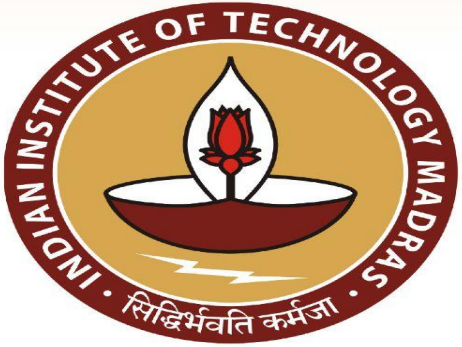

IIT Madras
BSc Degree

Beyond HTML

HTML Evolution

Markup languages

- Origins from late 60s
- Mostly used for typesetting and document management systems
- Problems?
  - Lack of standardization
  - Target audience: coders, publishers, academics?
  - Target output: print, other forms of media
  - Machine readability

SGML

Standard Generalized Markup Language

- Meant to be a base from which any markup language could be designed
- Basic postulates:
  - Declarative: Specify structure and attributes, not how to process
  - Rigorous: strict definition of structure, like databases
- DTD - Document Type Definition
  - Used to specify different families within this umbrella
  - Each could have its own tags, interpretation
- SGML ***Applications***

HTML

- Originally intended to be an *application* of SGML
- Very lenient with parsing - meant to be forgiving of errors
  - Not valid SGML
- HTML 2.0 - attempt to become SGML compliant
- Legacy support
  - Not truly SGML compliant
- HTML4 official definition - true SGML application
  - Limited usage
- HTML5 - **not** an SGML application - defines its own parsing rules

XML

eXtensible Markup Language

- Based on SGML
- Custom tags - multiple *applications* defined
- Focus on simplicity, generality and usability
- Both human-readable and machine-readable
- Well structured: can be used to represent complex data relationships, data structures etc.
- Examples:
    - MathML, RSS, Atom, SVG

XML Example - RSS feeds

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
<channel>
 <title>RSS Title</title>
 <description>This is an example of an RSS feed</description>
 <link>http://www.example.com/main.html</link>
 <copyright>2020 Example.com All rights reserved</copyright>
 <lastBuildDate>Mon, 06 Sep 2010 00:01:00 +0000</lastBuildDate>
 <pubDate>Sun, 06 Sep 2009 16:20:00 +0000</pubDate>
 <ttl>1800</ttl>

 <item>
  <title>Example entry</title>
  <description>Here is some text containing an interesting description</description>
  <link>http://www.example.com/blog/post/1</link>
  <guid isPermaLink="false">7bd204c6-1655-4c27-aeee-53f933c5395f</guid>
  <pubDate>Sun, 06 Sep 2009 16:20:00 +0000</pubDate>
 </item>

</channel>
</rss>
```

XML Example: SVG

```
<svg version="1.1"
     width="300" height="200"
     xmlns="http://www.w3.org/2000/svg">
  <rect width="100%" height="100%" fill="red" />
  <circle cx="150" cy="100" r="80" fill="green" />
  <text x="150" y="125" font-size="60"
text-anchor="middle" fill="white">SVG</text>
</svg>
```

XHTML

- Based on XML - not directly SGML
- Reformulation of HTML4 as applications of XML
- Main goal: clean up HTML specification
  - Modular and more extensible
- XML Namespaces: allow inter-operability with other XML applications

HTML5

- Add support for latest features (multimedia support, canvases, ...)
- Remain easily readable and understandable to both human and machine
- Remain backward compatible
- Breaks away from SGML:
  - Not an SGML or XML application
  - Defines own parser

HTML5

- Add support for latest features (multimedia support, canvases, ...)
- Remain easily readable and understandable to both human and machine
- Remain backward compatible
- Breaks away from SGML:
  - Not an SGML or XML application
  - Defines own parser

The **last** version of HTML!

- HTML Living Standard maintained by WHATWG (Web Hypertext Application Technology Working Group) - split away from W3C

Extension?

- How to add new features? New tags?
- “Software defined”
  - Allow new tags to be added through JavaScript
  - Custom Elements - API supported by browsers
- Very powerful mechanism: arbitrary functionality possible
  - No new tags need to be brought into standard
- Potential problems:
  - Anyone can define a tag!?
  - Semantics (meaning) of tags may not be well thought out

Requirement? Javascript

JavaScript

Lightning overview -

not meant to be a tutorial

What is JavaScript

- High level programming language
  - Dynamic typing
  - Object orientation (prototype based)
- Multi-paradigm
  - Event-driven
  - Functional - composition of functions, functions as objects
  - Imperative - direct computation through procedures and functions
- Relatively easy to learn
  - similarities with Python, C/C++, Java (no direct relationship)

Why JavaScript?

- Most web browsers have a dedicated engine
  - Designed from the ground up for the web
- APIs:
  - Text, dates, regular expressions (pattern matching)
  - Standard data structures (dictionaries, ...)
  - Document Object Model - manipulate the browser
  - No native IO (no file access etc.) but provided through APIs
- Most power when used for DOM manipulation

Basic Examples

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript#inline_javascript_handlers

- Variables and basics: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Variables#what_is_a_variable
-

Learning Resources

- https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics
  - https://mdn.github.io/beginner-html-site-scripted/

- https://developer.mozilla.org/en-US/docs/Learn/JavaScript
- https://learnjavascript.online/

Custom Elements

Adding custom elements

- XML allows arbitrary namespaces and tag definitions
  - Applications can be defined on top of XML tag definitions
- But HTML5 does not use the same approach
- Requirement for elements:
  - Meaning: what does a tag mean - <title>, <h1> etc OK, but is <my-button> actually a button?
  - Rendering: how should a tag be shown: provide display details for each tag
  - States: checkbox can be checked or blank - what about custom tags?
  - Customized built-in element or Autonomous custom element?

https://html.spec.whatwg.org/multipage/custom-elements.html

Web Components

- Custom elements
  - JS API to create custom element tags
- Shadow DOM
  - API to keep styling of component separate from rest of page
- HTML Templates
  - <template> and <slot> tags to write markup templates

Web Component Examples

- https://github.com/mdn/web-components-examples
- https://mdn.github.io/web-components-examples/editable-list/
- https://mdn.github.io/web-components-examples/edit-word/
- https://mdn.github.io/web-components-examples/word-count-web-component/
- https://css-tricks.com/an-introduction-to-web-components/

Summary

- Custom Elements: API to extend HTML5 element/tag capabilities
- Shadow DOM: restrict scope of styling or modification of content
- HTML Templates

Combined: Web Components

- Goal: reuse
- Problem: limited standardization

Frameworks

Purpose of a framework

- Basic functionality already available
  - Python can create network listeners, manipulate strings etc.
  - JS can extend elements, use APIs to manipulate documents
- Problem:
  - Lots of code repetition - boilerplate
  - Reinventing the wheel - different coding styles, techniques
- Solution:
  - Standard techniques for common problems - design patterns
  - Frameworks: Flask for Python web apps, React for JS components
- SPA: Single Page Application
  - Many JS front-end frameworks focus on enabling this - also useful for mobile

Example - React

Library for building user interfaces

- Declarative
  - Opposed to imperative - specify what is needed, not how to do it
- Components
  - Different from WebComponents - similar ideas, different techniques
  - Webcomponents are more imperative: functions that specify behaviour
  - React is declarative: focus on UI, but allow composing views
- Examples: https://reactjs.org/

Frameworks

- React - numerically most popular at present
- Angular - origins from Google - well supported
- EmberJS - component + service framework
- Vue

https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction

Summary

- HTML5 is a living standard - no major changes, but continuous adaptation
- JS provides the adaptation layer
- HTML + CSS + JS = rule the world!
- But difficult to code
- Frameworks fill in the gaps

Front-end development for dynamic applications