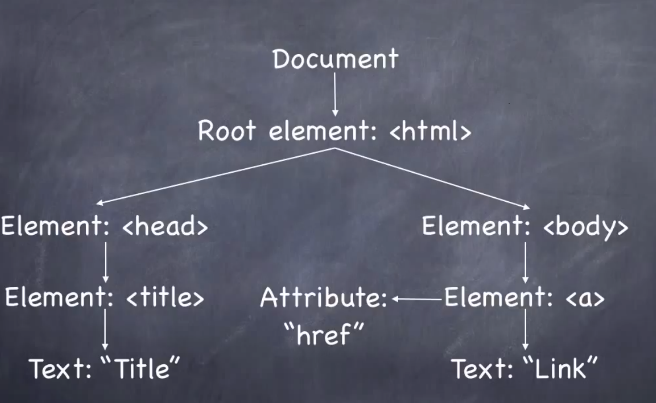
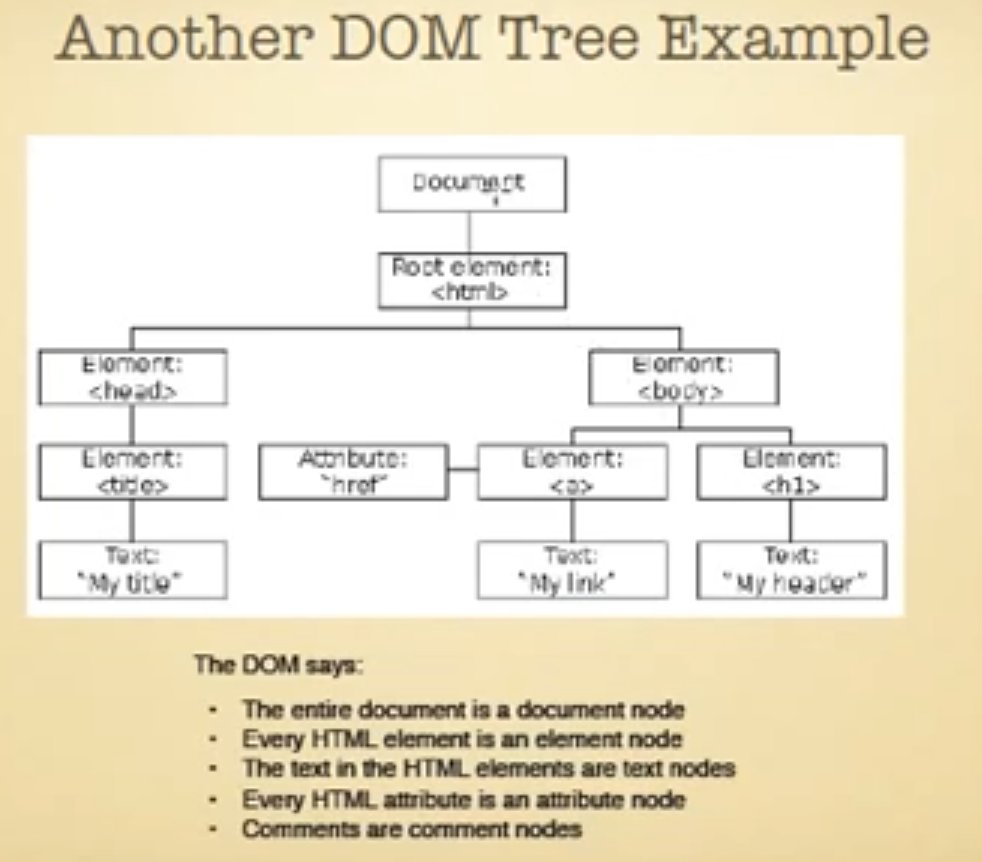
DOM in a tree



Another DOM Example



What do we do with Dom?

* Change HTML elements in a page
* Change HTML attributes in a page
* Change CSS styles in a page
* React to events in a page (e.g. button click)

3 ways of finding HTML elements

* By Tag name e.g. <h1>
* By ID e.g. <p id=”text”>
* By Class name e.g. <li class=”left”>

DOM (Document Object Model)

* Describe relationships in HTML
* Browsers interpret and organize HTML as a DOM
* API for CSS and JavaScript
* Dom is a representation of the whole document as nodes and attributes.
* We can access each of these nodes attributes and change or remove them.
* We can also create new ones or add attributes to existing ones.
* Everything in the dom is the node including text element
* Dom is the way to reach on the page from our script
* The DOM gives access to all the elements on a web page
* Inside the browser the whole web page, paragraphs, forms, tables are represented in object hierarchy.
* The DOM is a platform and language neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.

DOM (Document Object Module)

Document 🡪 Web page

Object 🡪 pieces

Model 🡪 agreed-upon or set of terms

* Document
* Is simply means the page not the site or web page?
* Document can have different representation.
* Objects
* Is just a thing or some thing
* Individual pieces of the document or whole things or we can say everything of the document is objects.
* Model
* Set of terms that we can agree on
* Set of standards that we can use
* Agreed-upon set of terms

What we can do with the Dom

- Get the title text

- Get the second paragraph

- Get the third link in the menu and set its CSS to display:none

- Change the background color of all paragraphs with a class of “important”

- Get all the <li> elements in the last unordered list

- Find the image with an id of “logo” and move it 40 pixels to the right

- Change a link so it performs a JavaScript function when clicked

- Create a new unordered list and insert between first and second paragraphs

DOM

- Dom means Document – Web page Object – pieces Model – agreed-upon set of terms

- The Document Object Model (DOM) is a programming interface for HTML, XML and SVG documents. It provides a structured representation of the document (a tree) and it defines a way that the structure can be accessed from programs so that they can change the document structure, style and content.

- The Document Object Model (DOM) is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML, and XML documents.[1] The nodes of every document are organized in a tree structure, called the DOM tree. Objects in the DOM tree may be addressed and manipulated by using methods on the objects. The public interface of a DOM is specified in its application programming interface (API).

- The DOM provides a representation of the document as a structured group of nodes and objects that have properties and methods. Nodes can also have event handlers attached to them, and once that event is triggered the event handlers get executed. Essentially, it connects web pages to scripts or programming languages.

- With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

- When a web page is loaded, the browser creates a Document Object Model of the page.

- The DOM defines a standard for accessing documents:

- The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page. This is an overview of DOM-related materials here at W3C and around the web.

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

* The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

HTML Validator

* Validator.w3.org

Check use list of htmnl5 and CSS3

* [www.caniuse.com](http://www.caniuse.com)

Important Site

* https://github.com/eligrey/classList.js

Managing Dom

* Google chrome – ctrl+shift+i
* Navigate Tabs of Console Panel
* Right side – ctrl+}
* Left side – ctrl+{
* Navigate console directly – ctrl+shift+j
* Toggle console under element tabs – Esc
* Dom Element
* 🡹 up arrow
* 🡻 down
* 🡸 collapse code
* 🡺 expand code
* Select an element to inspect – ctrl+shift+c
* To edit an element – enter or double click
* Next item – tab
* Reload the page again – ctrl+r
* Regular html editing mode – f2 or
* Right click mouse and select edit as html

Command on Console

* Clear – ctrl+L

To quickly identify an element using magnifying glass

HTML DOM querySelector() Method

* The querySelector() method only returns the first element that matches the specified selectors. To return all the matches, use the querySelectorAll() method instead.

Communicating with the console through JavaScript

* console.log(“message”) – see the message in console panel
* console.log(document.querySelector(“#main”)) – return the first main element
* console.dir(document.querySelector(“#main”)) – return the element in directory format
* console.info(“message”) – same as like console.log(“message”) see the message in console.
* console.warn(“message”) – warning message in console panel
* console.error(“message”) – error message in console panel

Enhancing the DOM

Selecting DOM Elements

Choosing by Id

* getElementById() most common
* Tags with a specific ID
* A single ID per page
* Use console.dir to explore the Node

Accessing DOM Elements by getElementById()

* Select an element by Id

document.getElementById('page\_artists')

* We can identify an element quickly by using magnifying glass
* To assign a node into a variable

var myNode = document.getElementById('comingtoevent')

* To see all available properties and methods

dir(myNode)

* To access an properties of a node

myNode.firstChild

* To see all the child nodes

myNode.childNodes

Choosing by Tag

* getElementByTagName()
* Groups elements by tags
* Returns an array
* Can be combined with getElement

Accessing DOM Elements by getElementsTagName()

* To grab all single list item

document.getElementsByTagName('li')

* To get a specific tag by using Id

document.getElementById('artlist').getElementsByTagName('li')

* Assign elements a variable

var myNode = document.getElementById('artlist').getElementsByTagName('li')

* To get a specific element use array key

myNode[2]

Elements by Class Name

* getElementsByClassName()
* Elements with a specific class
* Newer selector
* Not compatible with older browsers

Accessing DOM Elements by getElementsClassName()

* To get all class elements

document.getElementsByClassName()

Querying with CSS Selectors

* querySelector(), querySelectorAll()
* Nodes through CSS selectors
* Similar to jQuery
* Not compatible with older browsers

Accessing DOM Elements by querySelector()

* querySelector() returns first elements

document.querySelector('article')

return first article element

* querySelectorAll() return all matched elements

document.querySelectorAll('article') return article elements

querySelectorAll() return an array

document.querySelectorAll('article')[2].childNodes return child nodes of second article

* Select a descendant using querySelectorAll()

document.querySelectorAll('#artistlist li')

* Select a child using querySelectorAll()

document.querySelectorAll('#artistlist li>img')

Selecting named form elements

* Form elements can have name attributes
* DOM provides document.forms object
* Named elements can also be selected

Selecting named form elements

* Select forms

document.forms

* Select first forms

document.forms[0]

* Select form by using name

document.register

* Select an element of form by using name attribute

document.register.myname

* Change the value of a form element

document.register.myname.value = 'Ray Villalobos'

* Two input element can have the same name then identify the element by using array indexing

document.getElementsByName('subscribe')[0]

* Change the check attribute

document.getElementsByName('subscribe')[1].checked = "checked"

* Select select item

document.register.reference

* Set value in dropdown box

document.register.reference.value = 'facebook'

* Get the index of selected item

document.register.reference.selectedIndex

Node Properties

* node.nodeType – numerical value of a node
* node.nodeName – the name of the node
* node.nodeAttributes – array of node attributes
* node.nodeValue – element inside a node – stores the value of the node (null for element nodes and the text for text nodes)

Accessing node properties

var myNode = document.querySelectorAll('nav li a')[4]

* nodeType

myNode.nodeType

* nodeName

myNode.nodeName

* nodeAttributes

myNode.attributes

* myNode.attributes[0]

return first attributes

* attributes nodeType

myNode.attributes[0].nodeType

* attributes nodeName

myNode.attributes[0].nodeName

* return firstChild

myNode.firstChild

* Change the content of firstChild

myNode.firstChild = "registration"

* Change node value

myNode.firstChild.nodeValue = "registration"

Traversing up and down DOM Nodes

Moving up and down

* parentNode – Goes up a level
* childNodes – Array of children
* firstChild/lastChild – First/Last element
* previousSibling/nextSibling - Elements with same parent

Traversing up and down DOM Nodes

var myNode = document.querySelector('.artistlist li')

* One level up

myNode.parentNode

* Parent node children

myNode.parentNode.childNodes

* First child next sibling

myNode.parentNode.firstChild.nextSibling

* It’s difficult to identify a specific element from a list of child nodes in this case it’s preferable to use querySelectorAll()

var myNode = document.querySelectorAll('.artistlist li')

Targeting Node Elements

* firstElementChild – First element child
* lastElementChild – Last element child
* children – Only children that are elements
* previousElementSibling/nextElementSibling

Targeting Node Elements

* Return child that are only element not text

var myNode = document.querySelectorAll('.artistlist li')

myNode.parentNode.children

return only the child elements

* First element child returns

myNode.parentNode.firstElementChild

* Last element child returns

myNode.parentNode.lastElementChild

Changing HTML attributes

* Dot notation provides easy access
* img.src gives us an image location
* Read and write properties
* Add attributes that don’t exist
* Be careful of reserved words

Changing HTML attributes

* Select first image from a group of image

var myNode = document.querySelector('#artistlist img')

* Select the location of image using dot notation

myNode.src

* Change the image by using src attribute

myNode.src= 'images/hotel\_contempo.jpg'

* Add an attribute

myNode.id = "selected"

* JavaScript reserved is not worked

myNode.class = "myClass"

* Instead we should use

myNode.className = "myClass"

* Select a label element

var myNode = document.querySelector('label')

* Change the html for attributes

myNode.htmlFor = "companyname"

* Select a radio button

var myNode = document.querySelector("input[type='radio']")

* Change the checked attribute

myNode.checked = 0

Working with restricted or non-standard attributes

* Dot notation not convenient
* Some names are restricted in JavaScript
* node.getAttribute(attributeName) gets value
* node.setAttribute(attributeName, value) sets value
* node.hasAttribute(attributeName) Boolean
* node.removeAttribute(attributeName) deletes attribute
* dot notation is not work with dot notation so it’s more reliable to work with these method

Working with restricted attributes

* Select first label of register id

var myNode = document.querySelector('#register label')

* Select for attribute using htmlFor

myNode.htmlFor

* Instead of for attribute we can use getAttribute(‘for’) which work as like htmlFor

myNode.getAttribute('for')

* Set the value of for attribute

myNode.setAttribute('for','somethingelse')

* Check an attribute of the element

myNode.hasAttribute('id')

return Boolean true or false

* Remove an attribute

myNode.removeAttribute('for')

Detecting data attributes

* users can type anything as an attribute
* browsers ignore them, but it’s not valid html
* create our own attributes using data
* data-coolness valid attribute
* node.dataset property lets us access them

Note: All browsers can already use data. \* attributes and access them using getAttribute. “Suppoeted” refers to accessing the values using the dataset property. Current spec only refers to support on HTML elements, only some browsers also have support for SVG/MathML elements.

Detecting data attributes

* Select all image under artistlist id

var myNode = document.querySelectorAll('#artistlist img')

* Select second element

myNode[1]

* Select dataset

myNode[1].dataset.task

Controlling classes with classList

* Class properties can have more than one value
* Dot notation is not convenient
* HTML5 adds the datalist property to nodes
* jQuery like class modifications
* Lousy IE support
* node.classList.add(class) adds a class
* node.classList.remove(class) removes a class
* node.classList.toggle(class) turns class on/off
* node.classList.length how many
* node.classList.contains class name

Controlling classes with classList

var myNode = document.querySelector('.pixgrid img')

* Add a class with selected item

myNode.classList.add('faded')

* Add another class

myNode.classList.add('pulse')

* Output will be

<img src=”” alt=”” class=”faded pulse”>

* Remove a class

myNode.classList.remove('pulse')

* Toggle class

myNode.classList.toggle('hidden')

* Class name exists or not

myNode.classList.contains('hidden')

Targeting the attributes property

* node.attributes returns a node list
* Accessed in a variety of ways
* By numeric index
* By named index
* Using dot notation

Using text content modifiers

* To identify node

var myNode = document.querySelector('#abouttheevent')

* To identify innerHTML of an element excluding element

myNode.innerHTML

* To identify outerHTML of an element including element

myNode.outerHTML

* To insert adjacent element to with an element after end of the element

myNode.insertAdjacentHTML('afterend','<p>This is another paragraph</p>')

* To insert adjacent element to with an element before begin of the element

myNode.insertAdjacentHTML('beforebegin','<p>This is another paragraph</p>')

* To insert adjacent element to with an element after begin of the element

myNode.insertAdjacentHTML(afterbegin,'<p>This is another paragraph</p>')

* To insert adjacent element to with an element before end of the element

myNode.insertAdjacentHTML(beforeend,'<p>This is another paragraph</p>')

Using text content modifiers

* Most common and oldest one
* node.innerHTML changes text as HTML
* In newer browser we can use
* node.outerHTML includes element’s tags
* Insert an element before or after of the element
* node.insertAdjacentHTML(insertionPoint, htmlText)

Using text content modifiers

* node.innerText just the text of a node
* node.textContent in Firefox

if(node.innerText) {

myText = node.innerText;

} else {

myText = node.textContent;

}

Using text content modifiers

* To select the just text of the node

myNode.innerText

Creating and appending nodes

* document.createElement(element) Makes a new element
* Has to be added to the DOM
* node.appendChild(element) Adds element inside a node
* The appendChild() method appends a node as the last child of a node.
* The insertBefore() method to insert a new child node before a specified, existing, child node.

Creating and appending nodes

Create Element

* var myElement = document.createElement('img')

Add Source Attribute

* myElement.src="images/artists/LaVonne\_LaRue\_tn.jpg"

Add alt tag to the element

* myElement.alt="Photo of LaVonne\_LaRue\_tn"

Add non-standard attribute

* myElement.setAttribute('data-task','speaker')

Targeting the node

* var myNode = document.querySelectorAll('.artistlist ul li')

Append the element with the target element

* myNode[6].appendChild(myElement)

Controlling node insertions

* appendChild() lacks precision
* Need to insert a node anywhere in the node list
* Use insertBefore() for surgical insertions

Node Insertion

* Create a paragraph node

var pNode = document.createElement('p')

* Create a text node

var myText = document.createTextNode('within walking distance to major shopping venues')

* Append text node within paragraph node

pNode.appendChild(myText)

Previous and Next Sibling

* Once we’ve reached an element we can use this element as a starting point
* We can retrieves element on the same level with nextSibling and previousSibling
* We can reached the element the current one is embedded in with parentNode.

Children

* If an element contains other elements then these elements are its children
* childNodes is a list of all the first-level child nodes of the element – it does not cascade down into deeper levels.
* We can access a child element of the current element via the array counter or the item() method.

Chaining of DOM Methods

* We can chain methods in order to reach parts of the document faster:
* Example

document.getElementById(‘nav’).getElementsById(‘li’)[3].firstChild;

document.getElementById(‘nav’).getElementsByTagName(‘li’)[3].firstChild;

Changing the value of Text Nodes

* The nodeValue of text nodes can be read and changed.
* When we change the nodeValue the text of the node changes.
* It is a good idea to make sure that the node we try to change is a text node by checking that its nodeType is 3.
* Example

x=document.getElementById(‘nav’).getElementsByTagName(‘a’)[2].firstChild;

x.nodeValue = ‘Visuals’

Reading and Changing Attributes

* Element nodes have attributes that we can change.
* node.getAttribute(attribute) returns the current value of the attribute.
* node.setAttribute(attribute, value) sets the value of the attribute.

Creating New Nodes

* We can create new nodes with three methods.
* newNode=node.cloneNode(bool) creates newNode as a copy (clone)of node. If bool is true, the clone includes clones of all the child nodes of the original.
* document.createElement(element) creates a new element node.
* document.createTextNode(text) creates a new text node.

Adding, Replacing and Removing Nodes

* node.removeChild(oldNode) removes the child oldNode form node.
* node.appendChild(newNode) adds newNode as a new(last) child node to node.
* Node.insertBefore(newNode, oldNode) inserts newNode as a new child node of node before oldNode.
* node.replaceChild(newNode, oldNode) replaces the child node oldNode of node with newNode.