Tables and Forms

Chapter 5

Outline

- A. Advanced CSS selectors
- B. Tables
- C. Forms
- D. CSS cascade

A. Advanced CSS selectors

- Simple selectors
  - Type, id, class, attribute
  - Pseudo-class, structural
- Combined selectors
  - Descendant selectors: E1 E2
  - Child selector: E1 > E2
  - Adjacent sibling: E1 + E2
  - General sibling

CSS style rule


```
h1 { color: red; font-size: 2em; }
```

A **selector** specifies which elements the rule applies to.

Inside a pair of braces is a list of property declarations. A **property** refers to certain appearance of the element, and the **value** is the setting.

Selectors

- A simple selector consists of either a type selector or the universal selector followed by zero or more attribute selectors, id selectors, or class selectors.
  - Also include pseudo-classes and pseudo-elements
- A combined selector consists of two or more simple selectors separated by a combinator.
  - ' ', '>', '+', '~'

```
div#chap1 p.revised img {
   border: 1px solid red;
}
```

Type and universal selectors


```
em { color: red; }
```

- An type selector matches elements of a given type
  - E.g. set font color of em elements to red

```
* { padding: 0; margin: 0; }
```

- A universal selector matches any elements
  - E.g. reset padding and margins of all elements

id selectors


```
#first { color: blue; }
```

- An id selector matches a single element in an HTML doc by its 'id' attribute
  - **#first** is equivalent to ***#first**
- E#id matches a single element of type E that has the given ID
  - p#first selects the paragraph with the ID 'first'

class selectors


```
.info { color: red; }
```

- A class selector matches elements which belong to the class
  - **.info** is equivalent to ***.info**
  - A selector with both element type and class value, e.g. **p.info**, matches elements of that type that also belong to the class.
  - A selector can also specify more than one class value, e.g. **p.info.important**

Attribute selectors

- An attribute selector matches elements based on the value of an attribute
  - **E[attr]** matches elements E with the attribute
  - **E[attr="val"]** matches elements E whose given attribute *equals* to the value
  - **E[attr^="val"]** matches elements E whose given attribute *starts* with the value
  - **E[attr$="val"]** matches elements E whose given attribute *ends* with the value
  - **E[attr~="val"]** matches elements E whose given attribute *contains* the value

Examples

- Links to external web sites (absolute URL)

```
a[href^="http://"] { … }
<a href="http://www.gov.mo/">Macao gov</a>
```

- PNG images

```
img[src$=".png"] { … }
<img src="pic.png" alt="" />
```

- Input box of type 'password'

```
input[type="password"] { … }
<input type="password" name="pw" />
```

Pseudo-classes

- Pseudo-class enables selection of elements in a certain state or position in the DOM tree
  - Dynamic - :link, :visited, :focus, :active, :hover
  - Structural - :first-child, :nth-child( ), :last-child, etc
  - Negation - :not( )
  - :target
  - UI element states - :enabled, :disabled, :checked
- Ref: http://www.w3.org/TR/css3-selectors/
- Demo: http://www.quirksmode.org/css/contents.html

Dynamic pseudo-classes, 1


```
a:visited { color: #555555 }
```

- Select links that are visited or not
  - **a:link** matches an unvisited <a>
  - **a:visited** matches a visited <a>
- Only work for links
- A link can either be visited or unvisited, but not in both states at the same time

```
a:link { color: blue; }
a:visited { color: gray; }
```

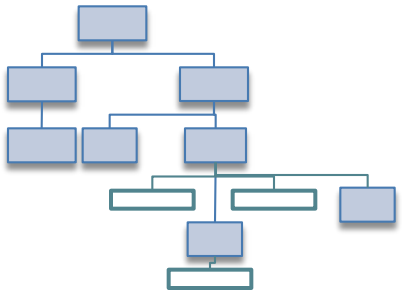
Dynamic pseudo-classes, 2


```
a:hover { text-decoration: underline; }
```

- Select elements based on user action
  - **a:hover** matches an <a> when the cursor is held over it
  - **a:active** matches an <a> when you click the link and do not release the mouse button
  - **a:focus** matches an <a> when it receives user focus
- :active and :focus also work on form controls
- :hover works on many elements like table rows, image

Structural pseudo-classes

- Match elements based on their position in the DOM tree
  - Only consider element children when determining the position



Position in the list of child elements	:first-child :last-child	:nth-child( ) :nth-last-child( )	:only-child
Position in the list of child elements of a certain type only	:first-of-type :last-of-type	:nth-of-type( ) :nth-last-of-type( )	:only-of-type
Others	:empty	:root	

Structural pseudo-classes


```
p:last-child { margin-bottom: 2em; }
```

- Select elements based on their position in the DOM tree
  - **E:first-child** matches any element E that is the first child of its parent
  - **E:last-child** matches any element E that is the last child of its parent
  - **E:nth-child(n)** matches any element E that is the nth child of its parent
  - **E:nth-last-child(n)** matches any element E that is the nth child of its parent, counting from the last child

Exercise

- Which elements do the following selectors match?

```
li:first-child
li:last-child
li:nth-child(2)
li:nth-last-child(3)
ol li:nth-child(1)
ul li:nth-last-child(1)
```

```
<body>
  <ol>
    <li>one</li>
    <li>two</li>
    <li>three</li>
    <li>four</li>
    <li>five</li>
  </ol>
  <ul>
     <li>A</li>
     <li>B</li>
  </ul>
</body>
```

:nth-child(an+b)

- :nth-child() and :nth-last-child() also accept a formula of the form an+b
  - Selects the elements in the position x, where x = an+b, n=0,1,2,3,4, … (Ignore x <= 0)
- Examples:
  - :nth-child(2n+1) matches elements at position 1, 3, 5, 7, …. I.e. elements at odd position. You can also write :nth-child(odd)
  - :nth-child(2n) matches elements at position 2, 4, 6, 8, …. I.e. elements at even position. You can also write :nth-child(even)
  - :nth-child(3n) matches elements at position 3, 6, 9. I.e. every third element

More examples of :nth-child()

- :nth-child(n+5) matches elements at position 5,6,7,… i.e. elements not in the first 4 position
- :nth-child(-n+5) matches elements at position 1,2,3,4 and 5. I.e. the first 5 children
- :nth-last-child(-n+5) matches the last 5 children
- :nth-last-child(n+5) matches elements except the last 4children
- :nth-chlid(n+2):nth-last-child(n+2) matches elements except the first and last child
  - May also be written as :not(:first-child):not(:last-child)

:nth-of-type() ,.etc

- Four similar pseudo-classes count elements of the same type only
  - **E:first-of-type** matches any element E that is the first sibling of its type
  - **E:last-of-type** matches any element E that is the last sibling of its type
  - **E:nth-of-type(n)** matches any element E that is the nth sibling of its type
  - **E:nth-last-of-type(n)** matches any element E that is the nth sibling of its type, counting from the last child

Exercise

- Which elements do the following selectors match?
- Can you select each h2 element using :nth-child( ) ?

```
h2
h2:first-of-type
h2:last-of-type
h2:nth-of-type(2)
p:nth-of-type(even)
```

```
<body>
  <h1>..</h1>
  <p>..</p>
  <h2>..</h2>
  <p>..</p>
  <p>..</p>
  <h2>..</h2>
  <p>..</p>
  <h2>..</h2>
  <p>..</p>
</body>
```

Example

- Add divider between consecutive

```
div:nth-child(n+2) {
  border-top: 1px dashed green;
}
```

OR

```
div:nth-last-child(n+2) {
  border-bottom: 1px dashed green;
}
```

```
<body>
  <div>first</div>
  <div>second</div>
  <div>third</div>
  <div>fourth</div>
</body>
```

First div

Second div

Third div

Fourth div

Negation E:not(s)

- Matches any E element that does not match the simple selector s.
  - p:not(:first-child) matches p that is not a first child
  - p:not(.important) matches p that is not in the class 'important'
  - :not(p) matches any elements other than p (e.g. div, em)
  - img:not([src$="png"]) matches img of format other than png (e.g. gif, jpg)
  - p:not(:first-child):not(:last-child) matches all p except the first and last child

Pseudo-element selectors

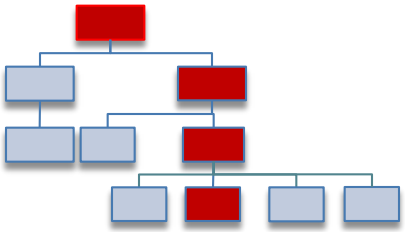
- Pseudo-element selectors match part of an element
  - p::first-line matches the first line of the <p>
  - p::first-letter matches the first letter of the <p>
  - CSS2 uses the syntax p:first-line and p:first-letter
  - Ref and examples:
    - http://www.w3.org/TR/css3-selectors/#pseudo-elements
    - http://www.w3schools.com/css/css_pseudo_elements.asp

Generated content

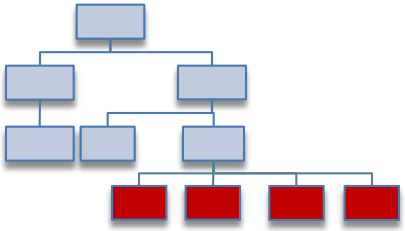
- Two special pseudo-elements ::before and ::after allows adding content before or after some elements
  - E.g. p.note::before { content: "Note: " } inserts "Note: " before the content of every paragraph in the class 'note'.
  - E.g. p:last-child::after { content: url(signature.png) } inserts an image at the end of the last paragraph
  - For more examples, search 'css generated content'

Combined selectors

- Combine two or more simple selectors
- Four kinds:
  - Descendant selectors: E1 E2
  - Child selector: E1 > E2



  - Adjacent sibling: E1 + E2
  - General sibling: E1 ~ E2



Descendant selectors


```
ul li { list-style-type: square; }
```

- Select elements that are descendants of another element
- Form: **E1 E2**, where E1 is a simple selector for the ancestor, and E2 is a simple selector for the descendant.
- Note: A space character separates the simple selectors in descendant selectors. There is no space within a simple selector. E.g. "p.revised" is different from "p .revised"

Example

- Note that there may be more than one elements between the ancestor and descendants.

```
div#toc span.attr { color: blue; }
…
<div id="toc">
  <h1>Core attributes</h1>
  <ul>
    <li><span class="attr">id</span></li>
    <li><span class="attr">class</span></li>
  </ul>
</div>
```

Child selectors


```
ul>li { border-left: 1px solid red; }
```

- Select elements that are direct child of another element
- Form: **E1 > E2**, where E1 is a simple selector for the parent, and E2 is a simple selector for the child.

Example

- Compare the difference between **ul>li** and **ul li**
- How to select the list items in the nested ol?

```
ul>li { border-left: 1px solid red; }
...
<!-- an ordered list inside an unordered list -->
<ul>
    <li>...</li>
    <li>...</li>
    <li>
      <ol><li>...</li> <li>...</li></ol>
    </li>
</ul>
```

Adjacent sibling selectors


```
h2+h3 { margin-top: -5mm; }
```

- Select a sibling element that is immediately preceded by another element
- Form: **E1 + E2**, where E1 is a simple selector for the preceding sibling, and E2 is a simple selector for the element to be selected. Both elements must have the same parent.

General sibling selectors


```
h1~p { font-size: larger; }
```

- Select a sibling element that is preceded by another element
- Form: **E1 ~ E2**, where E1 is a simple selector for the preceding sibling, and E2 is a simple selector for the element to be selected. Both elements must have the same parent. *There may be other siblings between E1 and E2.*

Example

- Compare the difference between **h1+h2** and **h1~h2** in this example

```
h1+h2 { margin-top: -5mm  }
…
<h1>Core attributes</h1>
<h2>The id attribute</h2>
<p> … </p>
<h2>The class attribute</h2>
<p> …</p>
```

Grouping selectors

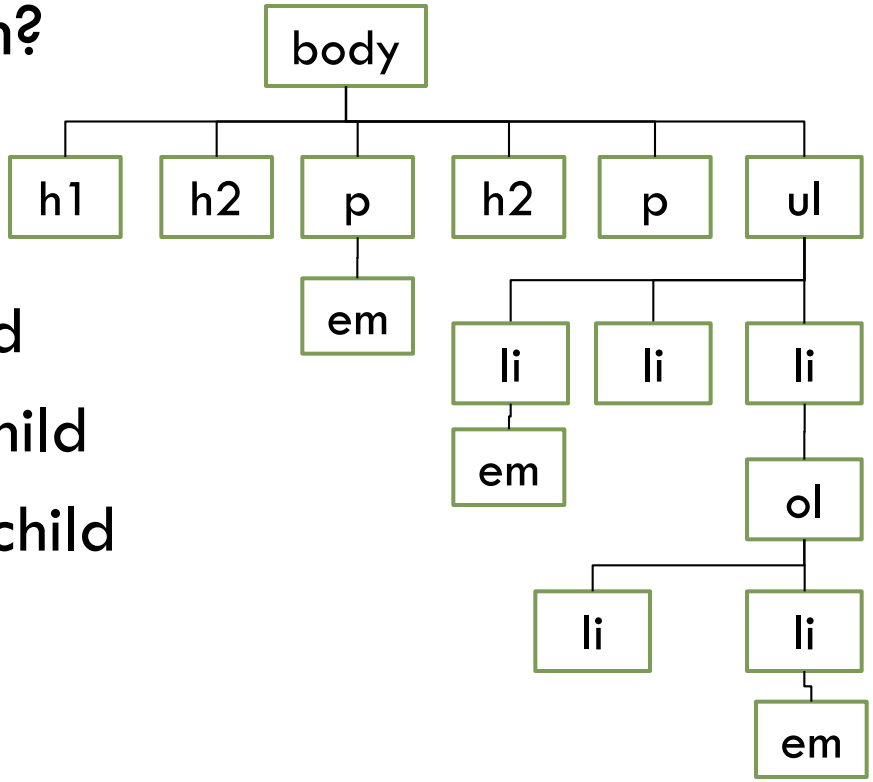
- You can group selectors using comma

```
#summary { color: red; }
.attr { color: red; }
h1 { color: red; }
```

```
#summary, .attr, h1 { color: red; }
```

Exercise

- Which elements do the following selectors match?
  - li em
  - ul li
  - ul>li
  - li li
  - h1+h2
  - h1~h2
  - ul * em
  - ul>*>em
  - li:first-child
  - ul li:first-child
  - ul>li:first-child
  - h2+p em
  - h1+h2+p
  - h2~ul ol em



Summary: selectors

Selectors	pattern
Type selector	E
id selector	#id
Class selector	.class
Universal selector	*
Attribute selector	E[attr], E[attr=val], …
Pseudo-class	:first-child, :nth-child( ), …
Pseudo-element	::first-letter, ::first-line, …
Descendant selector	E1 E2
Child selector	E1>E2
Adjacent sibling selector	E1+E2
General sibling selector	E1~E2

B. Table

- HTML table is a collection of data arranged in rows and columns
  - Row centric: list table cells in rows
  - Merging cells: a cell can span multiple rows and columns
  - Additional info: caption
  - Additional structure: columns, row groups, column groups
- Table styles
  - Formatting the grid of cells
  - Formatting the internal of cells

Name	Test 1	Test 2
Peter	90	70
Mary	100	80

Basic syntax

- A <table> contains several rows <tr>, each row contains several cells. A cell is either a data cell <td> or heading cell <th> .

```
<table>
  <tr><th>Name</th><th>Course</th><th>Exam</th></tr>
  <tr><th>Peter</th><td>70</td><td>80</td></tr>
  <tr><th>Mary</th><td>80</td><td>85</td></tr>
  <tr><th>John</th><td>65</td><td>70</td></tr>
</table>
```

Name	Course	Exam
Peter	70	80
Mary	80	85
John	65	70

2D structure of tables

- HTML tables are used to represent two-dimensional data
  - A header cell <th> on the first row describes the data in a column
  - A header cell <th> on the first column describes the data in a row
  - May be specified with the attribute 'scope'

Name	Course	Exam
Peter	70	80
Mary	80	85
John	65	70

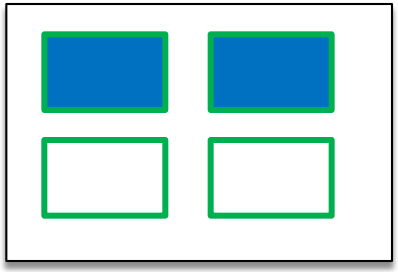
Basics of table styling

- Each table cell is a box
  - width and height properties indicate the minimum dimension
  - You can configure padding, border, but not margin
  - the default background is transparent.
  - a cell may contain inline and block elements, you can format them as in other block box.
- A table is also a box
  - Similar to block box
  - You can configure width, height, padding, border, margin

Table cells *on* a table

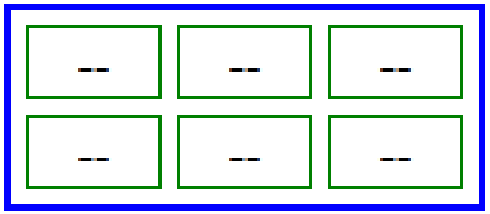
- Background of table cells appear on top of that of the table.
- Ref: http://www.w3.org/TR/CSS2/tables.html#table-layers

```
table { background-color: white;
   border: 1px solid black; }
th, td { border: 1px solid green; }
th { background-color: blue; }

<table>
   <tr><th>…</th><th>…</th></tr>
   <tr><td>…</td><td>…</td></tr>
</table>
```

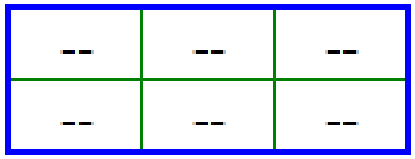


Border models

- Borders of table cells may be separate or shared
- Separated borders model (default):
  table { border-collapse: separate; }
- Collapsing border model:
  table { border-collapse: collapse; }



Separate borders model

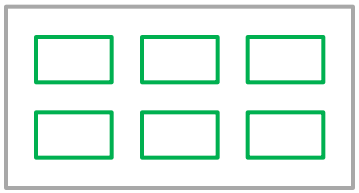


Collapsing border model

Separated borders models

- Each cell has its own border. The table has its border too.
  - Selected by table { border-collapse: separate }
  - Property border-spacing sets the spacing between cells. Background of the table shows through this spacing.
  - May hide empty cell by table { empty-cells: hide }

```
table td {
  border: 1px solid green; }
table {
  border: 1px solid darkgray;
  border-spacing: 5px 5px;
}
```



Collapsing border model

- Adjacent cells share a border
  - Selected by table { border-collapse: collapse }
  - Borders of the table and its table cells are merged.
  - You can set borders of multiple cells by setting the border of a row, row group, column, or column group.

```
table thead {
  border-bottom: 1px solid green; }
table colgroup {
  border-right: 1px solid darkgray; }
```

X	X	X	X
X	X	X	X
X	X	X	X
X	X	X	X

Alignment inside a table cell

- In a table cell, vertical-align aligns content vertically and text-align aligns inline content horizontally
  - td { vertical-align: middle; text-align: center }

	vertical-align:top	vertical-align:middle	vertical-align:bottom
text-align: left	top left	middle left	bottom left
text-align: center	center top	center middle	center bottom
text-align: right	right top	right middle	right bottom

Table head, body and foot

- Rows in a long table may be grouped
  - <thead> : table head contains headers
  - <tbody> : table body for the main data
  - <tfoot> : table footer contains summary info
- Table head is shown before table body. Table foot is shown after.
  - Note: HTML4 and XHTML requires putting <tfoot> before <tbody> in HTML source
- There can be more than one <tbody>

Example

Name	Course	Exam
Peter	70	80
Mary	80	85
John	65	70
Average	71.7	78.3

```
<table>
  <thead>
    <tr><th>Name</th><th>Course</th><th>Exam</th></tr>
  </thead>
  <tbody>
    <tr><th>Peter</th><td>70</td><td>80</td></tr>
    <tr><th>Mary</th><td>80</td><td>85</td></tr>
    <tr><th>John</th><td>65</td><td>70</td></tr>
  </tbody>
  <tfoot>
    <tr><th>Average</th><td>71.7</td><td>78.3</td></tr>
  </tfoot>
</table>
```

Table caption

- Add a caption to a table
  - To set the location of the caption, table { caption-side: bottom }

```
<table>
  <caption>Mark sheet for COMP113</caption>
  <thead> … </thead>
  <tbody> … </body>
  <tfoot> … </tfoot>
</table>
```

Mark sheet for COMP113

Name	Course	Exam	Grade
Peter	70	80	B+
Mary	80	85	A-
John	65	70	C
Average	71.7	78.3	B

Row groups

- You can group rows into more than one groups using <tbody>

```
<table id='T1'>
  <thead>  .. </thead>
  <tbody id='y1'> <tr>..<tr>..<tr>.. </tbody>
  <tbody id='y2'> <tr>..<tr>..<tr>.. </tbody>
  <tbody id='y3'> <tr>.. <tr>.. </tbody>
  <tbody id='y4'> <tr>.. </tbody>
</table>
```

Code	Course
COMP112	Programming I
COMP113	Web Technologies
COMP122	Data Structures and Algorithms
COMP212	Programming II
COMP221	Object Oriented Technologies
COMP222	Internet Programming I
COMP312	Internet Programming II
COMP321	Information System Implementation
COMP491	Final Year Project

Applying styles to cells in rows

- To apply styles to cells which are descendants of a row / row group
  - If the property is inherited (e.g. font-related), you only need to select the row / row group. Example: table thead { color: red }
  - If the property is not inherited (e.g. box properties), you have to select descendant cells, Example: table tbody td { border: 1px dashed green }

Merging table cells

- A table cell can span multiple rows and columns using the attributes rowspan and colspan

<th colspan='2'>

Student		Mark
p1030001	Ann	80
p1030002	Boris	
p1030003	Cecilia	76
p1030004	Debra	

<td rowspan='2'>

Example

Student		Mark
p1030001	Ann	80
p1030002	Boris	
p1030003	Cecilia	76
p1030004	Debra	

```
<table>
  <thead><tr><th colspan='2'>Student</th>
                <th>Mark</th></tr></thead>
  <tbody>
    <tr><td>p1030001</td><td>Ann</td>
          <td rowspan='2'>80</td></tr>
    <tr><td>p1030002</td><td>Boris</td></tr>
    <tr><td>p1030003</td><td>Cecilia</td>
          <td rowspan='2'>76</td></tr>
    <tr><td>p1030004</td><td>Debra</td></tr>
  </tbody>
 </table>
```

Notice that cell merging result in less table cells in some rows.

Optional tags

- End tags for <thead>, <tbody>, <tfoot>, <tr>, <th> and <td> are generally optional

```
<table>
  <caption>Mark sheet for COMP113</caption>
  <thead>
    <tr><th scope='col'>Name<th scope='col'>Course
        <th scope='col'>Exam<th scope='col'>Grade
  <tbody>
    <tr><th scope='row'>Peter<td>70<td>80<td>B+
    <tr><th scope='row'>Mary<td>80<td>85<td>A-
    <tr><th scope='row'>John<td>65<td>70<td>C
  <tfoot>
    <tr><th scope='row'>Average<td>71.7<td>78.3<td>B
</table>
```

Further reading

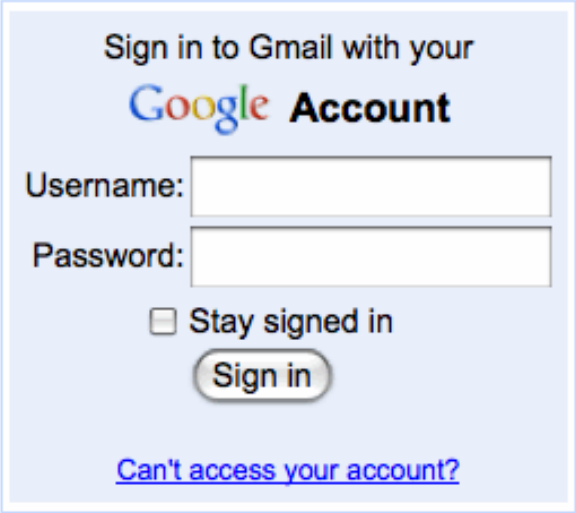
- CSS table formatting examples
  - http://www.smashingmagazine.com/2008/08/13/top-10-css-table-designs/
  - http://designshack.co.uk/articles/10-css-table-examples

C. Form

- Form and form controls
- Adding style to forms

Forms

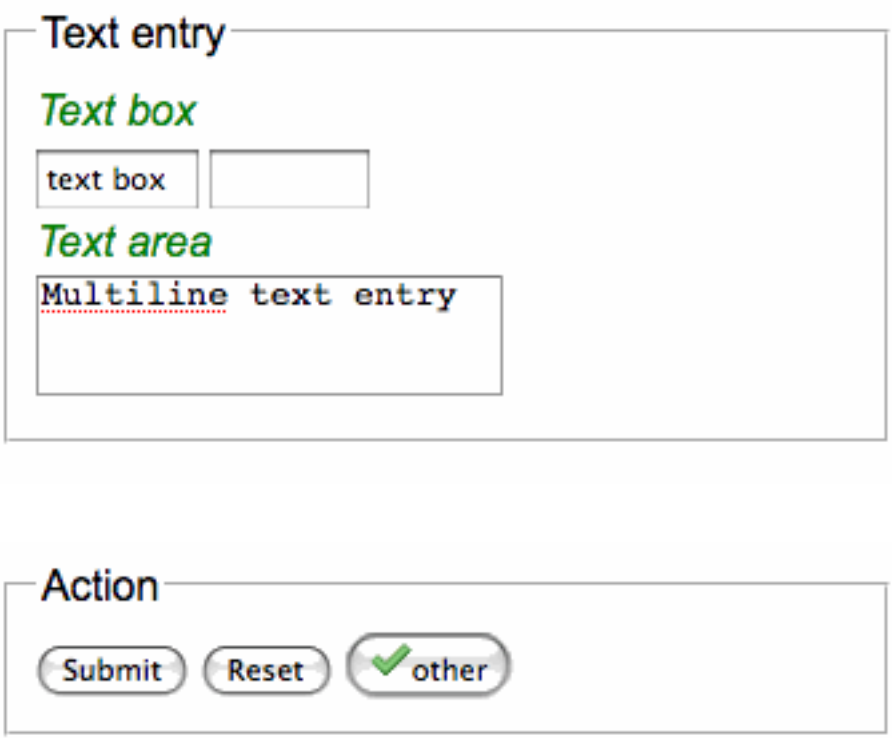
- Forms enable users to enter info for processing
  - A form contains various form controls to collect data
  - Form data may be processed by
    - Client-side JavaScript code, and/or
    - Server-side script at an URL

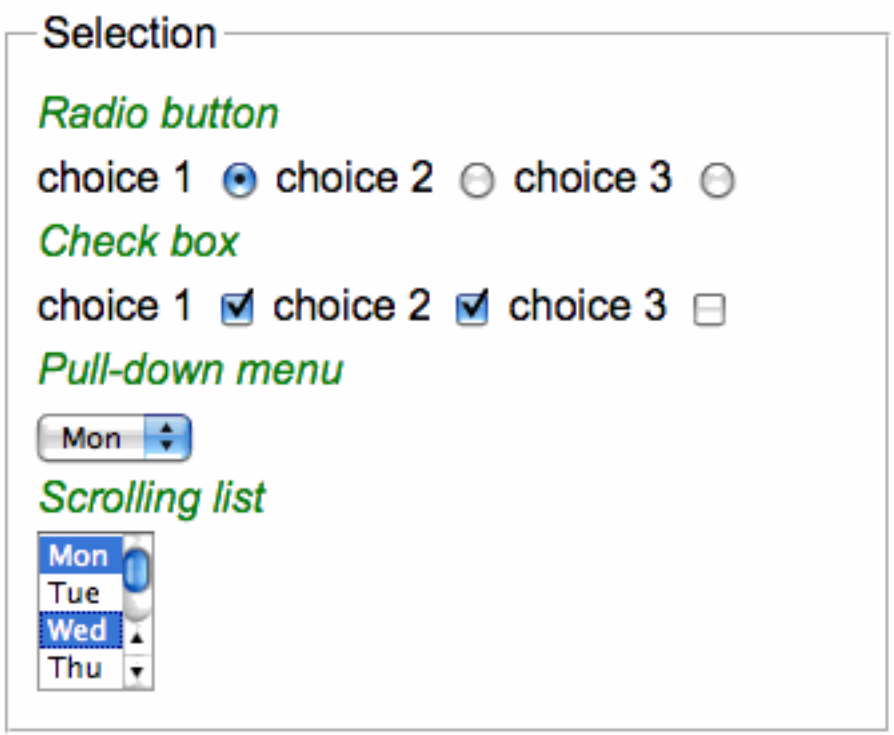


Login form of Gmail

Form controls

Three main kinds of form controls



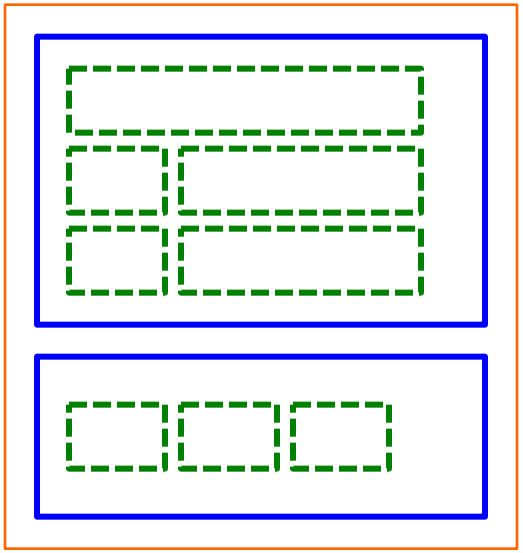


The form element

- <form> is a block element with three attributes
  - action (required) – URL of server side script to receive the form data when the form is submitted
  - method (optional) – HTTP method to send the data
    - get (default), or post
  - enctype (optional) – (encoding type) how form data is encoded as string
    - application/x-www-form-urlencoded (default) is suitable for small amount of data
    - multipart/form-data is suitable for file upload. Need method="post"

Content of the form element

- Usually we organize content in a <form> in blocks



<form>

Block elements, e.g. <p>, , <table>, <fieldset>

Text and inline elements, e.g. <a>, <img>, <input>, <select>, <textarea>

Form data

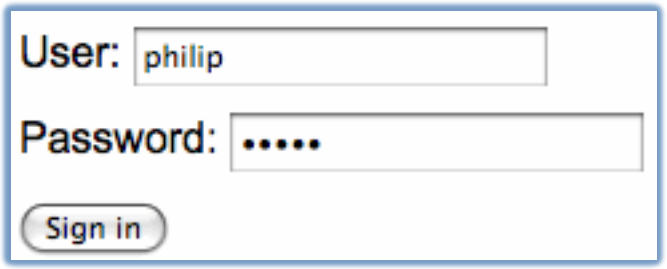
- A form collects data from form controls as name-value pairs
  - E.g. user=peter, passwd=123
  - Each form control must have the attribute name
    - attributes id and name have different meaning
  - Users interact with a control to set its value
  - Controls specify their initial values in different ways: attributes value, checked, selected and element content

Usage of common form controls

Task	UI	HTML element
Simple text entry	Text box	<input type="text" />
Password entry	Text box	<input type="password" />
Multiline text-entry	Text area	<textarea>
Yes / No	Checkbox	<input type="checkbox" />
Choose one from a group	Radio button	<input type="radio" />
	Pull-down menu	<select>
Choose many from a group	Checkbox group	<input type="checkbox" />
	Scrolling list	<select multiple="multiple">
Submit form	Submit button	<input type="submit" />
Other action	General button	 <button type="button">

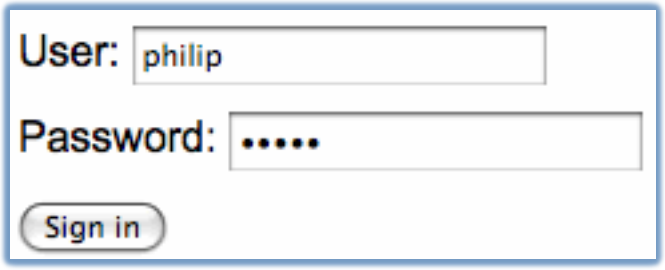
Simple text entry

- Use <input type="text" /> for simple text entry.
- Optional attributes:
  - size : length in number of characters
  - maxlength : max number of char the user can enter
  - value : initial value
  - placeholder: display when the text field is empty
- Use <input type="password" /> to mask the input



Example


```
<form action="login.jsp" method="get">
  <p>User: <input type="text" name="user" /> </p>
  <p>Password: <input type="password" name="passwd" /> </p>
  <p><input type="submit" value="Sign in"/> </p>
</form>
```



This form will send user=philip, passwd=12345

Label

- <label> links a text label to a control
  - When you click the label, the form control gains the input focus
  - Associate the label and the form control by the attribute for and id .

```
<p><label for="fname">Name</label>
<input type="text" name="fname" id="fname" />
</p>
<p><label for="fstudid">Student ID</label>
<input type="text" name="fstudid" id="fstudid" />
</p>
```

Name
Student ID

Note: the 'name' attribute is used for form submission only. It is not required to have the same value as the 'id' attribute.

Label, a shorthand

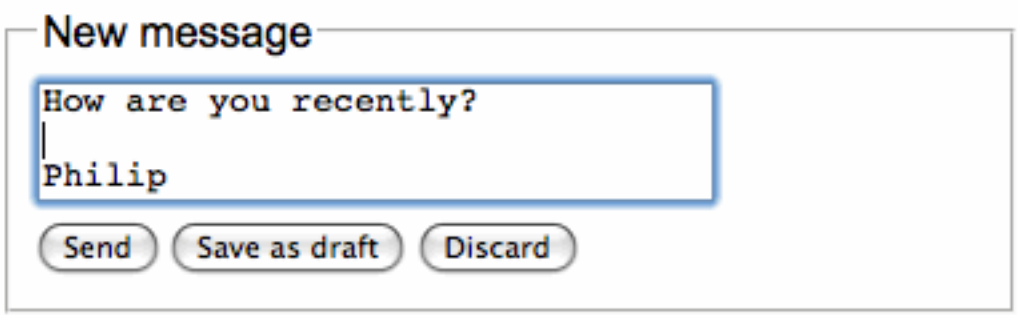
- If the <label> and its associated form control are near, you can put the control inside the <label>
  - Don't need to assign id to the control

```
<p>
  <label>Name
  <input type="text" name="fname" /> </label>
</p>
<p>
  <label>Student ID
  <input type="text" name="fstudid" /> </label>
</p>
```

Multiline text entry

- Use <textarea> to enter more than one lines of text
  - <textarea name= "mesg" rows="3" cols="40">initial value</textarea>

```
<p><textarea name="mesg" rows= "3" cols= "40" ></textarea> </p>
<p><input type= "submit" name= "action" value= "Send" />
<input type= "submit" name= "action" value= "Save as draft" />
<input type= "submit" name= "action" value= "Discard" /></p>
```



This form will send
mesg=How+are+you+recently%3F%0D%0A%0D%0APhilip
action=Send

Hidden field

- No user interface shown. Send data when the form is submitted
  - <input type="hidden" name="customerid" value="123"/>
- Useful to carry form data from one page to another

New type of <input>


```
<p>Mobile: <input type="tel" name="mobile" /> </p>
<p>Email: <input type="email" name="email" /> </p>
```

- HTML5 defines some new input types, but not all of them are implemented.
  - email, url, tel, number, color, date, range, …
- Current state: http://wufoo.com/html5/
- http://html5doctor.com/html5-forms-input-types/
- http://www.html5tutorial.info/index.php

Styling text fields

- Use attribute selector to select, e.g. input[type="text"]
- Select text field in focus: input:focus
- You can set the following CSS properties:
  - Box properties, incl. width, height, border, background
  - Font-related properties. Note: these properties are not inherited from parents for form controls.
- Similar for <textarea>

```
input[type="text"] {
  border: 1px solid gray;
  font-family: sans-serif;
}
input:focus { background-color: yellow; }
```

Checkbox

- Use a checkbox to ask a yes/no question.
  - <input type="checkbox" name="agree" />
  - To check it by default, add the boolean attribute checked
  - If checked, send name=on, or name=value if attribute value is set
  - If not checked, don't send anything.

```
Do you agree?
<input type="checkbox" name="agree" value="yes" checked="checked" />
```

Do you agree? ☑

This form will send agree=yes

Radio button

- Use radio buttons to select one choice from a group
  - <input type="radio" name="color" value="red"/>
  - Add the boolean attribute checked to preselect a choice
  - If checked, send name=value

```
Choose a color:
Red <input type="radio" name="color" value="red" checked/>
Green <input type="radio" name="color" value="green"/>
Blue <input type="radio" name="color" value="blue"/>
```

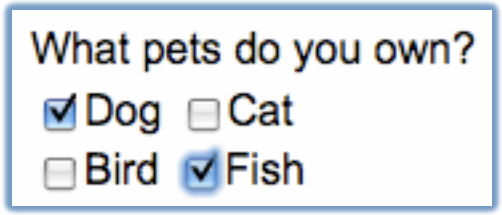
Choose a color: Red ◉ Green ○ Blue ○

This form will send color=red

Checkbox group

- A checkbox group allows a user to select more than one item

```
What pets do you own? <br/>
<input type="checkbox" name="pets" value="dog" />Dog
<input type="checkbox" name="pets" value="cat" />Cat <br/>
<input type="checkbox" name="pets" value="bird" />Bird
<input type="checkbox" name="pets" value="fish" />Fish
```



This form will send pets=dog, pets=fish

Adding labels to radio buttons

- When you click the label associated with a radio button, the form control is checked
  - Improve accessibility: make your radio button easier to check.
  - also work for checkbox

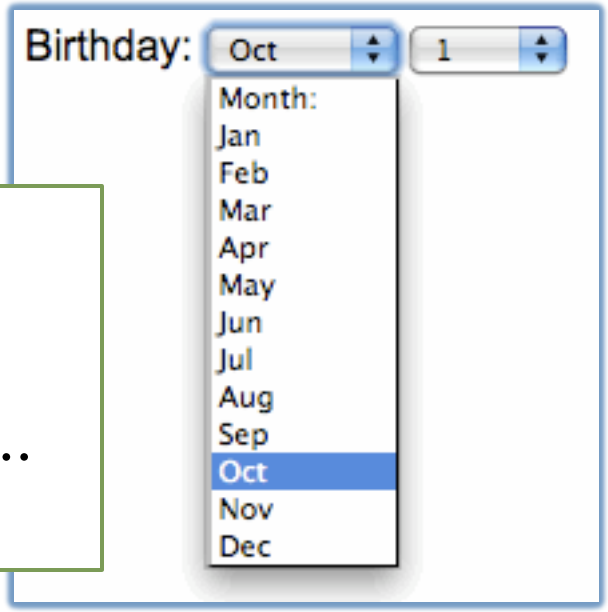
Choose a color: Red Green Blue

```
<label>Red
  <input type="radio" name="color" value="red" checked="checked"/>
</label>
<label>Green
  <input type="radio" name="color" value="green"/></label>
<label>Blue
  <input type="radio" name="color" value="blue"/></label>
```

Pull-down menu

- A compact control to select one option from many
  - <select name="month" > <option> … </select>
  - Use the boolean attribute selected to preselect an option
  - Send name=content of the selected option, or the option's attribute value if present.

```
<select name="month">
  <option selected="selected">Month:</option>
  <option>Jan</option>
  <option>Feb</option> <option>Mar</option> …
</select>
```



Example


```
<select name="month">
  <option>Jan</option>
  <option>Feb</option>
  <option>Mar</option> …
</select>
```

This form will send month=Jan, if the user selects 'Jan'

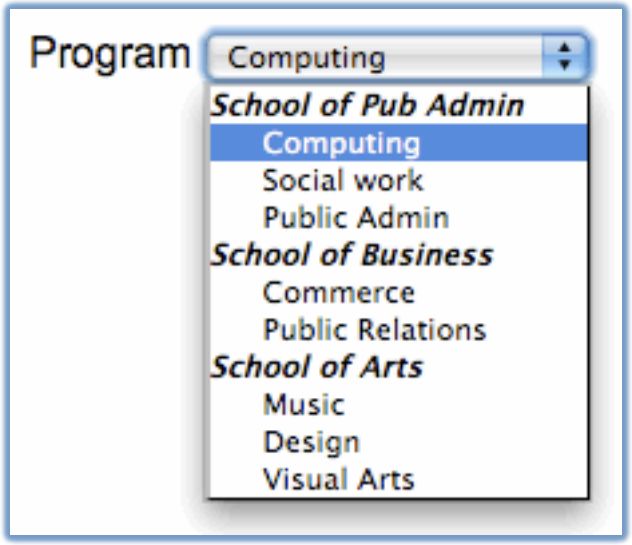
```
<select name="month">
  <option value="1">Jan</option>
  <option value="2">Feb</option>
  <option value="3">Mar</option> …
</select>
```

This form will send month=1. The attribute value is useful when the option content is too verbose.

Option group

- To improve accessibility, you can break a large number of options into groups.

```
<label for="fprog">Program</label>
<select name="fprog" id="fprog">
  <optgroup label="School of Pub Admin">
    <option>Computing</option>
    <option>Social work</option>
    <option>Public Admin</option>
  </optgroup>
  <optgroup label="School of Business">
    <option>Commerce</option>
    <option>Public Relations</option>
  </optgroup> …
</select>
```



Optional tags in option groups

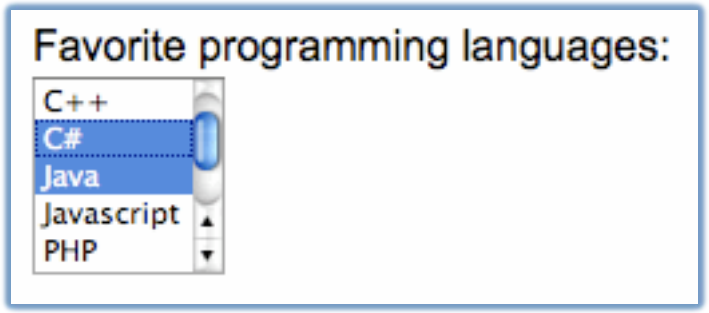
- The end tags of both <option> and <optgroup> are optional

```
<label for="fprog">Program</label>
<select name="fprog" id="fprog">
  <optgroup label="School of Pub Admin">
    <option>Computing</option>
    <option>Social work</option>
    <option>Public Admin</option>
  </optgroup>
  <optgroup label="School of Business">
    <option>Commerce</option>
    <option>Public Relations</option>
  </optgroup> ...
</select>
```

Scrolling list

- A compact control to select *multiple options* from many
  - Boolean attribute multiple
  - size is the number of options shown

```
<select name="proglang"
    multiple="multiple" size="5">
  <option>C++</option>
  <option>C#</option> …
</select>
```



Summary of selection controls

	A few options. You want to display all.	Too many options. You want to hide some.
Select one option only	Radio button <input type="radio" />	Pull-down menu <select >
Select multiple options	Check box group <input type="checkbox" />	Scrolling list <select multiple size="5">

```
Red <input type="radio" name="color"
value="red" checked />
Green <input type="radio" name="color"
value="green"/>
Blue <input type="radio" name="color"
value="blue"/>
```

```
<select name="month">
  <option value="1" selected >
Jan</option>
  <option value="2">Feb</option>
  <option value="3">Mar</option> …
</select>
```

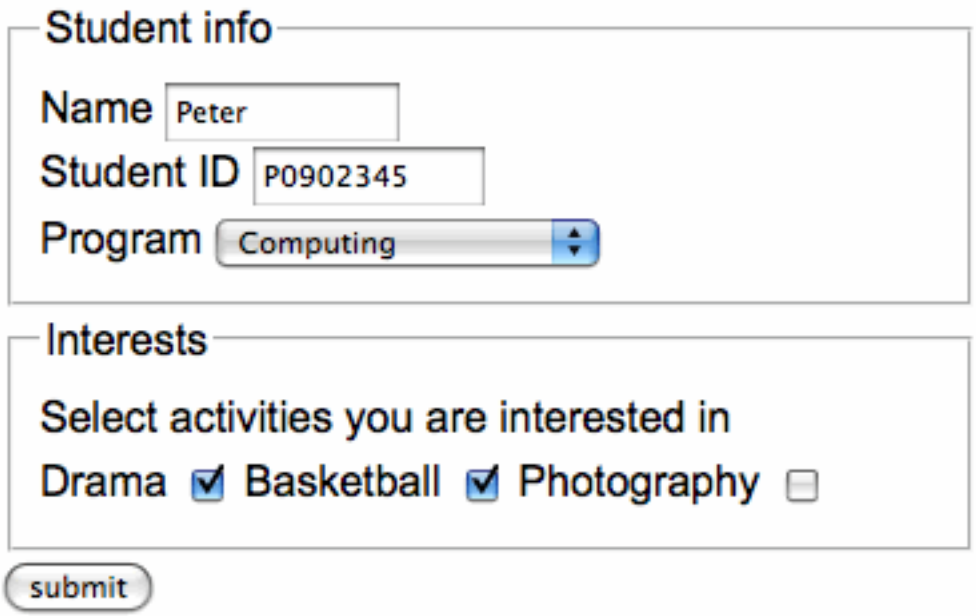
Styling selection controls

- You can configure font-related properties for <select>
  - Note: these controls don't inherit font properties from <body>
- Pseudo-class input:focus and input:checked work
- Browsers do not support much styling for checkboxes and radio buttons
- Ref
  - http://www.456bereastreet.com/archive/200701/styling_form_controls_with_css_revisited/
  - http://www.quirksmode.org/css/enabled.html
  - http://www.quirksmode.org/css/focus.html

Field set

- <fieldset> is a block element that groups related form controls
  - <legend> provides description of the group
  - You may style the border of the fieldset and text style of legend

```
<fieldset>
  <legend>Student info</legend>
  Name <input … /> …
  Student ID <input …/> …
  Program <select> …
</fieldset>
…
```



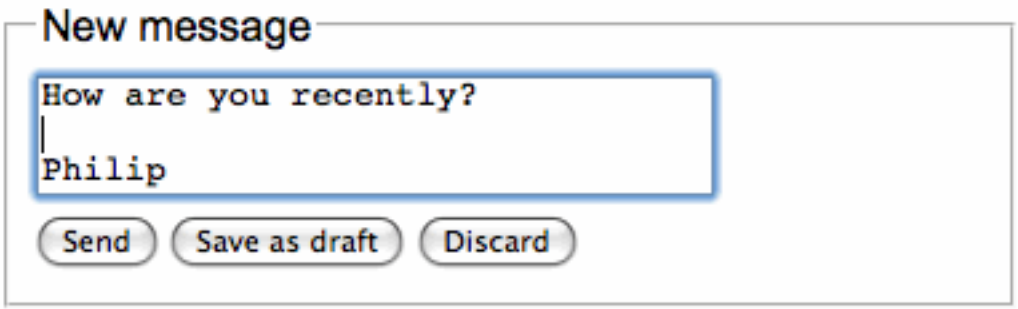
Buttons

- HTML defines three types of buttons
  - Submit buttons: submits a form
  - Reset buttons: resets all controls to their initial values
  - Push buttons: no default behavior. Web authors associate client-side scripts to the buttons.
- Two ways to make a button type=submit / reset / button
  - <input type="submit" value="content" /> uses the attribute vale as button content
  - <button type="submit"> content </button> allows images inside button

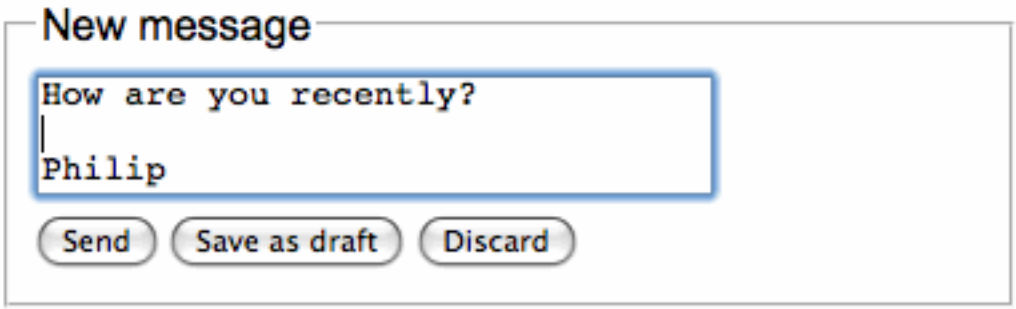
Submit button

- The user presses this button to submit the form
  - <input type="submit" name="action" value="Send"/>
  - When pressed, the form collects data from form controls and send it to server
  - Pressed button also sends name=value
  - Use several submit buttons with different values to distinguish user intention

Note: the 'value' is shown in the button



Example




```
<form action="#">
<fieldset>
   <legend>New message</legend>
   <p><textarea name="mesg" rows="3" cols="30"></textarea></p>
   <p><input type="submit" name="action" value="Send"/>
   <input type="submit" name="action" value="Save as draft"/>
   <input type="submit" name="action" value="Discard"/></p>
 </fieldset>
 </form> …
```

Reset button and Push button

- When the user presses the reset button, all form controls return to their initial values
  - <input type="reset" value="start over"/>
  - The form is not submitted
  - Not used much in modern web pages
- You can define a push button that triggers a JavaScript function when clicked
  - <input type="button" value="check spelling" onclick="alert('to be implemented'); "/>

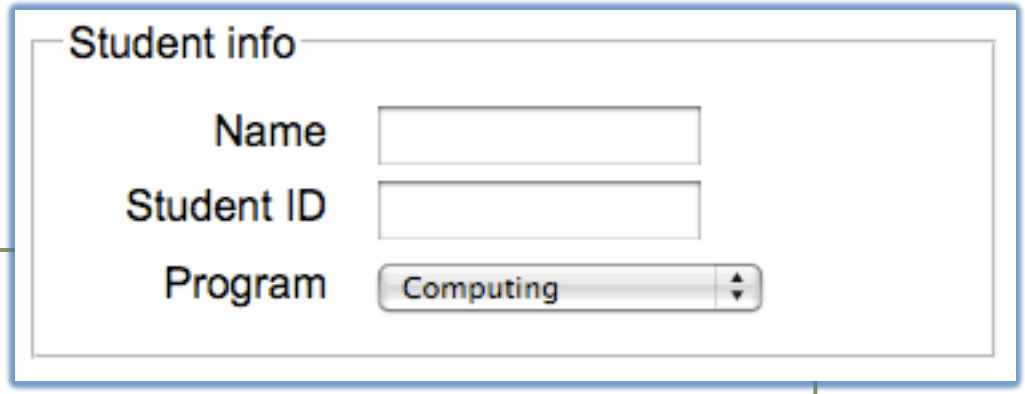
<button>

- The <button> element can use both image and text as content
  - <button type="submit" name="action" value="save">Save</button> submits form and send action=save.
  - <button type="button">Save</button> triggers JavaScript script associate to the click event of the button.


```
<button type="button">

Ok</button>
<button type="button">

Cancel</button>
```

Formatting form in a grid

- A common method to align form labels and controls is <table>
  - Easy to use, but makes the HTML code confusing
  - Pure CSS method in lab




```
<form action="#">
<table>
  <tr><td><label ...></td><td><input ... /></td></tr>
  <tr><td><label ...></td><td><input ... /></td></tr>
  <tr><td><label ...></td><td><select... ></td></tr>
</table>
</form>
```

Further reading

- More attributes for form controls
  - tabindex determines the order that the input focus shifts from control to control when the user presses the Tab key
  - disabled disables a control. The form will not send its value when submitted
  - readonly makes a control read-only. The form will send its value when submitted.

Further reading

- Styling form controls
  - Add style to text box, text area, and button
    - http://www.cssportal.com/form-elements/text-box.htm
  - Browsers differ in support of styles of form controls
    - http://www.456bereastreet.com/archive/200701/styling_form_controls_with_css_revisited/
  - The button element
    - http://speckyboy.com/2009/05/27/22-css-button-styling-tutorials-and-techniques/

Further reading

- CSS form design
  - http://wufoo.com/gallery/
  - Related articles in Smashing Magazine
    - http://www.smashingmagazine.com/tag/forms/
    - http://www.smashingmagazine.com/2006/11/11/css-based-forms-modern-solutions/
- Search 'UI kit'

D. CSS Cascade

Conflicting style declarations

- Several style rules that match an element may specify conflicting values for a property

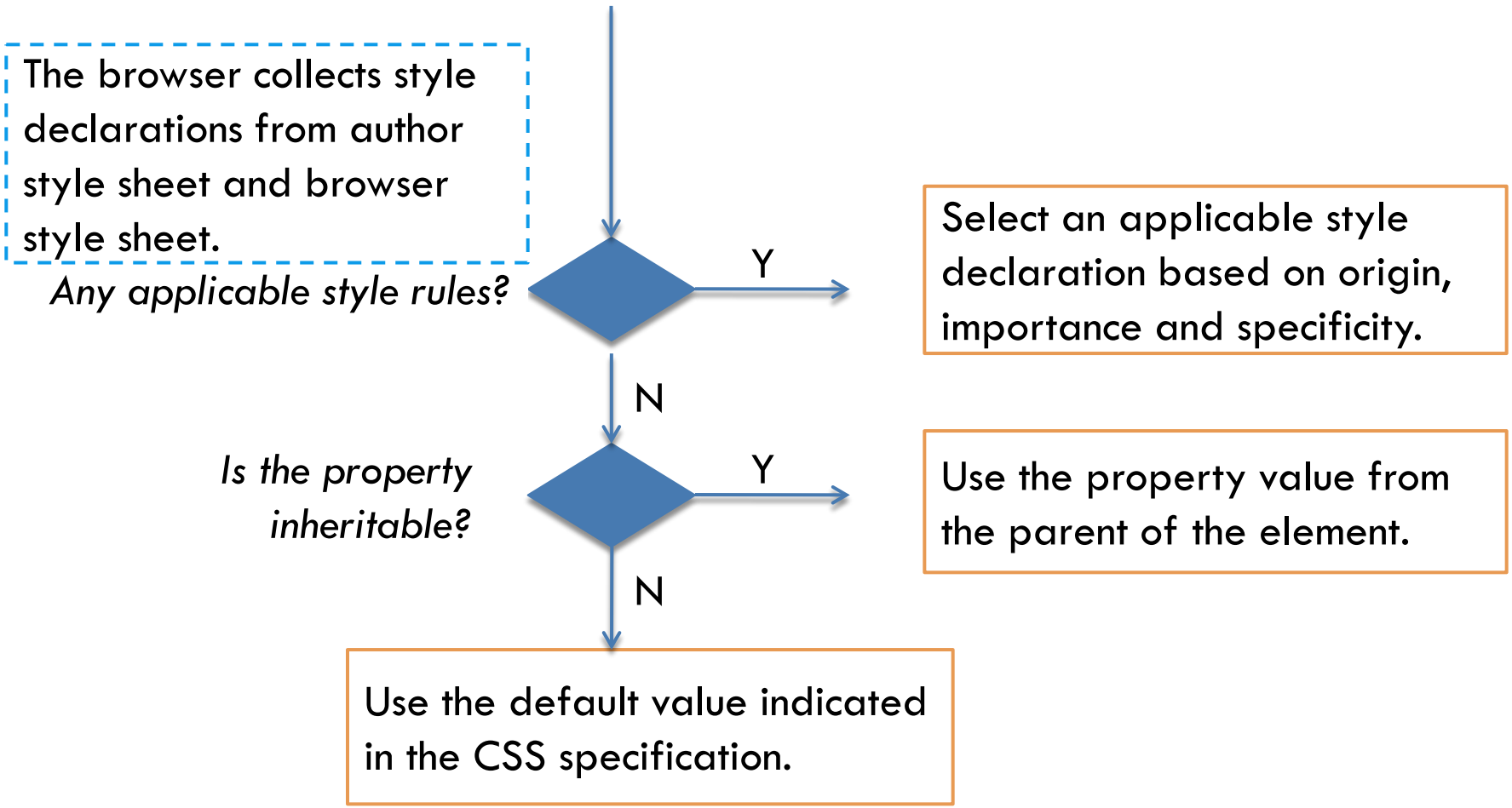
```
li { color: red; }
li li { color: green; }
li.hilte { color: blue; }

<ul>
    <li>one</li>
    <li>two:
        <ul><li>a</li><li>b</li>
            <li class="hilte">c</li></ul>
    </li>
    <li>three</li>
</ul>
```

The CSS Cascade

- The CSS cascade determines which style declaration wins in case of conflicting values for a property.
- Ref:
  - http://www.maxdesign.com.au/articles/css-cascade/
  - http://www.w3.org/TR/CSS2/cascade.html

CSS Cascade

To determine a property setting for an element …



Inheritance and default value

- In some cases, no style declaration for a property applies to an element
- If the property is inherited, the element obtains the property setting from its parent
  - E.g. most typographical properties
- If the property is *not* inherited, the element uses the default value
  - Ref: http://www.w3schools.com/CSS/css_reference.asp

Property	Default value
background-color	transparent
width	auto (stretch to fill container)
position	static (normal flow in chap 5)

Example

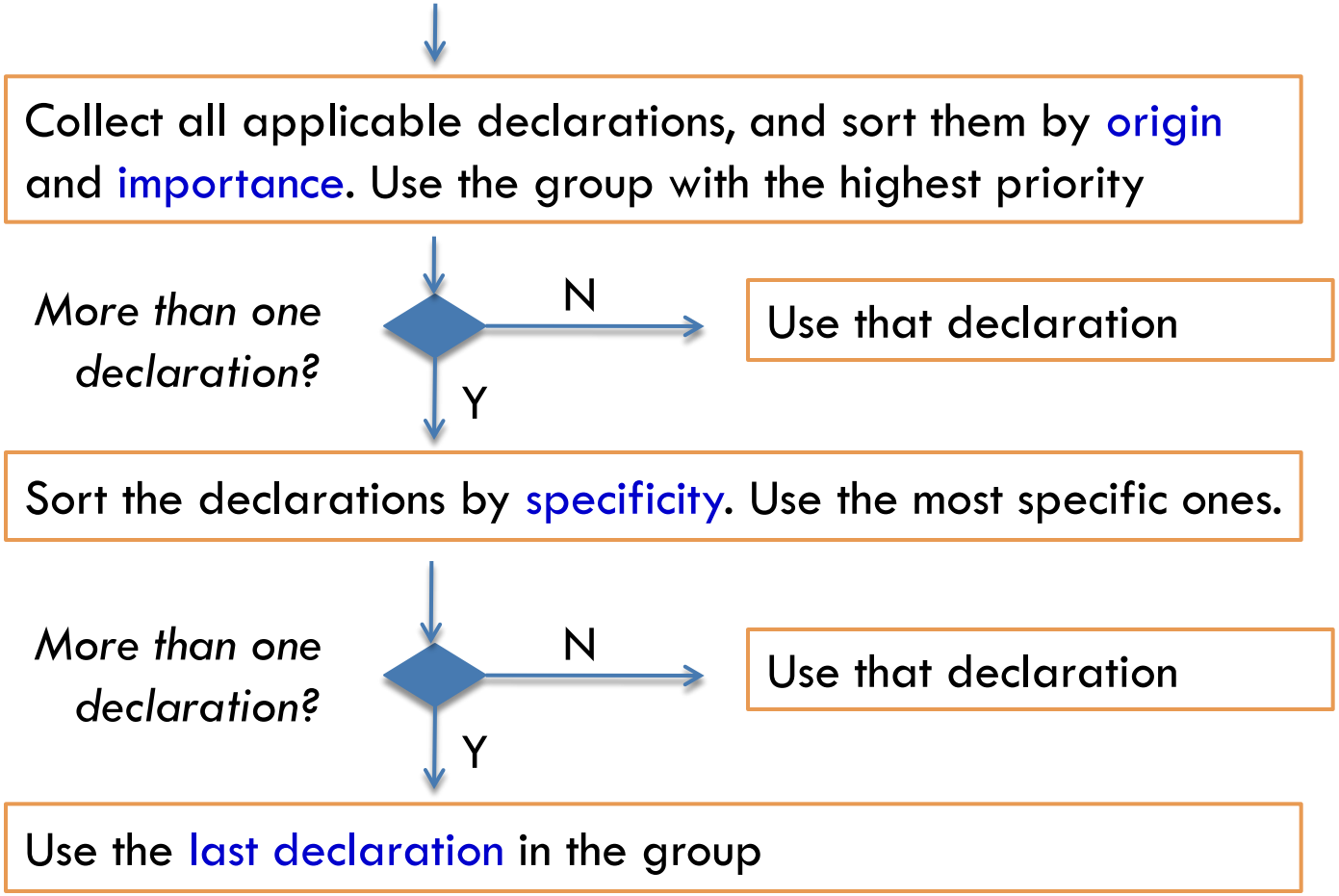

```
body { color: black }
em { color: blue; }
div { width: 400px; }
div p { margin: 50px 50px; }

<body>
  <div>
      <p>first paragraph</p>
      <p><em>second</em> paragraph</p>
  </div>
</body>
```

Both <p> inherit the black text color, while does not.

There is no style declaration for width of the two <p>. However, they do not inherit the width (400px) from their parent. Instead, they take the default value 'auto'.

Select an applicable declaration

To select an applicable style declaration…



Origins of style declaration

- A style declaration comes from 3 possible origins
  - Browser style sheet (user agent style sheet)
    - Each browser has a default style sheet
    - Default of browsers may be different. To ensure consistent presentation, you may use a reset style sheet
  - Author style sheet
    - Added by the author of the page
    - Inline, embedded, external, @import
  - User style sheet
    - E.g. Google Chrome allows a user to change the style of a web page

Author style sheet

- The author of an HTML doc can specify style rules in three ways:
  - Inline style
  - Embedded style / internal style
  - External style

Inline style

- style attribute inside the start tag of an element
- Style applies to one element only
- Higher priority than embedded and external style declarations
- Mix up presentation and content.
  - May use it for quick testing. But not recommended in production version

```
<h1 style="color: red">HTML essential</h1>
…
<h1>CSS essential</h1>
```

Embedded style

- keep all rules in a <style> element inside <head>
- Style applies to one HTML doc only

```
<head>…
  <style> h1 { color: red; } </style>
</head>
<body>
  <h1>HTML essential</h1>
  <p>some text</p>
  <h1>CSS essential</h1>
</body>
```

External style

- keep all rules in a separate file and reference it in a **<link>** element
- Several HTML files can link to the same style sheet. Consistent style
- An external style sheet can also import another external style sheet with @import

```
<head>…
  <link rel="stylesheet" href="web.css"/>
</head>
<body>
  <h1>HTML essential</h1>
  <p>some text</p>
  <h1>CSS essential</h1>
</body>
```

```
/* web.css */
h1 { color: red; }
```

Importance

- An important declaration ends with !important
- An important declaration wins over a normal declaration

```
<head>…
  <style>
      #title { color: blue; }
      h1 { color: red !important; }
  </style>
</head>
<body>
  <h1 id='title'>HTML essential</h1>
  <p>some text</p>
</body>
```

Origin and importance

- Style declarations that match an element may come from several sources, in decreasing order of priority
  - Author style sheet with !important
  - Author style sheet
  - Browser style sheet
- A property declaration with higher priority wins.
- If two property declarations have the same priority, consider the specificity

Example

- What style will be used for the four links? Explain.

```
/* user agent style sheet */
a { text-decoration: underline; }
```

```
<style>
   a.plain { text-decoration: none; }
   a.broken {text-decoration: line-through !important; }
</style>

 <a href='p1.htm'>link 1</a>
 <a href='p2.htm' class='plain'>link 2</a>
 <a href='p3.htm' class='broken'>link 3</a>
 <a href='p4.htm' class='plain broken'>link 4</a>
```

Specificity

- Specificity of a CSS selector is a list of three numbers a-b-c
  - a is the number of ID selectors
  - b is the total number of class selectors, attributes selectors, and pseudo-classes
  - c is the total number of type selectors and pseudo-elements
  - Ignore the universal selectors
  - Selectors in :not() is counted, but :not() itself is not counted as a pseudo-class

Example

Selectors	specificity
em	0-0-1
div p em	0-0-3
p.revise	0-1-1
p.revise:first-of-type	0-2-1
a.plain[href^='http://']	0-2-1
#main	1-0-0
p#main	1-0-1

Selectors	specificity
ul#toc > li	1-0-2
ul#toc > li:nth-child(odd)	1-1-2
p::first-line	0-0-2
div:not(.important)	0-1-1
img:not([alt])	0-1-1
div>*>em	0-0-2
#toc > :first-child	1-1-0

Comparing specificity

- When there are more than one applicable style declaration, use the ones with the highest specificity
  - Example: suppose selector 1 has specificity a1-b1-c1 and selector2 has a2-b2-c2
  - If a1>a2, selector 1 wins. If a2>a1, selector 2 wins.
  - If a1=a2, compare b1 and b2. If b1>b2, selector 1 wins. If b2>b1, selector b2 wins.
  - If a1=a2 and b1=b2, compare c1 and c2. The higher wins.

Example: Specificity

Selectors	Specificity
div#content p#p2 span.note	high
div.redbox p#p2 span.note	
div p#p2 span.note	
div p:nth-child(2) span.note	
div p span.note	
div span.note	
div p span	
div span	
span	low

Exercise: compare the specificity of the following:

p:nth-child(2) span
a[href$='gif'] span
div#content p span
p > span.note
p > * > span.note