INFX 1606

INTRODUCTION TO WEBSITE CREATION

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

HTML & CSS - 6

Raghav V. Sampangi

Instructor, Faculty of Computer Science, Dalhousie University

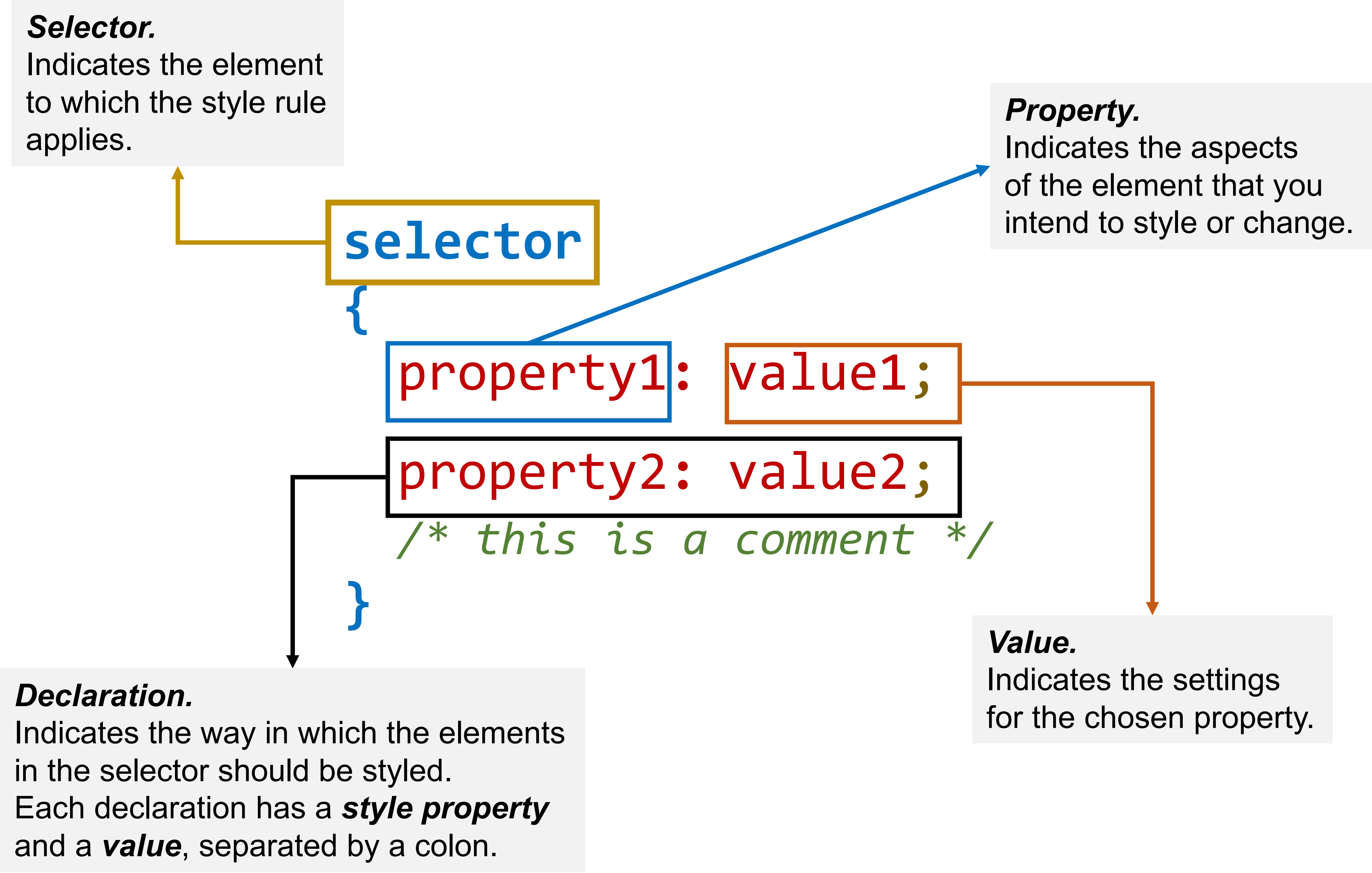
raghav@cs.dal.ca

Two things for today

- CSS Selectors and element relationships
- HTML Forms

CSS Selectors

Syntax of CSS Rules: Recap



Selector.
Indicates the element to which the style rule applies.

Property.
Indicates the aspects of the element that you intend to style or change.

```
selector
{
  property1: value1;
  property2: value2;
  /* this is a comment */
}
```

Value.
Indicates the settings for the chosen property.

Declaration.
Indicates the way in which the elements in the selector should be styled.
Each declaration has a ***style property*** and a ***value***, separated by a colon.

CSS Selectors: Recap

Selectors help identify the HTML element in the DOM to which the style rule applies.

```
selector
{
  property1: value1;
  property2: value2;
  /* A comment */
}
```

Types of selectors.

- ***Tag names***
  You can directly use the names of the HTML tags.
  E.g. **body**, **p**, **a**, **ol**, etc.

- ***Tag attributes***
  You can also use tag attributes to apply styles.
  Two attributes are commonly used – **class** and **id**.

CSS Selectors

Types of selectors.

- ***class***
  Identifies the style “class” to which an element belongs.
  Values to the HTML class attribute are ***space-separated***.

  E.g. `<div class=“c1 c2”></div>`

  Multiple HTML elements may have the same class value.

  In stylesheets, include a full-stop (or, period) just before the class name, if you use it as a selector.

  E.g.
```
<style>
    .c1 { font-style: italic; }
    .c2 { font-size: 125%; }
</style>
```

CSS Selectors

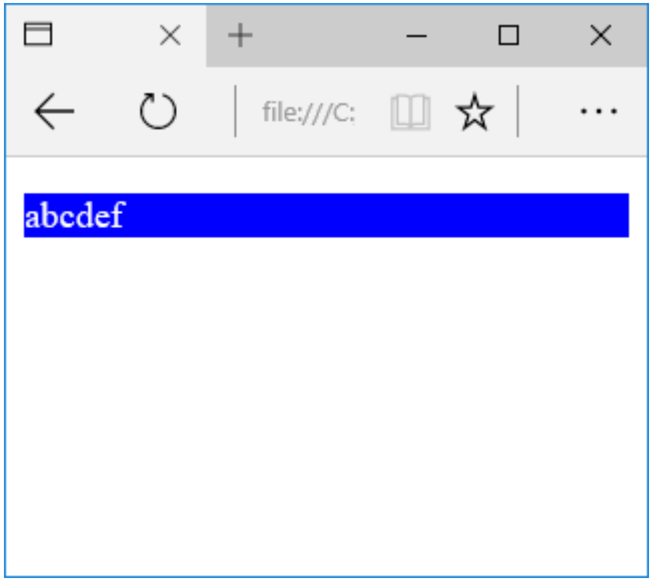
Types of selectors.

- ***class***
  Identifies the style "class" to which an element belongs.
  Values to the HTML class attribute are ***space-separated***.

  Styles are applied sequentially.
  (c1 first, c2 next and so on, in example below).

```
<style>
  .c1 { background-color: red; }
  .c2 { background-color: blue; color: white; }
</style>
```

```
<p class="c1 c2">abcdef</p>
```



abcdef

CSS Selectors (cont’d)

Types of selectors.

- ***id***

  You may also use the element’s ID to apply styles individually to an element. (Remember, IDs are unique to an element)

  E.g. `<div id=“myDiv”></div>`

  In stylesheets, include a hash sign (#) just before the ID, if you use it as a selector.

  E.g.
  ```
  <style>
      #myDiv { font-style: italic; }
  </style>
  ```

CSS Selectors (cont’d)

Types of selectors.

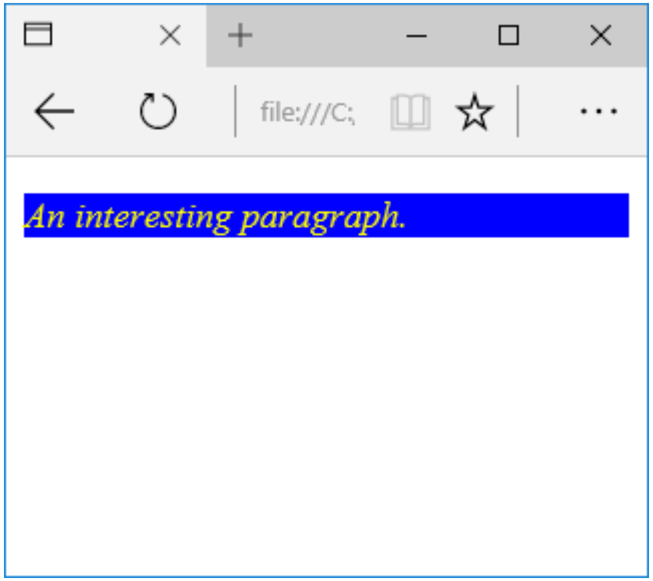
- ***id***

  You may also use the element’s ID to apply styles individually to an element. (Remember, IDs are unique to an element)

  If you choose to, you may also include style classes in addition to styling the element’s ID.

```
<style>
  #myParagraph { font-style: italic; color: yellow; }
  .c1 { background-color: red; }
  .c2 { background-color: blue; color: white; }
</style>
```

```
<p id=“myParagraph" class="c1 c2">An interesting
  paragraph.</p>
```



An interesting paragraph.

CSS Selectors (cont’d)

Types of selectors.

- ***Pseudo-classes***

This selector selects a ***special-state*** of the HTML element.
E.g. **hover** will apply the given style when the user “hovers” (or, moves the mouse on a given element)

In stylesheets, a colon will separate the element name and the pseudo-class.

Syntax:

```
selector:pseudo-class { property: value; }
```

No space between the selector and colon, and between the colon and the pseudo-class.

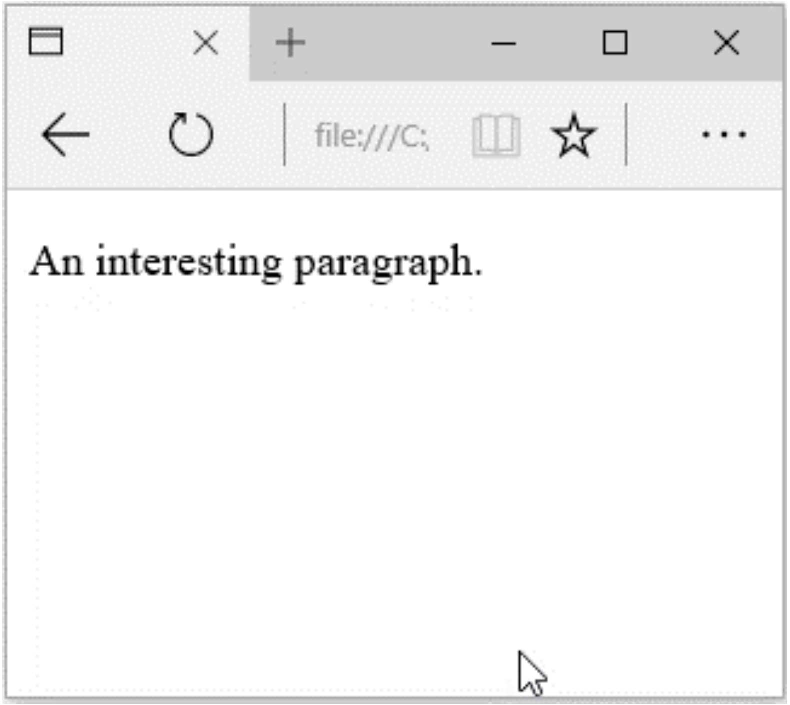
CSS Selectors (cont'd)

Types of selectors.

- ***Pseudo-classes***

E.g.

```
<style>
  p:hover { font-weight: bold; }
</style>
```



An interesting paragraph.

CSS Selectors (cont'd)

Types of selectors.

- ***Pseudo-classes***: List of pseudo-classes:

:link
:visited
:active
:hover
:focus
:first-child
:last-child
:nth-child
:nth-last-child
:nth-of-type
:first-of-type
:last-of-type
:empty
:target
:checked
:enabled
:disabled

More details: https://www.w3.org/TR/css3-selectors/

CSS Selectors (cont’d)

Types of selectors.

- ***Pseudo-classes***: Example – styling links:

```
<style>
  /* link – not visited by user */
  a:link { font-style: italic; color: yellow; }

 /* link – visited by user in the past */
  a:visited { color: red; }

 /* link’s behaviour when the mouse hovers over it */
  a:hover { font-weight: bold; color: black; }

 /* link’s behaviour when it is clicked, i.e. the moment it is active */
  a:active { color: red; }
</style>
```

```
<a href=“https://www.dal.ca">Link example</a>
```

For effective use, **hover** styling must be used after **link** and **visited** styling; and, **active** styling must be placed after **hover** styling.

CSS Selectors (cont’d)

Selectors based on element relationships (combinators).

You may also use selectors based on relationships between various elements in the HTML DOM.

Commonly used selectors based on relationships include:

`A E`	Any ***E*** element that is a descendent of ***A*** element (***Descendent Combinator* –** child, child or child, etc.)
`A > E`	Any ***E*** element that is a direct descendent of ***A*** element (***Child Combinator* –** direct children)
`E:first-child`	Any ***E*** element that is the first-child of its parent
`B + E`	Any ***E*** element that is the next sibling of a ***B*** element (***Adjacent Sibling Combinator***)
`B ~ E`	Any ***E*** element that is a sibling of a ***B*** element (***General Sibling Combinator***)

CSS Selectors (cont'd)

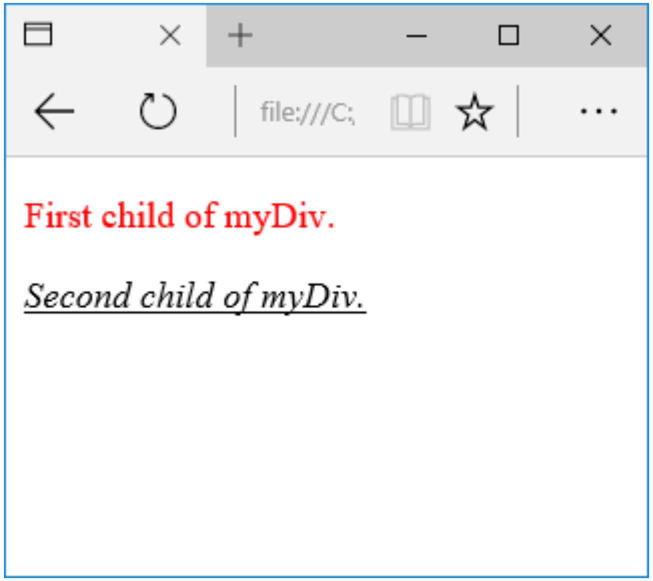
Selectors based on element relationships.

You may also use selectors based on relationships between various elements in the HTML DOM.

Example:

```
<style>
  div > p { color: red; }
  div div { text-decoration: underline; }
  #myDiv p:first-child { font-style: bold; }
  #myDiv p+div { font-style: italic; }
</style>
```

```
<div id="myDiv">
  <p>First child of myDiv.</p>
  <div>Second child of myDiv.</div>
</div>
```



First child of myDiv.

Second child of myDiv.

CSS Selectors (cont’d)

Selectors based on element relationships.

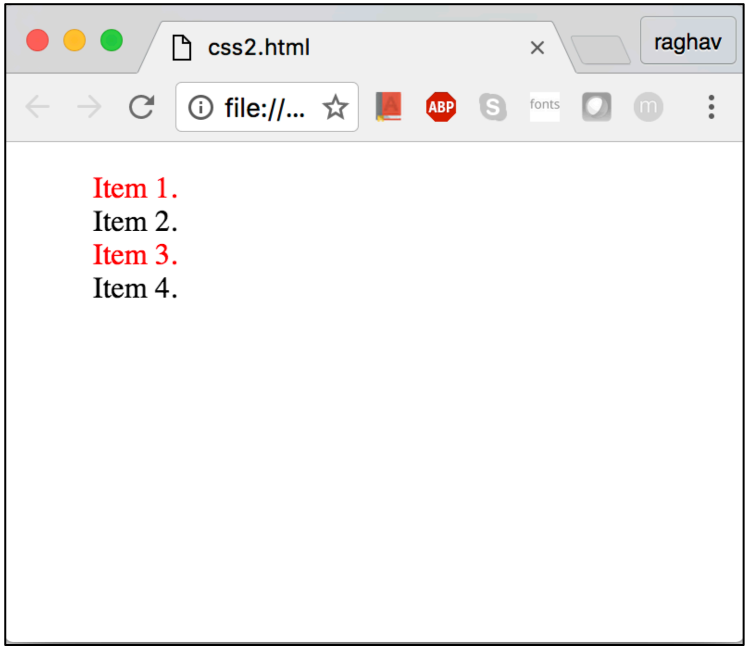
`:nth-child(an+b)`

Matches a number of child elements whose numeric position in the series of children matches the pattern an+b

i.e. if the selector was **li** and (an+b) = (2n+1), then, odd rows will be selected; if (an+b) = (2n), then, even rows will be selected.

```
<style>
  ul { list-style-type: none; }
  li:nth-child(2n+1) { color: red; }
</style>
```

```
<ul>
  <li>Item 1.</li>
  <li>Item 2.</li>
  <li>Item 3.</li>
  <li>Item 4.</li>
</ul>
```



```
Item 1.
Item 2.
Item 3.
Item 4.
```

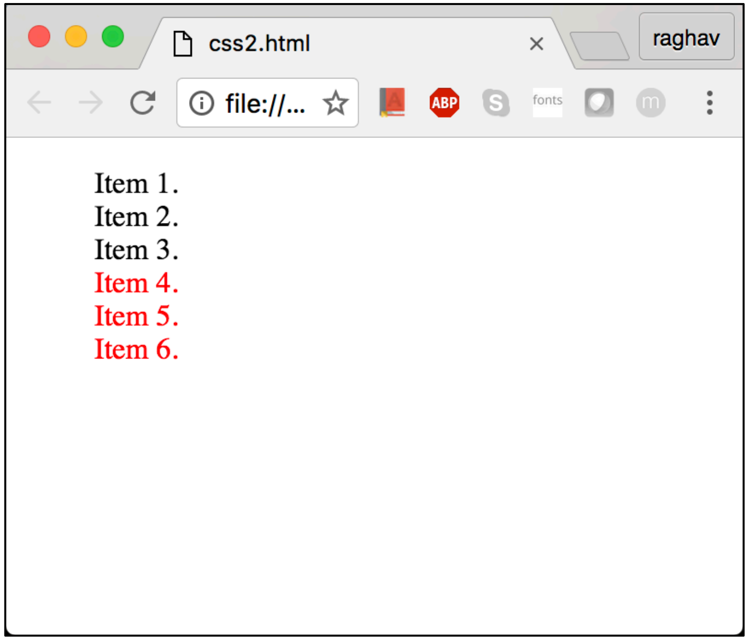
CSS Selectors (cont’d)

Selectors based on element relationships.

`:nth-last-child(an+b)`	Similar to :nth-child(an+b), but it selects elements of the specified type - counting backwards from the end of the series, not start. i.e. if the selector was **li** and (an+b) = (-(n-3)) = (-n+3), then, it selects the last three list items.

```
<style>
  ul { list-style-type: none; }
  li:nth-last-of-type(-n+3) { color: red; }
</style>
```

```
<ul>
  <li>Item 1.</li>
  <li>Item 2.</li>
  <li>Item 3.</li>
  <li>Item 4.</li>
  <li>Item 5.</li>
  <li>Item 6.</li>
</ul>
```



```
Item 1.
Item 2.
Item 3.
Item 4.
Item 5.
Item 6.
```

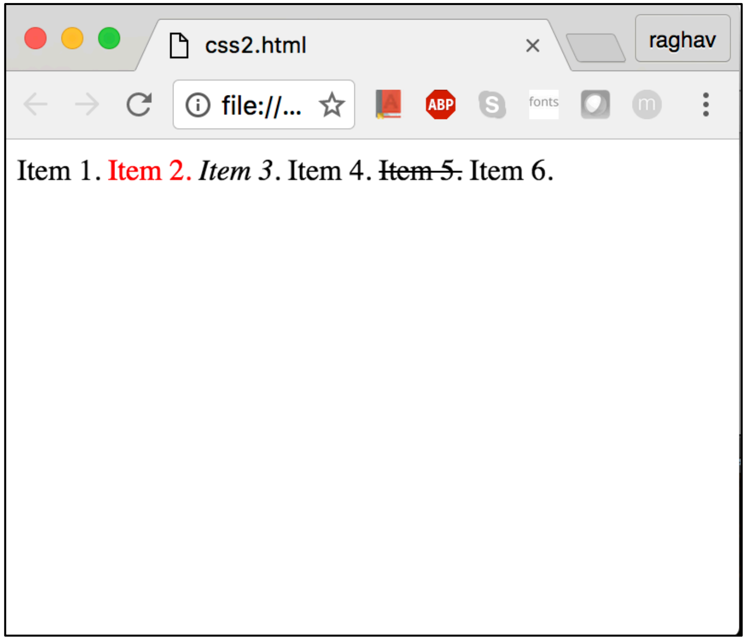
CSS Selectors (cont'd)

Selectors based on element relationships.

`:nth-last-of-type(an+b)`	Matches an element of the specified type that has (an+b-1) siblings of the same element type after it. i.e. if the selector was **span** and (an+b) = (3), then, it selects the span that has 2 more spans after it.

```
<style>
  span:nth-last-of-type(3) { color: red; }
</style>
```

```
<div>
  <span>Item 1.</span>
  <span>Item 2.</span>
  <em>Item 3.</em>
  <span>Item 4.</span>
  <strike>Item 5.</strike>
  <span>Item 6.</span>
</div>
```



Item 1. Item 2. *Item 3*. Item 4. ~~Item 5.~~ Item 6.

CSS Selectors (cont’d)

Selectors based on element relationships.

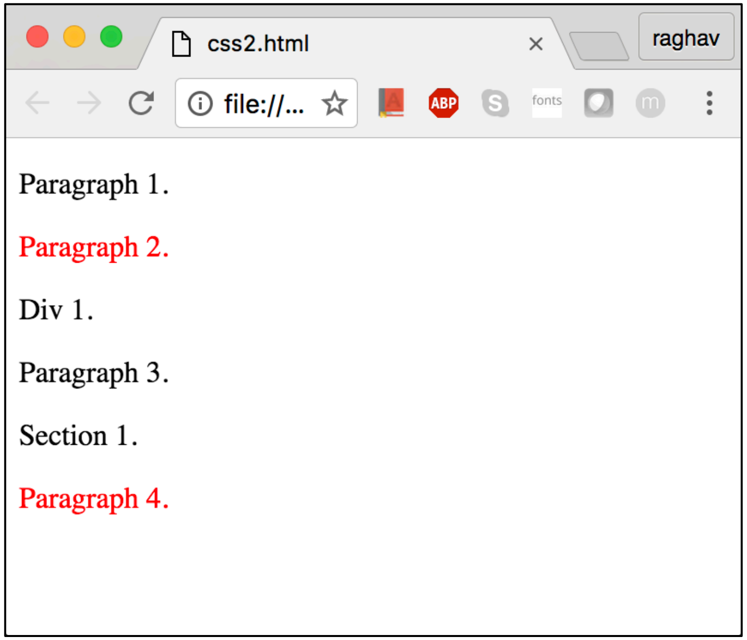
`:nth-of-type(an+b)`

A flexible pseudo-class. Matches an element that has (an+b-1) siblings of the same element type before it.

i.e. if the selector was **p** and (an+b) = (2n+1), then, it selects the last three list items.

```
<style>
  p:nth-of-type(2n+1) { color: red; }
</style>
```

```
<div>
  <p>Paragraph 1.</p>
  <p>Paragraph 2.</p>
  <div>Div 1.</div>
  <p>Paragraph 3.</p>
  <section>Section 1.</section>
  <p>Paragraph 4.</p>
</div>
```



```
Paragraph 1.
Paragraph 2.
Div 1.
Paragraph 3.
Section 1.
Paragraph 4.
```

CSS Selectors (cont'd)

Example: Navigation menu.

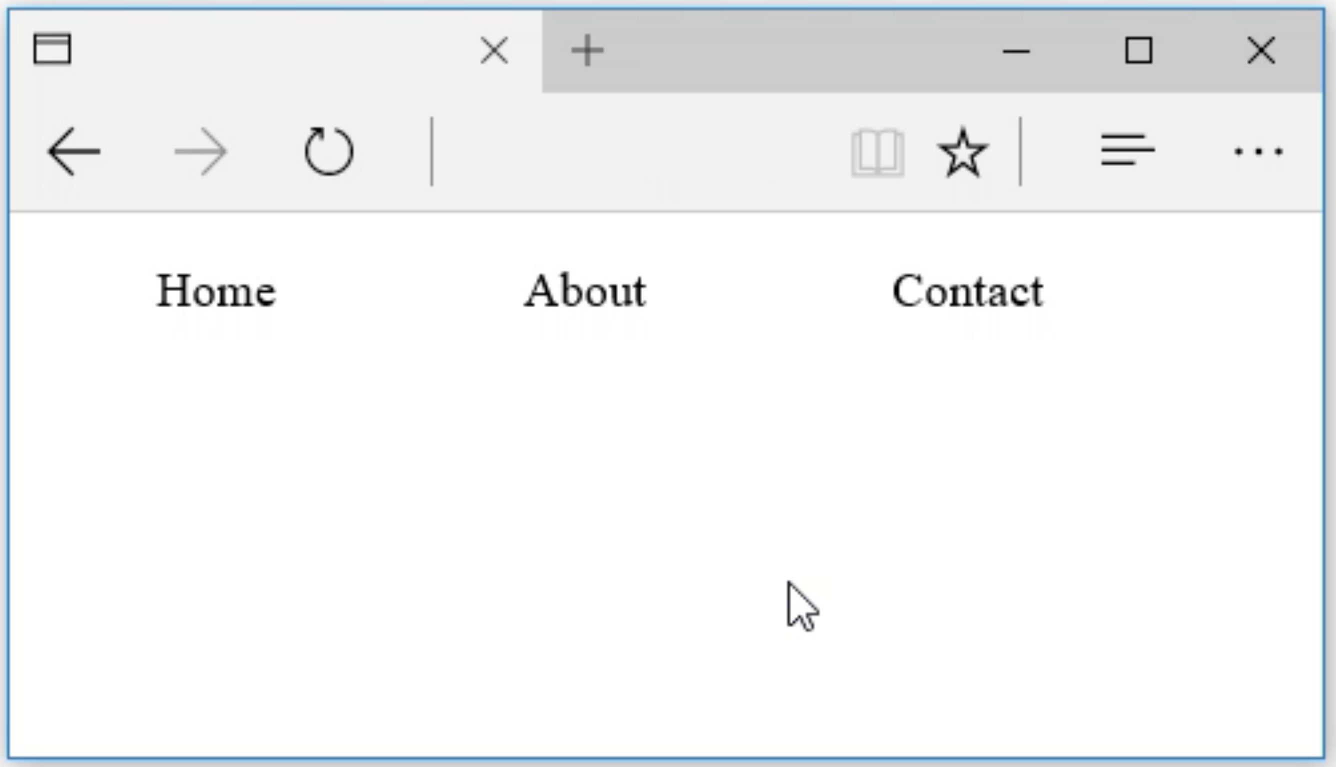
```html
<nav>
  <ul>
    <li>Home</li>
    <li>About
      <ul>
        <li>This website</li>
        <li>This foundation</li>
      </ul>
    </li>
    <li>Contact</li>
  </ul>
</nav>
```

```css
<style>
  /* horizontal - menu items */
  nav ul {
    list-style-type: none;
  }
  nav ul li {
    display: block;
    width:120px;
    float: left;
  }

  /* vertical - sub-menu items */
  nav ul ul {
    display: none;
    position: static;
  }
  nav ul li:hover > ul {
    display: block;
  }
  nav ul ul li {
    position: relative;
    display: block;
    height:30px;
  }
</style>
```

CSS Selectors (cont'd)

Example: Navigation menu.



Home About Contact

CSS Selectors (cont’d)

Selectors based on element relationships.

More examples available here:
https://developer.mozilla.org/en-US/docs/Web/CSS

Forms

Forms

What is a form?

A document that contains spaces where you can fill in information.

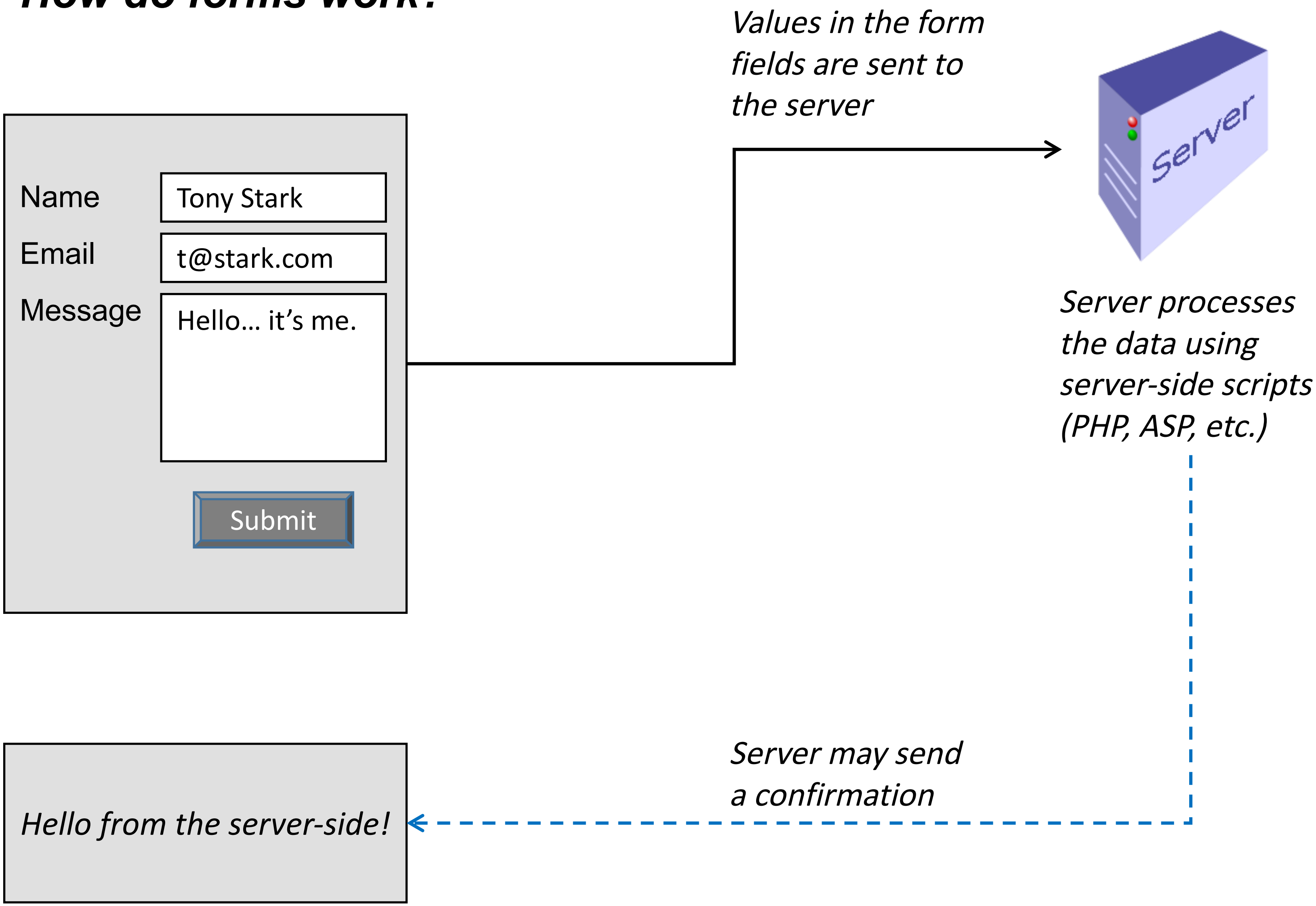
E.g. application forms, surveys, etc.

You typically complete a form and submit it to someone who is collecting the data, or the agency where you are applying for something, etc.

Key aspects of a form are: ***data input***, ***form submission***
and ***form data processing***

HTML Forms

How do forms work?



Name: Tony Stark

Email: t@stark.com

Message: Hello… it’s me.

Submit

Values in the form fields are sent to the server

Server processes the data using server-side scripts (PHP, ASP, etc.)

Server may send a confirmation

Hello from the server-side!

HTML Forms

HTML forms allow you to create various fields on the web page where users can fill in details.

Uses: account registration, payment, signing up for mailing lists, search

This is a concept where you apply your knowledge of other HTML elements – forms “contain” several other elements such as paragraph text, and sometimes even images.

HTML Forms

How do forms work?

The key elements of a form are → data input, form submission, and form data processing.

So, you will need a mechanism to collect data from the user and transfer them to the web server.

To collect information from users, you have to use ***form controls****.*

Form controls are fields or buttons or other elements that let users input data being collected by your website.

HTML Form Controls

The different types of form controls include:

Text → for single line text input

Text area → for multi-line or paragraph text input

Password → for user authentication forms (hides text)

Radio buttons → for choosing one from a set of options

Checkboxes → for choosing one or more options

Dropdown lists → for choosing one from a set of options (displayed as a list that “drops down” to expand and display all options

Submit buttons → for submitting data from the form to the server

Image buttons → allows you to use custom images as buttons

File upload → for uploading files to the server

HTML Form Controls

The different types of form controls include (**new in HTML5**):

Datalist → helps include pre-defined options for text input

Keygen → generates public/private key pair; used for security

output → to represent the result of a calculation

HTML Forms

So many options, but how do we go about creating forms?

Once again, begin with a ***wireframe*** – sketch the structure of your form. This helps you identify what elements your form should contain.

Next, define the **<form>** element, and all the other form control elements “live” inside this **<form>** element.

How do we submit the form data to the server?

For that, we will have to use PHP or other server side languages.

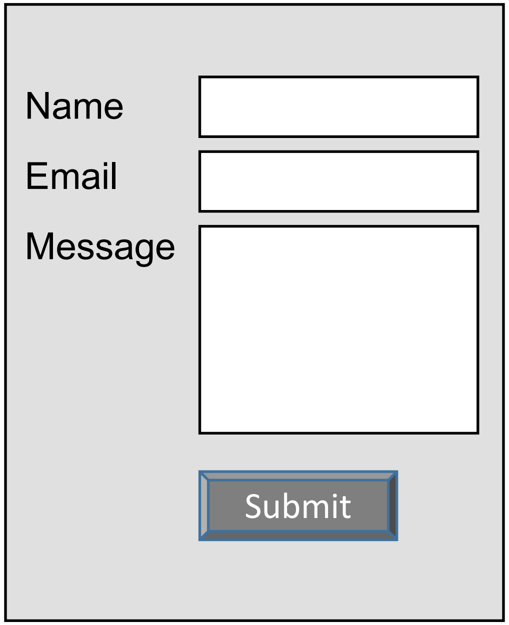
We may also use simple JavaScript functions to display alert messages when user submits a form (to be discussed later).

For our current discussion and examples, we will consider simple JavaScript functions.

HTML Forms

A simple HTML form:

Let’s consider that we have to create the following form…



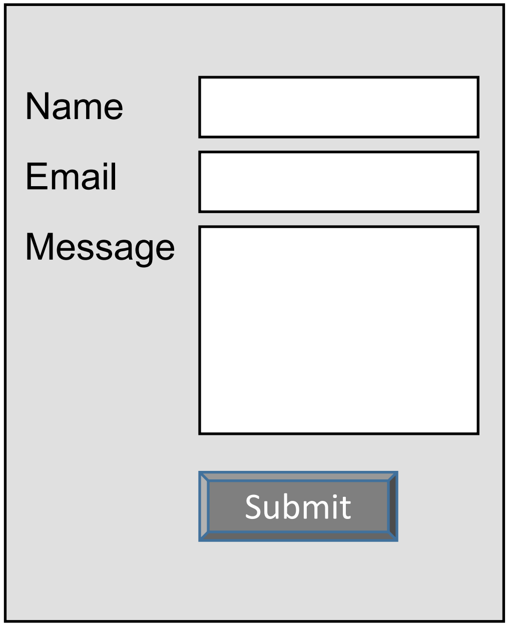
HTML Forms

A simple HTML form:

We begin by creating the form element…

```
<form action=“process.php” method=“post”>


</form>
```



HTML Forms

```
<form action=“process.php” method=“post”>

</form>
```

Main elements of this form element:

- ***action*** attribute → *required if you want your data to be processed*; its value will be the URL of the page on the server that will process the data sent by the form

- ***method*** attribute → good practice; default value is “get”, you can also use “post”

HTML Forms

Form: ***method*** attribute

- Defines the HTTP method when submitting forms

- If ***method="get"***, it means that the GET method is used (this is default)
  - GET method is useful if the query is ***passive***, i.e. if the input does not or is not expected to contain sensitive information (e.g. passwords)
  - When you use GET, the form data will be visible in the page URL after you submit the form, as shown below: http://www.mywebsite.com/process.php?name=Tony%20Stark

- If ***method="post"***, it means that the POST method is used
  - POST is useful if the query includes sensitive information (e.g. passwords)
  - POST is more secure, as the form data will not be available in the page URL after you submit the form

HTML Forms

Form: ***id*** attribute

- This is optional, but is often useful to include, as this helps in uniquely identifying a form among other page elements.
- Helps with styling and performing actions on the form as a whole.

```
<form id=“myForm” action=“process.php” method=“post”>

</form>
```

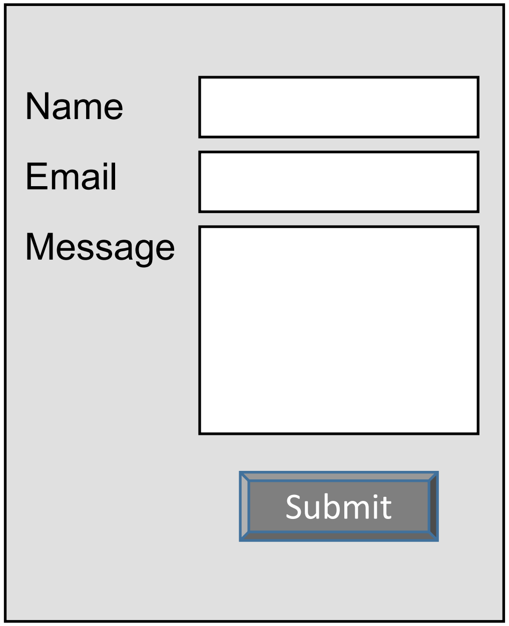
HTML Forms

A simple HTML form:

We begin by creating the form element…

```
<form action=“process.php” method=“post”>


</form>
```



Next, we add widgets (i.e. form controls) one by one.

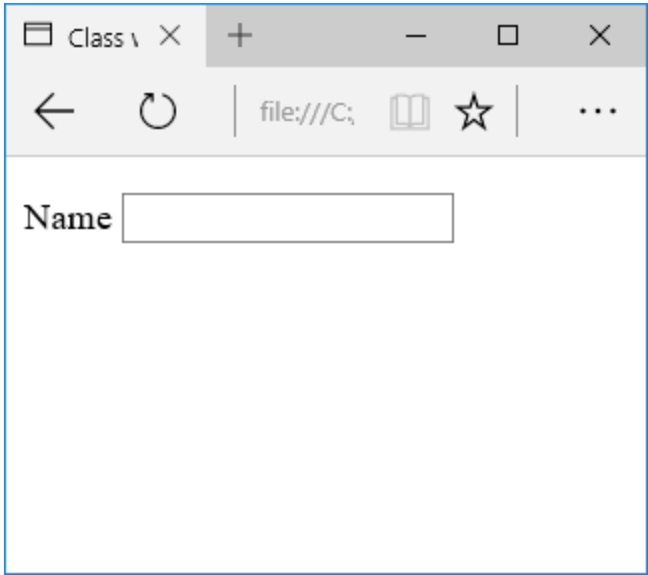
We need **two text boxes**, **one text area**, **one (submit) button**, and **three labels** (paragraph text).

We will begin by adding one text box, and one label.

HTML Forms

A simple HTML form – with one widget:

```
<form action=“process.php” method=“post”>
  <p>Name
    <input type=“text” name=“fullname” size=“16” />
  </p>
</form>
```



Name

HTML Forms

```
<form action=“process.php” method=“post”>
   <p>Name:
      <input type=“text” name=“fullname” size=“16” />
   </p>
</form>
```

Main elements of this form element (cont’d):

- ***paragraph*** element → optional; useful if you want to include text next to the input element

- ***Input*** element → optional; if you don’t use *input*, you may use other form controls in its place (e.g. textarea, button, radio button, etc.). If needed, <input> can be used as a valid child element of <p>.

HTML Form Controls

<input>

- *An important form element, when used.*
- There can be different types of input, as identified by the ***type*** attribute
  - ***type="text"*** → Single line text input
  - ***type="radio"*** → Radio button input (choosing one from a set of options)
  - ***type="submit"*** → Submit button, useful for submitting the form to the form-handler
  - ***type="password"*** → Defines a password field in the form
  - ***type="reset"*** → Helps create a button to reset (or, clear) form fields
  - ***type="hidden"*** → Defines a hidden field in the form (to transfer some reference data, e.g. order number, that the user does not input, but can be automatically computed)

And several others…

HTML Form Controls

<input>

- Other attributes (cont’d):
  - ***name*** → required; the value of this attribute tells the server as to from which form control a specific data value was received.

    For example, in our form, the input is set to text – if the name is set to “fullname” and the value entered by user is “Tony Stark”, then the value sent to the server will be in the form: ***fullname=Tony Stark***

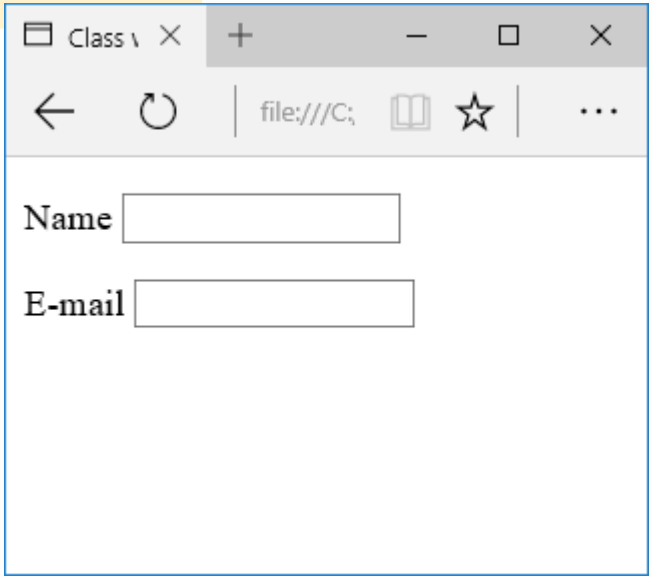
  - ***size*** → optional attribute; used to indicate the width of the text box, when type=“text” (default size is set to 20 characters).

  - ***maxlength*** → optional attribute; used to limit user input to ***maxlength*** number of characters, when type=“text”
    For example, if maxlength=“5”, the input will prevent the user from entering data after they have entered 5 characters.

HTML Forms

A simple HTML form – with two widgets (and with ***id***’s):

```
<form id="myForm" action="process.php" method="post">
  <p>Name
    <input type="text" id="fullname" name="fullname" size="16" />
  </p>
  <p>E-mail
    <input type="text" id="email" name="email" size="16" />
  </p>
</form>
```



Name

E-mail

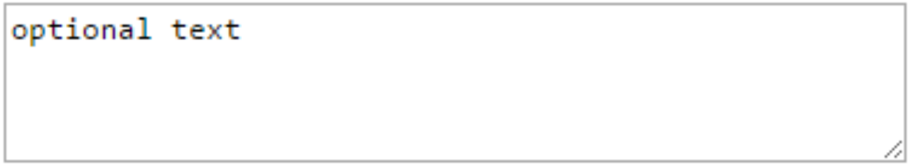
HTML Form Controls

<textarea>

- Defines multi-line or paragraph text input
- Dimensions of the text area can be defined using ***rows*** and ***cols*** attributes (to define the height and width in terms of rows and columns), or using the ***height*** and ***width*** *style definitions*
- Syntax:

```
<textarea id=“msg” name=“message”> optional text </textarea>
```

- Text area rendered for the above markup

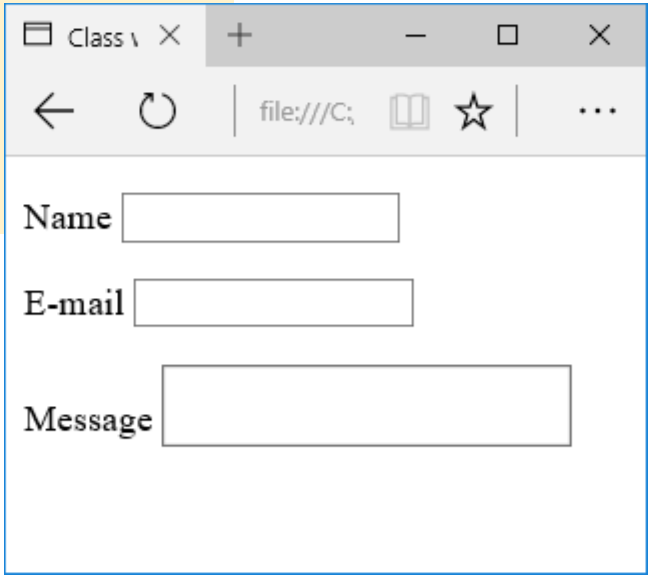


```
optional text
```

HTML Forms

A simple HTML form – with three widgets:

```
<form id="myForm" action="process.php" method="post">
  <p>Name
    <input type="text" id="fullname" name="fullname" size="16" />
  </p>
  <p>E-mail
    <input type="text" id="email" name="email" size="16" />
  </p>
  <p>Message
    <textarea id=“msg" name=“msg"></textarea>
  </p>
</form>
```



HTML Form Controls

<labels> instead of <p>

- You can also use <label> tag instead of <p> to include text labels for various form inputs
- Syntax:

```
<label for="id_name"> label text </label>
```

- The ***for*** attribute is used to connect or bind the label to the ***id*** of its corresponding input.
  For example, if we use a label for the full name text input, the markup will appear as follows:

```
<label for="fullname"> Full name: </label>
<input type="text" id="fullname" name="full_name" />
```

HTML Forms

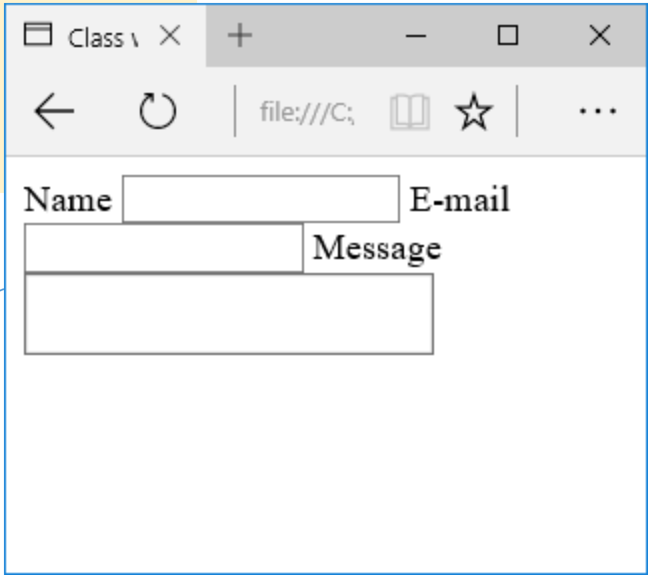
A simple HTML form – with labels instead of paragraphs:

```
<form id="myForm" action="process.php" method="post">
  <label for=“fullname">Name</label>
  <input type="text" id="fullname" name="fullname" size="16" />

  <label for ="email">E-mail</label>
  <input type="text" id="email" name="email" size="16" />

  <label for ="msg">Message</label>
  <textarea id=“msg" name=“msg"></textarea>
</form>
```

Whoops!!
What happened to the alignment???



HTML Form Controls

<labels> versus <p>

- Recall that paragraph elements are block elements, and have spacing to the adjacent elements.

- This was the reason why our use of <p> had aligned the elements correctly.

- If we use <label>, we have to create elements that will help with alignment and styling

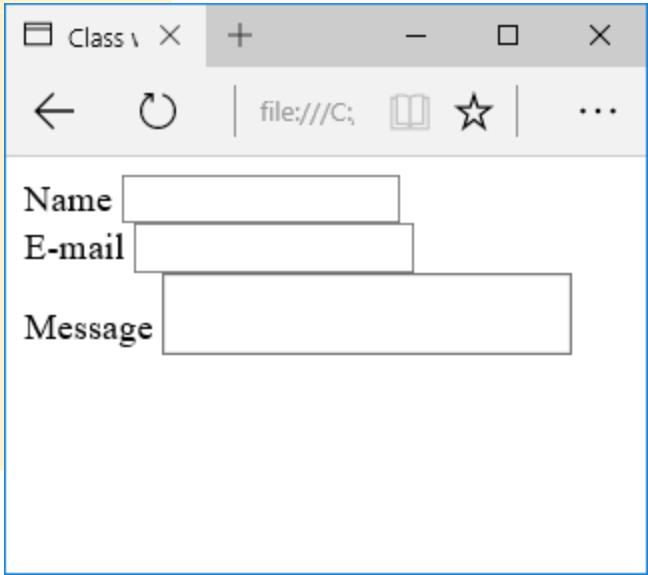
What element do we use to envelope such label+input combinations?

<div>
(grouping block elements)

HTML Forms

A simple HTML form – with labels instead of paragraphs:

```
<form id="myForm" action="process.php" method="post">
  <div>
    <label for=“fullname">Name</label>
    <input type="text" id="fullname" name="fullname" size="16" />
  </div>
  <div>
    <label for="email">E-mail</label>
    <input type="text" id="email" name="email" size="16" />
  </div>
  <div>
    <label for="msg">Message</label>
    <textarea id=“msg" name=“msg"></textarea>
  </div>
</form>
```



HTML Form Controls

<input type="submit"> versus <button>

- To create submit buttons, we can either use <input>, which allows a type called submit, or a <button> element

- If we use <input type="submit">, user clicking on the button makes the form send data to the web page defined by the "action" attribute in <form>

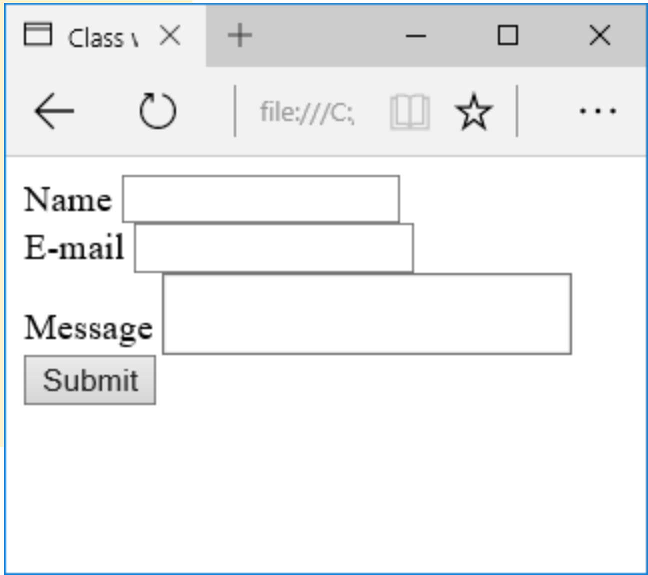
- If we use <button>, a user click does nothing by default.
  This is very useful, as it helps in creating custom actions using JavaScript.

- *Note: Such a process of receiving a user input and performing an action is called "event handling" → we will discuss more when we cover JavaScript.*

HTML Forms

A simple HTML form – with all requirements:

```
<form id="myForm" action="process.php" method="post">
  <div>
    <label for="fullname">Name</label>
    <input type="text" id="fullname" name="fullname" size="16" />
  </div>
  <div>
    <label for="email">E-mail</label>
    <input type="text" id="email" name="email" size="16" />
  </div>
  <div>
    <label for="msg">Message</label>
    <textarea id="msg" name="msg"></textarea>
  </div>
  <button>Submit</button>
</form>
```



HTML Form Controls

Try this later:

- *Create an HTML file, and write the markup from the previous slide.*
- *Include* **<input type=“reset” />** *after the <button></button> in our example.*

- *Open the HTML file in a browser.*

- *Enter some data, and click on “Reset”. See what happens…*

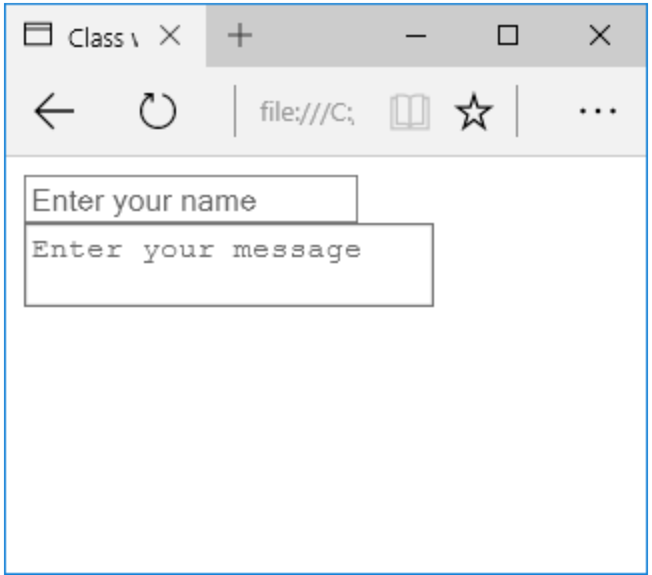
HTML Form Controls

Placeholder text

- You may also include placeholder text in text boxes and text areas

- For this, you would simply use the ***placeholder*** attribute

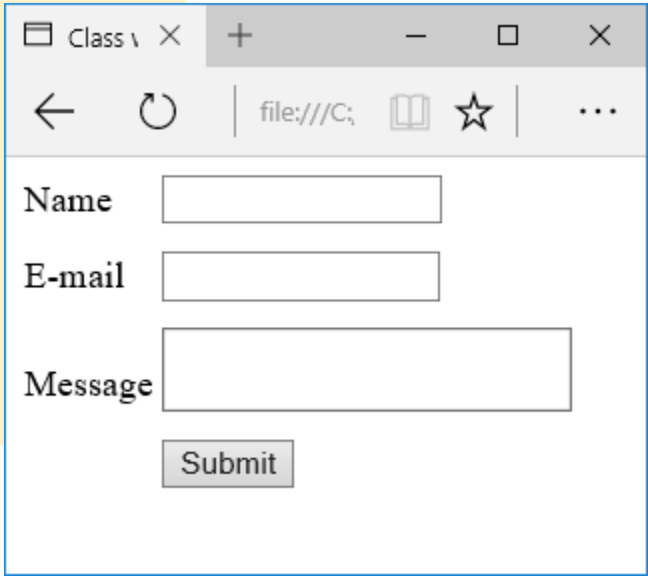
```
<input type="text" id="fullname" name="full_name"
placeholder="Enter your name" />
<textarea id="msg" name="msg" placeholder="Enter your message"></textarea>
```



HTML Forms

Okay, back to our form… final touches…

```
<form id="myForm" action="process.php" method="post">
  <div style=“padding-bottom:12px;">
    <label for=“fullname" style=“padding-right:18px;">Name</label>
    <input type="text" id="fullname" name="fullname" size="16" />
  </div>
  <div style=“padding-bottom:12px;">
    <label for="email" style=“padding-right:12px;">E-mail</label>
    <input type="text" id="email" name="email" size="16" />
  </div>
  <div style=“padding-bottom:12px;">
    <label for="msg">Message</label>
    <textarea id=“msg" name=“msg"></textarea>
  </div>
  <button style=“margin-left:60px;">Submit</button>
</form>
```



HTML Form Controls

Can we display a message when submitting a form?

- Sure! Let’s try a bit of JavaScript

- Include JavaScript in the page head, within **<script></script>** tags

- To show a message, we can use JavaScript’s ***alert*** function
  Note: a function in JavaScript is a sequence of steps that will be performed when that function is “invoked” or “called”

- How can a function be called or invoked?
  By user actions → for example, button click.

HTML Form Controls

Can we display a message when submitting a form?

- A function in JavaScript is defined as follows:

```
function function_name ( input_parameters )
{
    processing_step_1;
    processing_step_2;
    // and so on – anything after two forward slashes is a comment
}
```

- Such functions are shortcuts → instead of copying and pasting the same processing steps in different parts of the code, you will simply call or invoke the function.

- We will discuss this in detail in a later class.

HTML Form Controls

Three changes in our markup

1. We will remove the ***action*** attribute in the <form> *(for now – because we are not thinking of processing form data yet)*

```
<form id="myForm" ~~action="process.php"~~ method="post">
  <!-- actual form elements, i.e. div, labels, input 
and buttons -->
</form>
```

2. We will include a JavaScript function called ***alertMe()*** in the <script> element

```
<head>
   <title>My form</title>
   <script>
      function alertMe()
      {
         alert (“Message sent!”);
      }
   </script>
</head>
```

HTML Form Controls

Three changes in our markup

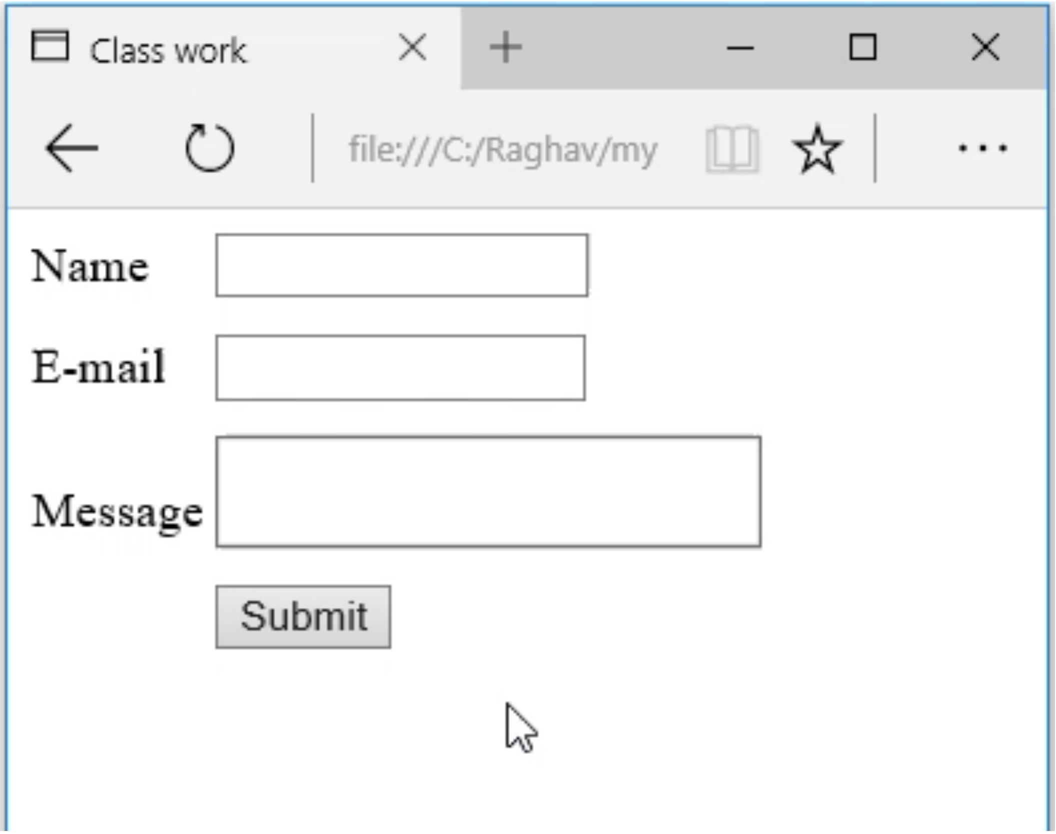
3. Include a way to recognize the button click and invoke or call the ***alertMe*** function.
   i.e. we are now setting up the event handler.
   To do this, include the ***onClick*** attribute in <button> as follows:
   (we will discuss this in a lot more detail when covering JavaScript)

```
<button style=“margin-left:60px;" onClick=“alertMe();">Submit</button>
```

Basically, you will include the function name followed by parentheses to create an event handler for that button.

HTML Forms

Okay, back to our form… working of the final form (with alert)



HTML Form Example: Full Markup (page head)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Class work</title>
    <script>
      function alertMe() {
        alert ("form submitted!");
      }
    </script>
  </head>
```

HTML Form Example: Full Markup (page body)

```
<body>
  <form id="myForm" method="post">
    <div style="padding-bottom:12px;">
      <label for="fullname" style="padding-right:18px;">Name</label>
      <input type="text" id="fullname" name="fullname" size="16" />
    </div>
    <div style="padding-bottom:12px;">
      <label for="email" style="padding-right:12px;">E-mail</label>
      <input type="text" id="email" name="email" size="16" />
    </div>
    <div style="padding-bottom:12px;">
      <label for="msg">Message</label>
      <textarea id="msg" name="msg"></textarea>
    </div>
    <button style="margin-left:60px;" onclick="alertMe();">Submit</button>
  </form>
</body>
</html>
```

HTML Forms: Advanced Concepts

Grouping form elements

We can use the **<fieldset></fieldset>** element to group certain form elements.

For example, if we wanted to group Name and E-mail fields in our form example, we can include those divisions inside <fieldset>.

To create a group heading for this new group, we have to use the **<legend></legend**> element, with the heading text between these tags.

E.g.

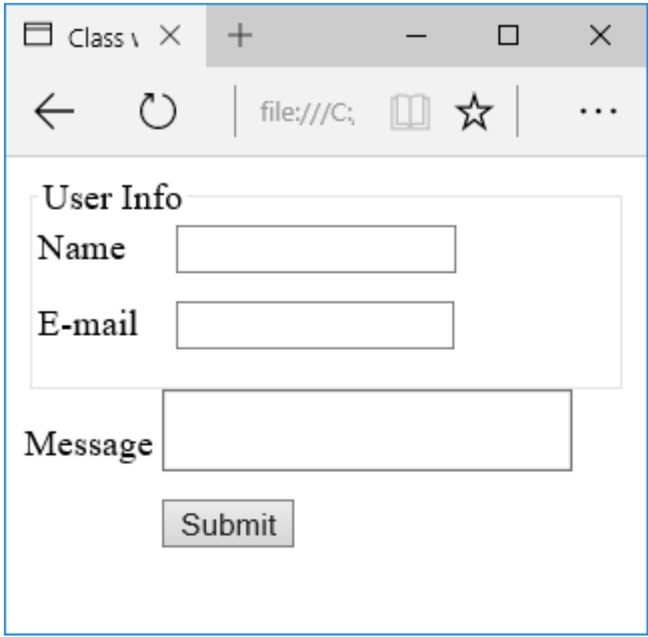
```
<fieldset>
   <legend>My Form Group</legend>
   <!-- form elements that need to be grouped -->

</fieldset>
```

HTML Forms: Advanced Concepts

Grouping form elements (*in our example*)

```
<form id="myForm" method="post">
  <fieldset>
    <legend>User Info</legend>
    <div style="padding-bottom:12px;">
      <label for="fullname" style="padding-right:18px;">Name</label>
      <input type="text" id="fullname" name="fullname" size="16" />
    </div>
    <div style="padding-bottom:12px;">
      <label for="email" style="padding-right:12px;">
       E-mail</label>
      <input type="text" id="email“
       name="email" size="16" />
    </div>
  </fieldset>
  <!-- Markup for message textarea and button -->
</form>
```



HTML Forms: Advanced Concepts

Form Validation

Traditionally, validating forms or the process of checking if the user input was correct – was accomplished using JavaScript.

HTML5 allows form validation and makes it the task of the browser.

A simple example is to make form fields mandatory, by simply including the ***required*** attribute for the input.

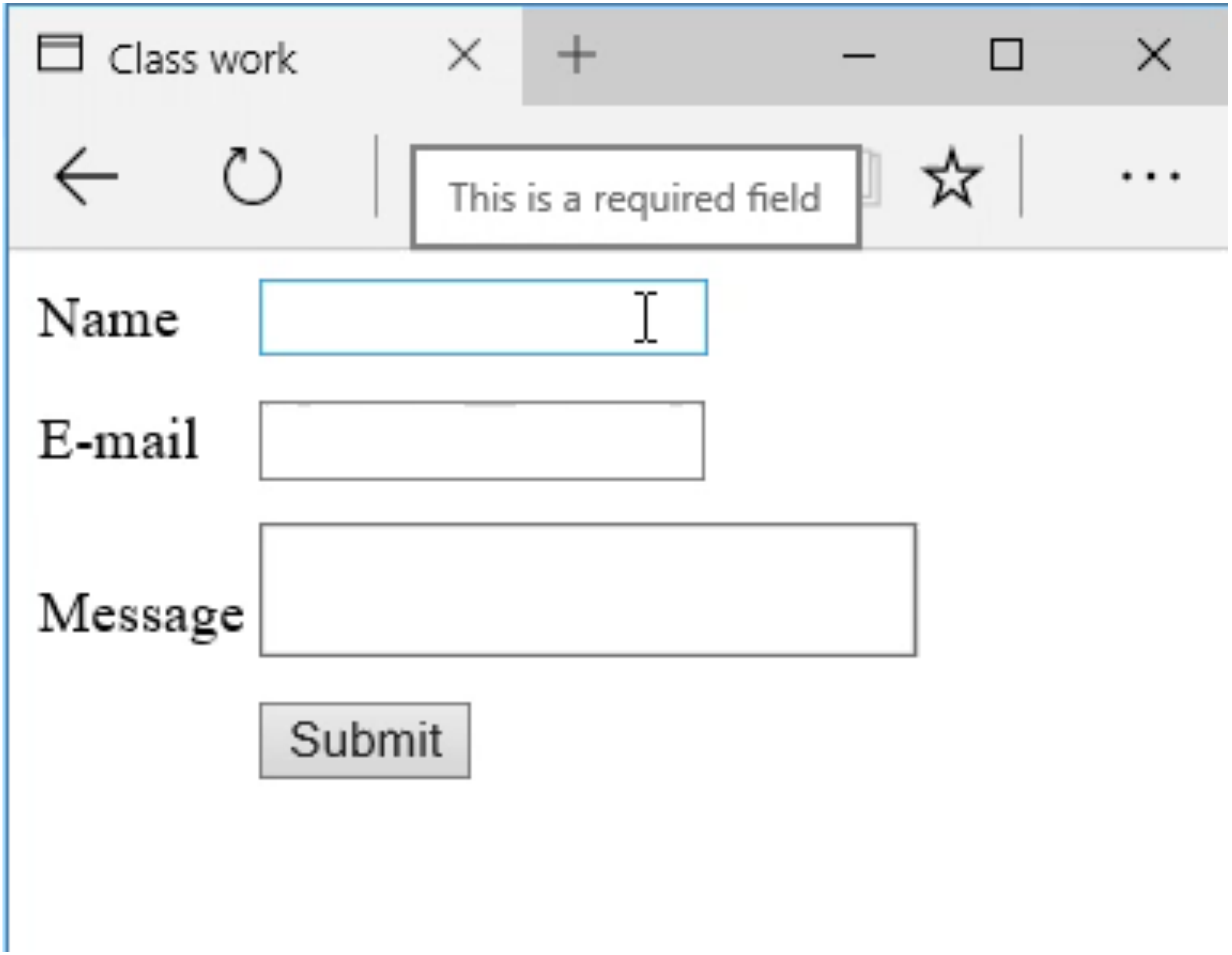
E.g. (in our form)

```html
<label for="fullname" style="padding-right:18px;">Name</label>

<input type="text" id="fullname" name="fullname" size="16" required />
```

HTML Forms: Advanced Concepts

Form Validation



This is a required field

HTML Forms: Advanced Concepts

E-mail address and URL validation

HTML5 also allows validating that email addresses have a specific format (e.g. abc@def.com), or URLs have a specific format (e.g. http://www.dal.ca)

E.g. (in our form)

```
<label for=“email" style="padding-right:18px;">E-mail</label>

<input type=“email" id=“email" name=“email" size="16" required />
```

For URLs:

```
<label for=“url">URL</label>

<input type=“url" id=“url" name=“webaddress" size="16" required />
```

You may also use “regular expressions” or JavaScript to check whether the input values have the expected pattern.

Key Ideas

The key elements of a form are → data input, form submission, and form data processing.

To collect information from users, you have to use ***form controls****.*

Form controls are fields or buttons or other elements that let users input data being collected by your website.

All form elements live inside the <form> element

You can use different types of <input>s, by setting the ***type*** attribute, or by using specialized tags, such as <textarea>