

# HTTP and Client-side Form Handling

## IERG4210 Lecture 4

Dr. Adonis Fung  
Information Engineering, CUHK  
Paranoids, Yahoo!

IERG4210 Web Programming and Security, 2015 Spring.  
Offered by Dept. of Information Engineering, The Chinese University of Hong Kong.

Copyright. Dr. Adonis Fung

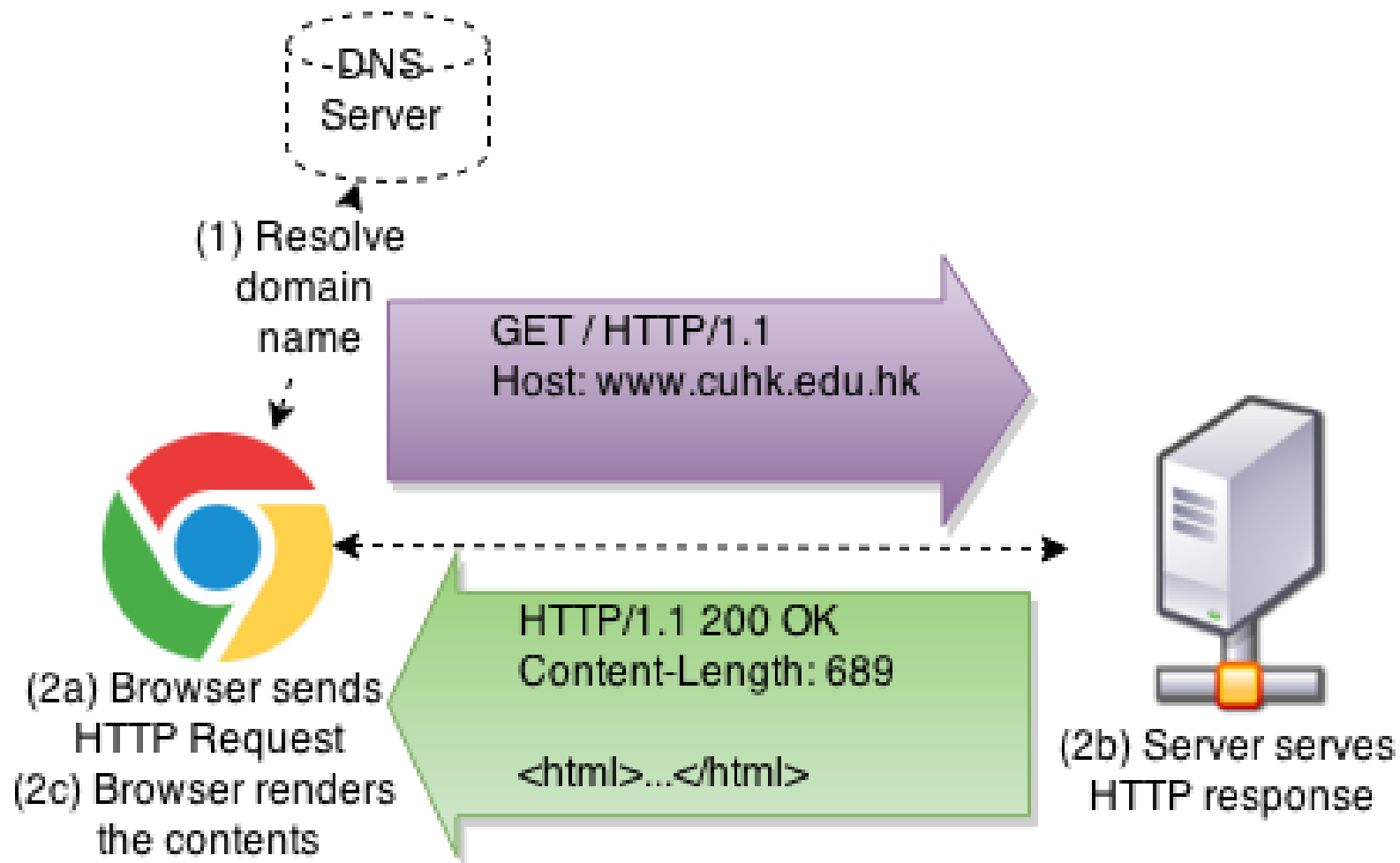
# Agenda

- HTTP
  - Introduction & Client-Server Model
  - HTTP Request and Response
- HTML Forms and Input Controls
- Client-side Restrictions
  - HTML: The use of form elements
  - HTML: HTML5 Validations
  - JS: Javascript Validations
- Form Submission Approaches
  - Traditional Form Submission
  - Programmatic Form Submission
  - AJAX Form Submission

# Introduction to HTTP

- Definition: HTTP is a text-based application-layer protocol that defines how content is requested from a client application and served by a web server.
  - Work on top of TCP/IP
  - Latest standard is HTTP/1.1, defined in [RFC2616](#). ([HTTP/2 drafting](#))
  - Specifications of HTTP Request and Response Headers
- Client-Server Model
  - Popular servers: Apache, Nginx, Node.js, IIS, AppEngine, etc
  - Popular clients/agents: Chrome, Firefox, IE, Safari, etc
  - (Demo) Using telnet to make a simple request

# Client-Browser Model



# Surfing the Web using Telnet

```
$ telnet www.cuhk.edu.hk 80
```

```
Trying 137.189.11.73...
```

```
Connected to www.cuhk.edu.hk.
```

```
Escape character is '^['.
```

```
GET / HTTP/1.1
```

```
Host: www.cuhk.edu.hk
```

```
HTTP/1.1 200 OK
```

```
Date: Mon, 26 Jan 2015 17:00:28 GMT
```

```
Server: Apache/2.0.52 (Unix) DAV/2 mod_ssl/2.0.52 OpenSSL/0.9.7d PHP/5.0.2
```

```
Content-Location: index.html.en
```

```
Vary: negotiate,accept-language
```

```
TCN: choice
```

```
Last-Modified: Mon, 14 Dec 2009 09:42:15 GMT
```

```
ETag: "10c43b-19d-16a9cbc0;c0440-10a-a080d780"
```

```
Accept-Ranges: bytes
```

```
Content-Length: 413
```

```
Content-Type: text/html
```

```
Content-Language: en
```

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>The Chinese University of Hong Kong</TITLE>
```

```
<META HTTP-EQUIV="CACHE-CONTROL" CONTENT="NO-CACHE">
```

```
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
```

```
<META HTTP-EQUIV="Expires" CONTENT="-1">
```

```
<META HTTP-EQUIV="Refresh" CONTENT="0; URL=/english/index.html">
```

```
</HEAD>
```

```
<BODY>
```

```
Being directed to The Chinese University of Hong Kong home page according to your browser language preference
```

```
</BODY>
```

```
</HTML>
```

# Typical HTTP Requests

- GET request:

```
GET /~ierg4210/lectures/incl/process.php?q=abc HTTP/1.1
Host: course.ie.cuhk.edu.hk
```

- POST request:

```
POST /~ierg4210/lectures/incl/process.php?q=abc HTTP/1.1
Host: course.ie.cuhk.edu.hk
Content-Length: 102
Content-Type: application/x-www-form-urlencoded
```

```
name=Adon&gender=M&email=phfung%40ie.cuhk.edu.hk&address=SHB%2C+CUHK%2C+NT@ion=NT&action=updateInfo
```

- Specifications:

- Request Version: HTTP/1.0, HTTP/1.1
- Request Method: GET, POST, PUT, HEAD, DELETE, CONNECT, etc...
- Request Parameters: query string v.s. body
- Request Headers: specifying hostname, content-length and content-type

# Typical HTTP Response

HTTP/1.1 200 OK

Date: Mon, 26 Jan 2015 17:00:28 GMT

Content-Length: 413

Content-Type: text/html

<HTML>...</HTML>

- Specifications:
  - **Response Version:** HTTP/1.0, HTTP/1.1
  - **Response Status:** 1xx for Informational, 2xx for Successful, 3xx for Redirection, 4xx for Client Error, and 5xx for Server Error
  - **Response Headers:** content-length, content-type, and many more for authentication, cookies, security, caching, redirection, etc...
  - **Response Body:** the content

# HTML Forms

- The most prevalent approach to solicit information from users
- Technically, a <form> tag that comprises different form controls including <input>, <textarea>, and <select>
- A typical example that asks for user's information:

```
<fieldset>
  <legend>Personal Information</legend>
  <form method="POST"
action="https://course.ie.cuhk.edu.hk/~ierg4210/lectures/incl
/process.php?q=abc"><ul>
  <li>
    <label>Name*</label>
    <div><input type="text" name="name" required /></div>
  </li>
  <li>
    <label>Gender*</label>
    <div><input required type="radio" name="gender"
value="M" /> M <input type="radio" name="gender" value="F" />
F</div>
  </li>
  <li>
    <label>Email*</label>
    <div><input type="email" name="email" required
placeholder="john@example.com" /></div>
  </li>
  <li>
    <label>Address*</label>
    <div><textarea name="address" required></textarea>
  </div>
</li>
```

HTML

## Personal Information

- Name\*

- Gender\*

☐ M ☐ F

- Email\*

- Address\*

- Region\*

Send

\* denotes a required field.



# <form> Attributes

```
<form method="POST" action="/~ierg4210/lectures/incl/process.php?q=abc">  
  <!-- include some input controls here -->  
</form>
```

HTML

- `method="POST"` or `method="GET"` (default: GET)
  - `POST` is mainly used to `make changes on server` data, while `GET` is used to `retrieve data` only
- `action="/~ierg4210/lectures/incl/process.php?q=abc"` (default: the current URL)
  - the value takes a URL that will accept the form request
- `onsubmit="return false"` is optional
  - Often used when the form is submitted over AJAX, to be discussed in later slides
  - Avoid it due to inline script restriction by [CSP](#)
- `enctype="multipart/form-data"` is optional
  - When `<input type="file"/>` is used for file upload

# Form Controls (1/4)

- A typical form control is defined as follows:

- `<!-- <label> is to focus on a field when clicked -->` HTML  
`<label for="field1">Field 1: </label>`  
`<input type="text" name="param1" id="field1" />`

## Most Common Controls:

- Text field

- **First Name:** `<input type="text" name="firstname" />` HTML

Password field (MUST use POST method)

- **Password:** `<input type="password" name="name" value="" />` HTML

## Hidden Field

**Hidden?** `<input type="hidden" name="action" value="updateData" />` HTML

Field 1:

First Name:

Password:

Hidden?

# Form Controls (2/4)

- Controls that offer **choices**:

- Radio box (limit to a single choice for a group of radios of the same name)

- ```
<input type="radio" name="sex" value="M" checked/
```

 HTML  

```
<input type="radio" name="sex" value="F" /> F
```

☒ M ☐ F

Checkboxes (multiple choices)

```
<input type="checkbox" name="item[]" value="A" checked /> A
```

 HTML  

```
<input type="checkbox" name="item[]" value="B" /> B
```

☒ A ☐ B

Note: the empty brackets [] are needed for PHP to receive the choices as an array

- Dropdown menu (single choice now; try adding a attribute **multiple**)

Which OS do you like:

 HTML  

```
<select name="OS">
```

```
  <option value="1">iOS</option>
```

```
  <option value="2" selected="true">Android</option>
```

```
</select>
```

Which OS do you like:

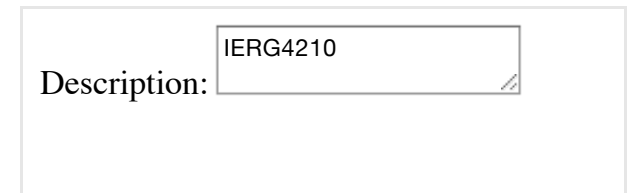
# Form Controls (3/4)

- Even More Controls:

- Textarea (Multi-line text field)

- **Description:**  
`<textarea name="desc">IERG4210</textarea>`

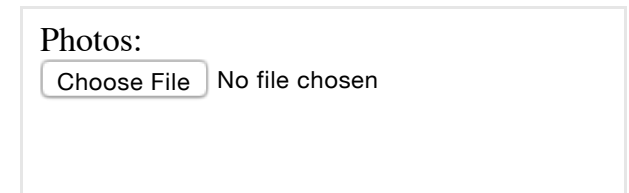
HTML

A form preview showing a label "Description:" followed by a text area containing the text "IERG4210". The text area has a small cursor icon at the bottom right.

## File Field

- **Photos:** `<input type="file" name="pics" />`

HTML

A form preview showing a label "Photos:" followed by a file input control. The control displays a "Choose File" button and the text "No file chosen".

## Submit Button

- `<input type="submit" value="Go" />`

HTML

A form preview showing a submit button with the text "Go".

## Image Submit Button (Image Credit: HSBC)

- `<input type="image" src="incl/04-go.gif" />`

HTML

A form preview showing an image submit button. The button is a blue rounded rectangle with the text "Go" in white.

# Form Controls (4/4)

- HTML 5 New Controls

- Email Field

- `<form>Email:*  
 <input type="email" name="email" required /></form>` HTML

Email:\*

URL Field with optional use of styling by new CSS pseudo selectors

- `<style>:valid{border:1px solid #0F0}  
 :invalid{border:1px solid #F00}</style>  
<form>URL: <input type="URL" name="url" /></form>` HTML

URL:

Search Field

- `<form><input type="search" name="q"  
 placeholder="Search..." /></form>` HTML

Custom Pattern Field with regular expressions

• `<form>Amount: $<input type="text" name="amount"  
 pattern="^[\\d,.]+ $" /></form>` HTML

Amount: \$

In a nutshell, HTML5 Forms introduced

- More specific-purpose textfields; Built-in support of client-side validations
- New CSS Pseudo Selectors: `:valid`, `:invalid`, `:required` and `:optional`

# Regular Expressions

- A language to recognize string patterns
- Always refer to the [Cheatsheet](#)

What you must know:

- **^** - start of string; **\$** - end of string (IMPORTANT to validations!)
- **+** - one or more times; **?** - 0 or 1 times; **\*** - 0 or more times

- **Examples:**

- Float (**\d** includes digits only):

`^[\\d\\.]+`

REGEX

- Alphanumeric (**\w** includes letters, digits, underscore):

`^[\\w\\-\\_ ]+`

REGEX

- Email:

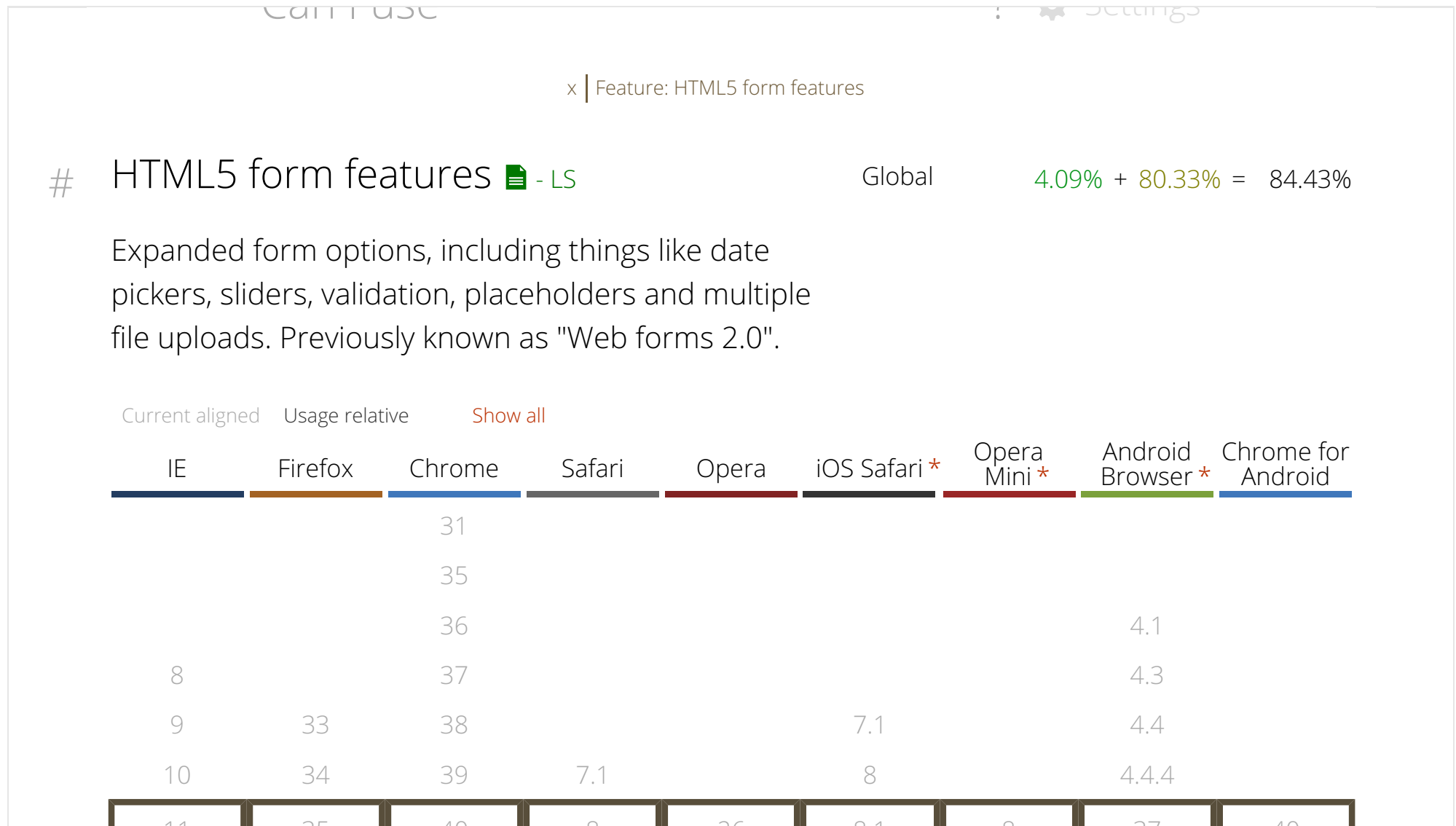
`^[\\w=+\\-\\/] [\\w=\\'+\\-\\/\\.]*@[\\w\\-]+(\\. [\\w\\-]+)* (\\. [\\w]{2,6})$`

REGEX

The regular expression for email address is readily available on Web.

**IMPORTANT: Consult creditable websites for reusable patterns!!**

# HTML5 Forms: Browser Support (1/1)



<http://caniuse.com/#feat=forms>

# HTML5 Forms: Browser Support (2/2)

## The Current State of HTML5 Forms

Browser support for the different features of HTML5 forms is quite varied. Let's explore.

[Types](#)[Attributes](#)[Elements](#)[Demo](#) • [Blog](#) • [FAQ](#) • [Features](#) • [Examples](#) • [Partners](#) • [Pricing](#) • [Gallery](#) • [Guides](#)

### The Introduction

HTML5 is the newest specification for HTML, the language that web browsers read to display web pages.

HTML5 has many new features intended to make creating websites easier and people's experience in using

### On this Page

1. [Browser Support for New](#)

<http://www.wufoo.com/html5/>



# Client-side Restrictions

# Client-side Restrictions

- To inform the users early on for input errors
  - To create a more interactive and responsive UI experience
  - Otherwise, input errors are prompted only after form submissions (round-trip delay)
- To imply a specific pattern that a user is expected to follow
  - To help users enter/choose the valid data that we need
  - **IMPORTANT: They can be bypassed by Parameter Tampering Attacks!! Don't count on them for security!!**  
Reason: A user has full control of any client-side code downloaded to his browser using the lovely **Firebug** :)
- Hence, you need input validations implemented on **BOTH**
  - **server-side for security enforcement**, and
  - client-side for better user experience.

# 3 Approaches of Client-side Restrictions

The use of different form controls (shown in previous slide)

- e.g. Radioboxes for genders implies either M or F
- e.g. Dropdown menu implies a restriction on some default choices

Validations with HTML5 (shown in previous slide)

- The first built-in support of client-side validations by IE 10+, Firefox 4+, Chrome, etc
- e.g. Email, URL, Search, and Custom fields

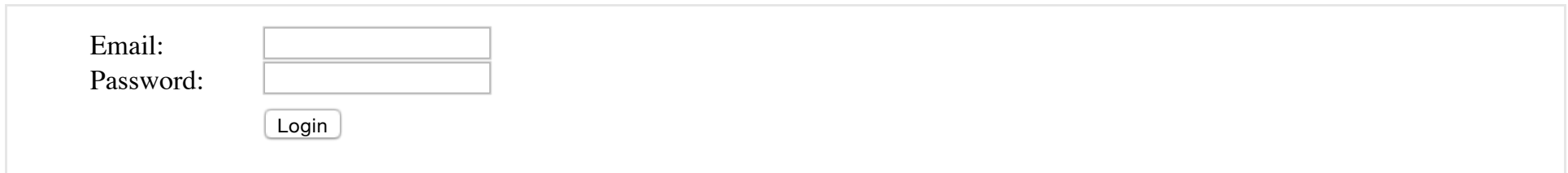
Validations with Javascript (to be discussed in next slide)

- The programmatic way to customize input patterns
- Well-supported across browsers

# Form Validations with Javascript (1/4)

- Recall the best practice: **Graceful Degradation** ([p.5 in Lect 2](#))
  - if (HTML5 supported) use the native HTML5 Validation
  - else if (JS supported) use the JS validation code
  - else the form still works without any validations

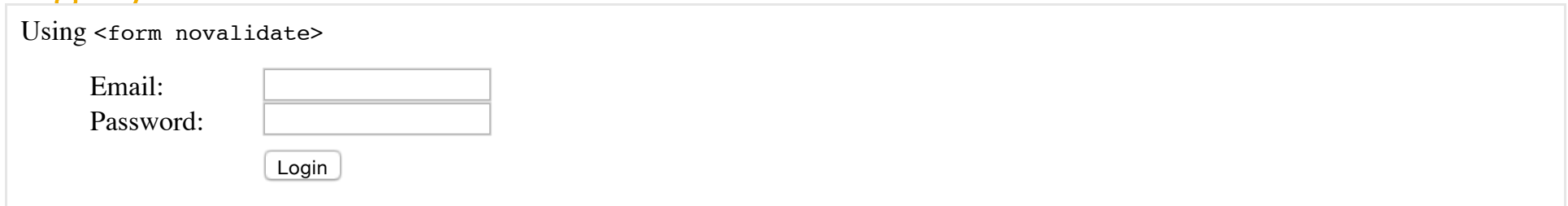
- HTML5 Mode:**



A login form in HTML5 Mode. It contains two input fields: 'Email:' and 'Password:'. Below the 'Password:' field is a 'Login' button. The form is enclosed in a light gray border.

Note: POST Parameters can be accessed only by server but not JS. Hence, nothing is shown here after submission. Firebug can show what was already sent.

- Legacy Mode:**



A login form in Legacy Mode. It contains two input fields: 'Email:' and 'Password:'. Below the 'Password:' field is a 'Login' button. The form is enclosed in a light gray border. Above the form, the text 'Using <form novalidate>' is displayed.

Note: Need some free [old-school IE browsers](#)/[saucelabs](#) for compatibility tests!?

# Form Validations with Javascript (2/4)

## The implementation:

- 1. Given a form that has a HTML5 Email field,

```
<form id="loginForm" method="POST">  
  Email: <input type="email" name="em" /><br/>  
  Password: <input type="password" name="pw" /><br/>  
  <input type="submit" value="Login" />  
</form>
```

HTML

Note: Unsupported type will fallback to an ordinary textfield

- 2. Add the title, HTML5 required and pattern attributes

```
<form id="loginForm" method="POST">  
  Email: <input type="email" name="em" title="valid email" required  
           pattern="^[\\w=+\\-\\/] [\\w=\\'+\\-\\/\\.]*@[\\w\\-]+(\\. [\\w\\-]+)* (\\. [\\w]{2,6})$" /><br/>  
  Password: <input type="password" name="pw" title="valid password" required/><br/>  
  <input type="submit" value="Login" />  
</form>
```

HTML

Note: Unsupported attributes will be ignored in legacy browsers

# Form Validations with Javascript (3/4)

3. To validate the form right before form submission:

HTML

```
<form id="loginForm" method="POST">...</form>
<script type="text/javascript">
var loginForm = document.getElementById('loginForm');
// Do this only if the HTML5 Form Validation is absent
if (!loginForm.checkValidity || loginForm.noValidate)
    // to listen on the submit event of "loginForm"
    loginForm.onsubmit = function(){
        // a private function for displayError
        function displayErr(el,msg){alert('FieldError: ' + msg);el.focus();return false}
        // looping over the array of elements contained in the form
        for (var i = 0, p, el, els = this.elements; el = els[i]; i++) {
            // validate empty field if required attribute is present
            if (el.hasAttribute('required') && el.value == '')
                return displayErr(el, el.title + ' is required');
            // validate pattern if pattern attribute is present
            if ((p = el.getAttribute('pattern')) && !new RegExp(p).test(el.value))
                return displayErr(el, 'in' + el.title);
        }
        // If false is returned above, the form submission will be canceled;
        // If false is NOT returned, the form will submit accordingly
    }
</script>
```

# Form Validations with Javascript (4/4)

- (Take-home) To also validate `radio` and `checkbox`

HTML

```
for (var i = 0, p, el, els = this.elements; el = els[i]; i++) {  
    // validate empty field, radio and checkboxes  
    if (el.hasAttribute('required')) {  
        if (el.type == 'radio') {  
            if (lastEl && lastEl == el.name) continue;  
            for (var j = 0, chk = false, lastEl = el.name, choices = this[lastEl],  
                choice; choice = choices[j]; j++)  
                if (choice.checked) {chk = true; break;}  
            if (!chk) return displayErr(el, 'choose a ' + el.title);  
            continue;  
        } else if ((el.type == 'checkbox' && !el.checked) || el.value == '')  
            return displayErr(el, el.title + ' is required');  
        }  
    if ((p = el.getAttribute('pattern')) && !new RegExp(p).test(el.value))  
        return displayErr(el, 'in' + el.title);  
}
```

[Code Demo](#). For your exercise/midterm/final, how to skip disabled/hidden controls??

# Form Submission Approaches



# Form Submission Approaches

## Traditional Form Submission (demonstrated in previous slide)

- Triggered by a submit button or the Enter key
- Fires the `submit` event, where one can register a listener through `addEventListener( 'submit' )` or `onsubmit` validate before a form submission

## Programmatic Form Submission

- Recommended to **use this only when submitting a form automatically**

```
<form method="POST" id="buildAutoPostReq"><!-- Some hidden fields here --></form>  
<script type="text/javascript">document.forms[0].submit();</script>
```

HTML

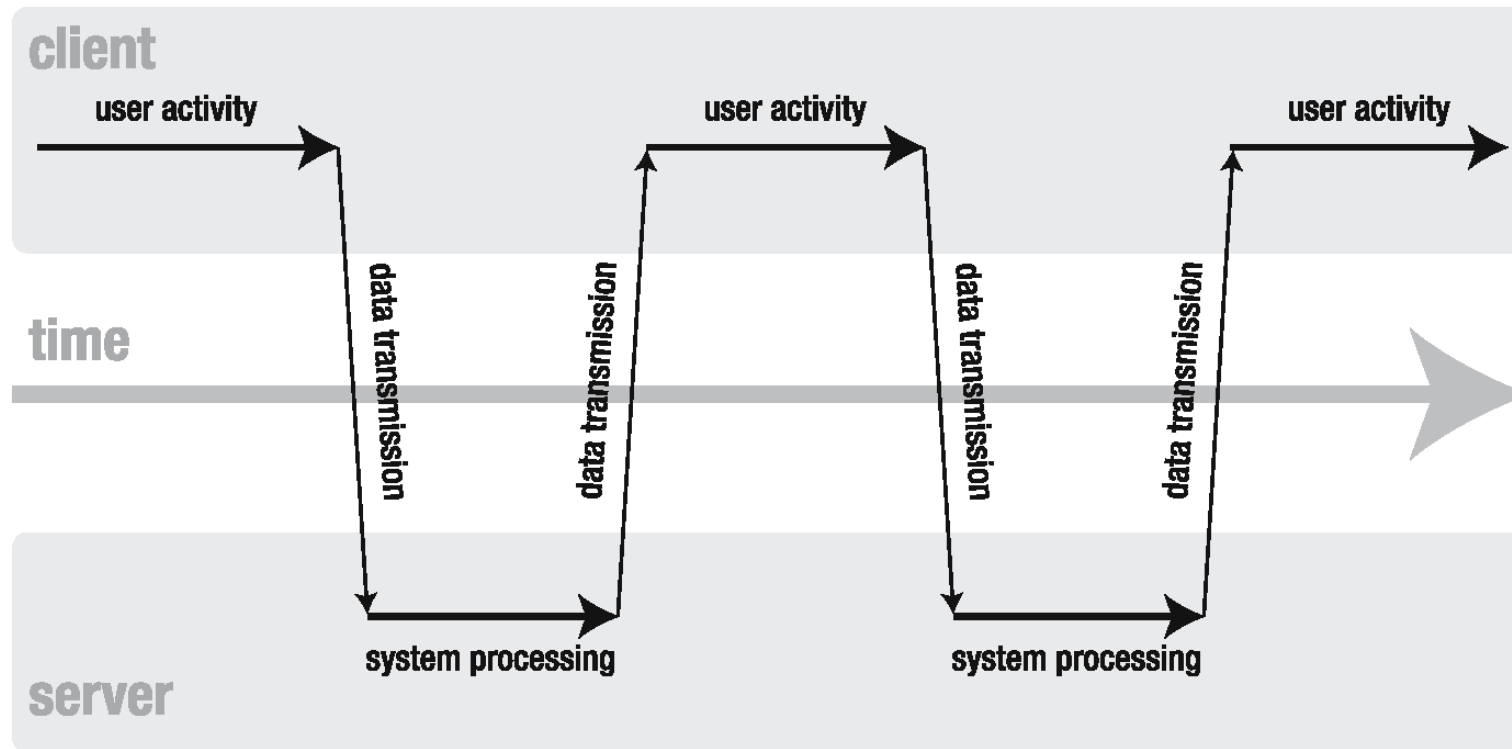
- Unfortunately, programmers (incl. HSBC) who don't know `<input type="image">` like to do this for images: When an image is clicked, `Form.submit()` will be finally called if a form is properly validated
- **Notice:** NO `submit` event is fired.

## AJAX Form Submission (implementation to be detailed next week)

- Asynchronous Javascript and XML
- In terms of implementation, it's ultimately the `XMLHttpRequest` API, but better wrap it with a library :)
- The concept on async is much more important

# Synchronous Workflow

## classic web application model (synchronous)



Synchronous calls are **blocking**. It hangs until the server returns, and could also block later requests, i.e. less efficient

# Asynchronous Workflow

Principle: Do something else while waiting

Dispatch many requests at a time. Do something else. Get notified when server returns, then render the results. The responses will likely be out of order.

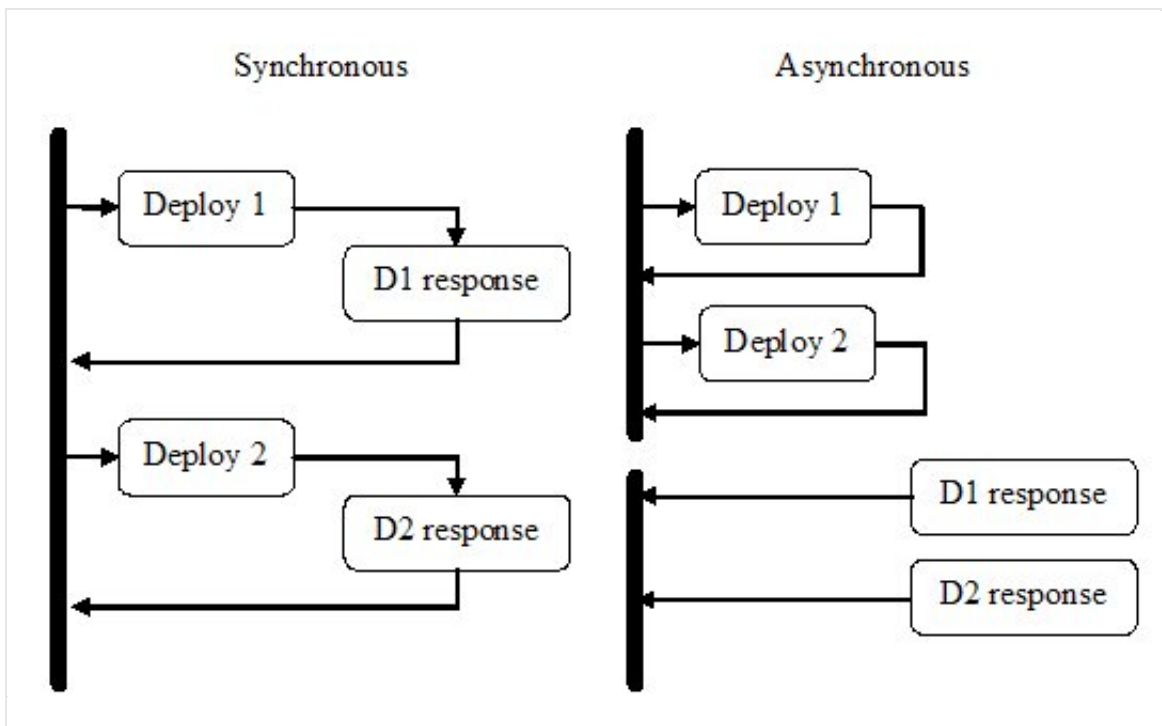


Image Source: [https://www.ibm.com/developerworks/websphere/library/techarticles/0611\\_lucas/images/fig1.jpg](https://www.ibm.com/developerworks/websphere/library/techarticles/0611_lucas/images/fig1.jpg)  
Video Source: [https://www.youtube.com/embed/cBVoNJ\\_IGV0](https://www.youtube.com/embed/cBVoNJ_IGV0) (start watching at 1m38s)

# AJAX usage

to be continued next week

# XMLHttpRequest

The screenshot shows the MDN website with the title 'Using XMLHttpRequest'. The page includes a navigation bar with links to 'TOPICS', 'DOCS', 'DEMOS', 'LEARNING', and 'COMMUNITY'. A search bar is visible with the text 'Search MDN'. The main content area is titled 'Using XMLHttpRequest' and contains a paragraph explaining that `XMLHttpRequest` makes sending HTTP requests easy. It also features a 'TABLE OF CONTENTS' sidebar with links to various topics like 'Types of requests', 'Handling responses', and 'Using FormData objects'.

**MOZILLA DEVELOPER NETWORK**  
MDN TOPICS ▼ DOCS ▼ DEMOS ▼ LEARNING ▼ COMMUNITY ▼

powered by Google™ Search MDN

Sign in mozilla ▼

MDN ▸ Document Object Model (DOM) ▸ XMLHttpRequest ▸ Using XMLHttpRequest Languages ▾ This page ▾

## Using XMLHttpRequest

HISTORY EDIT

`XMLHttpRequest` makes sending HTTP requests very easy. You simply create an instance of the object, open a URL, and send the request. The `HTTP status` of the result, as well as the result's contents, are available in the request object when the transaction is completed. This page outlines some of the common and even slightly obscure use cases for this powerful JavaScript object.

### Types of requests

A request made via `XMLHttpRequest` can fetch the data in one of two ways, asynchronously or synchronously. The type of request is dictated by the optional `async` property that is set on the `XMLHttpRequest.open()` method. If this property is set to `false`, then the `XMLHttpRequest` will be processed synchronously, otherwise the process will be done asynchronously. A detailed discussion and demonstrations of these two types of requests can be found on the [synchronous and asynchronous requests](#) page.

### Handling responses

There are several types of [response attributes](#) defined by the W3C specification for `XMLHttpRequest`. These tell the client making the `XMLHttpRequest` important

#### TABLE OF CONTENTS

- Types of requests
- Handling responses
  - Analyzing and manipulating the `responseXML` property
  - Analyzing and manipulating a `responseText` property containing an HTML document
- Using `FormData` objects
- Handling binary data
- Monitoring progress
- Cross-site `XMLHttpRequest`
- Bypassing the cache
- Security
  - `XMLHttpRequests` being stopped
- Downloading JSON and JavaScript from extensions

Ref: [https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Using\\_XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest)

# Some Logistics...

- Interactive Workshop next week
  - Will spare 30-45min for environment setup and coding
  - Some demos, then teaching teams walking around to help
  - Come with your laptop (sufficiently charged)
  - Hint: setting up [Node.js](#)
- Credit Card needed for AWS registrations. Follow Tutorial 3.