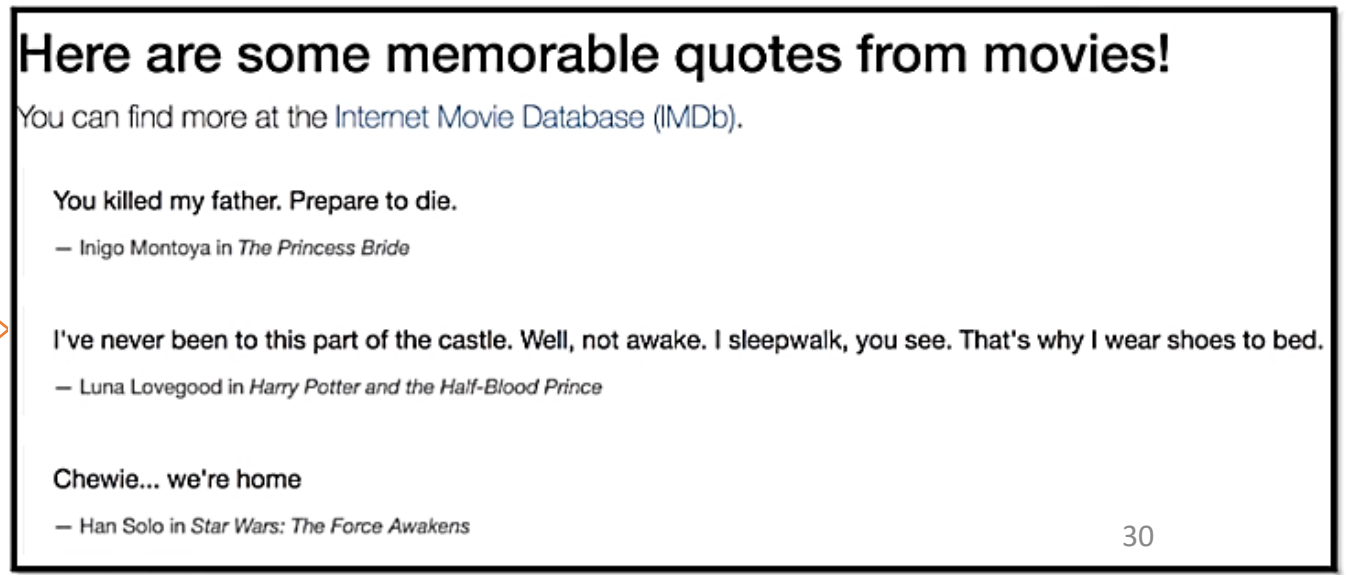
Chewie... we're home

Han Solo in *Star Wars: The Force Awakens*

Stylish HTML

Why CSS?

- **HTML ("The Content")**
  - What information does the page contain?
  - What is in the headings, body, etc.?
  - How is the information structured?
- **CSS ("The Presentation")**
  - What does the page look like?
  - What *color, formatting, text size, etc.* should the various parts have?



How does CSS work?

1. The Web Browser receives the HTML page from the server via HTTP
2. The HTML page can include CSS either in same file or with link to separate file
   - If it's a separate file, the web browser will request that file separately via HTTP
3. When all HTML and CSS files are available, the browser will render the page
4. For each element in the HTML page, the web browser will display the content and use the CSS to style it
5. Ideally, exactly one set of CSS styles will apply to any given element
6. If there are conflicting styles defined, complex rules determine which gets applied

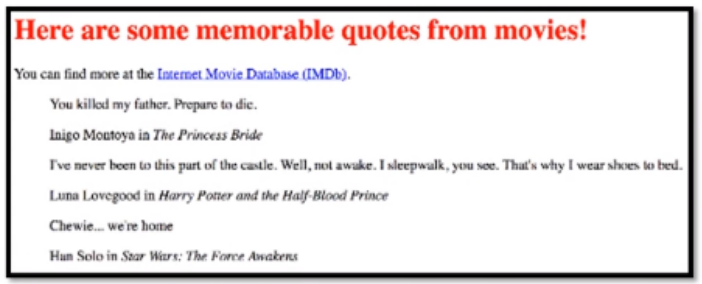
How do we use CSS?

1. **Inline:** use tag's "style" attribute to specify appearance
2. **Internal:** create `<style>` elements in HTML and assign to different tags, classes, etc.
3. **External:** specify styling in a separate CSS file

Review: Inline CSS as “style” attribute

- Include CSS within the element tag itself as a “style” attribute

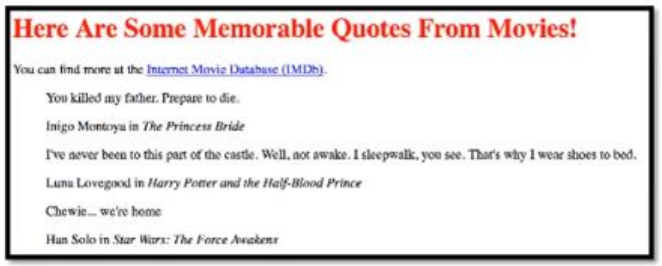
```
...
<h1 style="color:red">
  Here are some memorable
    quotes from movies!
</h1>
...
```



Review: Internal CSS using <style> tag

- Include CSS within the head of the HTML using <style> elements

```
<head>
...
  <style>
    h1 {
      color: red;
      text-transform: capitalize;
    }
  </style>
...
</head>
...
<h1>
  Here are some memorable
    quotes from movies!
</h1>
```



External CSS

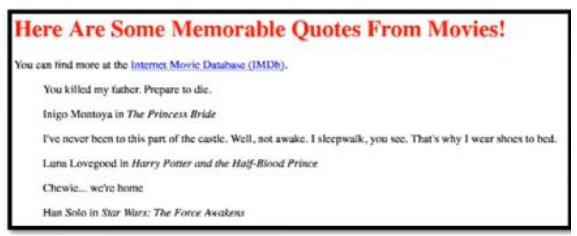
- Include CSS in a separate file and link the file in the head of the HTML file

```
<head>
...
<link rel="stylesheet"
      type="text/css"
      href="movie-styles.css" />
...
</head>
...
<h1>
  Here are some memorable
    quotes from movies!
</h1>
```

movies.html

```
h1 {
  color: red;
  text-transform: capitalize;
}
```

movie-styles.css



CSS Selectors

Type of Selector	What's in the CSS File?	What does this selector apply to?	What does the HTML file contain?
Element Selector	`h1 { color:red; }`	All <h1> elements	`<h1> ... </h1>`
Class selector	`.address { ... }`	All elements in class “address”	`<div class=“address”> ... </div>`
Id Selector	`#section1 { ... }`	Unique element with ID “section1”	`<p id=“section1”> ... </p>`

Summary

- **Cascading Style Sheets** (CSS) are a formatting language used to describe the appearance of content in an HTML file

- We can include CSS formatting in HTML in three ways:
  1. inline as style attributes
  2. internal using the `<style>` tag
  3. external in a separate file linked with the `<link>` tag

- A CSS element consists of a selector and property:value pairs

JavaScript

The programming language of HTML and the Web

JavaScript Basics

- Like many other programming languages, JavaScript includes:
  - variables, arrays, and objects
  - loops and conditional statements
  - functions
- Even if you know Java, there are still some important differences
  - defining functions and objects
  - interacting with HTML

Declaring a Variable

- The basic syntax for declaring any JavaScript variable is `var variableName = ...`

```
var age = 22;

var name = 'Jane Doe';

var isMale = false;
```

Viewing a variable's value (1)

- If using a `<script>` section in a HTML file, or an external .js file, **`document.write(var)`** will display a variable's value in the HTML

```
My age is:
<script>
   var age = 12;
   document.write(age);
</script>
```

My age is: 12

- However, this approach is discouraged
- We will see better alternatives later!

Viewing a variable's value (2)

- You can also use **`console.log(var)`** to print a variable's value in the browser's JavaScript console

```
<script>
   var age = 12;
   console.log(age);
</script>
```

Chrome=Ctrl+Shift+J

IE=F12

FireFox=Ctrl+Shift+K

Viewing a variable's value (3)

- Also, **`alert(var)`** will create a popup with the variable's value that appears on top of the browser

```
<script>
  var age = 12;
  alert(age);
</script>
```

- Last, if using the browser JavaScript console (REPL), just type the name of the variable

```
> var age = 12
> age
12
```

Changing a variable's type

- The type of each variable does not need to be specified and can be changed at any time.

```
var id = 33.2;

id = 'secret';
```

Primitive Types

Type	Example values
Number	5, 1.25, 1.1e5, +Infinity, -Infinity, NaN
String	'hello'
Boolean	true, false
Null	null
Undefined	undefined

Numbers

- All JavaScript numbers are stored using floating-point notation
  - i.e. 5 is stored internally as 0.5e1
- `+infinity` represents all numbers greater than `Number.MAX_VALUE` (around 10^{308})
- `-infinity` represents all numbers less than `Number.MIN_VALUE` (around 10^{-324})
- `NaN` represents any non-number value
  - `Number('tree')` would return `NaN`

Number Operations

- Basic arithmetic (+, -, *, /, %) can be used on JavaScript numbers
- Precedence will follow MDAS unless parentheses are used
- ++ and -- can be used to increment/decrement JavaScript numbers

```
var a = 4;
a++;                        // a = 5
var c = a - 3;              // 2
var d = c + 3 * a;          // 17
var e = ( c + 3 ) * a;      // 25
```

Strings

- JavaScript strings are series of 16-bit unsigned integers, each integer representing a character
- Convention is to use single quotes for strings unless single quotes exist within the string
  - `'I am a dolphin'` vs. `"I'm a dolphin"`
- Escape characters use backslash: `'\n \t \\'`
- All JavaScript strings are immutable
  - Any manipulation results in a new string

String Functions

- `+` or `.concat(`*`otherString`*`)` can be used to concatenate strings (add them together)
- `.toUpperCase()` and `.toLowerCase()` change the case of every character in a string
- *`var`*`.length` gets the length of a string

```
var firstName = 'John';
var lastName = 'doe';

var fullName= firstName.concat(' ', lastName); // 'John doe'
var greeting = 'HELLO, ' + fullName;

console.log(greeting.toUpperCase());     // 'HELLO, JOHN DOE'
console.log(greeting.toLowerCase());     // 'hello, john doe'

console.log(greeting.length);            // 15
```

Removing from an Array

- Elements can be removed from arrays using **`pop()`** and **`shift()`**
  - **`pop()`** will remove and return an element from the end of the array
  - **`shift()`** will remove and return an element from the beginning

```
var myArray = ['train', 'car', 'bike', 'scooter'];

var vehicle = myArray.pop();
console.log(vehicle);             // scooter
console.log(myArray);             // train,car,bike

vehicle = myArray.shift();
console.log(vehicle);             // train
console.log(myArray);             // car,bike
```

Modifying Objects

- Key-value pairs can be added to objects, even after their initial declaration

```
var pet = {
   name: 'Cooper',
   type: 'dog'
}

console.log(pet.age);        // undefined
pet.age = 11;
console.log(pet.age);        // 11

pet['status'] = 'good boy';
console.log(pet.status);     // "good boy"
```

Summary

- JavaScript **arrays** let us create ordered collections of values with numeric indices
- JavaScript **objects** are collections of associated values with semantically meaningful names/keys

Objects

- JavaScript objects are used to store key-value pairs
- Values can be of any type, including arrays and objects!
- Values can be accessed by ***`myObject.property`*** or ***`myObject['property']`***

```
var person = {
   name: 'John Doe',
   age: 25,
   isMale: true,
   personality: ['patient', 'loyal', 'happy'],
   company: { name: 'edX', id: 2984 }
}

console.log(person.age);              // 25
console.log(person['company'].id)     // 2984
```

control structures

Conditional Statements

```
var a = . . .

var b = . . .

var max; // undefined

if (a > b) {
   max = a;
}
else {
   max = b;
}

console.log(max);
```

Comparison and Logical Operators

Comparison Operators

Operator	Description
==	equal to
===	equal to and same type
!=	not equal to
!==	not equal to or different type
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

Logical Operators

Operator	Description
||	logical OR
&&	logical AND
!	logical NOT

Double-equals vs. Triple-equals

- Use double-equals (==) when you only want to compare **values**
- Use triple-equals (===) when you want to compare values **and** type

```
1 == '1' // true

1 === '1' // false! different types
```

Comparing Truthy/Falsy Values

- Recall that any value can be used as a boolean
  - "Falsy" values: `null, undefined, 0, NaN, ''`
  - "Truthy" values: `'cow', 'false', 5`, etc...

```
var x; // undefined
if (x) { . . . } // false! undefined is falsy

x = 0;
if (x) { . . . } // false! 0 is falsy

x = 39;
if (x) { . . . } // true! 39 is truthy


var y = null;
var z; // undefined

if (y == z) { . . . } // true! falsy equals falsy

if (y === z) { . . . } // false! different types
```

Comparing Objects

- Objects are only considered equal if the variables are **aliases**, i.e. refer to the same object

```
var cooper = { age: 11 }
var flanders = { age: 11 }

if (cooper == flanders) { . . . } // false!



var myDog = cooper;

if (myDog == cooper) { . . . } // true!
```

Comparing Numbers and Strings

- When comparing a string to a number, JavaScript will try to convert the string to a numeric form

```
5 < '20' // true
'5' < 20 // true
```

- Non-numeric strings are converted to NaN

```
5 > 'alligator' // false
5 < 'alligator' // also false!
```

- Non-numeric strings are compared alphabetically

```
'zebra' > 'giraffe' // true
```

Loops

```
var n = ...
var factorial = 1;
```

```
for (var i = 1; i <= n; i++) {
   factorial *= i;
}
```

```
var i = 1;
while (i <= n) {
    factorial *= i;
    i++;
}
```

```
var i = 1;
do {
    factorial *= i;
    i++;
}
while (i <= n);
```

Function

Applying Functions to Arrays

```
var nums = [ 4, 8, 12, 2 ];
```

```
function print(n) {
   console.log(n);
}
nums.forEach(print);
```

```
function isEven(n) {
   return n % 2 == 0;
}
nums.every(isEven); // true
```

```
function square(n) {
   return n * n;
}
var squares = nums.map(square); // [ 16, 64, 144, 4 ]
```

Declaring and Using Functions

```
function factorial(n) {
   var product = 1;
   for (var i = 1; i <= n; i++) {
       product *= i;
   }
   return product;
}

var x = . . .

var f = factorial(x);

console.log(f);
```

Pass-by-Value vs. Pass-by-Reference

- Primitive arguments are passed by **value**: the function cannot change them

```
function tryToChange(x) {
   x = 4;
}
var y = 11;
tryToChange(y);
console.log(y); // still 11
```

- Object arguments are passed by **reference**: the function **can** change them

```
function changeMe(obj) {
   obj.age++;
}
var p = { age: 30 };
changeMe(p);
console.log(p.age); // now 31
```

Functions as Objects

- JavaScript functions are objects
  - Therefore, functions can take advantage of the benefits of an object, such as having properties
- Since JavaScript functions are objects, we can have variables refer to them

```
var add = function (a, b) {
   return a + b;
};

console.log(add(3, 5));          // 8
```

Functions in Objects

- JavaScript functions can also be declared and used in objects

```
var johnDoe = {
   name: 'John Doe',
   age: '32',
   greeting: function () {
      return 'Hello! Nice Meeting You!';
   }
}

console.log(johnDoe.greeting());
```

Object Prototypes

- Every object in JavaScript has a **prototype,** accessed from the `__proto__` property in the object.
- The `__proto__` property is also an object, with its own `__proto__` property, and so on
- The root prototype of all objects is `Object.prototype`
- An object inherits the properties of its prototype

Creating a Prototype

- Prototypes are created like any other JavaScript function or object
- The **`this`** keyword refers to the current object
- The **`new`** keyword can be used to create new objects from the same prototype

```
function Person (name, age) { // prototype
    this.name = name;
    this.age = age;
    this.greeting = function () {
      return 'Hello! My name is ' + this.name;
    }
}

var johnDoe = new Person('John Doe', 32);
johnDoe.greeting(); // Hello! My name is John Doe

var janeDoe = new Person('Jane Doe', 28);
janeDoe.greeting(); // Hello! My name is Jane Doe
```

Extending a Prototype

- Prototypes can extend another prototype with more functionality
- To inherit a prototype, set the __proto__ property of an object to the parent prototype

```
function Student (name, age, school) {
   this.__proto__ = new Person(name, age);
   this.school = school;
}

var sarahBrown = new Student('Sarah Brown', 17, 'PennX');

sarahBrown.greeting();           //Hello! My name is Sarah Brown
sarahBrown instanceof Person;    //true
```

Prototype Properties

- Properties and methods can be added to prototypes by adding them to the prototype property

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

Person.prototype.planet = 'Earth';
Person.prototype.introduction = function () {
   return 'I am a ' + this.occupation;
}

var johnDoe = new Person('John Doe', 32, 'Dentist');

johnDoe.planet;          //Earth
johnDoe.introduction();    //I am a Dentist
```

Summary

- JavaScript supports functions
  - Primitives are passed by value
  - Objects are passed by reference
- Functions are objects and can be used to create objects
- JavaScript prototypes can be used to create "blueprints" for objects and can be modified dynamically

regular expressions

Review

- JavaScript strings are sequences of characters
- JavaScript strings are immutable
- Strings are objects and have their own functions

Modifying Strings

- We can modify a string but these functions return a **new** string (since strings are immutable!)

```
var friend = 'turtle';

friend.toUpperCase();    // 'TURTLE'
console.log(friend);     // 'turtle'
```

```
var message = ' hello everyone ';
message = message.trim(); // 'hello everyone'
```

```
var myAnimal = 'cat'.concat('mouse');
```

Strings and Characters

- We can get the number of characters in a string using the **`length`** property
- We can access each character by its (0-based) index using **`charAt`** or array notation

```
var name = 'toucan';

name.length;       // 6

name.charAt(3);    // 'c'
name[3];           // 'c'
```

- Remember! JavaScript strings are immutable!

```
var animal = 'cat';

animal[0] = 'r';

console.log(animal); // still 'cat'
```

Searching Strings

- We can determine whether a string starts with, ends with, or includes other strings

```
var msg = 'programming in JavaScript is fun';

msg.startsWith('programming');    // true
msg.startsWith('PROGRAMMING');    // false

msg.endsWith('is fun');           // true

msg.includes('JavaScript');       // true
```

- We can also get the starting index of a contained substring

```
var title = 'the title of my book';
var start = title.search('title');   // 4
start = title.search('banana');      // -1
```

Regular Expressions

- A **regular expression** is a pattern of characters
- A string **matches** a regular expression if it adheres to the same pattern
- Example: "consists of exactly three digits (0-9)"
  - '123' matches
  - 'abc' does not match
  - '12' does not match
  - '12345' does not match

Specifying Ranges of Characters

- We can also specify multiple valid characters that we want to consider for matching
- For instance, we can look for specific characters

```
var numbers = '5 8 2 5 7 6';
numbers.search(/[012]/);        // 4
/[012]/.test(numbers);          // true
```

- Or ranges of characters or special characters

```
var password = 'password4real';
password.search(/[a-z]/);       // 0
password.search(/\d/);          // 8
```

Simple Regular Expression Matching

- We can pass a regular expression to the string's **`search`** function to see if it matches the pattern
- In general, it is considered a match if **any** part of the string matches the regular expression

```
var status = 'I am working VERY hard';

status.search(/VERY/);   // 13

status.search(/very/);   // -1

status.search(/very/i); // 13
```

- Or, we can use the regex's **`test`** function

```
/script/.test('javascript is so much fun!'); // true
```

Using Ranges

- We can combine different ranges

```
var code = 'abc123d4e5';
code.search(/[0-9][a-z][0-9]/); // 5
```

- Or look for characters **not** in a range

```
var chars = 'abc123K456';
chars.search(/[^0-9a-z]/);     // 6
```

Quantifiers

- We may want to know whether the string contains an optional **single** occurrence

```
/[a-z][0-9]?[a-z]/.test('a1b');   // true

/[a-z][0-9]?[a-z]/.test('abc');   // true

/[a-z][0-9]?[a-z]/.test('a123b'); // false
```

- Or optional **multiple** occurrences

```
/[a-z][0-9]*[a-z]/.test('a123b'); // true
```

Summary

- JavaScript strings are immutable but provide functions that allow us to create new, modified versions of them
- Strings have **`startsWith`**, **`endsWith`**, **`includes`**, and **`search`** functions
- We can also use regular expressions' **`test`** function to check for matches in a string

startsWith and endsWith Matches

- Regular expressions can tell us if a string **contains** a pattern, but we may want to know if the string **starts** or **ends** with the pattern

```
/^[a-z][0-9]/.test('a1b');        // true
/^[a-z][0-9]/.test('ab12');       // false

/[a-z][a-z]$/.test('123abc');     // true
/[a-z][a-z]$/.test('123abc456'); // false
```

- This lets us detect **exact** matches

```
/^[a-z][0-9][a-z]$/.test('a1b');        // true

/^[a-z][0-9][a-z]$/.test('a1b2c');      // false

/^[a-z][0-9a-z]*[a-z]$/.test('a1b2c'); // true
```

Introduction to the DOM

How do we use JavaScript and HTML?

- We motivated this part of the course by saying that we wanted a way to dynamically generate HTML
- Now that we've seen JavaScript, how can we use it to access/modify HTML elements?
- This is done by using the **DOM**

What is the DOM?

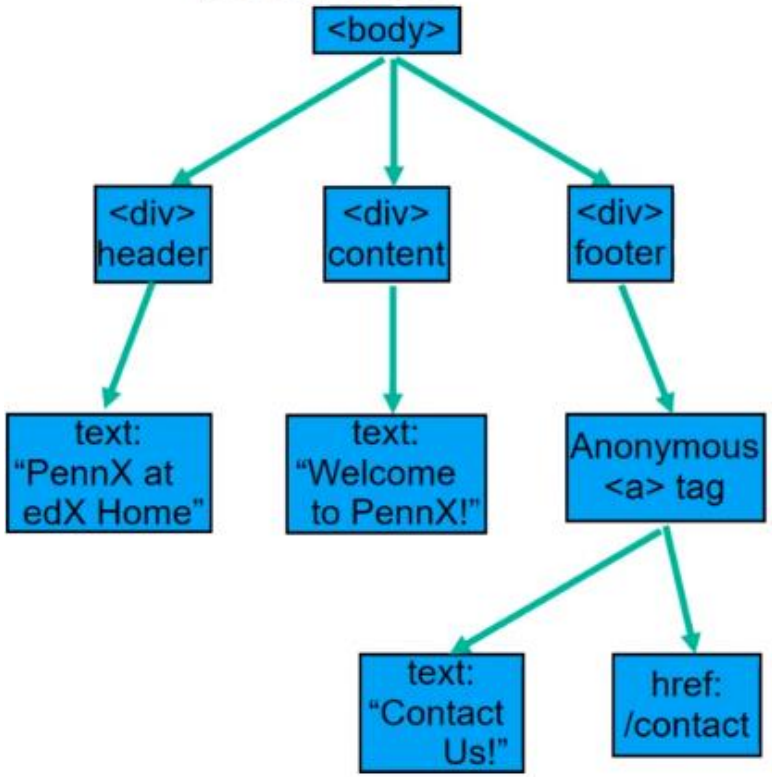
- The **D**ocument **O**bject **M**odel is a structured tree representation of a web page
- The HTML of every web page is turned into a DOM representation by the browser

What does the DOM look like?

- HTML

```
<!DOCTYPE html>
<html>
  <body>
    <div id="header">
        PennX at edX Home
    </div>
    <div id="content">
        Welcome to PennX!
    </div>
    <div id="footer">
      <a href="/contact">
         Contact Us!
       </a>
    </div>
</html>
```

- DOM tree



- DOM in console

```
<!DOCTYPE html>
…<html> == $0
  <head></head>
 ▼<body>
    <div id="header">
        PennX at edX Home
    </div>
    <div id="content">
        Welcome to PennX!
    </div>
  ▼<div id="footer">
      <a href="/contact">
          Contact Us!
      </a>
    </div>
  </body>
</html>
```

Why the DOM?

- Remember, HTML specifies the **structure** of the content on the Web page

- The DOM provides a way for us to programmatically access that structure in JavaScript

Objects as JSON

- JSON = **J**ava**S**cript **O**bject **N**otation
- JSON is a **textual** representation of a JavaScript Object that can be stored as a string, in a .json file, or be exchanged between programs
- A sample JSON file or string might look like this:

```
{
    "name": "John Doe",
    "age": 25,
    "isMale": true,
    "personality": ["patient", "loyal", "happy"],
    "company": { "name": "EdX", "id": 2984 }
}
```

DOM Example

- The root DOM object can be accessed by the object called **`document`**
- Elements in this DOM tree can be retrieved and manipulated

```
<html>
<body>

<div>
  The current date/time is <span id="dateTime"> </span>.
</div>

<script>
  var dateTimeField = document.getElementById('dateTime');
  dateTimeField.innerHTML = new Date();
</script>

</body>
</html>
```

- Data can be stored in the browser across multiple page requests using **`localStorage`**

```
<div>
  You have visited this page <span id="report"> </span> times.
</div>

<script>
  var timesVisited = 0;
  if (localStorage.timesVisited) {
    timesVisited = parseInt(localStorage.timesVisited);
  }
  timesVisited += 1;
  localStorage.setItem('timesVisited', timesVisited);

  var report = document.getElementById('report');
  report.innerHTML = timesVisited;

  if (timesVisited > 10)
      report.style.backgroundColor = 'red';
</script>
```

Converting between JSON and Objects

- JavaScript objects can be converted to a JSON string via **`JSON.stringify(myObject)`**
- String representations can be converted back to an object via **`JSON.parse(jsonString)`**
- All values must be a string, number, array, boolean, null, or another valid JSON object

DOM and JSON

```
You have accessed this page <span id="report"></span> times.
<p>
Your last visit was <span id="lastVisitDate"></span>.

<script>
var timesVisited = 0;
var lastVisitDate = 'never';

if (localStorage.lastVisit) {
   var lastVisit = JSON.parse(localStorage.lastVisit);
   timesVisited = lastVisit.numVisits;
   lastVisitDate = lastVisit.date;
}

document.getElementById('lastVisitDate').innerHTML = lastVisitDate;

timesVisited++;
document.getElementById('report').innerHTML = timesVisited;

var myLastVisit = { }
myLastVisit.date = new Date();
myLastVisit.numVisits = timesVisited;
localStorage.lastVisit = JSON.stringify(myLastVisit);
</script>
```

Storing JSON

- A great application of JSON usage is to store JSON strings in local browser storage:

```
localStorage.myJSON = JSON.stringify(myObject);

// … in a later session
myObject = JSON.parse(localStorage.myJSON);
```

- Later on, as you learn about server-side JavaScript, you will also learn how to use JSON data to communicate with a server or API.

Summary

- JavaScript can use the DOM to retrieve/modify HTML elements
  - `document.getElementById('id')` returns the specific HTML element with that ID
  - *`element`*`.innerHTML` can be modified to change the element's HTML/content
  - *`element`*`.style` can be modified to change the element's CSS/appearance
- We can use `localStorage` to save values across page requests
- Objects can be converted to string representations known as JSON

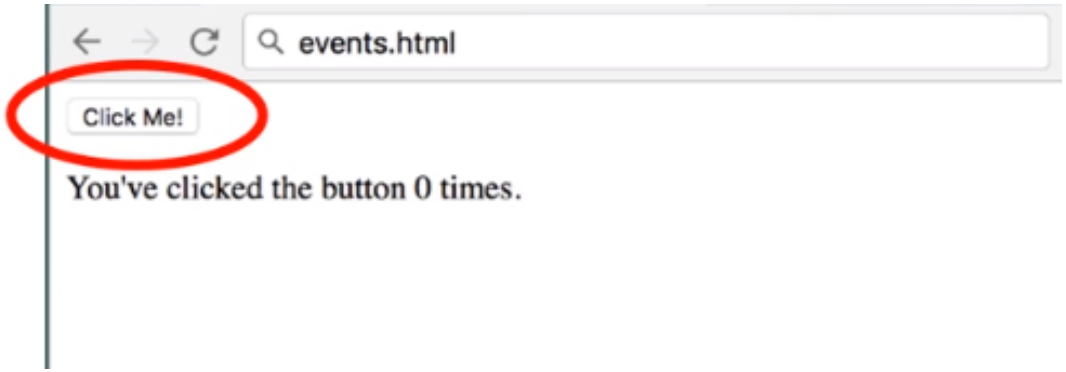
DOM events

Review: HTML, JavaScript, DOM

- Previously we saw that JavaScript can use the DOM to retrieve/modify HTML elements
  - `document.getElementById('id')` returns the specific HTML element with that ID
  - `element.innerHTML` can be modified to change the element's HTML/content
  - `element.style` can be modified to change the element's CSS/appearance

- How can we do this in response to user events?

```
<html>
<body>

<button id="clickMe">Click Me!</button>
<p>
You've clicked the button <span id="numClicks">0 times</span>.

<script>
var clicks = 0;

function clickHandler() {
  clicks++;
  var numClicksSpan = document.getElementById('numClicks');
  if (clicks == 1)
    numClicksSpan.innerHTML = 'once';
  else
    numClicksSpan.innerHTML = clicks + ' times';
}

var button = document.getElementById('clickMe');
button.addEventListener('click', clickHandler);

</script>

</body>
</html>
```



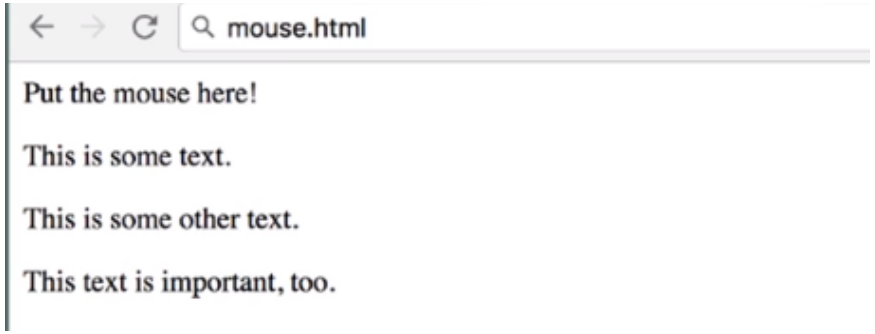
Event-Driven Programming

- Ordinarily we think of a program as a sequence of instructions and function calls
- **Event-Driven programming** is when a program's behavior is based on *events*
- In web programming, these events are generally user actions
- Different events/actions invoke different **callback functions** which handle that event/action

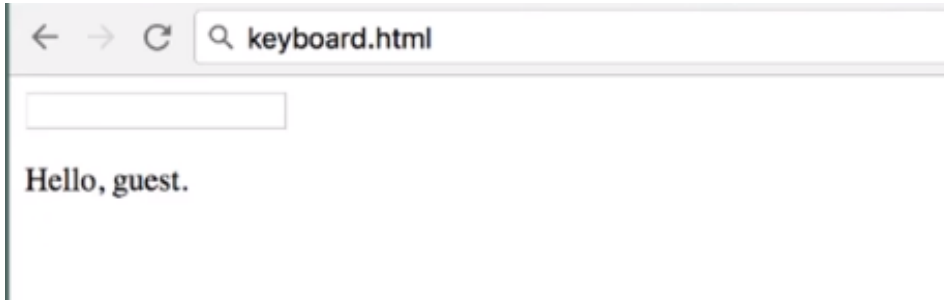
Event-Driven Programming

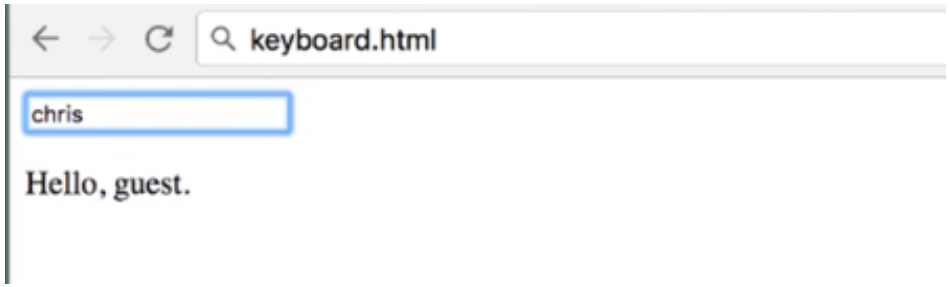
- Event-Driven programming is a form of asynchronous programming
  - Synchronous Way:
    - You are expecting a user input
    - You continuously re-check a text field until the user has put in the required information
    - You run some code on the user input
  - Event-Driven/Asynchronous Way:
    - You are expecting a user input
    - You tell your browser to let your program know when the user has put in the required information
    - You (possibly) run other code until your browser notifies you
    - When the user has entered the information, you run the associated callback function

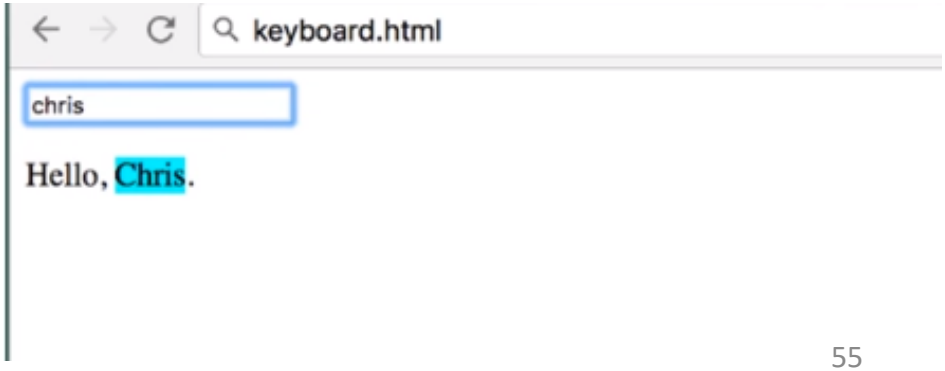
```
<div id="mouseOverMe"> Put the mouse here! </div> <p>
<div class="highlightText"> This is some text. </div> <p>
<div> This is some other text. </div> <p>
<div class="highlightText"> This text is important, too.</div>

<script>
 function makeBold() {
   var divs = document.getElementsByClassName('highlightText');
   for (var i = 0; i < divs.length; i++) {
       divs[i].style.color = 'red';
       divs[i].style.fontWeight = 'bold';
   }
 }

 function restore() {
   var divs = document.getElementsByClassName('highlightText');
   for (var i = 0; i < divs.length; i++) {
       divs[i].style.color = 'black';
       divs[i].style.fontWeight = 'normal';
   }
 }

 var mouseOverMeDiv = document.getElementById('mouseOverMe');
 mouseOverMeDiv.addEventListener('mouseover', makeBold);
 mouseOverMeDiv.addEventListener('mouseout', restore);
</script>
```



```
<html>
<body>

<input id="nameInput"></input>
<p>
Hello, <span id="nameField">guest</span>.

<script>

function nameHandler(e) {
  if (e.keyCode == 13) {
    var nameInput = document.getElementById('nameInput');
    var nameField = document.getElementById('nameField');
    nameField.innerHTML = nameInput.value;
    nameField.style.backgroundColor = 'cyan';
    nameField.style.textTransform = 'capitalize';
  }
}

document.addEventListener('keyup', nameHandler);

</script>

</body>
</html>
```







Summary

- We can use **event-driven programming** in JavaScript to modify HTML based on user activity

- We do this by defining **callback functions** and associating them with various events by adding event listeners
  - *element*.addEventListener(*event*, *function*)
  - Events: ‘click’, ‘mouseover’, ‘mouseout’, ‘keyup’

A fast, small, and feature-rich JavaScript library

Review

- Previously we've seen how to use JavaScript, the DOM, and event-driven programming to modify HTML based on user activity

- However…
  - different browsers may work in different manners
  - the syntax can be a bit clunky
  - many features are hard to implement

- Is there an easier way?

jQuery

- Simplifies JavaScript usage on webapps
- More intuitive way of DOM manipulation
- Great cross-browser support (Except IE6)
- Additional Utilities
- Effects and Animations
- Customizable plugins

Introduction to jQuery

Using jQuery

- Download the latest version of jQuery from jquery.com

- Add the downloaded .js file to your HTML webpage using a script tag
  - `<script src="jQueryFile.js"></script>`

Selecting DOM Elements

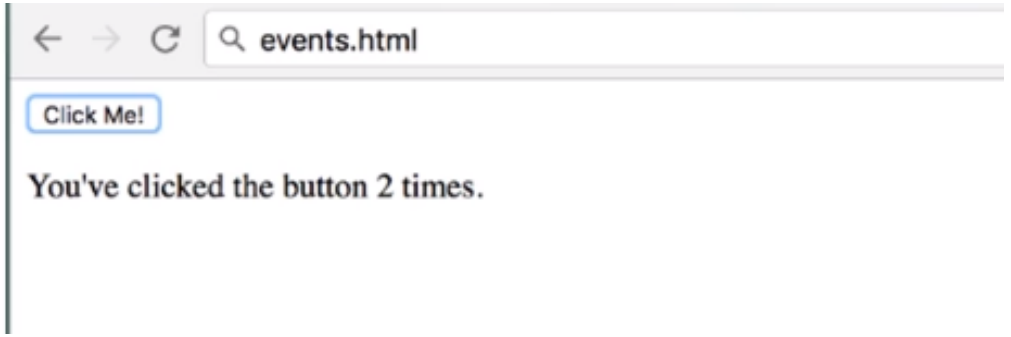
- In jQuery, **$** is used to select DOM elements for manipulation, along with basic CSS element syntax
  - `$("*")` selects all elements
  - `$(this)` selects the current element
  - `$("div")` selects all elements
  - `$(".title")` selects all elements with class="title"
  - **`$("#name")` selects the element with id="name"**

jQuery DOM Manipulation

- To manipulate DOM contents, the general format is `$(selector).action(arguments…)`

```
$("#name").html("Hello");
$("#name").append(" World!");
$("#name").addClass("greeting");
$("#name").hide();
$("#name").show();
```

- To add an event listener to an element, the general format is `$(selector).event(callback)`




```
<html>
<head><script src="jquery.js"></script></head>
<body>
<button id="clickMe">Click Me!</button>
<p>
You've clicked the button <span id="numClicks">0 times</span>.

<script>
var clicks = 0;

function clickHandler() {
  clicks++;
  var numClicksSpan = document.getElementById('numClicks');
  if (clicks == 1)
    numClicksSpan.innerHTML = 'once';
  else
    numClicksSpan.innerHTML = clicks + ' times';
}

var button = document.getElementById('clickMe');
button.addEventListener('click', clickHandler);

</script>

</body>
</html>
```

```
function clickHandler() {
  clicks++;
  var numClicksSpan = $('#numClicks');
  if (clicks == 1)
    numClicksSpan.innerHTML = 'once';
  else
    numClicksSpan.innerHTML = clicks + ' times';
}
```

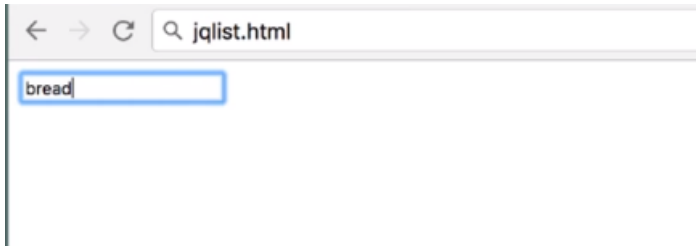
```
function clickHandler() {
  clicks++;
  var numClicksSpan = $('#numClicks');
  if (clicks == 1)
    numClicksSpan.html('once');
  else
    numClicksSpan.html(clicks + ' times');
}
```

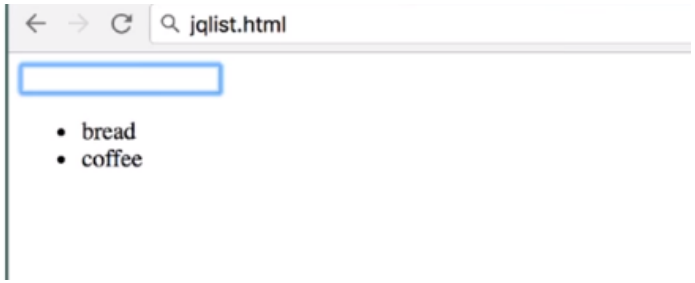
```
}

var button = $('#clickMe');
button.addEventListener('click', clickHandler);

</script>
```

```
}

$('#clickMe').click(clickHandler);


</script>
```

Example(1)

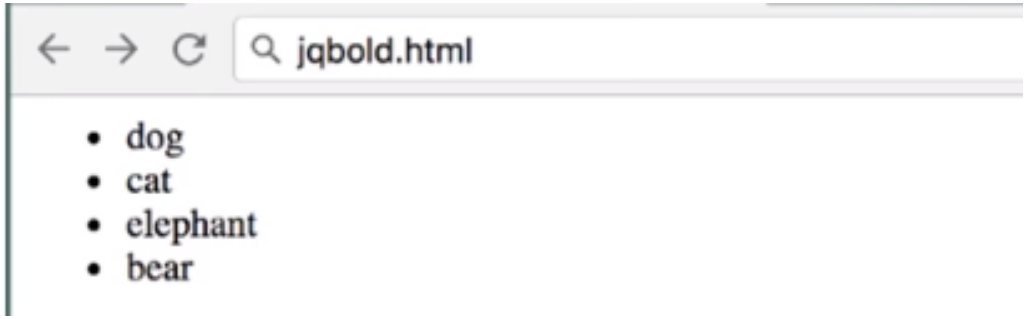
```
<html>
<head><script src="jquery.js"></script></head>

<body>
<input id="itemField"></input>
<p>

<ul>
<span id="list"></span>
</ul>

<script>
function keyPressHandler(e) {
  if (e.keyCode == 13) {
    $('#list').append('<li>'   + $('#itemField').val() + '</li>');
    $('#itemField').val('');
  }
}

$('#itemField').keyup(keyPressHandler);

</script>

</body>
</html>
```





Example(2)

```
<html>

<head><script src="jquery.js"></script></head>

<body>

<ul>
<li>dog</li>
<li>cat</li>
<li>elephant</li>
<li>bear</li>
</ul>

<script>

$('li').click(function() {
  $(this).css('font-weight', 'bold');
});

</script>

</body>
</html>
```



Summary

- jQuery is a powerful library that allows us to select DOM elements using CSS notation
- We can then modify their content and appearance programmatically
- We can also register event listeners for different elements

event handling(1)

jqhighlight.html

- dog
- cat
- elephant
- bear

jqhighlight.html

- dog
- cat
- elephant
- bear

jqhighlight.html

- dog
- cat
- elephant
- bear

Review

- Previously we saw how to use jQuery to select and modify DOM elements and add event listeners

- How else can we specify selectors and add event listeners?

```
<html>
<head><script src="jquery.js"></script></head>
<body>
<ul>
<li>dog</li>
<li>cat</li>
<li>elephant</li>
<li>bear</li>
</ul>

<script>

$('li').mouseenter(function() {
  $(this).css('color', 'red');
  $(this).css('font-size', '120%');
});

$('li').mouseleave(function() {
  $(this).css('color', 'black');
  $(this).css('font-size', '100%');
});

</script>

</body>
</html>
```

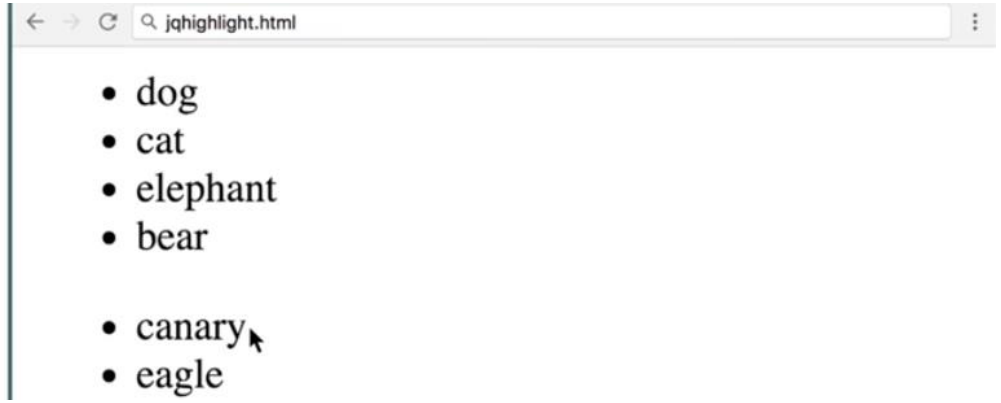
Example(1)

```
<html>
<head><script src="jquery.js"></script></head>
<body>
<ul>
<li>dog</li>
<li>cat</li>
<li>elephant</li>
<li>bear</li>
</ul>

<script>

$('li').mouseenter(function() {
  $(this).css('color', 'red');
  $(this).css('font-size', '120%');
});

$('li').mouseleave(function() {
  $(this).css('color', 'black');
  $(this).css('font-size', '100%');
});

</script>

</body>
</html>
```

```
<script>

$('li').hover(
  function() {
    $(this).css('color', 'red');
    $(this).css('font-size', '120%');
  },
  function() {
    $(this).css('color', 'black');
    $(this).css('font-size', '100%');
});

</script>
```

```
<script>
$('li').on({
  mouseenter: function() {
    $(this).css('color', 'red');
    $(this).css('font-size', '120%');
  },
  mouseleave: function() {
    $(this).css('color', 'black');
    $(this).css('font-size', '100%');
  },
  click: function() {
    $(this).css('background-color', 'yellow');
  }
});
</script>
```

Example(1)




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

<body>
<ul>
<li class="highlight">dog</li>
<li class="highlight">cat</li>
<li class="highlight">elephant</li>
<li class="highlight">bear</li>
</ul>

<ul>
<li>canary</li>
<li>eagle</li>
</ul>

<script>

$("li.highlight").on({
mouseenter: function() { . . . },
mouseleave: function() { . . . },
click: function() { . . . }
});

</script>

</body>
</html>
```

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

<body>
<ul class="highlight">
<li>dog</li>
<li>cat</li>
<li>elephant</li>
<li>bear</li>
</ul>

<ul>
<li>canary</li>
<li>eagle</li>
</ul>

<script>

$("ul.highlight").find("li").on({
mouseenter: function() { . . . },
mouseleave: function() { . . . },
click: function() { . . . }
});

</script>

</body>
</html>
```

Advanced Selectors

- `$(someNodes).find(selector)` will search *someNodes*' children for selector.
- **$** selectors can be chained
  - `$("div.book")` selects the div with class="book"
  - `$("div, .book")` selects all divs **and** all elements with class="book"
- : can be used to specify element properties
  - `$("p:hidden")` selects all `<p>` elements that are visually hidden
- These selectors are all CSS selectors!
  - All other CSS selectors also work in **$**

Summary

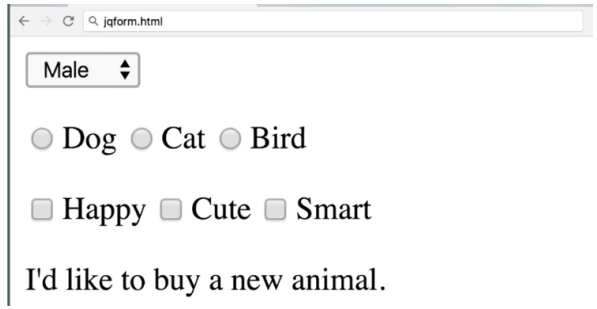
- jQuery can be used to retrieve and modify DOM elements and add event listeners
- We can use advanced selectors and handlers to simplify functionality

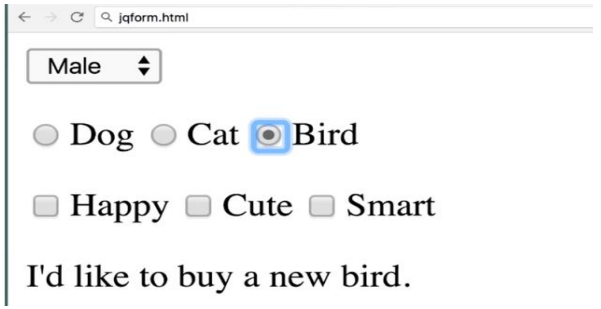
event handling(Cont.)

User Events in the Browser

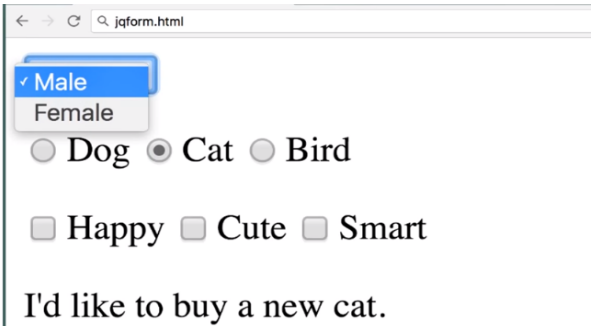
Event Type	Events
Mouse	click, dblclick, mousedown, mouseup, mouseover, mouseout
Keyboard	keydown, keypress, keyup
Form	focus, blur, change, reset, submit
Window / Element	load, resize, scroll, unload

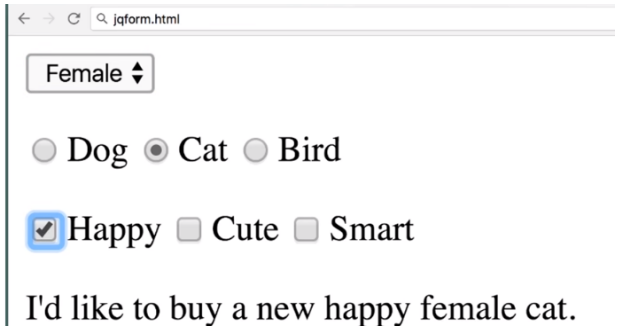
```
<head>
<script src="jquery.js"></script>
</head>

<body>

<form>

<select name="choose">
  <option value="male">Male</option>
  <option value="female">Female</option>
</select>
<p>
<input type="radio" name="species" value="dog">Dog</input>
<input type="radio" name="species" value="cat">Cat</input>
<input type="radio" name="species" value="bird">Bird</input>
<p>
<input type="checkbox" value="happy">Happy</input>
<input type="checkbox" value="cute">Cute</input>
<input type="checkbox" value="smart">Smart</input>

</form>

<p>
I'd like to buy a new <span id="featureSpan"></span>
<span id="genderSpan"></span> <span id="speciesSpan">animal</span>.
```



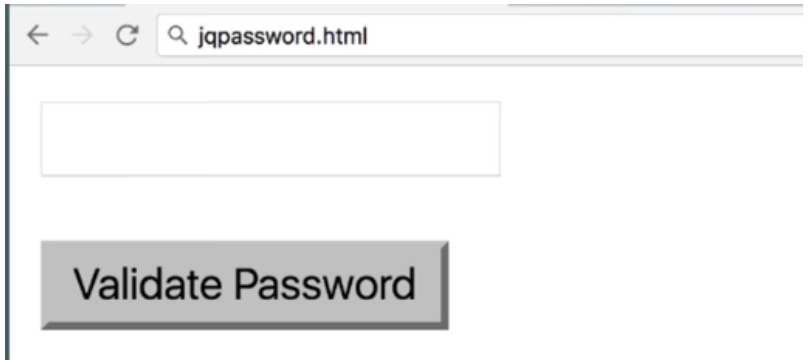


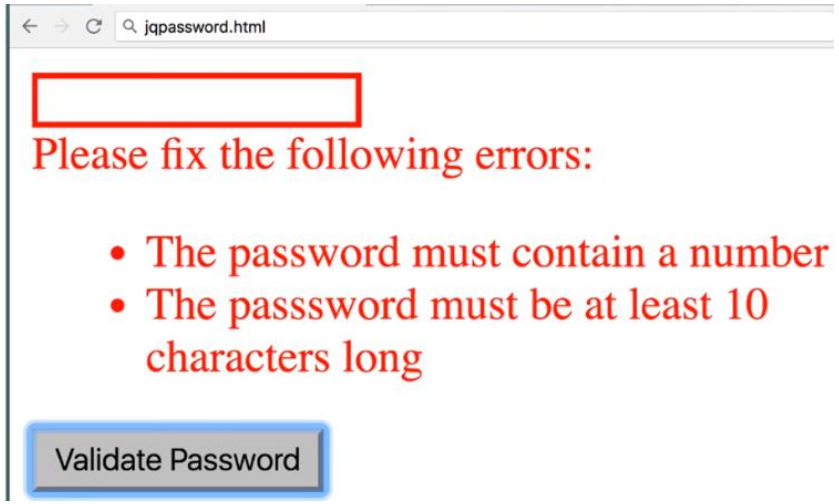

```
<select name="choose">
  . . .
</select>
<p>
<input type="radio" name="species" value="dog">Dog</input>
<input type="radio" name="species" value="cat">Cat</input>
<input type="radio" name="species" value="bird">Bird</input>

. . .

<p>
I'd like to buy a new <span id="featureSpan"></span>
<span id="genderSpan"></span> <span id="speciesSpan">animal</span>.

<script>

    // handling select box
    $("select[name='choose']").change(function() {
       $('#genderSpan').html($(this).val());
    });

    // handling radio buttons
    $("input:radio[name='species']").change(function() {
       if ($(this).prop('checked')) {
          $('#speciesSpan').html($(this).val());
       }
    });
```

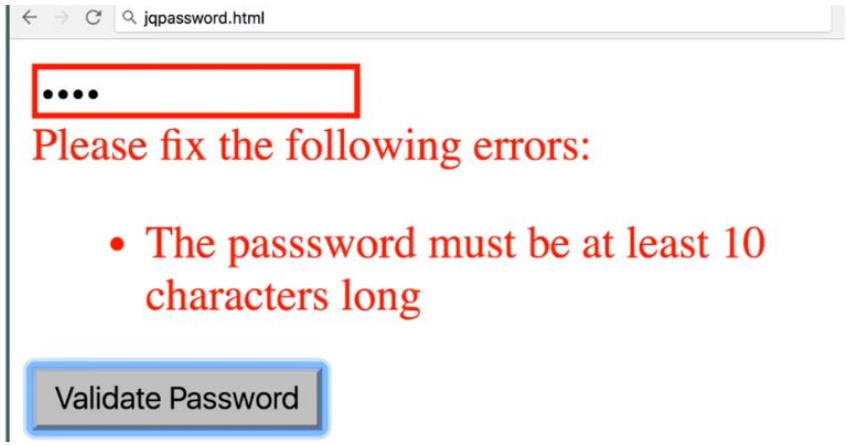




```
<p>
I'd like to buy a new <span id="featureSpan"></span>
<span id="genderSpan"></span> <span id="speciesSpan">animal</span>.

<script>
 . . .
 var allChecked = [];

 $('input:checkbox').change(function() {
    var value = $(this).val();
    if ($(this).prop('checked')) {
        allChecked.push(value);
    }
    else {
        var index = allChecked.indexOf(value);
        if (index != -1)
           allChecked.splice(index, 1);
    }
    $('#featureSpan').html('');
    for (var i = 0; i < allChecked.length; i++) {
       $('#featureSpan').append(allChecked[i]);
       if (i < allChecked.length - 1)
          $('#featureSpan').append(', ');
       else
         $('#featureSpan').append(' ');
    }
  });
</script>
```







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

<style>

.errorText {
color: red;
}

.errorBox {
border: 2px solid red;
}

.goodBox {
border: 2px solid green;
}

</style>

</head>
```

```
<body>

<input type="password" name="password"></input>

<br>

<span id="errorMessage" class="errorText" hidden>
     Please fix the following errors:</span>
<ul>
<li id="needsNumber" class="errorText" hidden>
    The password must contain a number</li>

<li id="atLeast10Chars" class="errorText" hidden>
    The passsword must be at least 10 characters long</li>
</ul>

<span id="successMessage" hidden>The password is okay!</span>

<p>

<button name="submit">Validate Password</button>
```

```
<script>

$("button[name='submit']").click(function() {
   var passwordField = $("input[name='password']");
   var password = passwordField.val();
   var isOkay = true;
   if (password.length < 10) {
     isOkay = false;
     $('#atLeast10Chars').show();
   }
   if (/\d/.test(password) == false) {
     isOkay = false;
     $('#needsNumber').show();
   }
   if (isOkay == false) {
     $('#successMessage').hide();
     $('#errorMessage').show();
     passwordField.removeClass("goodBox").addClass("errorBox");
   }
   else {
     $('.errorText').hide();
     $('#successMessage').show();
     passwordField.removeClass("errorBox").addClass("goodBox");
   }
   return false;
});

</script>
```

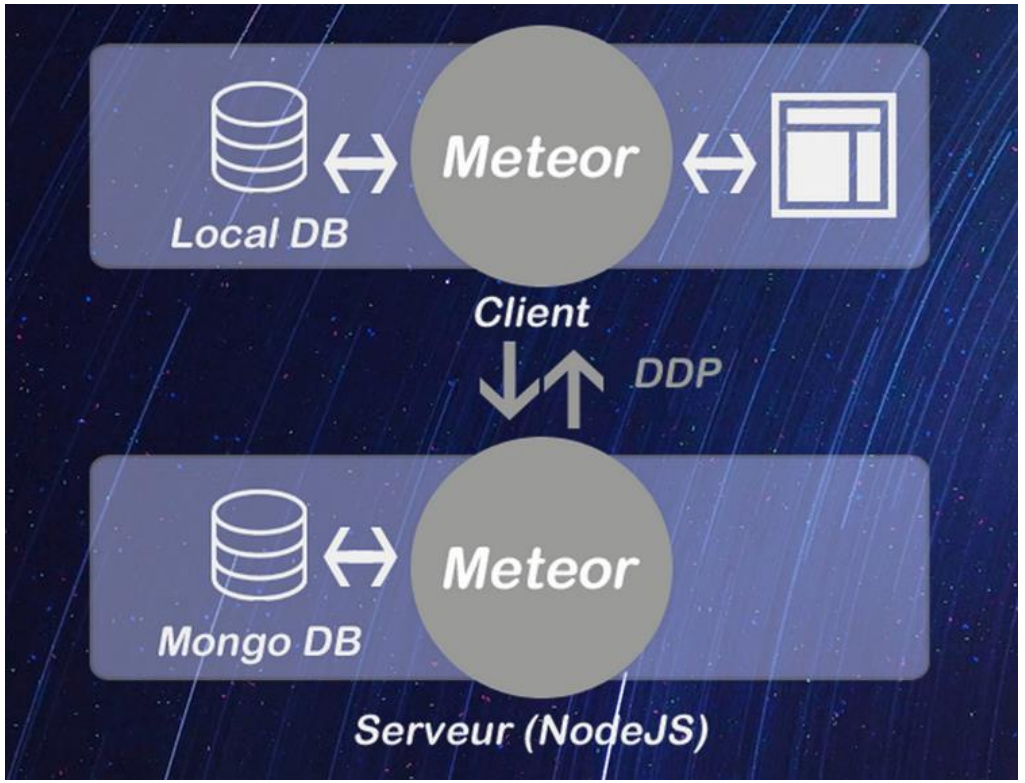
Summary

- jQuery can be used to retrieve and modify DOM elements and add event listeners
- We have seen how to handle events from the mouse, keyboard, and forms

METEOR

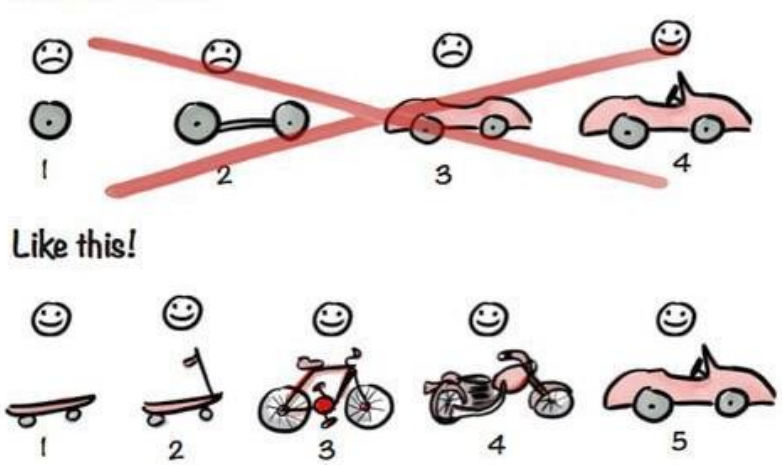
An open-source platform for building top-quality web apps in a fraction of the time, whether you're an expert developer or just getting started.

webperformance



Why is Meteor?

1.It's fast to build with.



2.Develop with just one language

3.Easy to learn.

3.Real-time by default

Fullstack Reactivity

Traditional programming	Reactive programming

var a = 2;
var b = 5;
var c = a + b;
console.log(c);
# c is 7 	
	var a = 2;
	var b = 5;
	var c = a + b;
	console.log(c);
	# c is 7

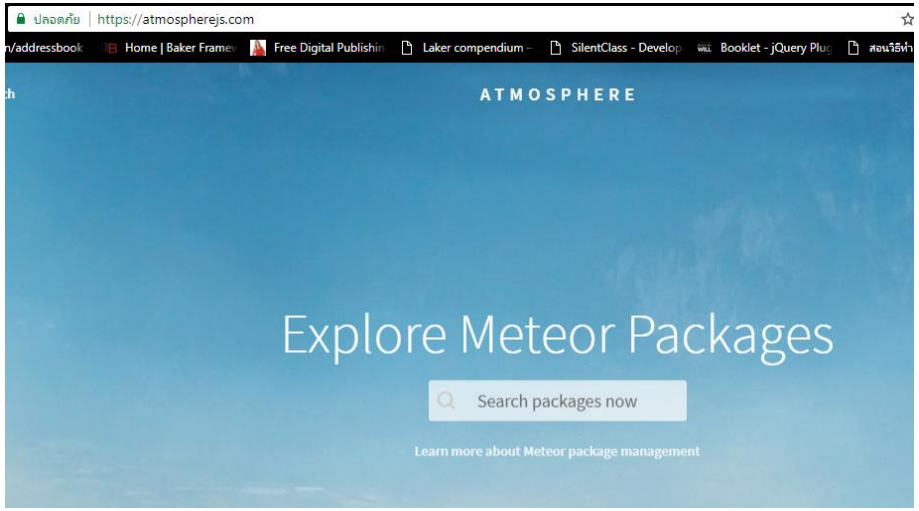
a = 5;
console.log(c);
# c is still 7 	
	a = 5;
	console.log(c);
	# c is magically 10

c = a + b;
console.log(c);
# c is finally 10 		

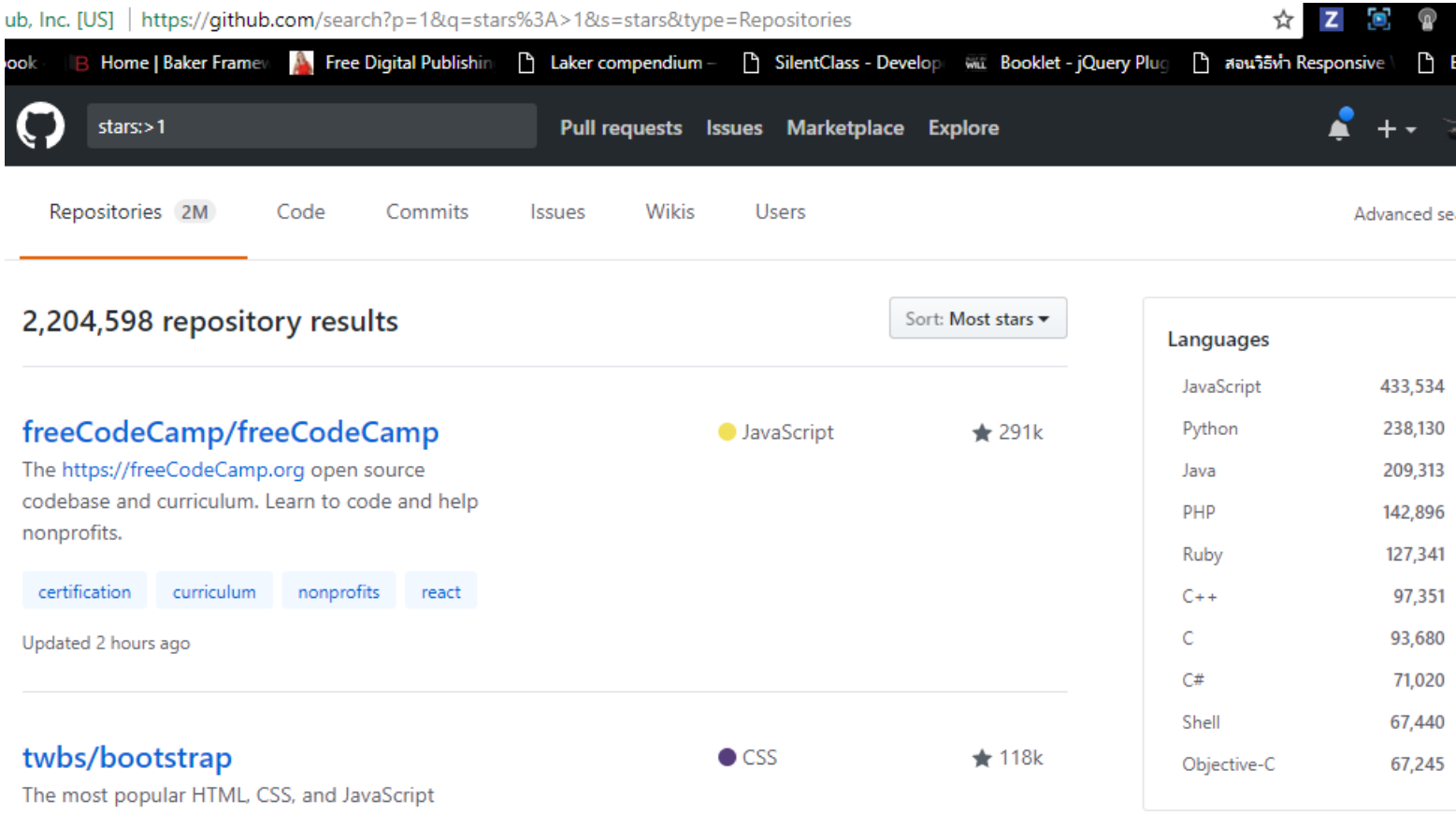
Why is Meteor(Cont.)?

5.Build native smartphone

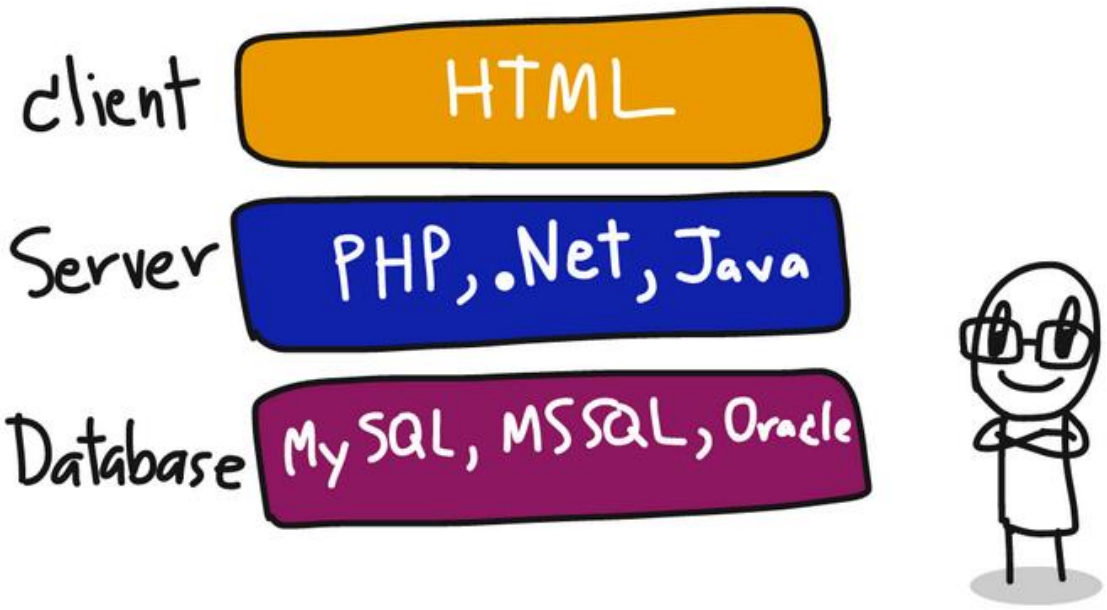
6.Smart packages that save you time



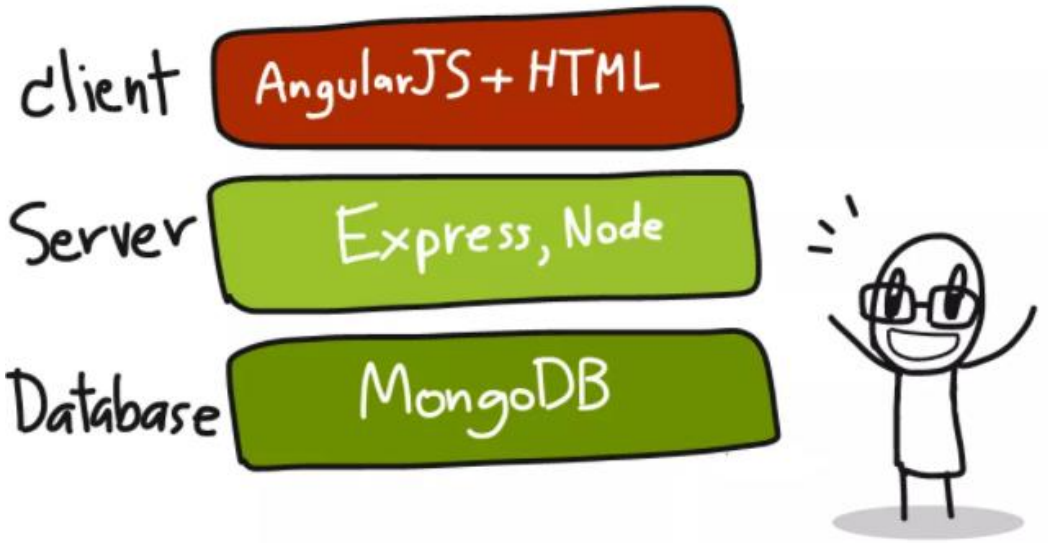
7.Active and supportive community



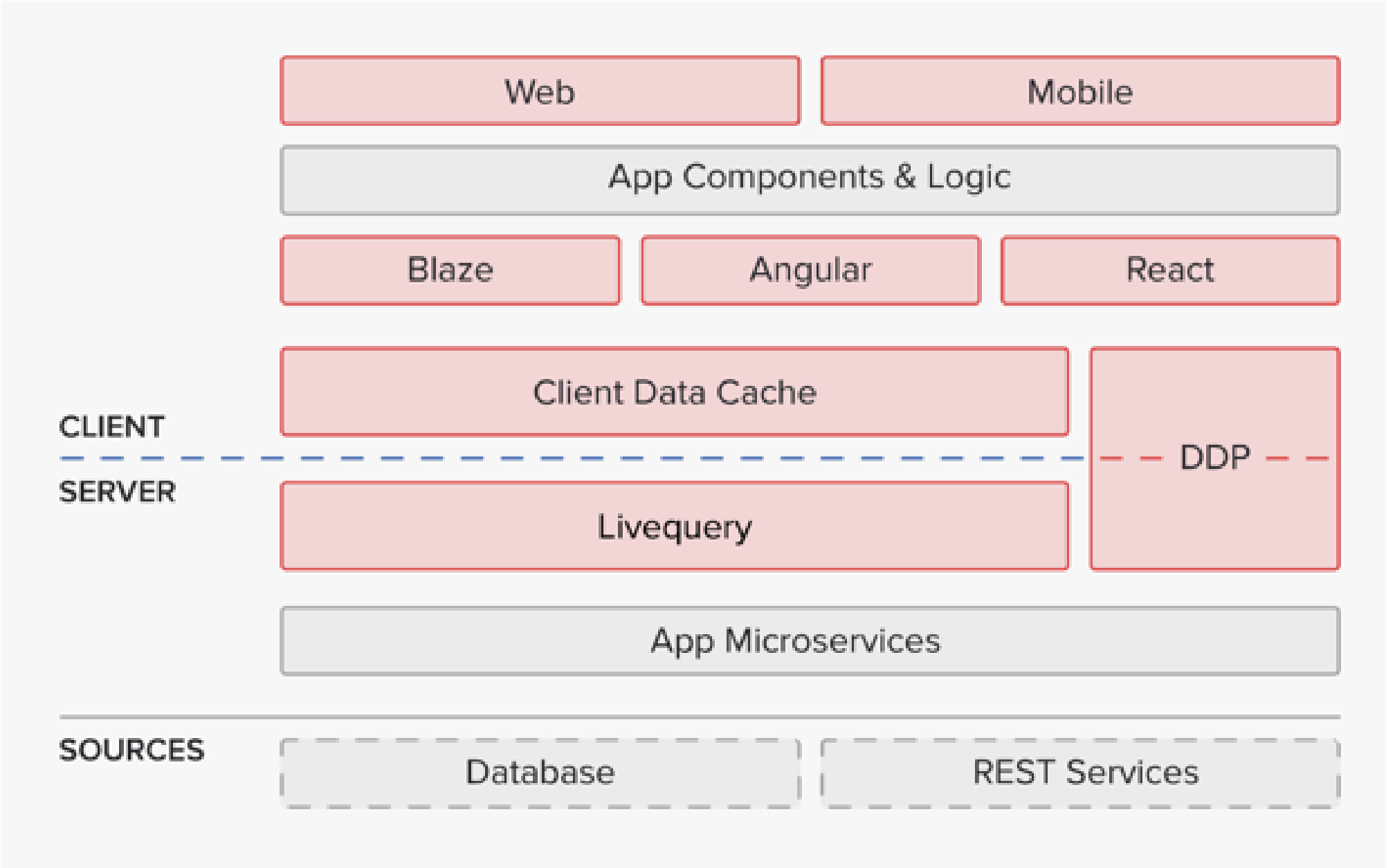
8.Meteor is the future.

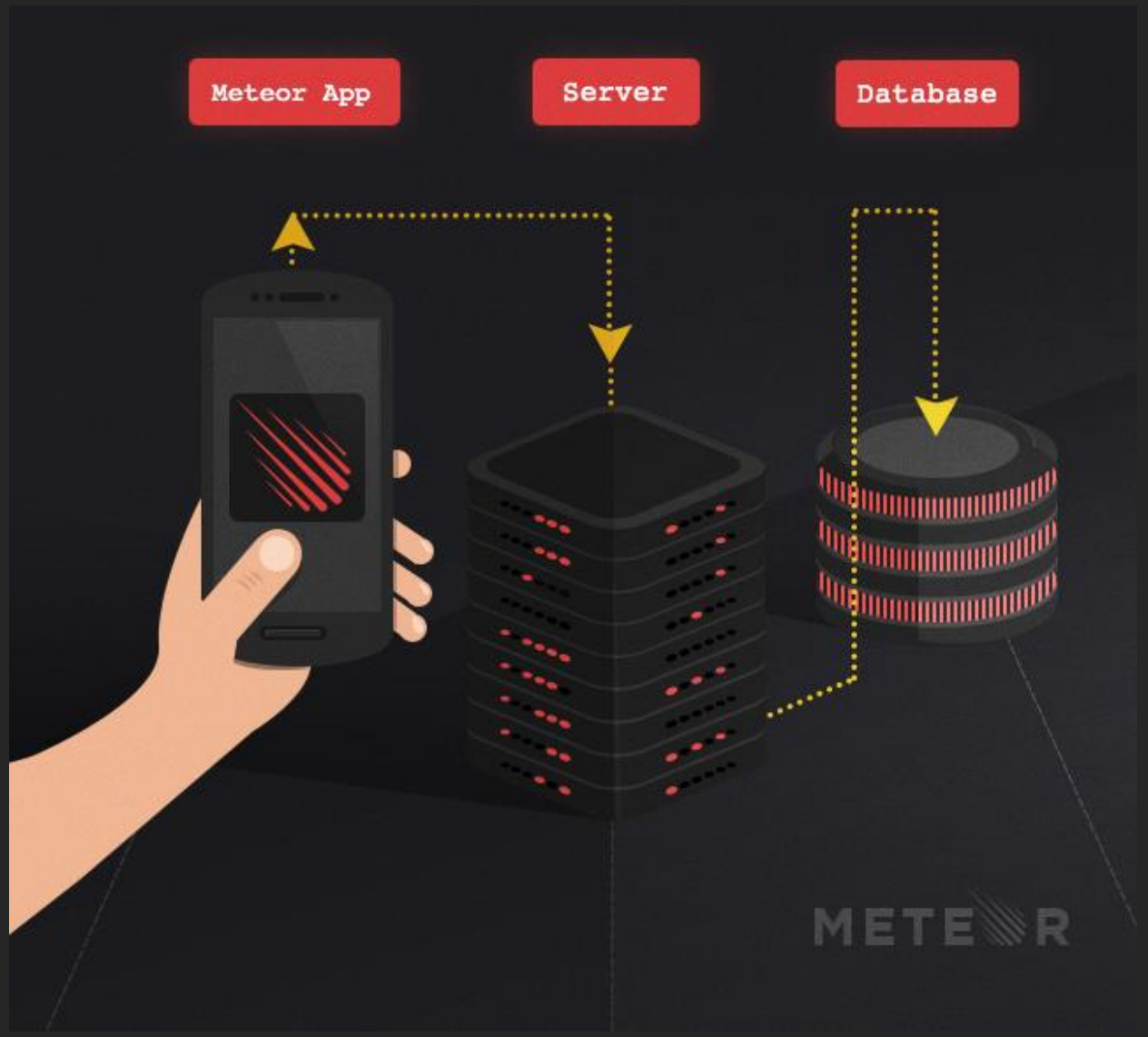


Web Application Stack



Full JavaScript Stack


Web
Mobile
App Components & Logic
Blaze
Angular
React
Client Data Cache
DDP
CLIENT
SERVER
Livequery
App Microservices
SOURCES
Database
REST Services


Meteor App
Server
Database
METEOR


realtime sync
Close
Close
Syncing
DisplayModal()
Close
realtime sync


node
JS


mongoDB.

GoDaddy™

Microsoft

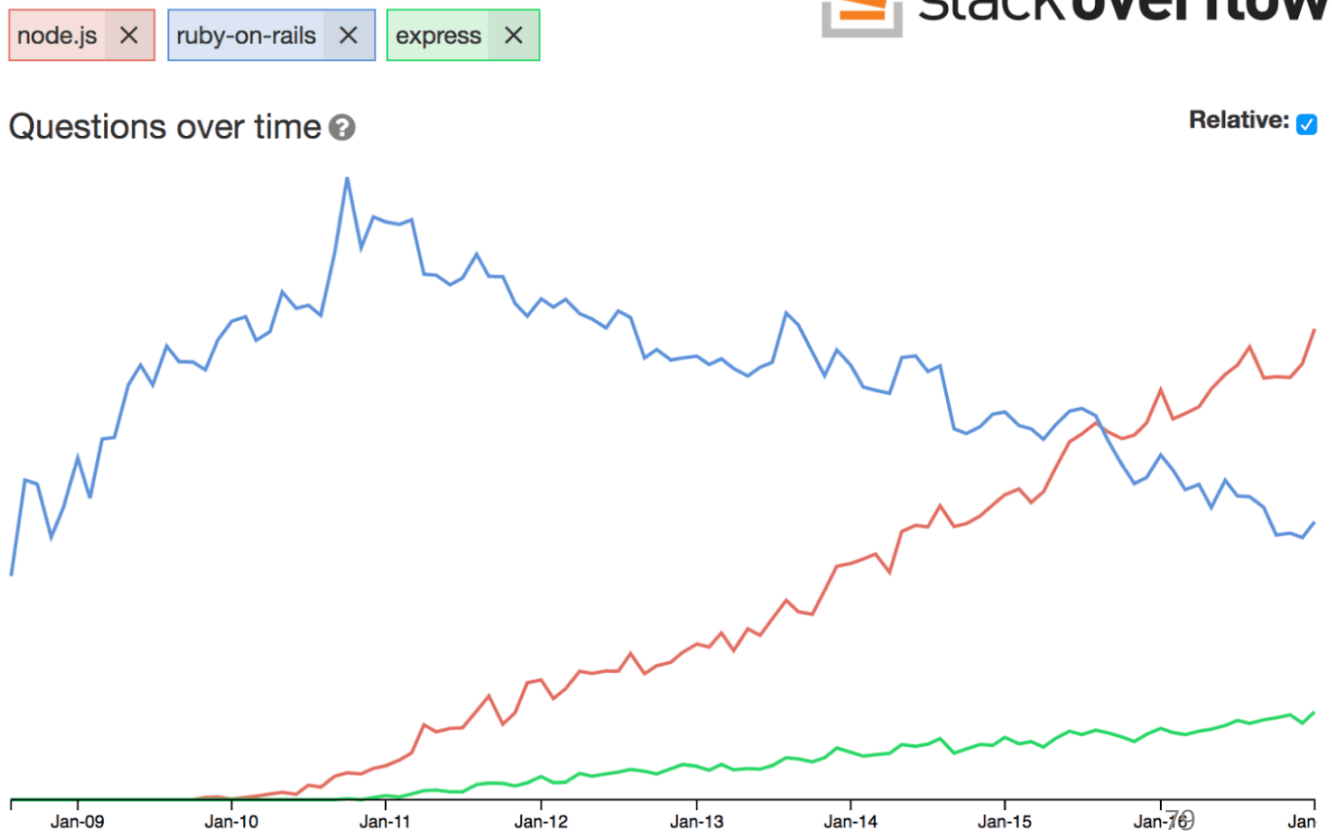
ebay

NETFLIX

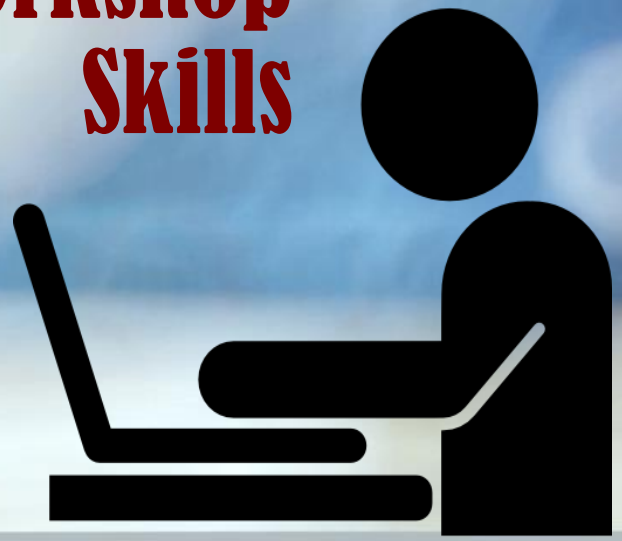
UBER

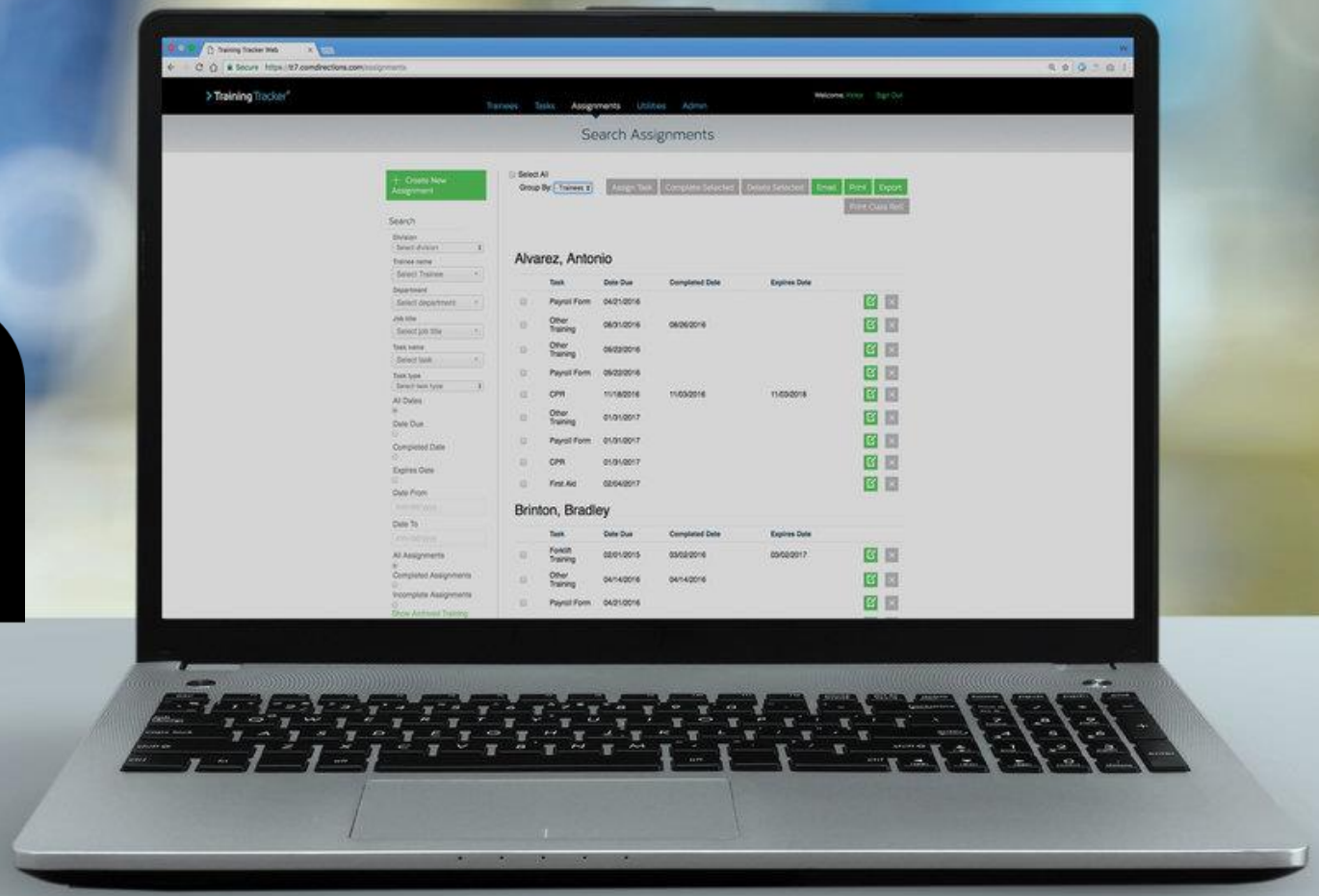
YAHOO!

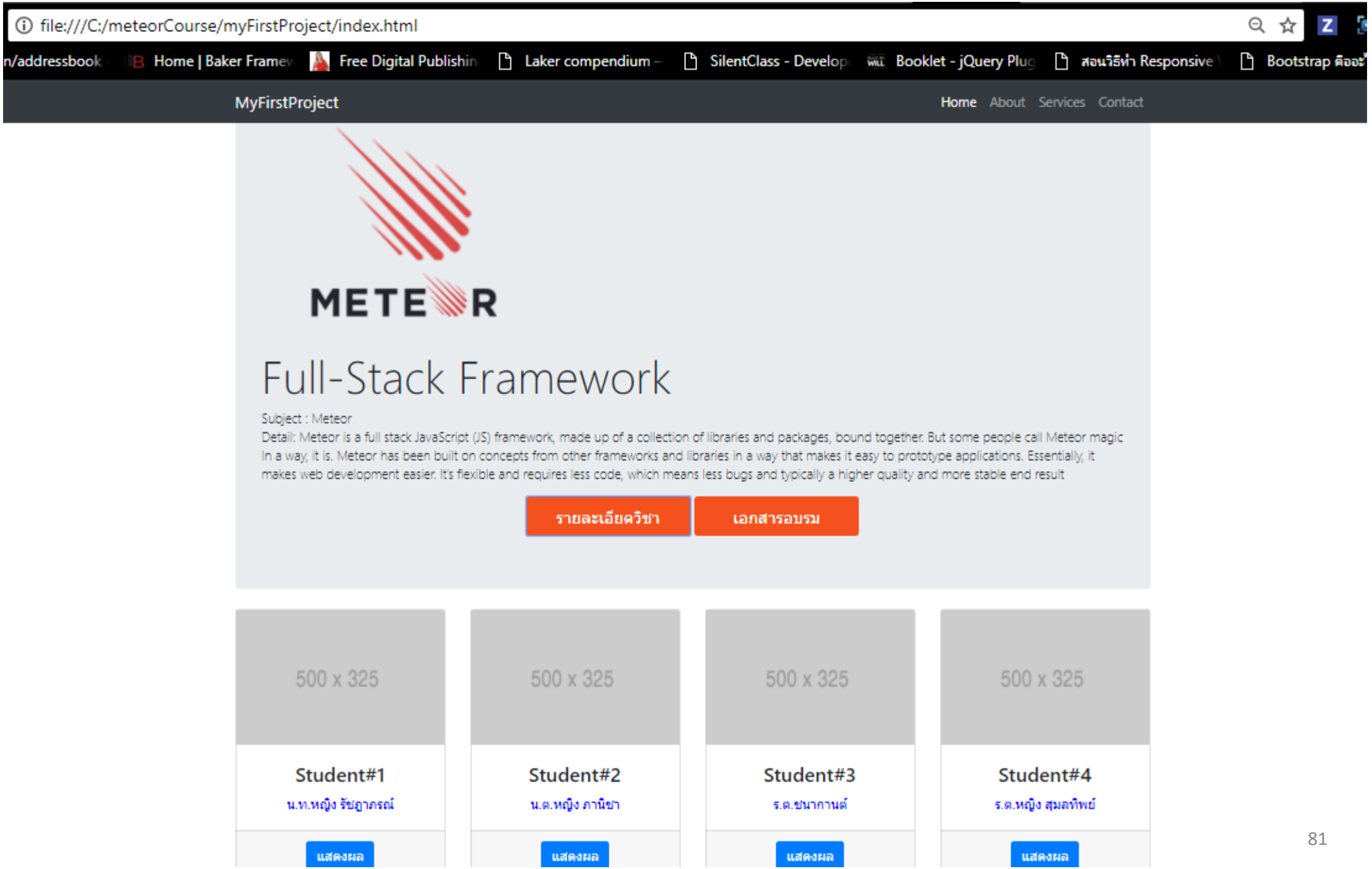
stackoverflow



Meteor Workshop Skills






file:///C:/meteorCourse/myFirstProject/index.html
n/addressbook
Home | Baker Framew
Free Digital Publishin
Laker compendium –
SilentClass - Develop
Booklet - jQuery Plug
สอนวิธีทำ Responsive
Bootstrap คืออะ
MyFirstProject
Home
About
Services
Contact
METEOR
Full-Stack Framework
Subject : Meteor
Detail: Meteor is a full stack JavaScript (JS) framework, made up of a collection of libraries and packages, bound together. But some people call Meteor magic In a way, it is. Meteor has been built on concepts from other frameworks and libraries in a way that makes it easy to prototype applications. Essentially, it makes web development easier. It's flexible and requires less code, which means less bugs and typically a higher quality and more stable end result
รายละเอียดวิชา
เอกสารอบรม
500 x 325
Student#1
น.ท.หญิง รัชฎาภรณ์
แสดงผล
500 x 325
Student#2
น.ต.หญิง ภานิชา
แสดงผล
500 x 325
Student#3
ร.ต.ชนากานต์
แสดงผล
500 x 325
Student#4
ร.ต.หญิง สุมลทิพย์
แสดงผล