JavaScript HTML DOM

JavaScript and HTML Documents

Dynamic Documents with JavaScript

Outline

- JavaScript and HTML Documents:
  - JavaScript Execution Environment
  - DOM
  - How to access elements in HTML Using JS?
  - Events and Event Handling
  - DOM Tree Traversal and Modification
- Dynamic Documents with JavaScript:
  - Mouse Events
  - JavaScript timers
  - Pattern Matching By Using RegEx

JavaScript and HTML Documents

Chapter 5 from supplementary book + ch22 textbook

JavaScript Execution Environment

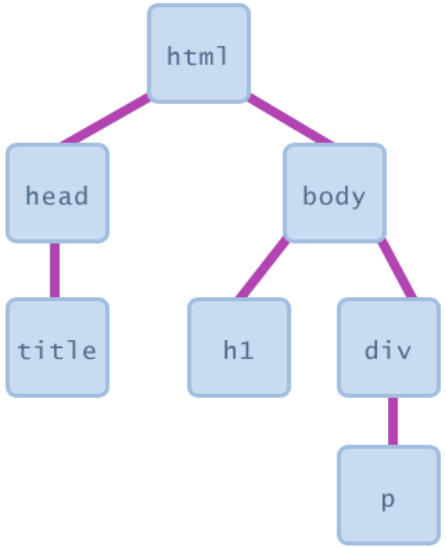
- JavaScript executes in a browser
- The **Window** object represents the window displaying a document
  - All properties are visible to all scripts
  - Global variables are properties of the Window object
- The **Document** object represents the document displayed
  - The document property of Window
  - Property arrays for forms, links, anchors, …

DOM

- The **Document Object Model (DOM)** is a **programming interface** that provides a way to access and manipulate the contents of a document.
- It provides a structured **map of the document** and a set of **methods** for interacting with them.
- It can be used with other XML languages, and it can be accessed by other programming languages (like PHP, Ruby, etc.).

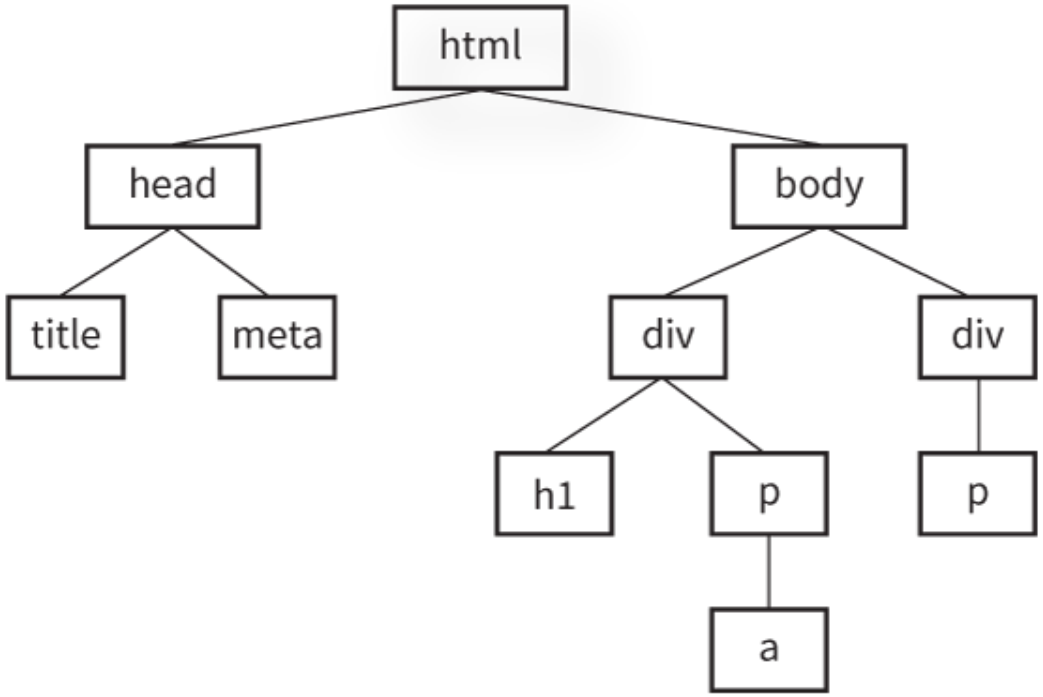
Document Object Model (DOM)

- A set of JavaScript **objects** that represent each element on the page
- We can **examine** elements' state
  - e.g. see whether a box is checked
- We can change **state**
  - e.g. insert some new text into a div
- We can change **styles**
  - e.g. make a paragraph red



Node Tree

The DOM treats the structure of a document like a **tree** with branches:

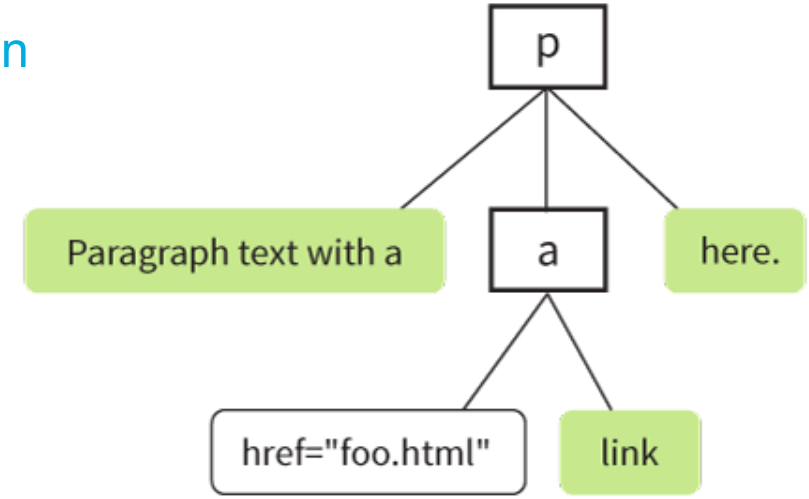


Node Tree (cont'd)

Every element, attribute, and piece of content is a node on the tree and can be accessed for scripting:

```
<p>Paragraph text with a <a href="foo.html">link</a> here.</p>
```

The nodes within a `p` element



How to access elements in HTML Using JS?

Accessing Nodes

- To point to nodes, list them separated by periods (.)
- In this example, the variable `foo` is set to the HTML content of an element with `id="beginner"`:

```
var foo= document.getElementById("beginner").innerHTML;
```

- The **`document`** object points to the page itself.
- **`getElementById`** specifies an element with the `id` "beginner".
- **`innerHTML`** stands for the HTML content within that element.

Accessing Nodes (cont'd)

Methods for accessing nodes in the document:

`getElementsByTagName()`

Accesses all elements with the given tag name

Example: `document.getElementsByTagName("p");`

`getElementById()`

Accesses a single element by the value of its `id` attribute

Example: `document.getElementById("special");`

`getElementsByClassName()`

Access elements by the value of a class attribute

Example: `document.getElementsByClassName("product");`

Accessing Nodes (cont'd.)

`querySelectorAll()`

Accesses nodes based on a **CSS selector**

Example: `document.querySelectorAll(".sidebar p");`

Click Here

`getAttribute()`

Accesses the value of a given **attribute**

Example:

`document.getElementById("logo").getAttribute("src")`

Click Here

Using forms array

- Consider this simple form:

```
<form action = "">
    <input type = "button"  name = "pushMe">
 </form>
```

- The input element can be referenced as `document.forms[0].element[0]`

Using `name` Attributes

- All elements from the reference element (such as form tag) up to, but not including, the body must have a name attribute

- Example

```
<form name = "myForm"  action = "">
   <input type = "button"  name = "pushMe">
</form>
```

- Referencing the input

```
document.myForm.pushMe
```

Events and Event Handling

Basic Concepts of Event Handling

- *Event-driven programming* is a style of programming in which pieces of code, `event handlers`, are written to be activated when certain ***events*** occur
- `Events` represent activity in the environment including, especially, user actions such as moving the mouse or typing on the keyboard
- An `event handler` is a program segment designed to execute when a certain event occurs
- Events are represented by **JavaScript objects**
- ***Registration*** is the activity of connecting a script to a type of event

Events, Attributes and Tags

- Particular events are associated to certain attributes
- The attribute for one kind of event may appear on different tags allowing the program to react to events affecting different components
- e.g. A text element gets focus in three ways:
  1. When the user puts the mouse cursor over it and presses left mouse button.
  2. When the user tabs to the element using the keyboard.
  3. By executing the JavaScript `focus()` method.
     (e.g., document.getElementById("myAnchor").focus();)
- Losing the focus is ***blurring***

Events, Attributes and Tags

Event	*Tag Attribute*
blur	onblur
change	onchange
click	onclick
focus	onfocus
load	onload
mousedown	onmousedown
mousemove	onmousemove
mouseout	onmouseout
mouseover	onmouseover
mouseup	onmouseup
select	onselect
submit	onsubmit
unload	onunload

Setting a Handler (Registration)

- Using an attribute, a JavaScript command can be specified:

```
<input type="button" name="myButton"
  onclick=
  "alert('You clicked the button!')"/>
```

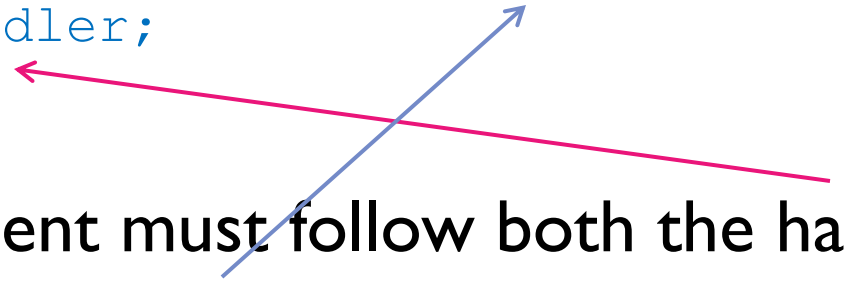
- A function call can be used if the handler is longer than a single statement

```
<input type="button" name="myButton"
  onclick="myHandler()"/>
```

Setting a Handler (Registration)

- The event handler could also be registered by assignment to associated event property:

```
document.getElementById("MyButton").onclick =
myButtonHandler;
```



- This statement must follow both the handler function and element.

Comparing Registration Methods

- Assigning to a node property **helps** separate HTML and Java code (JavaScript) that interacts with it.

- Assigning to a node property **allows** reassignment later if the handler needs to be changed

Unobtrusive JavaScript

- JavaScript event code seen previously was *obtrusive* (in the HTML)
  - This is bad style (mixes content and behavior)

- *Use* *unobtrusive* JavaScript code:
  - HTML with minimal JavaScript inside
  - Uses the DOM to attach and execute all JavaScript functions
  - Clean HTML code, clear separation of content, presentation, behavior

Example

- Radio
- Validator

Click Here

The `navigator` object

- Information about the web browser application
- The navigator object is a property of the window object.
- Properties:
  - appName, appVersion, browserLanguage, cookieEnabled, platform, userAgent
  - Try: ex

Example:

- Navigate.html

DOM Tree Traversal and Modification

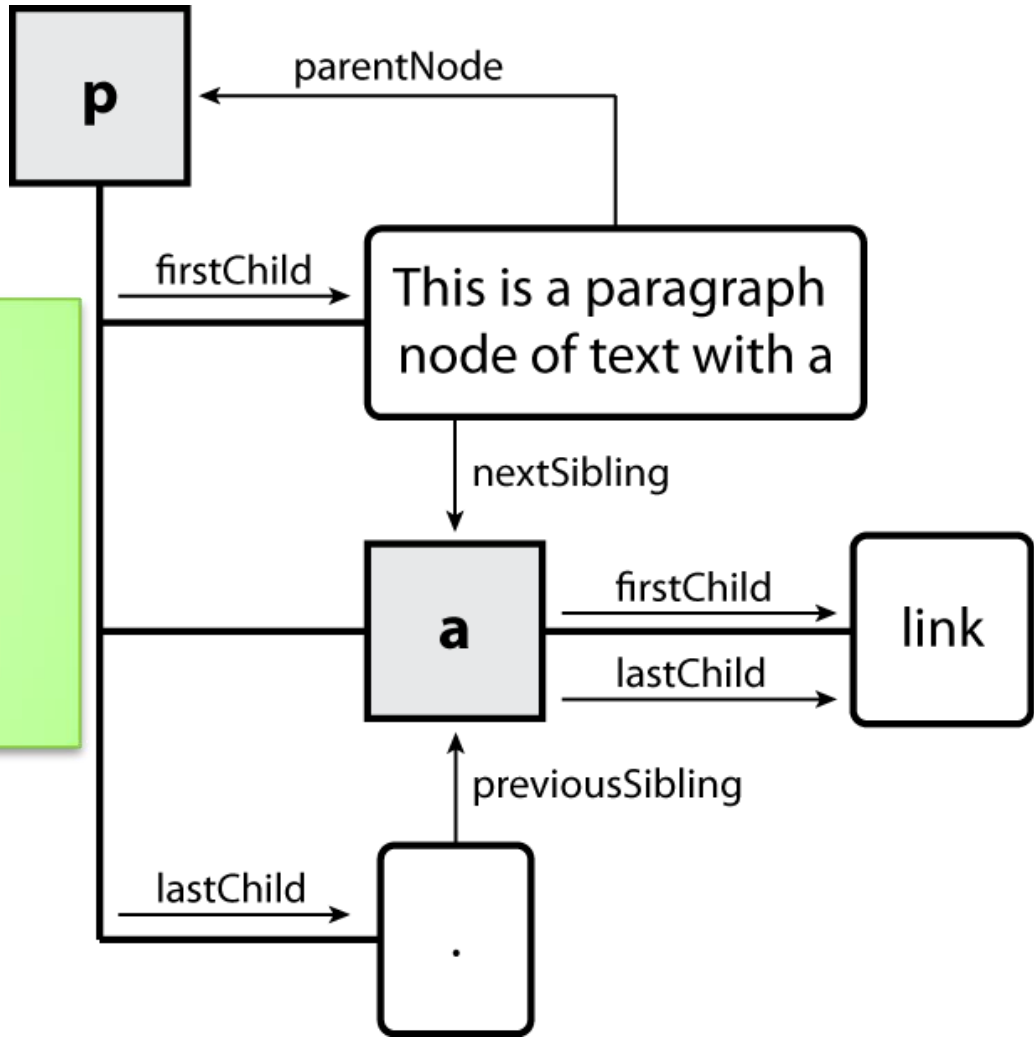
DOM Tree Traversal

- Every node's DOM object has the following properties:

name(s)	description
`firstChild`, `lastChild`	start/end of this node's list of children
`childNodes`	array of all this node's children
`nextSibling`, `previousSibling`	neighboring nodes with the same parent
`parentNode`	the element that contains this node

DOM Tree Traversal Example

```
<p id="foo">
This is a paragraph of text with a
<a href="/path/to/another/page.html">
link</a>.</p>
```



Modifying the DOM tree

- Every DOM element object has these methods:

name	description
`appendChild(`*node*`)`	places given node at end of this node's child list
`insertBefore(`*new*`, `*old*`)`	places the given new node in this node's child list just before `old` child
`removeChild(`*node*`)`	removes given node from this node's child list
`replaceChild(`*new*`, `*old*`)`	replaces given child with new node

Dynamic Documents with JavaScript

Chapter 6 from supplementary book

Moving Elements

- Remember positioning elements [absolute, relative, and static] from CSS.
- JavaScript code can move elements by changing the **top** and **left** properties of style object.
- Note that the position mode has to be relative or absolute for this to work.
- Example movertext, mover illustrates dynamically placing elements. ClickHere

Element Visibility [CSS]

- Sets whether an element should be shown onscreen
- The element will still take up space onscreen, but will not be shown
- To make it not take up any space, set display to ***none*** instead
- Can be *visible* (default) or *hidden*
- Can be used to show/hide dynamic HTML content on the page in response to events
- Example: visible
- document.getElementById("myP").style.visibility = "hidden";

Changing Colors and Fonts

- Colors and font properties can be manipulated through the **style** property of an element.
  - e.g. document.getElementById(‘fname').style.color= ‘red’;

Dynamic Content

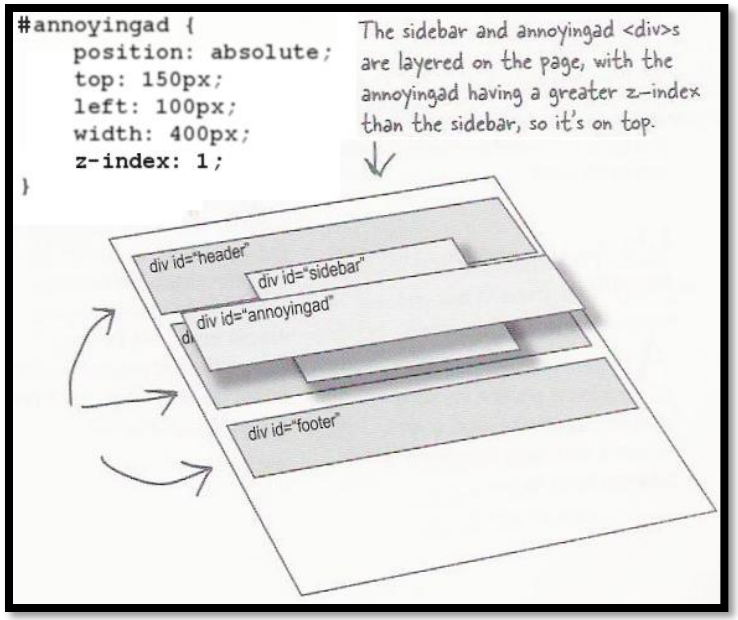
- By manipulating the DOM tree representing the document, the document content can be changed.

- The content of an element is accessed through the value property of its associated JS object.
  - e.g. document.getElementById(‘fname').value= ‘Adam’;

- Example dynamiccolor, dynamicfont. ClickHere

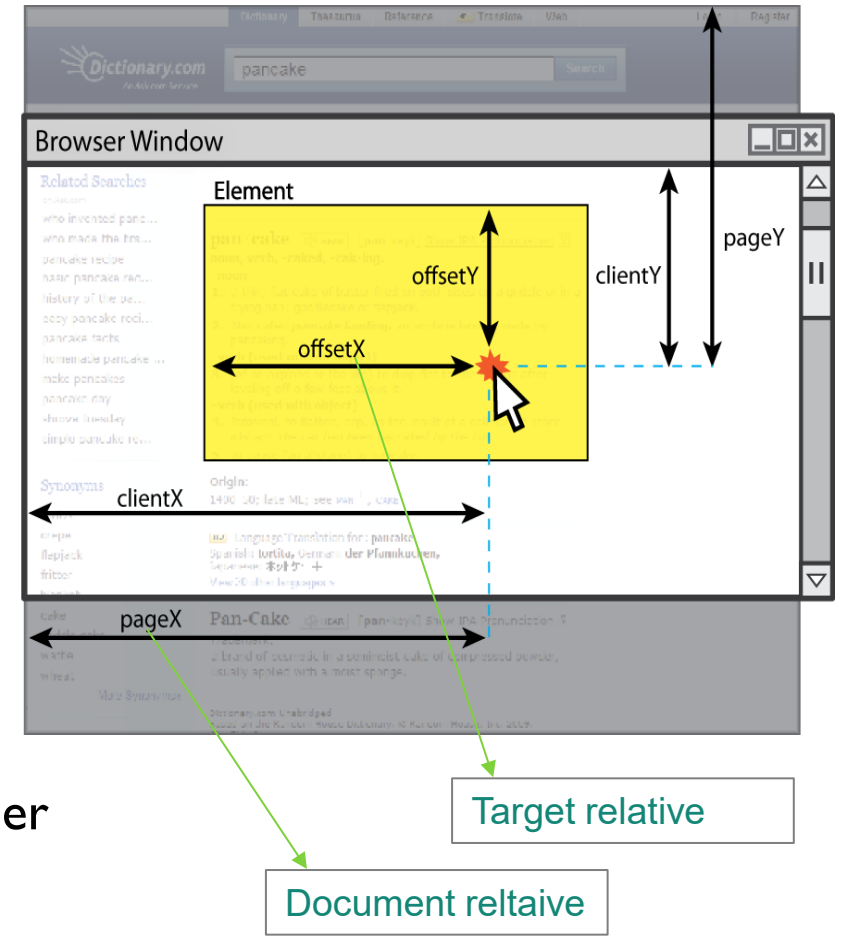
Stacking Elements (zIndex)

- Sets which absolute positioned element will appear on top of another that occupies the same space
- Higher z-index **wins**
- Can be auto (default) or a number
- Can be adjusted in DOM: object.style.zIndex = "value";
- The stacking example illustrates manipulating the z-index property dynamically



Locating the Mouse Cursor & Reacting to Clicks

- An event object created for a mouse related event has properties that give the coordinates of the mouse position at the time of the event
  - **clientX** and **clientY** give the position in pixels relative to the upper left-hand corner of the browser window
  - **screenX** and **screenY** give the mouse position relative to the upper left-hand corner of the screen



Try it by click here

Locating the Mouse Cursor & Reacting to Clicks

- The event object is available as an object named event In Mozilla/Firefox the object can be accessed by passing it as a parameter to the event handler

- In Internet Explorer, the object is a global property
- The anywhere example is an example using mouse location information

JavaScript timers

- Timer concepts:
  - **Timer**: executes an action after a delay, or repeatedly at given intervals.
  - JavaScript's implementation of timers:
    - setTimeout, setInterval, clearTimeout, clearInterval functions.
    - An event handler function and a delay (ms) are passed as parameters to the above functions.
    - 1 second = 1000 milliseconds.
    - The function is called after the delay.

Timer functions

- **setTimeout**(function, delay, [param1, param2, ...]);
  arranges to call the given function after the given delay in ms, optionally passing it the parameters provided

- **setInterval**(function, delay, [param1, param2, ...]);
  arranges to call the given function repeatedly, once every delay ms
  - Both setTimeout and setInterval return an object representing the timer.

- **clearTimeout**(timer); and **clearInterval**(timer);
  stops the given timer object so it will not call its function any more

setTimeout Example

```
function delayMsg()
{ setTimeout(surprise, 5000); }

function surprise() { // called when the timer goes
  off alert(“WoW!"); }

<button onclick="delayMsg();">Click me!</button>
```

- **setTimeout** returns instantly; surprise() function is executed once after 5 seconds

Clearing a Timer

```
var timer;
function repeatedMessage()
{ timer = setInterval(rudyRudy, 1000); }
function rudyRudy() { alert("Rudy!"); }
function cancel() { clearInterval(timer); }


<button onclick="repeatedMessage();">Rudy
  chant</button>
<button onclick="cancel();">Make it stop!</button>
```

- setInterval returns an object representing the timer can be stored in a global variable
- To cancel the timer, call clearInterval and pass the timer object

Common Timer Errors

- Many students mistakenly write () when passing the function
- setTimeout(surprise(), 2000);
  setTimeout(surprise, 2000);

- What does it actually do if you have the ()?
  - It calls the function immediately, rather than waiting the 2000ms!

The DOM is NOT JavaScript

- Black is for JavaScript, Red is for DOM

```
var anchorTags =
  document.getElementsByTagName("a") for (var
  i = 0;i <anchorTags.length; i++)
 {
  alert("Href of this a element is:" +
  anchorTags[i].href);
  }
```