# SOEN 287: WEB PROGRAMMING


Chapter 5
JavaScript & HTML Documents

# Javascript

We will cover in this part:

- Document Object Model (DOM)
- Document access
- Event handling

# DOM (**D**ocument **O**bject **M**odel)

- The DOM is a W3C (World Wide Web Consortium) standard.

- The DOM defines a standard for accessing documents:

  *"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."*

  (http://www.w3schools.com/js/js_htmldom.asp)

- The DOM is an abstract model that defines the interface between HTML documents and application programs—an API

# DOM (**D**ocument **O**bject **M**odel)

- The HTML DOM is a standard for how to get, change, add, or delete HTML elements.
- HTML DOM methods are **actions** you can perform (on HTML Elements)
- HTML DOM properties are **values** (of HTML Elements) that you can set or change

# DOM

- A language that supports the DOM must have a binding to the DOM constructs.

- In the JavaScript binding, HTML elements are represented as **objects** and element attributes are represented as **properties**

e.g., `<input type = "text" name = "address">`

would be represented as an **object** with two **properties**, `type` and `name`, with the values `"text"` and `"address"`

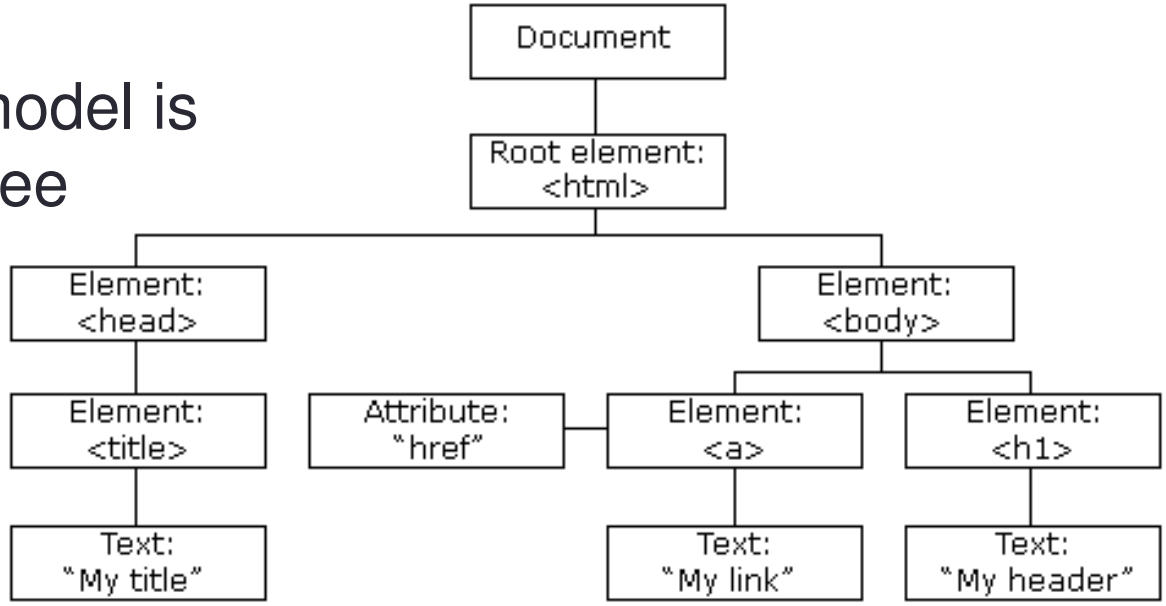
# HTML DOM

http://www.w3schools.com/js/js_htmldom.asp

- When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

- DOM is a logical structure of a document.

- The **HTML DOM** model is constructed as a tree of o**bjects**:

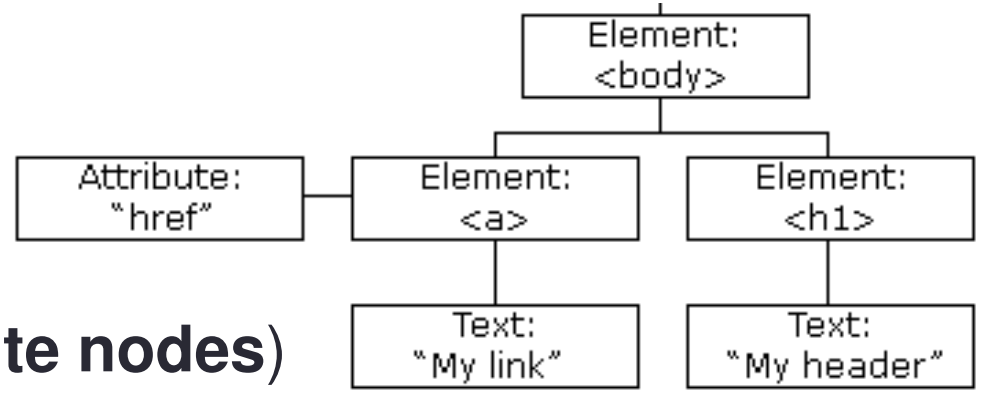


# HTML DOM (Document Object Model)

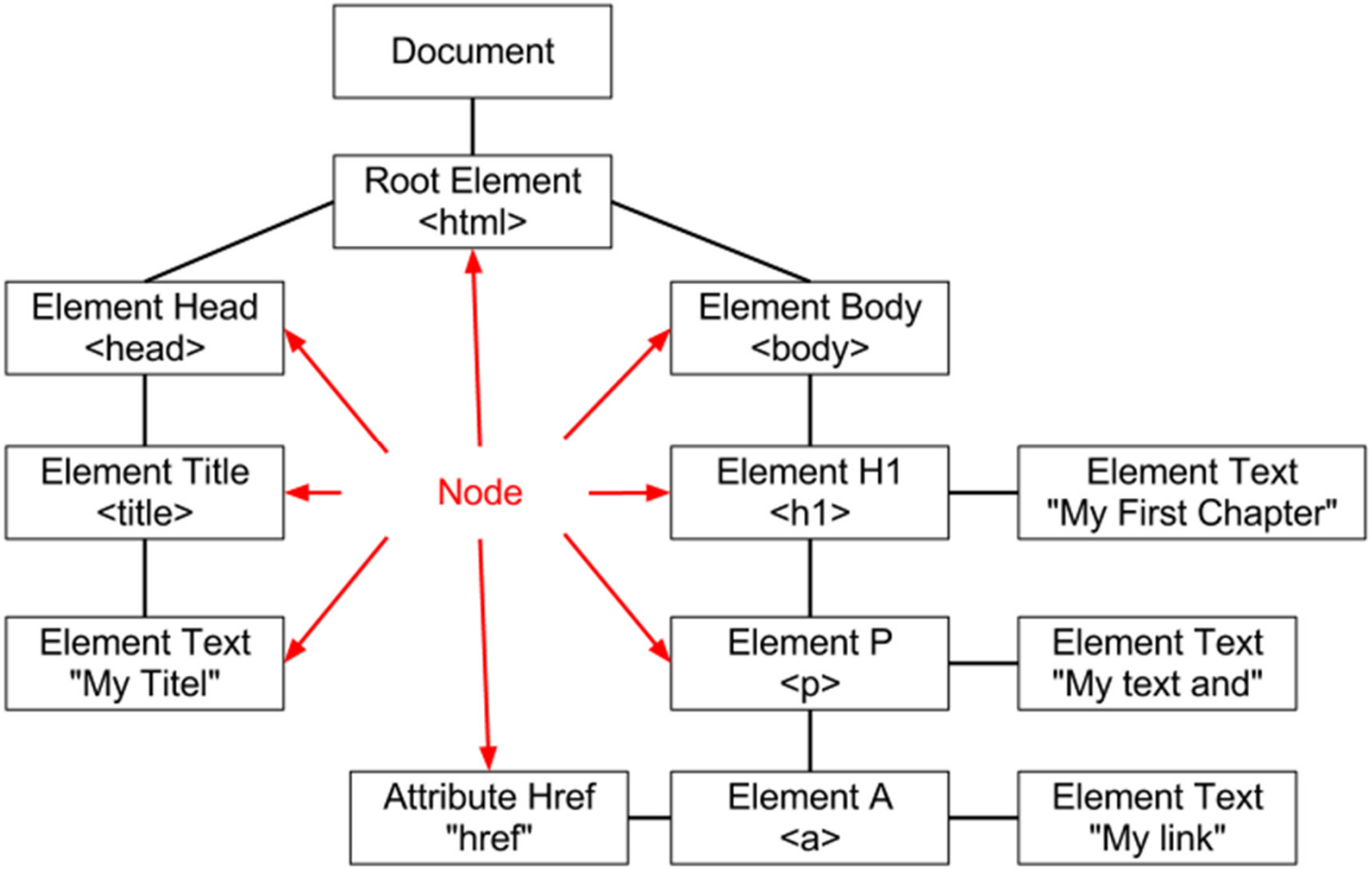
- In the HTML DOM, the **Element object** represents an HTML element.

- Each element object can have **child nodes**
  - Element node
  - Text node

- Elements can also have attributes (**attribute nodes**)



# Example of a DOM




http://gerardnico.com/wiki/xml/node (Link no longer active as of Feb 2016)

# DOM

(http://www.w3schools.com/js/js_htmldom.asp)

- The HTML DOM can be accessed with JavaScript (and with other programming languages).
- In the DOM, all HTML elements are defined as **objects**.
  - A **property** is a value that you can get or set (like changing the content of an HTML element).
  - A **method** is an action you can do (like add or deleting an HTML element).

# DOM

(http://www.w3schools.com/js/js_htmldom.asp)

- The most common way to access an HTML element is to use the id of the element:
  method `getElementById()`

- The easiest way to get the content of an element is by using the **innerHTML** property.

- Combining these two …..

# How to locate a DOM element

(http://www.w3schools.com/js/js_htmldom.asp)

- getElementById()

  Finding an element by element id

- getElementsByTagName() & getElementsByClassName()
  - returns a NodeList of all elements with the specified name or classname.
  - Elements can be accessed through their node number

- Ex:
```
document.getElementById('p1')
document.getElementsByTagName('div');
```

# How to edit/change a DOM element

(http://www.w3schools.com/js/js_htmldom.asp)

- getElementById(*id*).innerHTML=
  Changing the inner HTML of an element

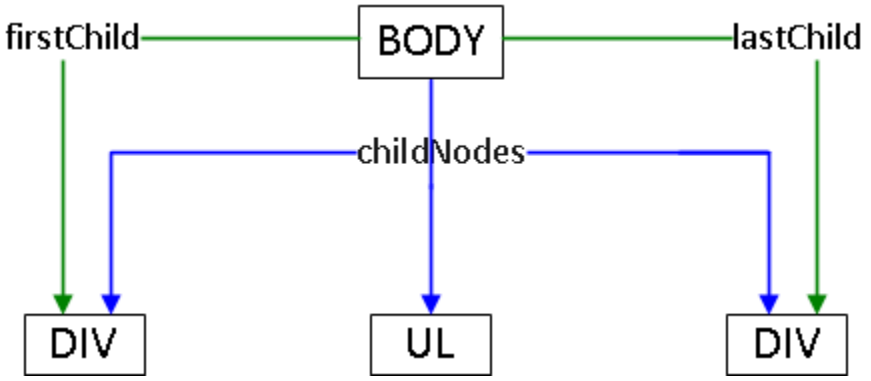
- getElementById(*id*).set*Attribute*(“attribute ”, “newvalue”);
  Changing the attribute of an element
- Example:

- getElementById(*id)*.style.*attribute*=
  Changing the style of an HTML element

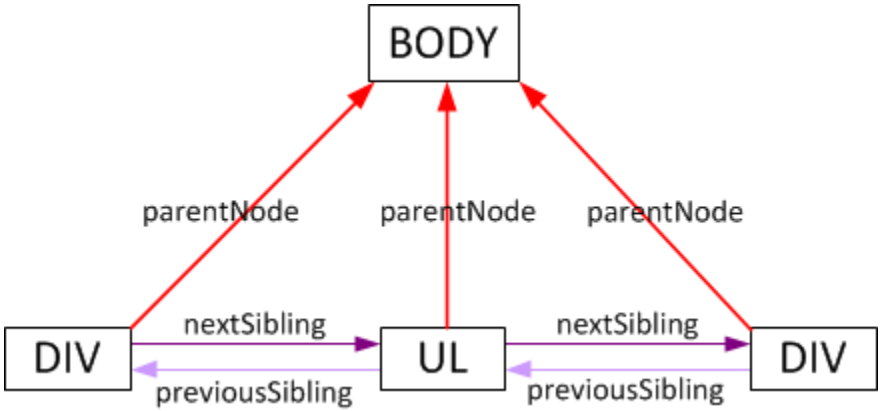
demo1.html

# How to access a specific node relative to current node

- firstChild
- lastChild




- nextSibling
- previousSibling
- parentNode




http://javascript.info/tutorial/traversing-dom

# Creating a node list

(http://www.w3schools.com/js/js_htmldom.asp)

- The getElementsByTagName() method returns a **node list**.
- A node list is an array of nodes.
- Example:

```
var h1Tags = document.getElementsByTagName("h1");
```

To manipulate the 3rd h1 element: `h1Tags[2]`

bytagname.html

# Question

Given the following code segment, how could I access the textnode `Hello World!`?

```
<body>
      Hello World! </ br>
      <p>First another thing</p>
      <div>
            <p>Second another thing</p>
      </div>
      . . . . .
```

demo2.html

# How to add/delete a DOM element

(http://www.w3schools.com/js/js_htmldom.asp)

- document.createElement()

  Create an HTML element

  Ex: `var node= document.createElement("p");`

- removeChild()

  Remove an HTML element

  demo3.html

  Ex:
  ```
  var list=document.getElementById("myList");
  list.removeChild(list.childNodes[0]);
  ```

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_node_removechild

- appendChild()

  demo4.html

  Add an HTML element

  Ex:

  `document.getElementById("myList").appendChild(`*`new Element`*`);`

# How to add/delete a DOM element

(http://www.w3schools.com/js/js_htmldom.asp)

- replaceChild()

  Replace an HTML element

- insertBefore()

  insert an HTML node as a child, before a specified existing child

demo5.html
What changes from demo4.html?

# How to see a DOM tree?

- IE9+, FX3+, and Chrome have ways to show the tree of a document
- Let's look at how Chrome display the DOM tree:

  File: **table3.html**

# DOM & JavaScript

http://www.w3schools.com/js/js_htmldom.asp

- Thanks to the object model, JavaScript has what it needs to create *dynamic* HTML.

- JavaScript can …
  - change all the HTML elements in the page
  - change all the HTML attributes in the page
  - change all the CSS styles in the page
  - remove existing HTML elements and attributes
  - add new HTML elements and attributes
  - react to all existing HTML events in the page
  - create new HTML events in the page

# DOM & JavaScript


How do we tell system to *take* the input of 2 text fields and *perform* a requested operation?

- User does something and the page reacts!
- Purpose of JavaScript!!!!
- But needs a way to know when user does something so that an *event* can happen and where to get data from

# Handling Events from …

- `body` elements
  - `load` and `unload`
- `button` elements
  - `mouseover`, `click`, `mouseout`, `dblclick`, `mousedown`, `mouseup`, `reset`, `...`
- `text box` and `password` elements
  - `blur`, `focus`, `change` and `select`

# Events and Event Handling

- An ***event*** is a notification that something specific has occurred, either with the browser or an action of the browser user

- An ***event handler*** is a script that is implicitly executed in response to the appearance of an event

- The process of connecting an event handler to an event is called ***registration***

# How do we access elements?

There are several ways to do it

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

1. DOM address

   Use the `forms` and `element` arrays of the `Document` object

   ```
   document.forms[0].elements[0]
   ```

   *Problem*: document changes

# How do we access elements? …

2. **Element names** – requires the element and all of its ancestors (except `body`) to have **`name`** attributes

Example:

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

```
document.myForm.pushMe
```

*Problem*: XHTML 1.1 spec doesn't allow the `name` attribute on `form` elements

# How do we access elements? …

3. **`getElementById`** Method (defined in DOM 1)

Example:

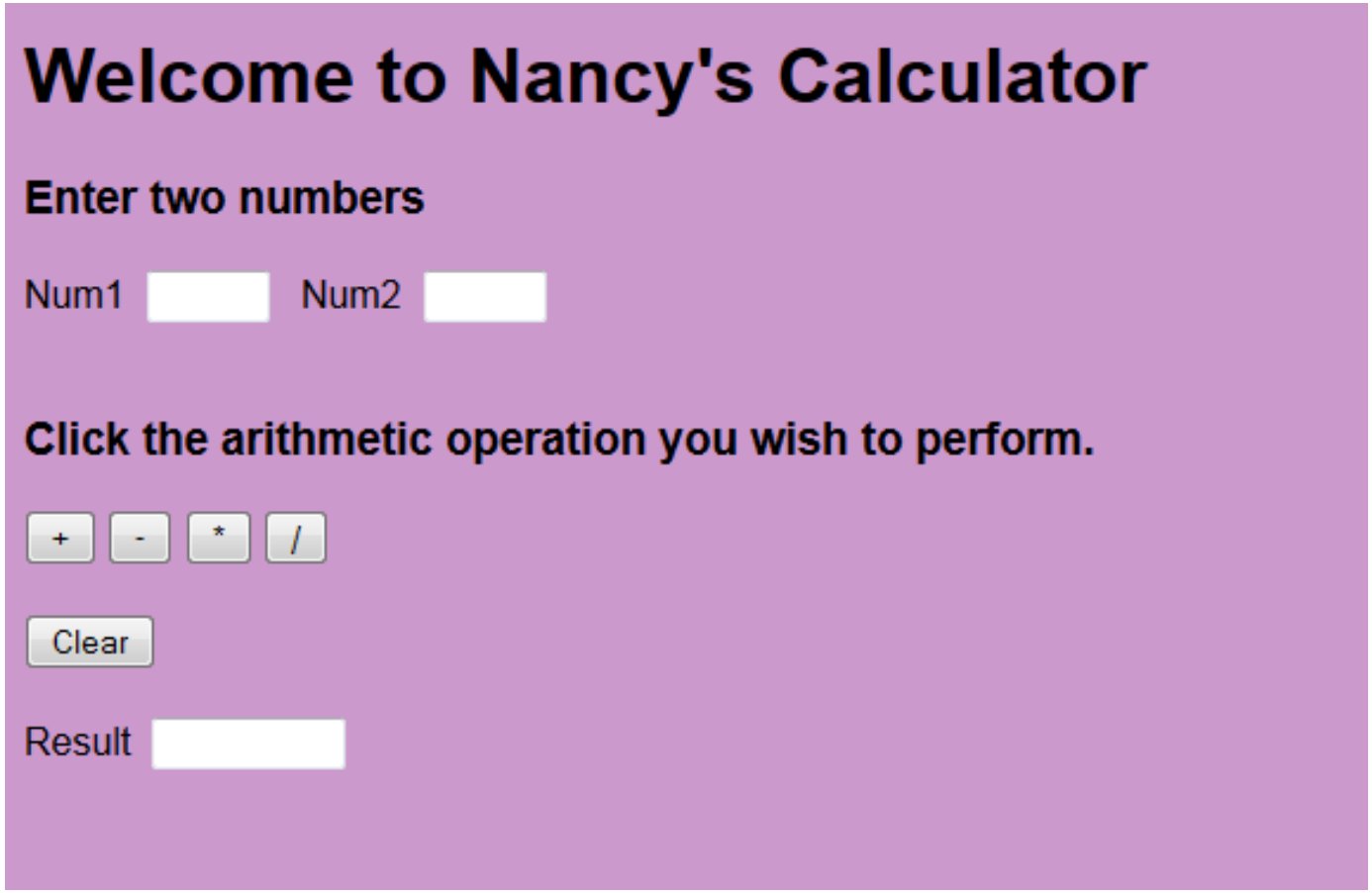
```
<form action = "">
    <input type = "button"  id = "pushMe" />
</form>

    document.getElementById("pushMe")
```

# How do we access elements? …

Checkboxes and radio button have an implicit array, which has their name

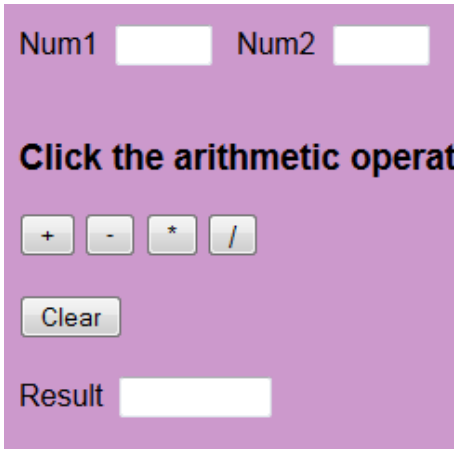
```
<form id = "topGroup" >
  <input type = "checkbox"  name = "toppings"
         value = "olives" />
  ...
  <input type = "checkbox"  name = "toppings"
         value = "tomatoes" />
</form>
...
var numChecked = 0;
var dom = document.getElementById("topGroup");
for index = 0; index < dom.toppings.length; index++)
  if (dom.toppings[index].checked]
    numChecked++;
```

# Adding functionality to an HTML file




`calculator.html` (uses **name** attribute in form element)
`calculator1.html` (uses **id** attribute in form element)

# Processing Text: Calculator.html




In the form have:

- *Input*: 2 operands into 2 text fields

```
<input type="text" size="5" name="num1" />

<input type="text" size="5" name="num2" />
```

- *Action*: The four buttons each have an `onclick` event handler

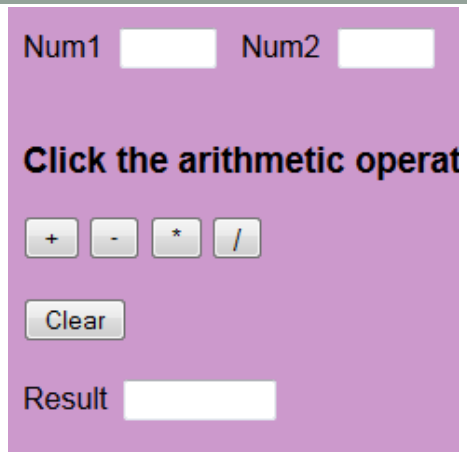
```
<input type="button" value="+" onclick="add()" />
```

- *Output*: Calculated results appears in a text field

```
<input type="text" size="10" name="result" />
```

- *Function(s)*:`add()` is a function written using JavaScript

# Code for `calculator.html` form




uses **name** attribute in form element

```
<form name = "calc" >

  Num1<input type="text" size="5" name="num1" />
  Num2<input type="text" size="5" name="num2" /><br /><br />

  <input type="button" value="+" onclick="add()" />
  <input type="button" value="-" onclick="subtract()" />
  <input type="button" value="*" onclick="multiply()" />
  <input type="button" value="/" onclick="divide()" /> <br /><br />

  <input type="reset" value="Clear" /><br /><br />

  Result<input type="text" size="10" name="result" />

</form>
```

# add() function

```
<script type="text/javascript">
<!--
function add(){
  var num1=parseFloat(document.calc.num1.value);
  var num2=parseFloat(document.calc.num2.value);
  document.calc.result.value=num1+num2;
}
. . . . .

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

Uses form name

Why do we `need parseFloat( )`?
What happens if we try to add 1.5 + 2?

# multiply() function

```
<script type="text/javascript">
<!--
function multiply(){




}
. . . . .
//-->
</script>
```

Let's look at the complete code: calculator.html
(Pay attention to the `divide()`)

# Difference between `calculator.html` & `calculator1.html`

- `<form name = "calc">` (calculator.html)

```
function add(){                                   (calculator.js)
   var num1=parseFloat(document.calc.num1.value);
   var num2=parseFloat(document.calc.num2.value);
   document.calc.result.value=num1+num2; }
```

- `<form id = "calc">` (calculator1.html)

```
function add(){                                   (calculator1.js)
    var dom=document.getElementById("calc");
    var num1=parseFloat(dom.num1.value);
    var num2=parseFloat(dom.num2.value);
    dom.result.value=num1+num2;  }
```

# Validating a Form/Processing Options

- Usually will validate a form before submitting it

**Welcome to my thrift store!**

**What would you like to buy?**

- ☐ Item1 $14.99
- ☐ Item2 $12.99
- ☐ Item3 $13.99
- ☐ Item4 $11.99

**Choose Payment Method**

- ○ Money Order (20% service charge)
- ○ Personal Check (10% service charge)
- ○ Visa (Preferred (20% discount)
- ○ MasterCard (10% discount)

Process Order | Reset Form

What type of validation would we need?

What would the DOM of this page look like?

What elements are there?

# Complete code for form

```
<form name="payform">
  <h2>Welcome to my thrift store!</h2>
  <h4>What would you like to buy?</h4><br /><br />

  <input type= _____ name="item" value="14.99" />
           Item1 $14.99<br />

  <input type= _____ name="item" value="12.99" />
           Item2 $12.99<br />

  <input type= _____ name="item" value="13.99" />
           Item3 $13.99<br />

  <input type= _____ name="item" value="14.99" />
           Item4 $11.99<br /><br />
```

# Complete code for form

```
<h4>Choose Payment Method</h4><br />
<input type= _____ name="pay" value="1.2" />
        Money Order (20% service charge)<br />

<input type= _____ name="pay" value="1.1" />
        Personal Check (10% service charge)<br />

<input type= _____ name="pay" value=".8" />
        Visa (Preferred (20% discount)<br />

<input type= _____ name="pay" value=".9" />
        MasterCard (10% discount)<br />  <br /><br />

<input type= _____ value="Process Order"
        onclick="total()" />

<input type= _____value="Reset Form" />

</form>
```

# Function `total()`

```
function total() {
  var subtotal=0;
  var buy=false;

  var elmnts=document.payform.elements;

  for(var x=0 ; x<=3 ; x++) {
       if (elmnts[x].checked) {
         subtotal=subtotal+parseFloat(elmnts[x].value);
         buy = true;
       }
    }
    if(!buy) {
       window.alert("You must buy something.");
       return false;
    }
 ... ... ...
}
```

shop.html

# Registering event handler

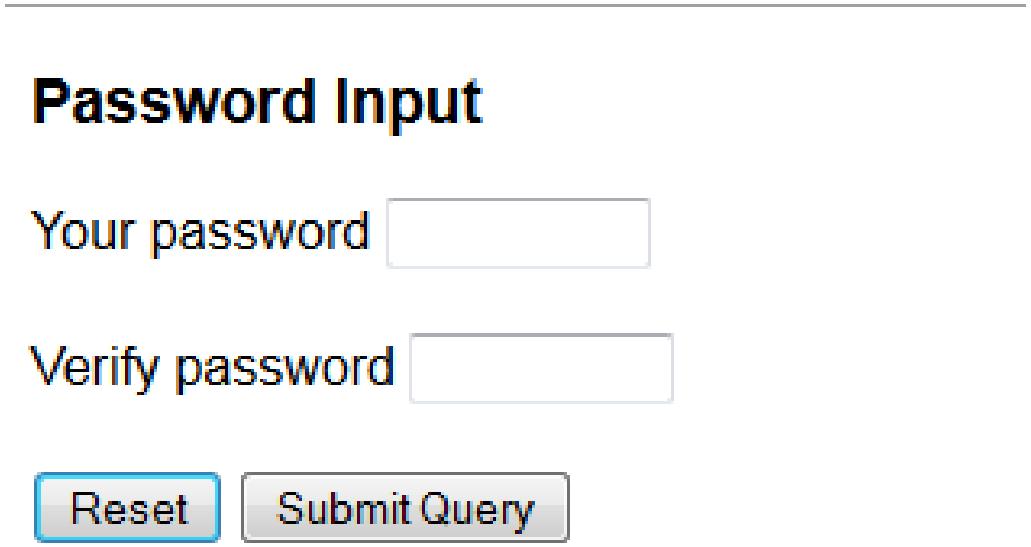
- Method 1: Assigning the event handler to an event tag attribute

```
<input type="button" value="+" onclick="add()" />
```

- Method 2: Assigning the event handler's name to the associated event property on the button object.

Our next example will illustrate this.

# Are the passwords the same?




pswd_chk.html / pswd_chk.js / pswd_chkr.js

# Validating Text

- DOM tree?

Name:

E-mail:

Postal Code:

Verify Input Clear

# Code to validate email

- Format of an email?
- Some methods/attributes to help:
  - *varname*.length
  - countchar(*varname*,"c") where c is a character
  - *varname*.indexOf("*string looking for*")
  - *varname*.lastindexOf("*string looking for*")

```
var email =
if(
```