



ICOM 6034

Website engineering

Dr. Roy Ho

Department of Computer Science, HKU

Session 0: Course overview

Session 1: Enabling standards and technologies (Part 1)



Session 0

Course overview

Staff

■ Instructor

- Dr. Roy S.C. Ho
- Department of Computer Science, HKU
- Email: scho@cs.hku.hk
- A little bit about me:
 - Freelance (web) programmer in 1995-1998
 - Have been managing web projects since 2000
 - Obtained PhD (research areas: operating systems and networking) in 2007
 - Have been involved in a number of open-source projects, including network protocol development for Internet2 (for bulk data transfer)
 - Have been teaching ICOM6034 since 2007

■ Teaching assistant

- Mr. Steven Chu
- Email: stevenchu@ecom-icom.hku.hk
- A little bit about Steven:
 - Has been managing web projects since 2000
 - Has been the TA of ICOM6034 since 2007

Dates and arrangements

- 10 sessions:

Part 1 Web development basics	Jan 16	Saturday afternoon
	Jan 20	Wednesday evening
	Jan 27	Wednesday evening
Part 2 Design and implementation of web applications	Jan 29	Friday evening
	Feb 3	Wednesday evening
	Feb 5	Friday evening
Part 3 Interoperability of web applications and services	Feb 8	Monday evening
	Feb 19	Friday evening
	Feb 24	Wednesday evening
Part 4 Optimizations	Feb 27	Saturday afternoon

- **Teaching mode:** hybrid (i.e., F2F lectures + online streaming + audio recording)
- Yellow/green = labs/demos = Lecture (~1.5 - 2 hours) + lab/demo (~1 - 1.5 hours)
 - **You are very welcome to attend the labs in-person if you need more technical help and if the situation allows**
- Venue: P603
 - **Please bring a notebook for the labs.**
- Exam: May 3-18, 2021 (TBA)

Course materials

- No textbook
- Lecture notes will be provided
- Pre- or post-class self-learning resources given in Moodle
 - Mainly online, technical materials, not covered in the exam

Assessment

- [30%] 1 take-home, individual assignment
 - An extended version of the lab work
 - (so doing the labs will save you time)
- [40%] 1 group project, details to be announced
 - Each group may have 1-3 members (2 by default)
 - A “single-person mode” (with reduced workload) for those who wish to work on their own
 - But forming a proper group is highly encouraged as you can obtain much better peer support when you encounter any technical problems
 - (so it would be good for you to get to know each other earlier)
- [30%] an open-book, online examination
 - Scope: all lecture slides
 - All pre-/post-class readings, external links, labs, assignments, and demos will not be covered



Organization

- **Part 1:** web development basics
 - 3 sessions
- **Part 2:** design and implementation of web applications
 - 3 sessions
- **Part 3:** interoperability
 - 3 sessions
- **Part 4:** optimizations
 - 1 session

Part 1: Web development basics

(3 sessions)

- A very quick review of the basic web technologies
 - X/HTML 4/5, CSS, JavaScript, server-side scripting, web security, mobile/multi-device supports, internationalization, etc.
- Introduce the standardization process of web technologies, and the importance of standard-compliance
- Selection criteria of web technologies
 - There are so many web technologies, and new ones appear every month – how to select the appropriate ones?

Part 2: Design and implementation of web applications (3 sessions)

- Q: Was it possible to create this web page using **plain** HTML/CSS/JavaScript introduced in Part 1?

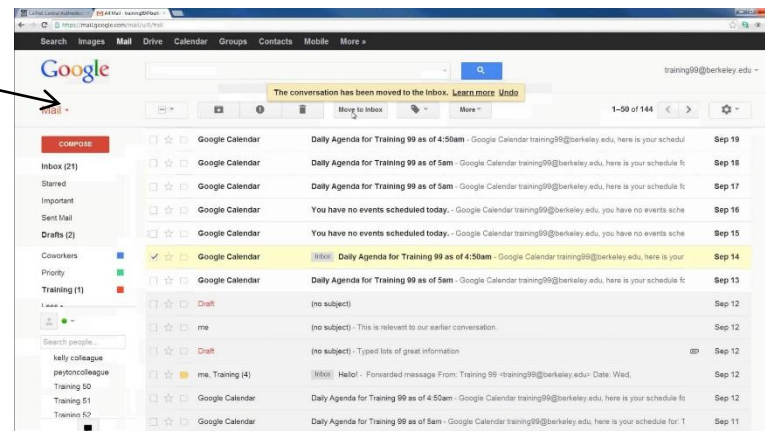
A: Yes – it could be done by using HTML tables alone.



Yahoo.com
in 1999

- Q: What about this?

A:



Gmail.com
~2010 -
present

Part 2: Design and implementation of web applications (3 sessions)

■ Web 2.0 in focus

- In “Web 1.0”, websites simply displayed static information (texts + pictures), but now (in “Web 2.0”) many web sites (or “web applications”) function like complete, desktop applications
- Much more real-time interactions with users
 - I.e., The implementation at the client-side (or “frontend”) has become much more complicated
- Much more complicated application logic at the server-side

■ Those basic technologies (introduced in Part 1) are too low-level, and inefficient, for developing large-scale, feature-rich websites/applications

■ We need tools for rapid and maintainable website/webapp development

- E.g., Huge Inc. (<https://www.hugeinc.com/>)
 - A possible tool: pagePiling.js (<https://alvarotrigo.com/pagePiling/>)

■ Objective: to (re)use established design patterns, frameworks and libraries.

■ Some popular tools will be introduced:

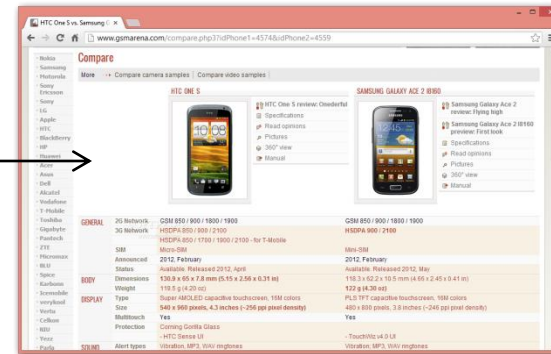
- Client-side JavaScript libraries (e.g., jQuery, plugins, etc.)
- Server-side frameworks:
 - MVC frameworks (e.g., Laravel, Ruby on Rails)
 - Content management systems (CMS) => building websites using “plug-and-play” modules, e.g., Drupal

Part 3: interoperability (3 sessions)

- Goal #1: there are lots of contents and functions already available in other websites; we want to retrieve and include them in our own websites
 - Examples: YouTube videos, online calendars, maps, Amazon product info, weather info, flight/hotel info, updated event lists,, etc.
 - E.g., obviously, you do not want to re-create a “Google Maps” by yourself just for displaying the physical locations of your stores
 - **Reusing existing contents/functions** is crucial for i) shortening development time; and ii) providing updated/comprehensive contents and stable functions
- Goal #2: how to share the contents and functions of your own websites to others?
- Topics covered:
 - Standard data formats for data sharing
 - **Web API protocols, and “mash-ups”**
 - Cloud services for **simplifying** website development

Part 4: optimizations (1 session)

I've created an awesome website, but the traffic is low...
:(:~



I've made a great website/webapp, but it is loaded slowly in browsers. The user experience is bad...

Part 4: Optimizations

- Search engine optimizations (SEO)
- Techniques for improving the rendering speed of webpages

Summary of scope

Part 1 (lectures 1-3):



Websites becoming “web apps” -> more sophisticated logic, lots of user interactions, “Web 2.0”, etc.

Part 2 (lectures 4-6):



Integration and interoperability issues -> how to reuse remote data?

Part 3 (lectures 7-9):

Part 4 (lectures 10):

You have a great website, how to make it more popular, and loaded fast at clients' computers?

Optimizations

YouTube, Gmail, Amazon, online databases, Maps, updated event lists, YOUR websites,

The web/cloud(s)

Our approaches (1/2)

- “Breadth” over “depth”
 - Website engineering is a very broad subject
 - For any given task, there are tens of tools available
 - Impossible to cover everything in-depth.
 - And probably not useful to do so?
 - We will cover a broad spectrum of technical topics – standards, design frameworks, rapid development tools, security, interoperability, performance and popularity, etc.
 - For each topic: core concepts + 1 or 2 example technologies (covered in labs/demos)
- By focusing on the “big picture” and the concepts, one can:
 - (For developers) Easily master new technologies because most of them are conceptually similar to each other (e.g., JSP is similar to PHP, etc.), and know which topics have to be studied further
 - (For project managers) Oversee the quality (of the design and implementation) and the completeness of a website project

Our approaches (2/2)

- Programming details
 - Programming details will be covered to a depth only for illustrating the engineering concepts
 - e.g., we can't explain every function call in JavaScript
 - Labs, demos and assignments to help
 - Self-study is expected
 - Please refer to the post-class readings (all are online tutorials and e-books)
- A purely **open-source** and **standard-compliant** approach:
 - All tools for development are standard-compliant and/or open-source
 - To experience with open-source tools as cost-effective alternatives to proprietary solutions

Limitations & assumptions

- With 20+ years of active development, “website engineering” is a very broad subject
 - It is difficult to even give an *introduction* in 10 lectures
- We assume that you have:
 - Knowledge on basic web technologies (including HTML, CSS, JavaScript, and PHP); and
 - The experience of coding based on documentation.

A friendly reminder:

- The labs, assignment and group project will involve web programming, which could be difficult if you do not have any experience with software development. So, if you do not have time to self-study technical details, then this course might not be suitable for you.



Pre-lecture survey

What are your expectations?

- “Gain some hands-on experience on basic web programming”
 - *Yes, these will be covered.*
- “Latest / up-to-date technologies” / “new trends”
 - *The covered technologies may not be the “newest” ones (new ones appear almost every month; many of them don’t last), but they are mature and widely-used in real websites, and able to help illustrate the engineering concepts that we want to introduce.*
 - *These concepts should enable us to easily master those “new” technologies since most of them are conceptually similar to the “old” ones.*
- “UI / UX / graphics designs” / “trendy websites” / “modern designs”
- Technologies for web accessibility (e.g., those for people with disabilities)
 - *We will not cover these topics due to time constraints (and because I’m not a UX/graphics designer). However, the techniques for accommodating flexible/adaptive UI designs (e.g., for desktops/mobile devices, and the concept of “separation of concerns”) will be introduced.*
 - *We will name quite a number of rapid development tools in Part 2; many of them provide convenient tools for building decent UI elements. Playing with their demos is a good way to feel what updated UI of websites can look like.*
 - *A simple example: pagePiling.js*



What are your expectations?

- Expecting an introduction to website engineering, and some good practices:
 - “How HTML, CSS, JavaScript and PHP tie together to form what we see online”
 - *These will be covered.*
- “Understand what elements that make a ‘good’ website”
 - *We will focus on the good elements/practices related to the internal implementation of websites but not UI (graphics) designs*
- Expecting to become a professional developer
 - “To set up a fully-functional e-commerce website”
 - *This course can help one to learn to be a developer or a web project manager*
 - *But being a professional developer (or to set up a production e-commerce website) requires much more time-consuming, hands-on work experience which can’t be gained/taught in 10 lectures...*
 - *The post-class readings, which cover in-depth technical details can help; please use them and code.*
- Project management / business issues, e.g., “whole process of website project” / “project lifecycle”, “resource planning (both computing and human resources)”, “e-marketing tools”, DevOps, etc.
 - *Due to time constraints, we may not be able to cover these areas, but feel free to contact me for related readings.*

Previous experiences in web development

- “I have used Laravel, React, jQuery, Angular, Ruby on Rails, WordPress, or PHP/MySQL, etc., before in work”
- “I have done SEO before in work”
 - *Then some parts of this course may be too easy for you.*
 - *This course will focus on “breadth”, which may enrich your knowledge in other areas.*
- “I have no experience in computer programming or HTML/CSS/JS/PHP, etc., - even in assignments/projects”
 - *This course will require you to do programming, which will be difficult to those who have no prior experience.*
 - *If you plan to self-learn the required skills, please make sure that you know basic HTML/CSS/JS/PHP programming before the end of Part 1. We will also do a quick review of these technologies, and provide related post-class readings in Part 1.*
 - *If you have little technical experience and do not have spare time for self-studies, then you may still be able to follow the lectures (because we will focus on the concepts which aren’t difficult), but it will be difficult for you to finish the assignment/project.*

Specific topics expected

- HTML, web APIs, client/server-side scripting, MVC, JSON, SEO, etc.
 - *We will introduce the basic usage of these technologies; more programming details and advanced features can be learnt through the post-class readings.*
- Python / Java / SQL / server maintenance / Apache Tomcat configuration / mobile apps, etc.
- How to fix errors/bugs in websites
 - *We probably won't have time to cover these general (or platform-dependent) topics in CS/IT.*
- Specific website components (e.g., membership management facilities, chatbot, shopping cart, marketplace, payment gateway, etc.)
 - *We can't cover the implementation of these components one-by-one (because there are so many commonly-used components). But most of these components are readily-available on the web – the concepts/tools covered in this course should enable you to explore and incorporate most of these components into your websites.*

Feel free to contact me if you wish to learn about any web technologies that are not covered.

Specific topics expected

- “Full-stack” web technologies
- MEAN / MERN stacks (MongoDB, Express, Angular/React, Node.js)
- React / Vue / Gatsby, etc.
 - *We will focus on the “classic” web stack with a client-side JS library (e.g., jQuery) and a server-side MVC framework (Laravel), which is both widely-used and versatile.*
 - *The following technologies involve more advanced concepts and/or programming; might be too technical to be covered in this course:*
 - *NoSQL (for big data) and so MongoDB (MongoDB is a NoSQL database)*
 - *Express, Angular, Gatsby, and Node.js*
 - *However, the concept of client-side rendering (which Express, Angular, and Vue are based on), and also React (which Gatsby is based on) will be introduced in Part 3 but not in-depth.*
- Progressive web apps (PWA)
 - *We will focus on general web apps. Platform-dependent and other advanced topics (e.g., access to mobile device’s sensors, offline operations, background computation with “workers”, etc.) maybe too technical to be covered in this course.*

Feel free to contact me if you wish to learn about any web technologies that are not covered.

Questions?

- Discussion group on Moodle
- Email
 - ☐ Me about the course content
 - ☐ Steven about the labs and the assignments
 - ☐ Any of us if you wish to make an appointment for meeting (f2f or online)
- Discussions before/during/after lectures
 - ☐ I will try to appear 15-30 minutes before each lecture
- Discussions on related topics not covered are welcome.



Session 1

Enabling standards and
technologies



Session 1 objectives

- How web technologies are standardized and the importance of standard-compliance
- Quick review of some enabling client-side technologies:
 - HTML/XHTML
 - CSS
 - JavaScript
- Lab 1A: Our development platform, and some simple HTML and CSS customizations



Web standards

Compatibility & interoperability

- The web is all about **compatibility** and **interoperability**
 - Tens of platforms and implementations of clients and servers
 - Desktops, tablets, mobile phones, etc...
 - Ideally, **all** information sent from **any** source (server) can be displayed correctly in **any** destination (client browser)
 - Impossible to achieve without **standards**

W3C

- World Wide Web Consortium (W3C) - the web standardization body
 - First formed to refine HTML
 - Has standardized almost all important standards relating to the web:
 - E.g., HTML, XML, XHTML, HTML5, CSS, JavaScript, etc.
 - Development of standards:
 - Working drafts (least mature)
 - To be revised and modified by the community
 - Candidate recommendations
 - Further revisions; big changes not expected
 - Proposed recommendations
 - To be submitted to W3C Advisory Council for final approval
 - Recommendations (most mature, similar to “standards”)
 - E.g., HTML5 was still a candidate recommendation in 2012, but has become a recommendation since 2014.

IETF

- Internet Engineering Task Force
- Mainly responsible for standardizing the “lower-level” stuff, e.g., network protocols.
 - TCP/IP, HTTP.....
- Develops Internet standards
 - Request for comments (RFC) (least mature)
 - Proposed standards
 - Draft standards
 - Standards (most mature)

WHATWG

- The Web Hypertext Application Technology Working Group (**WHATWG**) was formed to draft the **Web Applications 1.0 specification** in 2004, part of which later became HTML5
- Founded by the community and vendors such as Apple, the Mozilla Foundation and Opera Software.
- Formed in response to the slow work of W3C, and its plan to abandon HTML in favor of XML-based technologies (e.g., XHTML).
 - Many developers were tired of writing XML at that time...
- In 2007, upon WHATWG's recommendation, the W3C HTML working group adopted the WHATWG's HTML5 to form W3C's HTML5.
- Inputs related to HTML5 (and “web applications” in general) are still being sent from WHATWG to W3C nowadays

Advantages of standards adoption

- Interoperability & compatibility
- “Open-ness”: most open standards (e.g., W3C’s) are evolved by the **community**
 - People really use them – and improve them
 - Higher chance that there are good **cross-platform reference implementations**
- A widely-adopted standard usually has better forward and backward compatibility
 - There exists a **proper upgrade path** even if a standard becomes obsolete
 - A (bad) example – Microsoft VBScript (something similar to JavaScript, but proprietary)
 - Fairly popular in late 1990s and early 2000s
 - Replaced by ASP.NET for web development since 2007
 - Loss of investment in software and expertise

Some selection criteria of web technologies

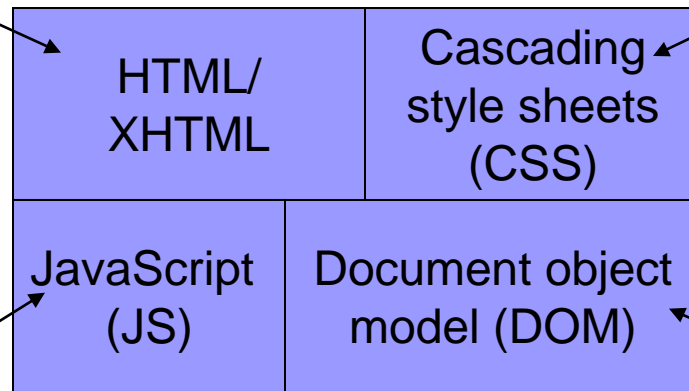
- There are so many technologies for web development; how to select?
- Fit your needs – function, performance, adoption/deployment/maintenance costs,
- **Standards-compliant** as far as possible
 - Also check the “future”/roadmap of the underlying standards
- Have **good reference implementation(s)**
 - What do we mean by “good”?
 - Cross-platform, cross-browsers
 - The reference implementations have been widely used for some time
 - Stability
 - Most problems already have solutions or workarounds
 - The technology is still and **will be** actively maintained and developed
 - (Q: is Adobe Flash a good choice in 2021?)
- **Open-source** (with a license approved by the Open Source Initiative (OSI))
 - Better quality, reliability, and flexibility
 - Can know which features are standard-compliant and which are not
 - Community support – Interoperability/compatibility issues are known/solved more easily



Enabling client-side technologies

HTML + CSS, JS, DOM

Define the **structure**
of web documents



Define the **presentation**
details of individual
HTML elements

Define the **logic** and
webpage's **behaviors**

Naming of web page
components (for use
by JS)

These four components give control over how HTML elements are presented at the client-side, and allow them to change - **without** relying on the server (i.e., without loading a new page)



Quick review:

1. HTML/XHTML

HTML – the core of client-side technologies

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

HTML was originally designed to define the **structure** of web documents – nothing else.

A brief history of HTML:

HTML Versions	
HTML	1991
HTML+	1993
HTML2.0	1995
HTML3.2	1997
HTML4.01	1999
XHTML1.0	2000
HTML5	October 2014

HTML was invented.

Early days of the web; mainly used by the academia.

The “Dot-com bubble” time, when the web started to become really popular.

It took 15 years for HTML to be updated from version 4 to 5...

Main problems of HTML 4 (and why XHTML was proposed)

- HTML 4 was the version of HTML that was widely-used during the “Dot-com Bubble” time in late 1990s to early 2000s
- Problem 1: the web was still a new thing at that time even to developers, many HTML 4 pages were created with syntax errors; and
- Problem 2: browsers were too “forgiving”...
- E.g., the following HTML document still worked in most browsers (even without `</h1>` and `</html>`)!

```
<html><body><h1>Bad HTML</body>
```

- Results:
 - Browsers had to spend much time in **correcting errors** – they became very complicated and slow
 - Difficult for programs to **extract** and **reuse** the content stored in HTML files

XHTML

- Extensible Hypertext Markup Language
 - Designed to replace HTML 4
 - Has a much stricter syntax than HTML 4
 - Conforms to the XML (extensible markup language) standard
- XHTML 1.1 is the most common XHTML variant used nowadays
- No error correction. A browser should display an error once it encounters one in an XHTML document, so it can run faster
- To avoid errors, documents can be **pre-validated** by XML parsers => cleaner and error-free

Differences between XHTML and HTML (1/2)

- XHTML elements must be **properly nested**

`<i>bold and italic</i>` is *wrong*

- This is correct:

```
<html>
<head> ... </head>
<body><h1>...</h1> ... </body>
</html>
```

- Tag names must be in **lowercase**
- All XHTML elements must be **closed**
 - `<h1>...</h1>`
 - Non-containers still have to be closed:
`
`, `<hr />`, ``

Differences between XHTML and HTML (2/2)

- Attribute names must also be in lower case
 - Example: `<table width="100%">`
- Attribute values must be quoted
 - Example: `<table width="100%">`
- Attribute minimization is forbidden
 - Example: `<frame noresize="noresize">`, cannot be minimized to `<frame noresize>`
- For client-side manipulation (i.e., by JavaScript), the `id` attribute is used instead of the `name` attribute
 - Wrong: ``
 - Right: ``
 - Right: ``

A simple XHTML document

```
■ <!DOCTYPE html PUBLIC
  "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>A simple document</title>
  </head>
  <body>
    <p>A simple paragraph.</p>
  </body>
</html>
```

} DOCTYPE declaration

DOCTYPE declaration (1/2)

- A **DTD**, or “Document Type Definition” defines which HTML elements are allowed for the current XHTML document
- Every XHTML document must begin with one of the following **DOCTYPE** declarations (DTDs):
 - `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">`
 - `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`
 - `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">`

DOCTYPE declaration (2/2)

■ Strict

- Used for clean markup – with structural information only (e.g., <body>, <p>, <h1>, etc.). **No** presentation information is allowed (i.e., no , color, or font size information, , <i>, etc.)
- Instead, use CSS to define how the webpage components should look like
- Best approach for separating document structure (HTML) and presentation details (CSS)

■ Transitional and Frameset

- Allow some obsolete HTML elements and presentation definitions (e.g., font size, etc.)
 - Not encouraged; should be avoided – more on this later

■ List of HTML tags allowed in each DTD:

- <http://w3schools.sinsixx.com/tags/default.asp.htm>

Validation tool

- W3C HTML Validation Tool
<http://validator.w3.org/> is an HTML form for checking the syntax of HTML and XHTML documents

Given HTML5 (which we will introduce in Session 2), is XHTML still relevant nowadays?

- Yes and No.
- Yes?:
 - For professional developers, knowing XHTML is important because there are still many websites written in XHTML (or even HTML 4) waiting to be updated to HTML5
 - Examples: china.com.cn, setn.com, tvb.com (HTML 4!), some sub-domains in amazon.com,, etc.
 - Some e-book formats (e.g., EPUB) and email clients are using XHTML, too.
 - Many web APIs (which we will introduce in Part 3) serve only XHTML/XML data.
 - => So XHTML can still be considered relevant - at least for these few years.
- No?:
 - HTML5 supports many powerful features that XHTML doesn't. So, HTML5 should always be used in new projects.



Quick review:

2. CSS

CSS for **separation** of structure and presentation

- HTML was originally designed to define the **structure** of a document. E.g., `<p>` means “this is a paragraph”; `<h1>` means “this is a heading”; `<table>` means “this is a table”, and so on.
- But “newer” browsers (in late 1990s to early 2000’s) and HTML standards (versions ≤ 4) added some formatting details, e.g., ``, ``, `<i>`, etc., which mixed **presentation details** with the **document structure**
 - But, what if the same page is to be displayed in another browser or device?
 - Difficult to support user-defined formats (e.g., “themes”)
 - → Long HTML files with clumsy JS codes for correcting display
 - → **messy, or “spaghetti” code**



- Cascading Style Sheets (**CSS**) came to rescue
- CSS defines **how** HTML elements (e.g., headings, text, etc.) are displayed (e.g., font size, color, position, etc.)

Cascading Style Sheets

■ Advantages:

- **Separate** presentation details from document structure...
- So that HTML files can be left clean (i.e., maintainable and extensible)
- Can be used for XHTML and HTML 4/5 pages
- One can:
 - Apply the same CSS to all pages in a website => all share the same “look and feel”
 - Apply different CSS files in different conditions (e.g., themes, mobile devices, etc.)

■ However, most browsers support CSS a bit differently

- Check browser compatibility before using some uncommon CSS features (e.g., see caniuse.com)

CSS syntax

- Very simple – just a list of **selectors** (which select HTML tags) and **descriptors** (to tell what to do with the selected elements):
 - Example: `h1 {font-family: Verdana; color: green}` means that everything included in `h1` (HTML heading level 1) tags should use the Verdana font and be colored green
- The general syntax is:
 - `selector { property: value }`
 - or
 - `selector, ..., selector {`
 - `property: value;`
 - `...`
 - `property: value`
 - `}`
 - A semicolon for separating property:value pairs

Selectors

- A selector can be any HTML tag:

```
body { background-color: #ffffff }
```

- You can use multiple selectors:

```
h1, h2 {color: red}
```

You can repeat selectors:

```
h1, h2, h3 {font-family: Verdana; color: red}  
h1, h3 {font-weight: bold; color: pink}
```

- When multiple styles are assigned to the same element, the last one overrides earlier ones

More examples

- `/* This is a comment - mypage.css */`
- `h1,h2,h3 {font-family: Arial, sans-serif;} /* use 1st available font */`
- `p, table, li, address {` `/* apply to all these tags */`
 `font-family: "Courier New";` `/* quote values containing spaces */`
 `margin-left: 15pt;` `/* specify indentation */`
 `}`
- `p, li, th, td {font-size: 80%;}` `/* 80% of size in containing element */`
- `p {background-color:#FAEBD7}` `/* colors can be specified in hex */`
- `h1,h2,h3,hr {color:saddlebrown;}`
- `a:link {color:darkred}` `/* an unvisited link */`
- `a:visited {color:darkred}` `/* a link that has been accessed */`
- `a:active {color:red}` `/* a link now being "clicked" */`
- `a:hover {color:red}` `/* when the mouse hovers over it */`

The class attribute

- The `class` attribute allows you to define different styles for elements of the same HTML tag
 - In X/HTML:

```
<p class="important">The is an important note!</p>  
<p class="footnote">Offer ends 2/10/2018.</p>
```
 - In CSS:

```
p.important {font-size: 24pt; color: red}  
p.footnote {font-size: 8pt}
```
- To define a selector that applies to any element of that class (be it a `<p>` or a `<h1>`, etc.), simply omit the tag name (but keep the dot):

```
.footnote {font-size: 8pt}
```

The id attribute

- The `id` attribute identifies a specific element:

- In X/HTML:

`<p id="important">... </p>`

- In CSS:

`p#important {font-style: italic}` or
`#important {font-style: italic}`

- `class` and `id` can both be used, and can share the same name:

`<p class="important" id="important">`

- **But:** Unlike `class`, `id` should be unique – no two elements should share the same `id`.

div and span

- **div** and **span** are HTML container elements whose only purpose is for hosting CSS information
- **div** ensures there is a line break before and after (similar to a paragraph); **span** does not
- Example:
 - HTML: `<div>This div is treated like a paragraph, but this span is not.</div>`
 - CSS: `div {background-color: #66FFFF}`
`span.color {color: red}`
- To select an element inside a `<div>`:

```
<div class="header">  
  <p>This website uses CSS!</p>  
</div>
```

```
div.header p { /* some styles */ }
```

All “p” elements that are “child elements” of “div.header”

Ways to use CSS (1): External CSS

- CSS is placed in a separate file
- Best **separation**; recommended
- In HTML, within the `<head>` element:
`<link rel="stylesheet" type="text/css" href="Style Sheet URL">`
- Or in XHTML:
`<?xml-stylesheet href="Style Sheet URL" type="text/css"?>`
- Note: `"text/css"` is the MIME type

Ways to use CSS (2): Embedded CSS

- CSS is placed within the header section of the HTML doc
- In HTML, within the `<head>` element:
`<style type="text/css">`
`<!--`
CSS Style Sheet
`-->`
`</style>`

Ways to use CSS (3):

Inline CSS

- CSS is defined inline within an HTML element
- The `style` attribute can be used in any HTML element:
`<html-tag style="property: value">` or
`<html-tag style="property: value;
property: value; ...; property: value">`
- Useful if you only want a small amount of markup, or used as a “dirty fix” for some urgent client requests
- Disadvantages:
 - Mixes presentation information with HTML – poor **separation**, not recommended
 - Some CSS features cannot be used in inline CSS

Cascading order

- CSS styles are applied to HTML elements in the following order:
 1. Browser default
 2. External CSS
 3. Embedded CSS (inside the `<head>` tag)
 4. Inline CSS
- When styles conflict, the last one wins

Example of cascading order

- External style sheet:

```
h3 { color: red;
      text-align: left;
      font-size: 8pt
    }
```
- Embedded style sheet:

```
h3 { text-align: right;
      font-size: 20pt
    }
```
- Resultant attributes:

```
color: red;
text-align: right;
font-size: 20pt
```

Shorthand properties

- In general, properties of the same type (e.g., font-related) can be combined:

```
h2 { font-weight: bold; font-variant: small-caps;  
font-size: 12pt; line-height: 14pt; font-family: sans-  
serif }
```

can be written as:

```
h2 { font: bold small-caps 12pt/14pt sans-serif }
```

Font properties and values

(for reference only)

■ font-family:

- inherit (same as parent)
- Verdana, "Courier New", ... (if the font is available)
- serif | sans-serif | cursive | fantasy | monospace
(Generic: your browser decides which font to use)

■ font-size:

- inherit | smaller | larger | xx-small | x-small | small | medium | large | x-large | xx-large | **12pt**

■ font-weight:

- normal | **bold** | bolder | lighter | 100 | 200 | ... | 700

■ font-style:

- normal | italic | oblique

Colors and lengths

(for reference only)

■ color: and background-color:

- aqua | black | blue | fuchsia | gray | green | lime | maroon | navy | olive | purple | red | silver | teal | white | #FF0000 | #F00 | rgb(255, 0, 0) | ***Additional browser-specific names (not recommended)***

■ Units for measurement:

- px, %
 - pixels, percentage of the container's size
- in, cm, mm, pt, pc
 - inches, centimeters, millimeters, points (1/72 of an inch), picas (1 pica = 12 points)

Some text properties and values

(for reference only)

- **text-align:**

- left | right | center | justify

- **text-decoration:**

- none | underline | overline | line-through

- **text-transform:**

- none | capitalize | uppercase | lowercase

- **text-indent**

- ***length*** | **10%** (indents the first line of text)

- **white-space:**

- normal | pre | nowrap

Use proper names for HTML elements

- CSS is designed to *separate document structure from style*
 - Therefore, the names (e.g., class/id names) used in HTML should always describe *document structure, not style*
- (Bad) example:
 - Suppose you define `div.huge {font-size: 36pt}` and you use `<div class="huge">` throughout a large number of HTML documents
 - Now you discover your clients don't like this, so you change the CSS to be `div.huge {font-color: red}`
 - The “div” is no longer “huge” - the name is now “wrong” => you will need to modify all your documents...
 - By contrast, if you had used `div.important {font-size: 36pt}`, the name would not be “wrong”



Quick review:

3. JavaScript

Reasons that we need **scripting** (or programming) in browsers

- HTML, a **static** markup language (not a programming language), cannot perform operations like:
 - Changing text/image in a web page without reloading it
 - E.g., “mouse-over” effects
 - Reacting to events, e.g., confirmation before a user leaves a website, etc.
 - Validating input data in HTML forms before submission
 - Detecting the type/version of a client browser
 - Numerical calculations, e.g., unit conversion
 - Other “**user interactions**” and “**logic**”...
- We prefer most of these operations to be done at the **client side**
 - I.e., we prefer not to rely on the server entirely – why?
- **Client-side scripting** was introduced to support the above operations

JavaScript

- In order to support those client-side operations, “**scripts**” (which means “short programs”) are embedded in the HTML file and run by the browser.
 - Many variants in the past, e.g., JavaScript, JScript, ActiveX, VBScript, etc.
 - Nowadays, JavaScript is the most popular client-side scripting language.
- JavaScript modifies the behavior/appearance of HTML elements dynamically based on user inputs (event-driven)
- JavaScript is platform-independent
 - But expect different browsers to behave differently
 - Test your programs in different browsers
 - Use JS libraries and frameworks (e.g., jQuery) whenever possible (Session 4)

A simple JavaScript program

- JavaScript code can be included in `<script>` tags in HTML:

```
<script type="text/javascript">
<!--//
var weightInPounds;
var weightInKilos;

//assign values
//weightInPounds = 150;
weightInPounds = prompt("Enter your weight",150);
weightInKilos = weightInPounds*0.45359;

//display the result
document.write("You weigh " + weightInKilos + "kg");

//-->
</script>
```

Where to put JavaScript

- JavaScript can also be put in a separate `.js` file
 - And put this in the `<head>` of HTML:
 - `<script src="myJavaScriptFile.js"></script>`
 - An external `.js` file allows you use the same JavaScript in multiple HTML pages
- JavaScript can be put in the `<head>` or in the `<body>` of an HTML document
 - JavaScript *functions* should be defined in the `<head>`
 - This ensures that the functions are loaded before they are called
 - JavaScript in the `<body>` section will be executed when the page loads
- JavaScript can be put in an HTML *form object* (e.g., a button)
 - This JavaScript will be executed when the form object is used

JavaScript basics

- JavaScript is not Java, but similar:
 - Case-sensitive; support primitive data types: number/string/boolean
- Variable declaration:
 - `var` name="Peter", pi=3.14;
- *Variable names* are untyped: the type of a variable depends on the value it is currently holding
- Loop controls: while, for, do {} while, break, continue, etc. - similar to C
- Simple string concatenation:
 - `greeting = "Hello, " + name;`

Ways to create an **array**

- Use an array literal:

```
var colors = ["red", "green", "blue"];
```

- Use `new Array()` to create an empty array:

- ☐ `var colors = new Array();`

- ☐ `var colors = new Array(3);`

- You can add elements to the array later:

```
colors[0] = "red"; colors[2] = "blue"; colors[1]="green";
```

- You can define array elements within `new Array()`:

```
var colors = new Array("red", "green", "blue");
```

Array length and functions

- If `myArray` is an array, you can retrieve its length by `myArray.length`
- Other array functions:
 - `myArray.sort()` sorts the array alphabetically
 - `myArray.reverse()` reverses the array elements
 - `myArray.push(...)` adds any number of new elements to the end of the array, and increases the array's length
 - `myArray.pop()` removes and returns the last element of the array, and decrements the array's length
 - `myArray.toString()` returns a string containing the values of the array elements, separated by commas

Ways to create an **object**

- Define an object literally:

- `var course = { number: "ICOM6034", teacher: "Roy" };`

- Or, use `new Object()` to create a “blank” object, and add attributes to it later:

- `var course = new Object();`
`course.number = "ICOM6034";`
`course.teacher = "Roy";`

- Or, define a constructor:

- `function Course(n, t) {`
`this.number = n;`
`this.teacher = t;`
`}`

- `var course = new Course("ICOM6034", "Roy");`

Functions

- Functions should be defined in the `<head>` of an HTML page, to ensure that they are loaded first

- Syntax:

`function name(arg1, ..., argN) { statements }`

- E.g., `function multiply(i, j) { var k=2; return i * j * k; }`

`result = multiply(3, 4); // result = 24`

- Any variables declared within a function are local to it

- A function can also be defined *literally*, e.g.,

- `var f = function(x, y) { return x * y; }`

`result = f(3, 4);`

- Simple parameters are passed *by value*, objects are passed *by reference*

Debugging

- JavaScript, in general, is difficult to debug because its execution environment (i.e., the browsers) is not designed to be a development environment
- But Firefox/Chrome/Safari in general have better debugging tools than Edge/IE
 - E.g., Firefox Developer Tools, Chrome DevTools, Safari Web Inspector
- So developers tend to develop/debug using Firefox/Chrome/Safari, and after debugging, test the program in Edge/IE

Summary of Session 1

- The web is shaped by **standards**; standardization bodies include W3C, IETF, WHATWG, etc.
- **HTML/XHTML** define the structure of web documents
- **CSS** defines how the document is presented; it **separates** presentation details from the structure (HTML) of the document
- **JavaScript** adds client-side logic and behaviors to individual HTML elements

Covered in this Session

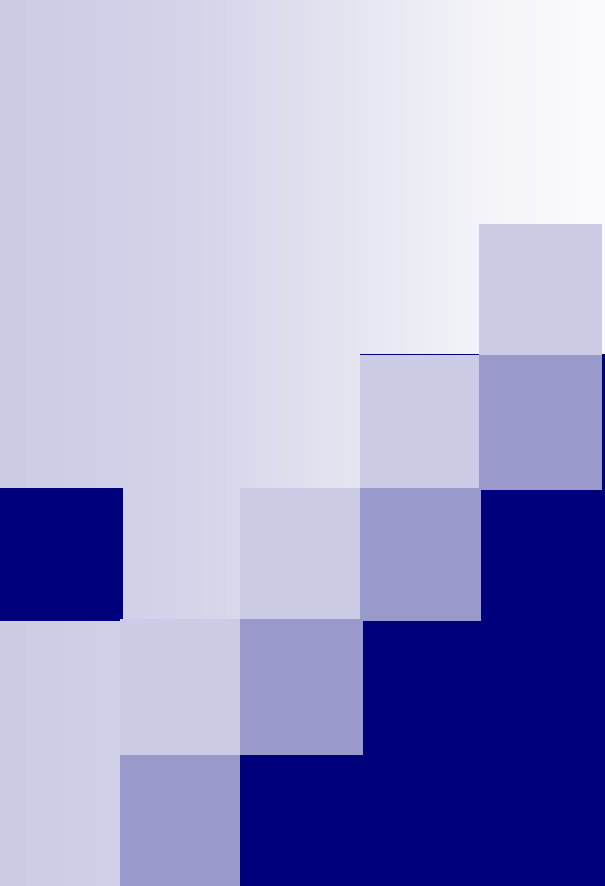
The four low-level, **client-side technologies** of the web

- JavaScript relies on **Document Object Model (DOM)** for naming/manipulating X/HTML elements
- The login page of Facebook.com as an integrated example of HTML + CSS + JavaScript + DOM
- Introduction of HTML5 and CSS3

Next lecture

Post-class self-learning resources

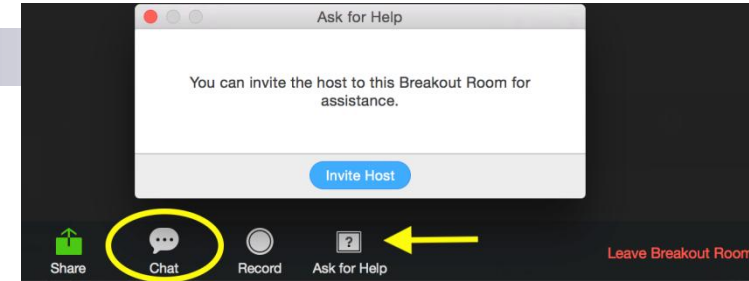
- For those who wish to explore further; not covered in exam
 - Please go to Moodle for the links
-
- Tutorials and examples (easy to read, with interactive examples):
 - <http://www.w3schools.com>
 - Cover HTML/XHTML, CSS, JavaScript, and more.
 - References:
 - List of HTML tags: http://www.w3schools.com/tags/ref_byfunc.asp
 - Browser compatibility tables: <http://caniuse.com/>



Lab arrangements in the “hybrid” teaching mode

Lab arrangements

- Your lab work does not need to be submitted
- “**Model answers**” are provided for all lab tasks => please do take a look and try to understand how the code works
 - The assignment/project will be based on the lab, so understanding the labs will save you a lot of time
- All students (including online students and those in the classroom) will be grouped in **Zoom Breakout Rooms**
 - Each group/room may have 4 members
 - Members in a group may discuss, help each other, and finish the lab tasks together
 - Please consider enabling **audio** (and optionally **screen sharing** or video if you like) to facilitate the discussion
 - Steven and I will also move around and join your discussion when we have time
- If you have any questions during the lab:
 - **Discuss** with other group members in your Breakout Room
 - Look at the “**model answers**” (if that can help)
 - If you attend the lab in-person – easy, simply raise your hand and let us know
 - So, **you are welcome to attend the lab in-person if you need more technical help and if the situation allows**
 - If you attend the lab online – use the “**Ask for Help**” function in Zoom to inform us
 - But we may not be able to appear immediately if there are many students asking for help. So please keep trying (re-clicking the “Ask for help” button) until we join your Breakout Room
 - You may **optionally** grant us the “remote control” capability if you think that would facilitate us to better help you
 - <https://support.zoom.us/hc/en-us/articles/201362673-Requesting-or-giving-remote-control>
- If you have any questions (on the lab/assignment/project) after the lab:
 - Post to Moodle’s discussion board
 - Email us anytime



} sometimes **screenshots** may help