**Document Object Model**

**The True power of dynamic web pages**

**Table of Contents**

* Document Object Model (DOM)
* The DOM API
* Selecting DOM elements
  + getElementsByXXX()
  + querySelector()
* NodeLists
* LiveNodeList
* StaticNodeList

**Document Object Model (DOM) - The way to access HTML with JavaScript**

**Document Object Model**

* The Document Object Model is an **API for HTML and XML** documents
  + Provides a **structural representation** of the document
  + Developers can **modify the content and UI** of a web page
* The DOM consists of objects to manipulate a web page
  + All the properties, methods and events are **organized into objects**
  + Those objects are accessible through **programming languages** and **scripts**
* How to use the DOM of an HTML page?
  + Write JavaScript to interact with the DOM
    - JavaScript uses the DOM API (native implementation for each browser)

# DOM API - Using the DOM with JavaScript

**The DOM API**

* The DOM API consist of objects and methods to interact with the HTML page
  + Can add or remove HTML elements
  + Can apply styles dynamically
  + Can add and remove HTML attributes
* DOM introduces objects that represent HTML elements and their properties
  + document.documentElement is <html>
  + document.body is the body of the page

**DOM Objects**

* Each of the HTML elements have corresponding DOM object types
  + **HTMLLIElement** represents <li>
  + **HTMLAudioElement** represents <audio>
* Each of these objects have the appropriate properties
  + **HTMLAnchorElement** has href property
  + **HTMLImageElement** has src property
* The **document** object is a special object
  + It represents the entry point for the DOM API

**HTML Elements**

* HTML elements have properties that match attributes
  + id, className, draggable, style, onclick, etc…
* Different HTML elements have **their specific attributes**
  + **HTMLImageElement** has property src
  + **HTMLInputElement** has property value
  + **HTMLAnchorElement** has property href
* HTML elements have properties for content
  + innerHTML
    - Returns as a string the content of the element, without the element
  + outerHTML
    - Returns as a string the content of the element, with the element
  + innerText / textContent
    - Returns as a string the text content of the element, without the tags

# *DOM Objects – Live Demo*

# Selecting DOM Elements

**Selecting HTML Elements**

* HTML elements can be found and cached into variables using the DOM PI
  + Select single element

var header = document.getElementById('header');

var nav = document.querySelector('#main-nav');

* + Select a collection of elements

var inputs = document.getElementsByTagName('li');

var radiosGroup = document.getElementsByName('genders[]');

var header = document.querySelectorAll('#main-nav li');

* + Using predefined collections of elements

var links = document.links;

var forms = document.forms;

**Using getElementsByXXX() Methods**

* DOM API contains methods for selecting elements based on some characteristic
  + getElementBybyId(id):
    - Returns a **single element** or null

var header = document.getElementById('header');

* + getElementsByClassName(className):
    - Returns a **collection of elements**

var posts = document.getElementsByClassName('post-item');

* DOM API contains methods for selecting elements based on some characteristic
  + getElementsByTagName(tagName);
    - Returns a **collection of elements**

var sidebars = document.getElementsByTagName('sidebar');

* + getElementsByName(name);
    - Returns a **collection of elements**

var gendersGroup = document.getElementsByName('genders[]');

# *Selecting Elements with getElementsByXXX() - Live Demo*

**Using querySelector Methods**

* The DOM API has methods that use CSS-like selectors to find and select HTML elements
  + querySelector(selector) returns the **left most element** that matches the selector
  + querySelectorAll(selector) returns **a collection of all elements** that match the selector
* Both methods take as a string parameter
  + The CSS selector of the element

//the element with id="header"

var header = document.querySelector('#header');

//li elements contained in element with id=main-nav

var navItems = document.querySelectorAll('#main-nav li');

# *Selecting Elements with querySelector() - Live Demo*

**Selecting Nested Elements**

* The DOM API can be used to select elements that are inside other elements
  + Select all divs that are inside an element with id "wrapper"

var wrapper = document.getElementById('wrapper');

var divsInWrapper = wrapper.getElementsByTagName('div');

* All methods can be used on HTML elements
  + Except **getElementById(id)**

# *Selecting Nested Elements - Live Demo*

# NodeLists (The result of querying DOM elements)

**NodeLists**

* A NodeList is a collection returned by the DOM selector methods:
  + getElementsByTagName(tagName)
  + getElementsByName(name)
  + getElementsByClassName(className)
  + querySelectorAll(selector)

var divs = document.getElementsByTagName('div');

var queryDivs = document.querySelectorAll('div');

for(var i=0; i < divs.length; i++){

//do stuff with divs[i]…

}

* NodeList looks like an array, but is not
  + It's an object with properties similar to array
    - Has length and indexer
  + Traversing an array with for-in loop works unexpected:

for (var i in divs) {

console.log('divs[' + i + '] = ' + divs[i]);

}

//divs[0] = ...

//divs[1] = ...

//divs[length] = ...

//divs[item] = ...

# *NodeLists - Live Demo*

# Static NodeList and Live NodeList - What is that? What is the difference?

**Static NodeList and Live NodeList**

* There are two kinds of NodeLists
  + getElementsByXXX() return a **LiveNodeList**
  + querySelectorAll() returns a StaticNodeList
* Live lists keep track of the selected elements
  + Even when changes are made to the DOM
  + While static list contain the elements as they were at the execution of the method
* Keep in mind that LiveNodeList is slower that regular array
  + Need to cache its length for better performance

# Static NodeList and *Live NodeList - Live Demo*