

Introduction to Web Technology

HTML

Joseph Tonien

School of Computing and Information Technology
University of Wollongong

HTML

Objectives:

- learn the HTML language to create web site;
- understand the concept of HTML tags and attributes;
- use HTML tags to format text, add graphics, create links, display tables, lists, etc.

HTML

- is a markup language for describing web documents
 - used to mark parts of documents to indicate how they should appear on a display
- HTML stands for **Hyper Text Markup Language**
- HTML documents are described by **HTML tags**
- Each HTML tag describes different document content

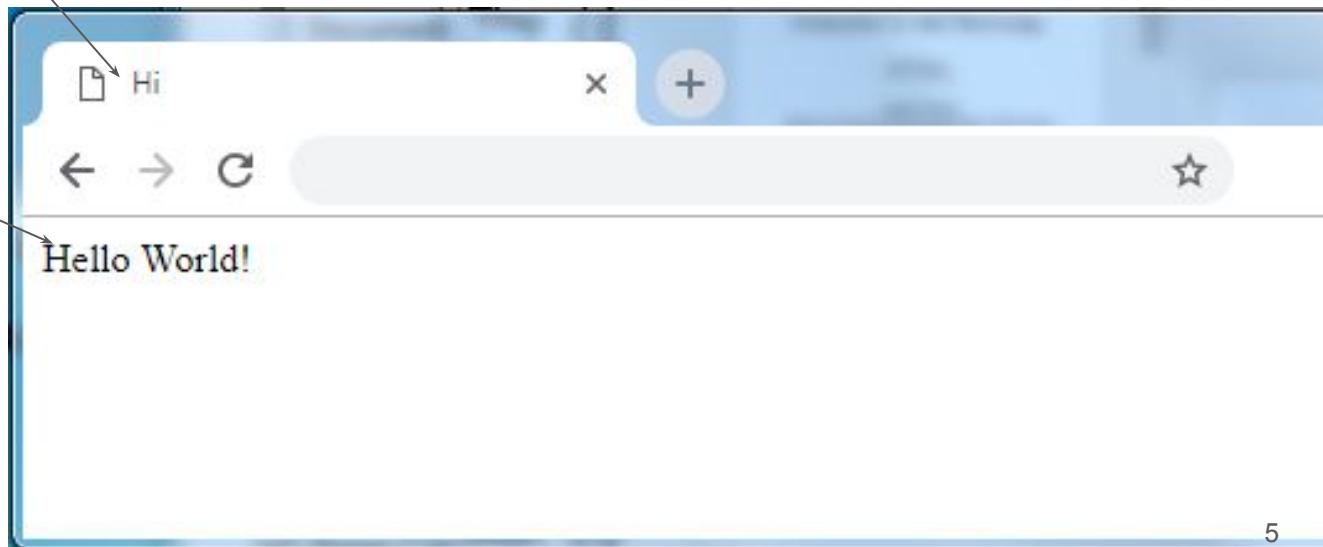
The first HTML document

```
<html>
  <head>
    <title>Hi</title>
  </head>

  <body>
    Hello World!
  </body>
</html>
```

The first HTML document

```
<html>  
  <head>  
    <title>Hi</title>  
  </head>  
  <body>  
    Hello World!  
  </body>  
</html>
```

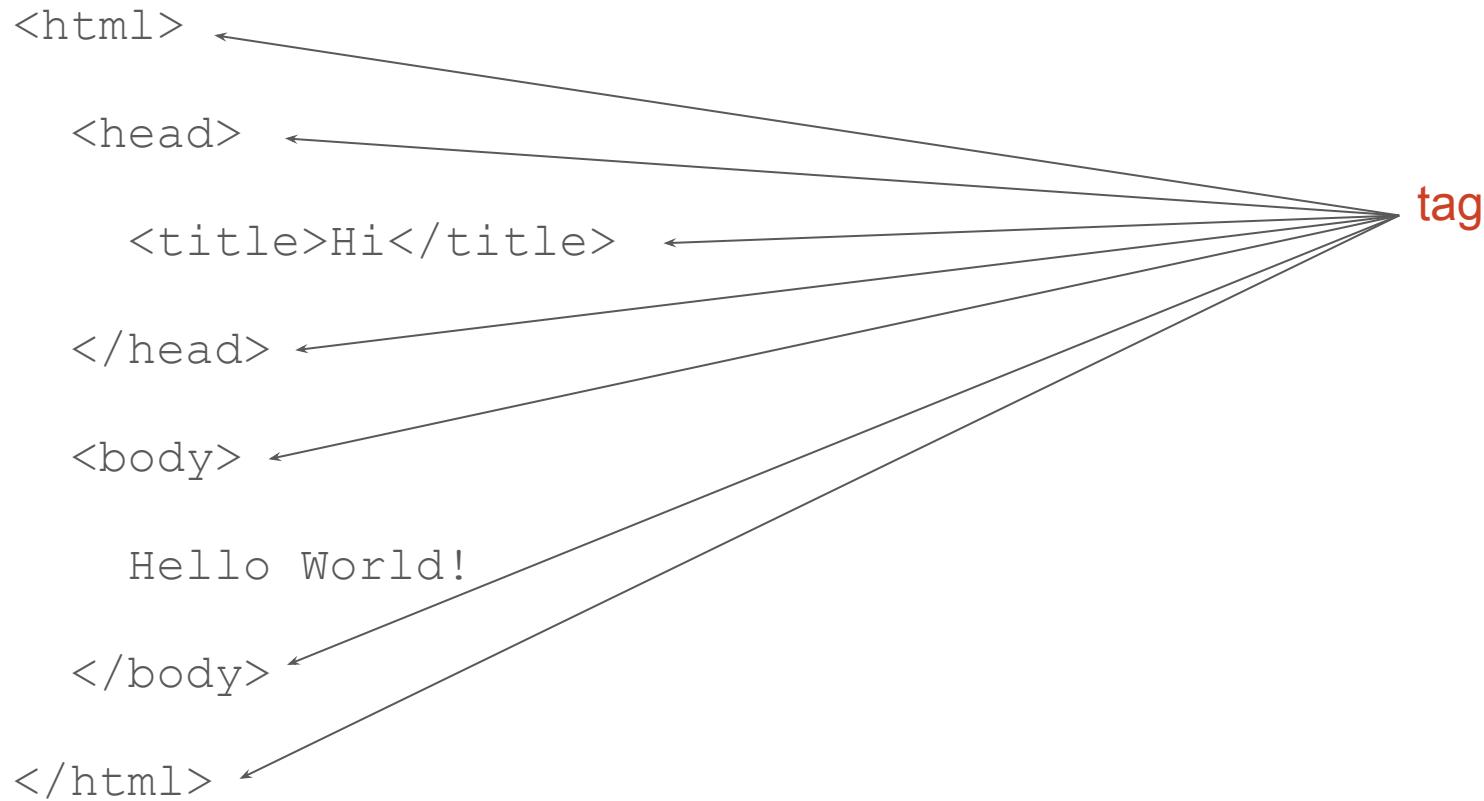


HTML document structure

```
<html>  
  
  <head>  
  
    <title>Hi</title>  
  
  </head>  
  
  <body>  
  
    Hello World!  
  
  </body>  
  
</html>
```

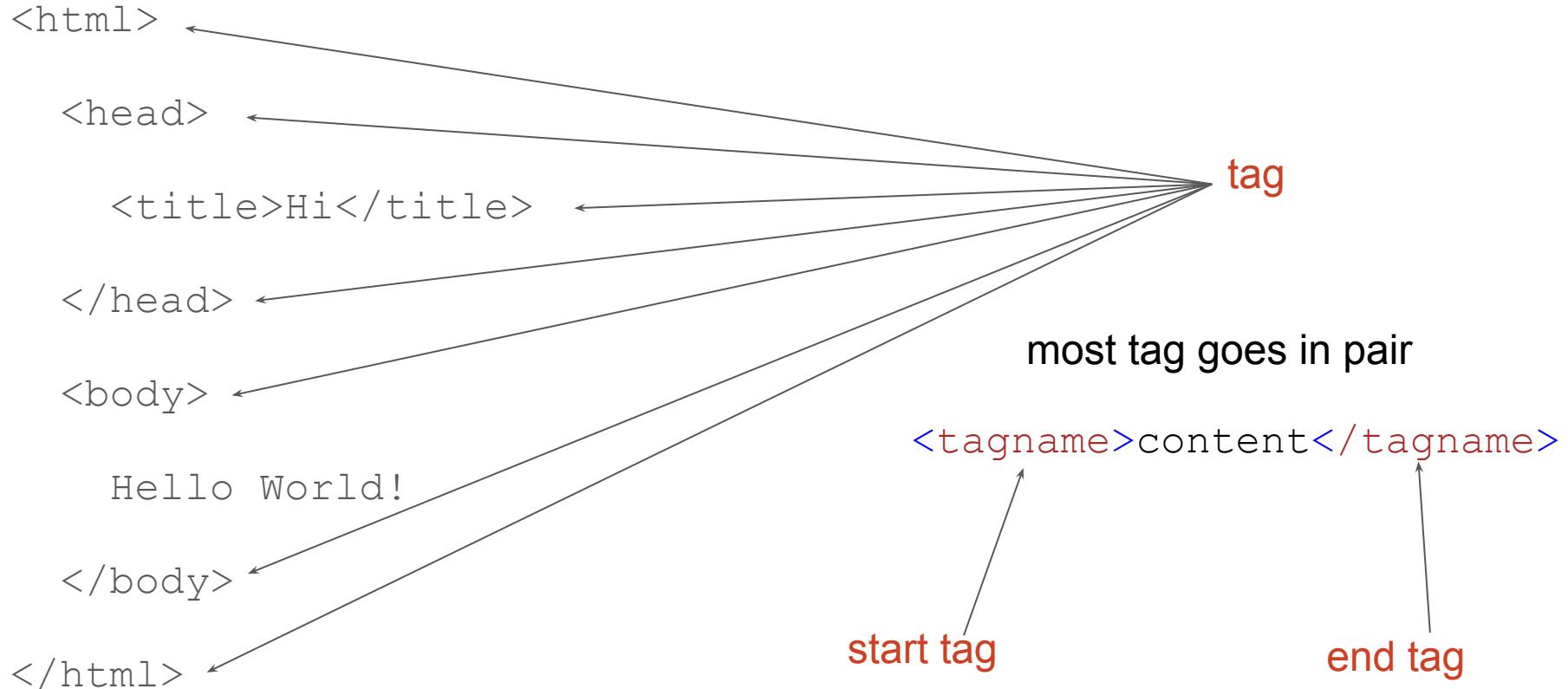
- A HTML document starts with **<html>** and ends with **</html>**
- A HTML document has a **head** and a **body**
 - The visible part of the HTML document is between **<body>** and **</body>**

HTML tags



HTML tags

tags are **NOT** case sensitive



HTML tags

```
<html>
```

```
  <head>
```

```
    <title>Hi</title>
```

```
  </head>
```

```
<body>
```

```
  Hello World!
```

```
</body>
```

```
</html>
```

HTML **documents** are made up by HTML **elements**.

The HTML **element** is everything *from the start tag to the end tag*.

Where is the HTML element?

Where is the HEAD element?

Where is the BODY element?

Where is the TITLE element?

What elements does the HTML element contain?

Self-closing tag

```
<tagname></tagname>
```

- Sometime we have an element with no content (**an empty element**)
- With an empty element, we can combine the start-tag and end-tag into one **self-closing tag**: `<tagname />`
- Example:
 - `
` tag defines a **line break**
 - `<hr />` tag defines a **horizontal line**
 - `` tag defines an **image**

Tag attributes

```
<tagname attribute1="value1" attribute2="value2">  
  ...  
</tagname>
```

- HTML elements can have **attributes**
- Attributes provide **additional information** about an element
- Attributes are always specified in **the start tag**
- Attributes come in name/value pairs like: **name="value"**

Heading tags

```
<body>  
  
<h1>Heading 1</h1>  
  
<h2>Heading 2</h2>  
  
<h3>Heading 3</h3>  
  
<h6>Heading 6</h6>  
  
Normal text...  
  
</body>
```

Heading tags: **<h1>, <h2>, . . . , <h6>**

<h1> the most important heading

<h6> the least important heading

Unordered List

My timetable:

<**ul**>

<**li**>MATH222: Mon 8:30-10:30 lecture</**li**>

CSCI204: Tue 9:30-11:30 lab

ISIT206: Wed 8:30-10:30 lecture

</**ul**>

My timetable:

- MATH222: Mon 8:30-10:30 lecture
- CSCI204: Tue 9:30-11:30 lab
- ISIT206: Wed 8:30-10:30 lecture

Unordered List

Unordered List

My timetable:

<**ul**>

<**li**>MATH222: Mon 8:30-10:30 lecture</**li**>

<**li**>CSCI204: Tue 9:30-11:30 lab</**li**>

<**li**>ISIT206: Wed 8:30-10:30 lecture</**li**>

</**ul**>

List Item

My timetable:

- MATH222: Mon 8:30-10:30 lecture
- CSCI204: Tue 9:30-11:30 lab
- ISIT206: Wed 8:30-10:30 lecture

Ordered List

My timetable:

<**o1**>

<**li**>MATH222: Mon 8:30-10:30 lecture</**li**>

CSCI204: Tue 9:30-11:30 lab

ISIT206: Wed 8:30-10:30 lecture

</**o1**>

My timetable:

1. MATH222: Mon 8:30-10:30 lecture
2. CSCI204: Tue 9:30-11:30 lab
3. ISIT206: Wed 8:30-10:30 lecture

Ordered List

My timetable:

Ordered List

List Item

- MATH222: Mon 8:30-10:30 lecture
- CSCI204: Tue 9:30-11:30 lab
- ISIT206: Wed 8:30-10:30 lecture

My timetable:

1. MATH222: Mon 8:30-10:30 lecture
2. CSCI204: Tue 9:30-11:30 lab
3. ISIT206: Wed 8:30-10:30 lecture

Description List

My timetable:

<**dl**>

<**dt**>MATH222</**dt**>

<**dd**>Mon 8:30-10:30 lecture</**dd**>

<**dt**>CSCI204</**dt**>

<**dd**>Tue 9:30-11:30 lab</**dd**>

<**dt**>ISIT206</**dt**>

<**dd**>Wed 8:30-10:30 lecture</**dd**>

</**dl**>

My timetable:

MATH222

Mon 8:30-10:30 lecture

CSCI204

Tue 9:30-11:30 lab

ISIT206

Wed 8:30-10:30 lecture

Description List

My timetable:

<dl>

<dt>MATH222</dt>

<dd>Mon 8:30-10:30 lecture</dd>

Description List

Description Term

Description Definition

<dt>CSCI204</dt>

<dd>Tue 9:30-11:30 lab</dd>

<dt>ISIT206</dt>

<dd>Wed 8:30-10:30 lecture</dd>

</dl>

My timetable:

MATH222

Mon 8:30-10:30 lecture

CSCI204

Tue 9:30-11:30 lab

ISIT206

Wed 8:30-10:30 lecture

Formatting text

```
<i>italic text</i> <br />
```

```
<b>bold text</b> <br />
```

```
<mark>highlighted text</mark> <br />
```

```
<del>deleted text</del> <br />
```

```
<ins>inserted text</ins>
```

italic text

bold text

highlighted text

deleted text

inserted text

Formatting text

```
<ins>My timetable:</ins>
```

```
<ul>
```

```
  <li><b>MATH222</b>: Mon 8:30-10:30 <i>lecture</i></li>
```

```
  <li><b>CSCI204</b>: Tue 9:30-11:30 <i>lab</i></li>
```

```
  <li><b>ISIT206</b>: Wed 8:30-10:30 <i>lecture</i></li>
```

```
</ul>
```

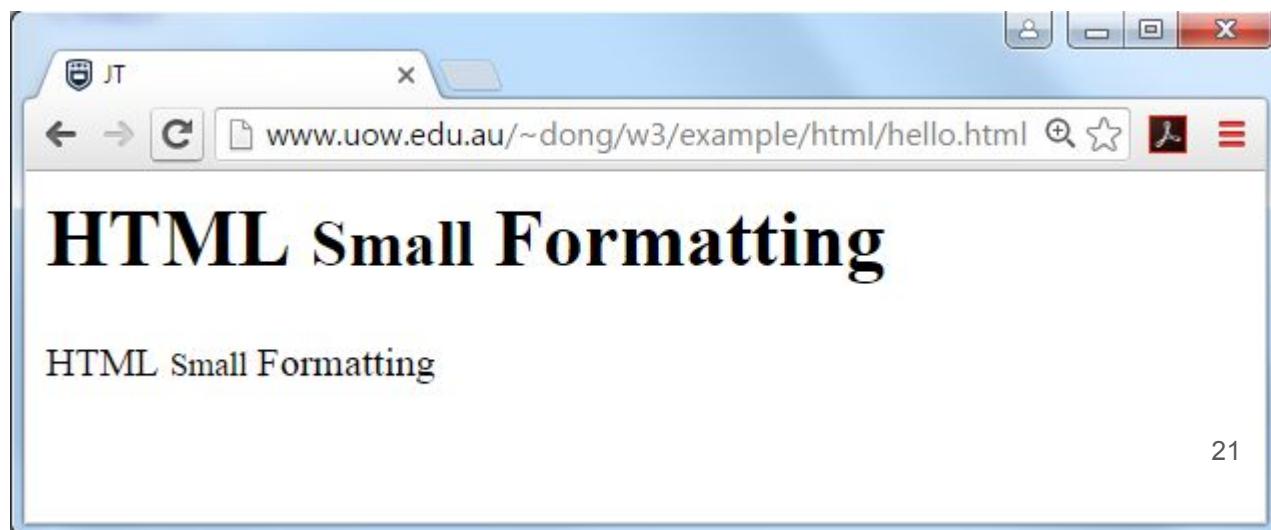
My timetable:

- **MATH222**: Mon 8:30-10:30 *lecture*
- **CSCI204**: Tue 9:30-11:30 *lab*
- **ISIT206**: Wed 8:30-10:30 *lecture*

Formatting text

```
<h1>HTML <small>Small</small> Formatting</h1>
```

```
HTML <small>Small</small> Formatting
```



Formatting text

Changing font

```
<font size="1">This is the smallest size</font>
```

```
<font size="2">smaller than the default</font>
```

```
<font size="3">This is the default font size</font>
```

```
<font size="4">larger than the default</font>
```

```
<font size="5">larger than the default</font>
```

```
<font size="6" color="red">larger than the default</font>
```

```
<font size="7" color="blue">This is the largest size</font>
```

Formatting text

```
<body>
```

Some math

```
x<sub>1</sub><sup>n</sup> +  
x<sub>2</sub><sup>n</sup> =  
y<sub>1</sub><sup>n</sup> +  
y<sub>2</sub><sup>n</sup>
```

```
</body>
```



Table

border attribute

```
<table border="1">
  <tr>
    <th>Username</th>
    <th>First name</th>
    <th>Last name</th>
  </tr>
  <tr>
    <td>jsmith</td>
    <td>John</td>
    <td>Smith</td>
  </tr>
  <tr>
    <td>mlee</td>
    <td>Mary</td>
    <td>Lee</td>
  </tr>
</table>
```

Username	First name	Last name
jsmith	John	Smith
mlee	Mary	Lee

*Turn off the border:
border="0"*

Username	First name	Last name
jsmith	John	Smith
mlee	Mary	Lee

Table

```
<table border="1">
  <tr> ←
    <th>Username</th>
    <th>First name</th>
    <th>Last name</th>
  </tr>
  <tr>
    <td>jsmith</td>
    <td>John</td>
    <td>Smith</td>
  </tr>
  <tr>
    <td>mlee</td>
    <td>Mary</td>
    <td>Lee</td>
  </tr>
</table>
```

Table Row

row 1

row 2

row 3

Username	First name	Last name
jsmith	John	Smith
mlee	Mary	Lee

Table

```
<table border="1">
  <tr>
    <th>Username</th> ← Table Header
    <th>First name</th>
    <th>Last name</th>
  </tr>
  <tr>
    <td>jsmith</td> ← Table Data
    <td>John</td>
    <td>Smith</td>
  </tr>
  <tr>
    <td>mlee</td>
    <td>Mary</td>
    <td>Lee</td>
  </tr>
</table>
```

Username	First name	Last name
jsmith	John	Smith
mlee	Mary	Lee

Table

```
<table border="1">
  <tr>
    <td>jsmith</td>
    <td>John</td>
    <td>Smith</td>
  </tr>
  <tr>
    <td>mlee</td>
    <td>Mary</td>
    <td>Lee</td>
  </tr>
</table>
```

*Sometime we do not need
table header*

Table Data

jsmith	John	Smith
mlee	Mary	Lee

Table

```
<table border="1" width="50%">
<caption>User information</caption>
<tr>
  <th width="20%">Username</th>
  <th width="40%">First name</th>
  <th width="40%">Last name</th>
</tr>
<tr>
  <td align="center">jsmith</td>
  <td align="right">John</td>
  <td align="right">Smith</td>
</tr>
<tr>
  <td align="center">mlee</td>
  <td align="right">Mary</td>
  <td align="right">Lee</td>
</tr>
</table>
```

User information

Username	First name	Last name
jsmith	John	Smith
mlee	Mary	Lee

Table

```
<table border="1" width="50%">
  <caption>User information</caption>
  <tr>
    <th width="20%">Username</th>
    <th width="40%">First name</th>
    <th width="40%">Last name</th>
  </tr>
  <tr>
    <td align="center">jsmith</td>
    <td align="right">John</td>
    <td align="right">Smith</td>
  </tr>
  <tr>
    <td align="center">mlee</td>
    <td align="right">Mary</td>
    <td align="right">Lee</td>
  </tr>
</table>
```

Table caption

User information

Username	First name	Last name
jsmith	John	Smith
mlee	Mary	Lee

Table

```
<table border="1" width="50%>
  <caption>User information</caption>
  <tr>
    <th width="20%">Username</th>
    <th width="40%">First name</th>
    <th width="40%">Last name</th>
  </tr>
  <tr>
    <td align="center">jsmith</td>
    <td align="right">John</td>
    <td align="right">Smith</td>
  </tr>
  <tr>
    <td align="center">mlee</td>
    <td align="right">Mary</td>
    <td align="right">Lee</td>
  </tr>
</table>
```

Table width
within the page

Column width
within the table

User information

Username	First name	Last name
jsmith	John	Smith
mlee	Mary	Lee

Table

```
<table border="1">
  <caption>User information</caption>
  <tr>
    <th width="150px">Username</th>
    <th width="200px">First name</th>
    <th width="200px">Last name</th>
  </tr>
  <tr>
    <td align="center">jsmith</td>
    <td align="right">John</td>
    <td align="right">Smith</td>
  </tr>
  <tr>
    <td align="center">mlee</td>
    <td align="right">Mary</td>
    <td align="right">Lee</td>
  </tr>
</table>
```

Column width
using pixels

User information		
Username	First name	Last name
jsmith	John	Smith
mlee	Mary	Lee

Table

```
<table border="1" width="50%">
  <caption>User information</caption>
  <tr>
    <th width="20%">Username</th>
    <th width="40%">First name</th>
    <th width="40%">Last name</th>
  </tr>
  <tr>
    <td align="center">jsmith</td>
    <td align="right">John</td>
    <td align="right">Smith</td>
  </tr>
  <tr>
    <td align="center">mlee</td>
    <td align="right">Mary</td>
    <td align="right">Lee</td>
  </tr>
</table>
```

Horizontal alignment

User information

Username	First name	Last name
jsmith	John	Smith
mlee	Mary	Lee

Table

```
<table width="50%" border="1">
<tr>
  <td valign="middle">jsmith</td>
  <td>
    John Smith <br />
    DOB: 23/01/2000 <br />
    Course code: 875
  </td>
</tr>
<tr>
  <td valign="middle">mlee</td>
  <td>
    Mary Lee <br />
    DOB: 15/04/2001 <br />
    Course code: 741
  </td>
</tr>
</table>
```

Vertical alignment

jsmith	John Smith DOB: 23/01/2000 Course code: 875
mlee	Mary Lee DOB: 15/04/2001 Course code: 741

Table

```
<table width="50%" border="1">
<tr>
  <td valign="top">jsmith</td>
  <td>
    John Smith <br />
    DOB: 23/01/2000 <br />
    Course code: 875
  </td>
</tr>
<tr>
  <td valign="top">mlee</td>
  <td>
    Mary Lee <br />
    DOB: 15/04/2001 <br />
    Course code: 741
  </td>
</tr>
</table>
```

Vertical alignment

jsmith	John Smith DOB: 23/01/2000 Course code: 875
mlee	Mary Lee DOB: 15/04/2001 Course code: 741

Table

```
<table width="50%" border="1">
<tr>
  <td valign="bottom">jsmith</td>
  <td>
    John Smith <br />
    DOB: 23/01/2000 <br />
    Course code: 875
  </td>
</tr>
<tr>
  <td valign="bottom">mlee</td>
  <td>
    Mary Lee <br />
    DOB: 15/04/2001 <br />
    Course code: 741
  </td>
</tr>
</table>
```

Vertical alignment

jsmith	John Smith DOB: 23/01/2000 Course code: 875
mlee	Mary Lee DOB: 15/04/2001 Course code: 741

Table

Column span

```
<table border="1" width="40%">
  <tr>
    <td colspan="2">STUDENT DETAILS</td>
  </tr>
  <tr>
    <td width="30%">STUDENT NAME</td>
    <td>John Lee</td>
  </tr>
  <tr>
    <td>STUDENT NUMBER</td>
    <td>1234567</td>
  </tr>
  <tr>
    <td>UOW EMAIL</td>
    <td>jlee@uowmail.edu.au</td>
  </tr>
</table>
```

STUDENT DETAILS	
STUDENT NAME	John Lee
STUDENT NUMBER	1234567
UOW EMAIL	jlee@uowmail.edu.au

Table

Row span

```
<table border="1" width="50%">
```

```
  <tr>
    <td></td>
    <td>Monday</td>
    <td>Tuesday</td>
    <td>Wednesday</td>
  </tr>
```

```
  <tr>
    <td>8:30-9:30</td>
    <td rowspan="2">MATH 321 lecture</td>
    <td>INFO 104 lecture</td>
    <td>CS 222 lecture</td>
  </tr>
```

```
  <tr>
    <td>9:30-10:30</td>
    <td>CS 222 Lab</td>
    <td rowspan="2">MATH 321 tutorial</td>
  </tr>
```

```
  <tr>
    <td>10:30-11:30</td>
    <td>CS 222 lecture</td>
    <td>INFO 104 tutorial</td>
  </tr>
```

```
</table>
```

	Monday	Tuesday	Wednesday
8:30-9:30	MATH 321 lecture	INFO 104 lecture	CS 222 lecture
9:30-10:30		CS 222 Lab	MATH 321 tutorial
10:30-11:30	CS 222 lecture	INFO 104 tutorial	

Paragraph tag <p>

```
<p>This is a paragraph</p>
```

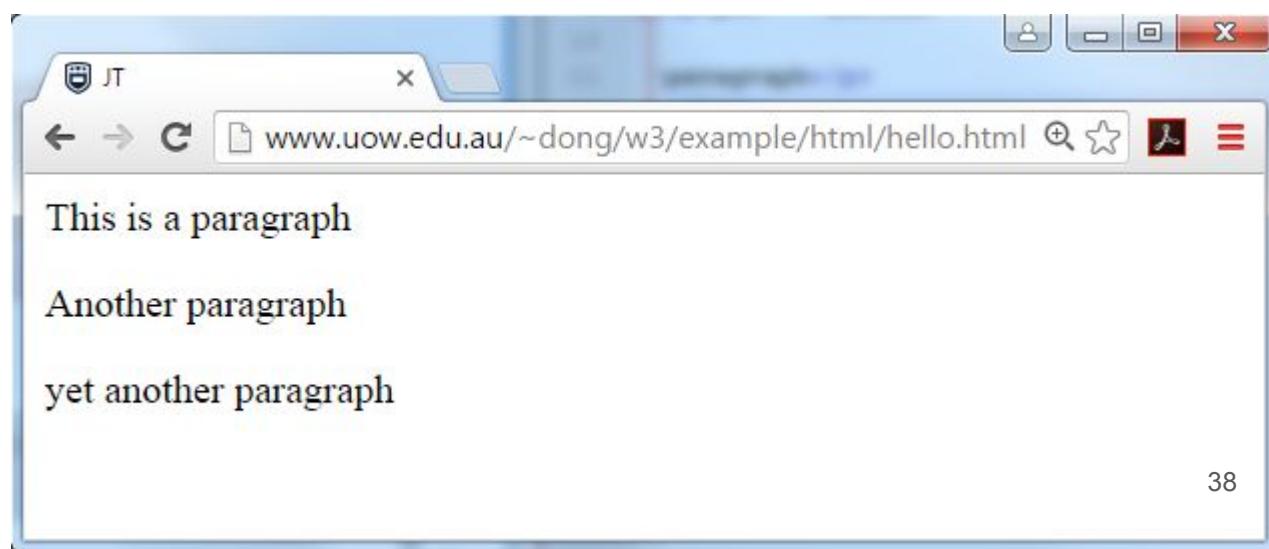
```
<p>Another
```

```
paragraph</p>
```

```
<p>yet another
```

```
paragraph</p>
```

**Extra spaces and lines will NOT
be displayed in paragraph**



Line break


```
<p>This is a paragraph</p>
```

```
<p>Another <br />
```

```
paragraph</p>
```

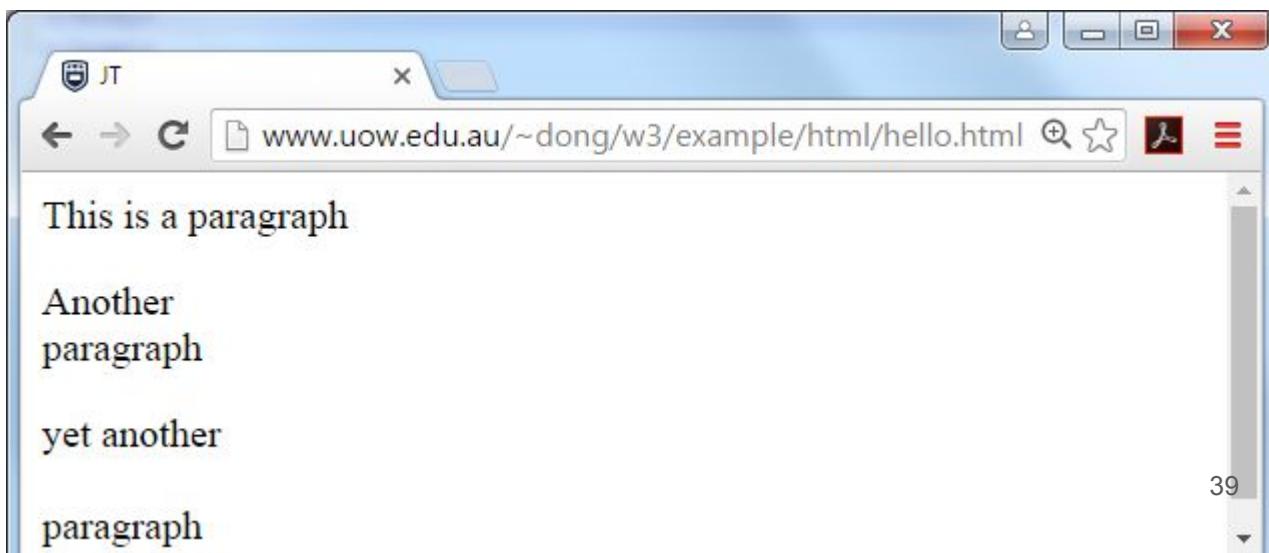
```
<p>yet another <br />
```

```
<br />
```

```
paragraph</p>
```

**
 tag defines a line break**

**
 is an empty element
(i.e. it is a tag with no content),
it combines the start and end
tags together**



Horizontal line <hr />

```
<p>This is a paragraph</p>
```

```
<p>Another <br />
```

```
paragraph</p>
```

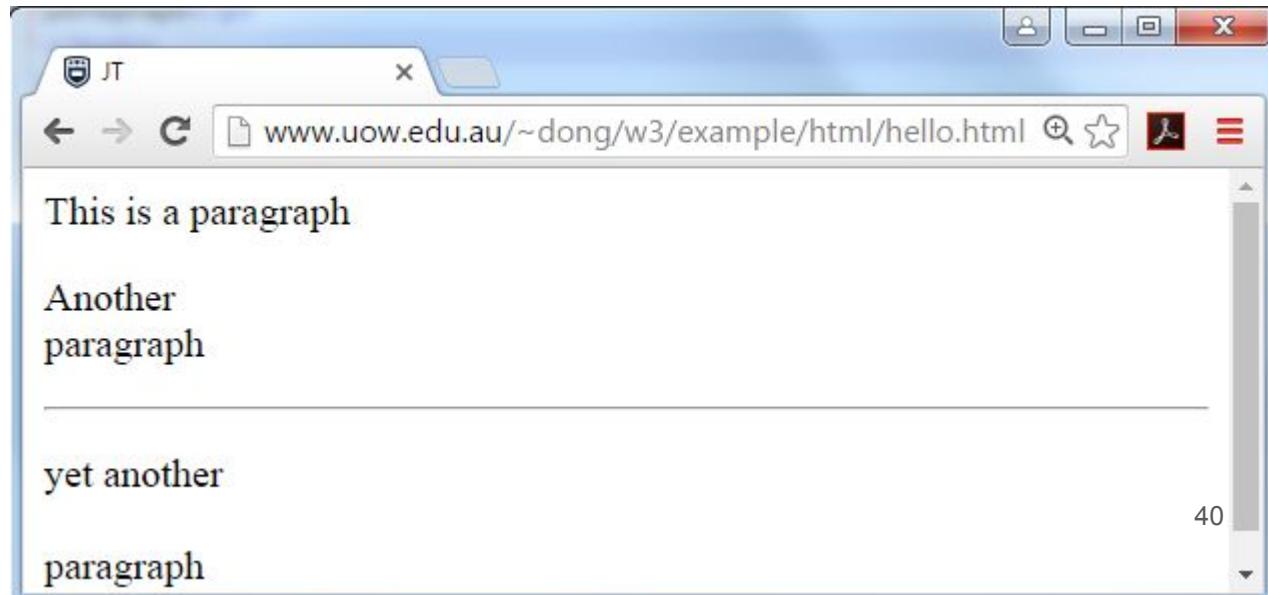
```
<hr />
```

```
<p>yet another <br />
```

```
<br />
```

```
paragraph</p>
```

similarly, we have the horizontal line tag <hr /> with no content



Non-breaking space

```
<p>This &nbsp; &nbsp; is a  
paragraph</p>
```

use for non-breaking space

```
<p>Another <br />
```

this is an example of **character entities**

```
paragraph</p>
```

```
<hr />
```

```
<p>yet another <br />
```

```
<br />
```

```
paragraph</p>
```

Character entity

- Some characters are reserved in HTML.
- Reserved characters in HTML must be replaced with character entities.

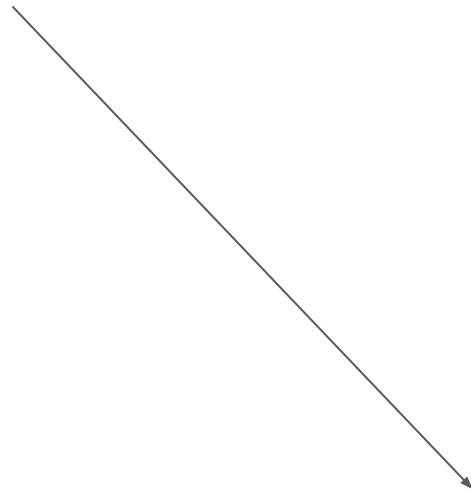
Character	Entity	Meaning
(non-breaking space)	 	Non-breaking space
<	<	Is less than
>	>	Is greater than
&	&	Ampersand
“	"	Double quote
‘	'	Single quote (apostrophe)
°	°	Degree
©	©	Copyright

Character entity

<body>

A HTML document starts with
<html> and ends with
</html>

</body>



A HTML document starts with <html>
and ends with <html>

Block quotations <blockquote>

```
<p>normal paragraph</p>
```

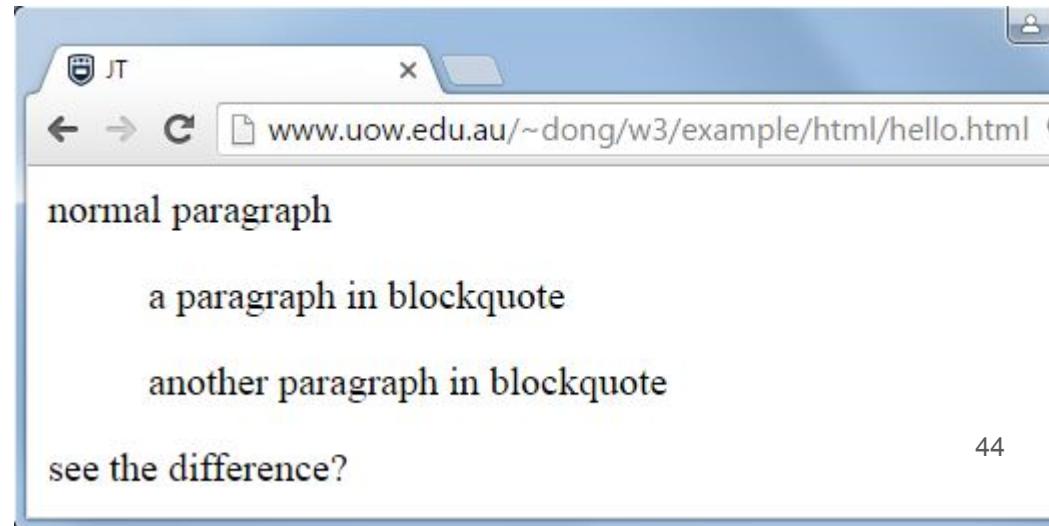
```
<blockquote>
```

```
<p>a paragraph in blockquote</p>
```

```
<p>another paragraph in blockquote</p>
```

```
</blockquote>
```

see the difference?



Preformatted text <pre>

```
<pre>
```

pre element is shown in monospace

```
Mary
```

it preserves the character and line spacing

```
    had    a
```

```
        little
```

```
            lamb
```

```
</pre>
```



Computer code

```
<pre>
```

`<code>`

```
a = 0;  
b = 3;  
c = 2;  
  
sum = a + b + c;
```

`</code>`

```
</pre>
```



what would happen if we use <code> ... </code> without <pre> ?

Computer code

```
<pre>
<code>
#include <iostream>

void main( ) {
    cout << "Hello World!" << endl;
}
</code>
</pre>
```

If you want to include special characters such as < > & " ' within **pre** tags, they should be substituted by character entities so that they are not subject to special interpretation by the browser.

```
#include <iostream>

void main( ) {
    cout << "Hello World!" << endl;
}
```

Image

```

```

Attribute	Description
src	URL of an image, for example <code>src="uow-logo.png"</code> <code>src="images/uow-logo.png"</code> <code>src="http://www.mycom.au/staff.png"</code>
alt	alternate text for an image
height width	optional. Specifies height, width for image in pixels, or in percentage

Image alt

```

```

- If a browser cannot find an image, it will display the `alt` text.
- Sometimes, to save bandwidth, user can disable image display, in this case, the `alt` text will be displayed.
- A screen reader is a software program that can read what is displayed on a screen which is very useful to people who are blind or visually impaired. Screen readers can read the `alt` text.

Image src

The URL of an image can be

- an **absolute** URL points to another website
- or a **relative** URL points to an image file within a website

Absolute URL

```
src="http://www.mycom.au/staff.png"
```

Relative URL

src="uow-logo.png" : the image file is in the same directory as the current html file

src="images/uow-logo.png" : the image file is in the subdirectory called **images** located at the same directory as the current html file

```
src="images/logo/uow-logo.png"
```

```
src="../../f1/bird.png"
```

Link

```
<a href="http://www.uow.edu.au" target="_blank">Visit UOW</a>
```

```
<a href="contact.html">Contact us</a>
```

```
<a href="http://www.uow.edu.au" target="_blank">
```

```

```

```
</a>
```

Link

```
<a href="http://www.uow.edu.au" target="_blank">Visit UOW</a>
```

The **href** in this example is an **absolute URL**.

If user clicks on this link, `http://www.uow.edu.au` will be opened in a new tab

target	description
_blank	open the link in a new window or tab
_self	open the link in the same frame (this is default)

Link

```
<a href="contact.html">Contact us</a>
```

The **href** in this example is a **relative URL**.

It is similar to the `src` attribute of the `img` tag:

```
href="contact.html"
```

```
href="assignment/a1.html"
```

```
href="..../handout/note5.html"
```

Link

```
<a href="http://www.uow.edu.au" target="_blank">  
    
</a>
```

Within the link tag `<a href...> `, we can put any text or image.

In the above example, it displays an image as a link to the address

`http://www.uow.edu.au`

Link - target within document

Within the html document we can use the attribute **id** to mark a specific location

```
<a href="#Proofs">1 Proofs</a>
<a href="#See_also">2 See also</a>
<a href="#Notes">3 Notes</a>
<a href="#References">4 References</a>
<a href="#External_links">5 External links</a>
```

```
<h3 id="Proofs">Proofs</h3>
```

...

```
<h3 id="See_also">See also</h3>
```

...

```
<h3 id="Notes">Notes</h3>
```

...

```
<h3 id="References">References</h3>
```

...

```
<h3 id="External_links">External links</h3>
```

...

Euler's theorem

Contents

- 1 Proofs
- 2 See also
- 3 Notes
- 4 References
- 5 External links

Proofs

....

See also

....

Notes

....

References

....

External links

Link - target within document

Within the html document we can use the attribute **id** to mark a specific location

```
<a href="#Proofs">1 Proofs</a>
<a href="#See_also">2 See also</a>
<a href="#Notes">3 Notes</a>
<a href="#References">4 References</a>
<a href="#External_links">5 External links</a>

<h3 id="Proofs">Proofs</h3>
```

...



The id value must be unique and must contain at least one character.
The id value must not contain any space characters.

Link - target within document

We can create a link to a specific location within a html page

For example:

```
<a href="https://en.wikipedia.org/wiki/Euler%27s_theorem#Proofs">  
Proof of the Euler theorem</a>
```

index.html

- index.html is a default page for a directory
- For security reason, it is better to have index.html for every directory
- It stops people from knowing the content and structure of your website

Example: If the directory abc has an index.html then when we go to
<http://the-web-address/abc>

it automatically displays the page <http://the-web-address/abc/index.html>

Comments

```
<body>
```

```
<!-- this is
```

```
a long comment
```

```
it will not be displayed on the web page
```

```
-->
```

```
</body>
```

XHTML

- XHTML stands for EXtensible HyperText Markup Language
- XHTML is stricter than HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
  
<html xmlns="http://www.w3.org/1999/xhtml">  
  <head>  
    <title></title>  
  </head>  
  <body>  
  </body>  
</html>
```



DOCTYPE and xmlns are mandatory

XHTML

- XHTML is stricter than HTML

elements must be properly nested:

```
<b><i>bold and italic - not nested correctly</b></i>
```

It should be like this:

```
<b><i>bold and italic - not nested correctly</i></b>
```

XHTML

- XHTML is stricter than HTML

elements must always be closed:

```
<p>paragraph not closed  
break not closed <br>  
horizontal line not closed <hr>  
image not closed 
```

It should be like this:

```
<p>paragraph not closed</p>  
break not closed <br />  
horizontal line not closed <hr />  
image not closed 
```

XHTML

- XHTML is stricter than HTML

elements must be in lower case:

```
<HEAD>
<TITLE>Web Technologies</TITLE>
<HEAD>
```

It should be like this:

```
<head>
<title>Web Technologies</title>
<head>
```

XHTML

- XHTML is stricter than HTML

attribute names must be in lower case:

```
<img SRC="logo.png" ALT="UOW logo" />
```

It should be like this:

```

```

XHTML

- XHTML is stricter than HTML

attribute values must be quoted:

```
<img src=logo.png alt=UOW logo />
```

It should be like this:

```

```

XHTML

- XHTML is stricter than HTML

attribute minimization is not allowed:

```
<input type="checkbox" name="subscribe" value="eletter"  
checked />
```

It should be like this:

```
<input type="checkbox" name="subscribe" value="eletter"  
checked="checked" />
```

HTML5

- version 5 of the HTML standard
- introduces new tags

```
<header> <footer> <article> <section>  
<svg> <canvas> <audio> <video> ...
```

- introduces new APIs
 - Drag and Drop
 - Local Storage
 - Geolocation
 -

References

- <http://www.w3schools.com/html>
- <http://developer.mozilla.org/en-US/docs/Web/HTML>

Introduction to Web Technology

CSS

Joseph Tonien

School of Computing and Information Technology
University of Wollongong

CSS

Objectives:

- understand the need of separation between the content and the style of your website
- learn 3 different ways to define web page styles
- use CSS language to define styles for your web pages

Cascading Style Sheets

CSS provides a separation between the HTML document **content** and document **presentation** (style).

3 ways to add styling to HTML elements:

- **Inline**

using a **style** attribute in HTML elements

- **Document**

using **<style>** element in the HTML **<head>** section

- **External**

using external **CSS files**

Inline CSS

By using a **style** attribute in HTML elements

```
<body style="background-color:lightgrey;">  
  
<h1 style="color:blue;">This is a Blue Heading</h1>
```

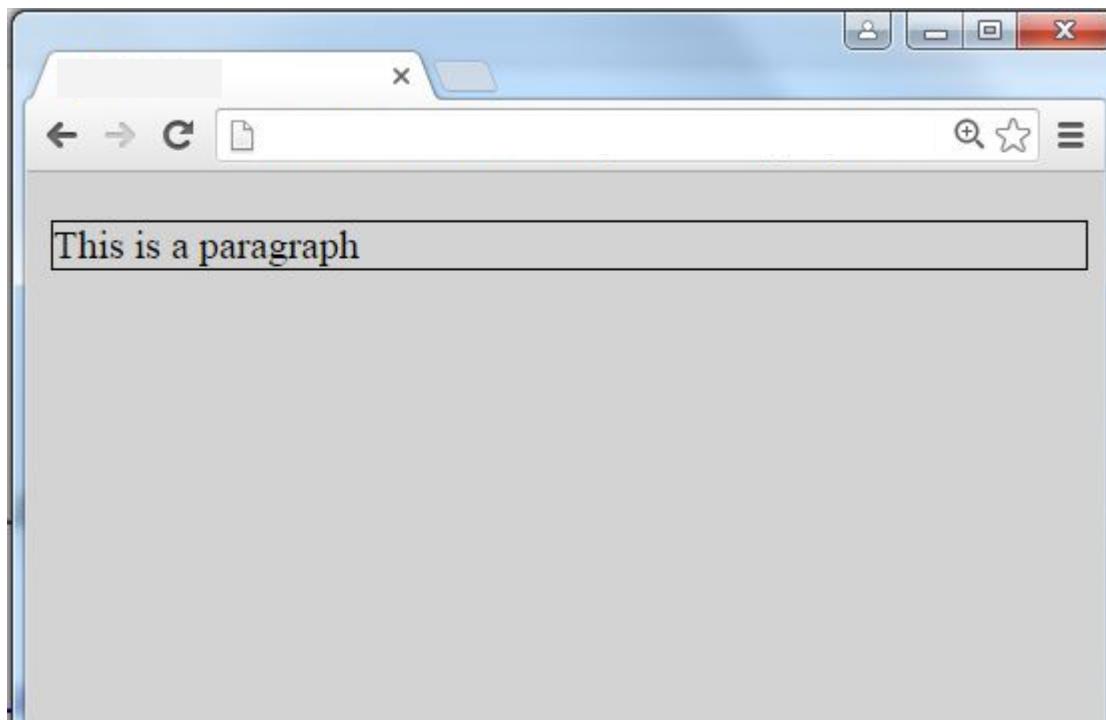


Inline CSS

```
<p style="border:1px solid black;">
```

This is a paragraph with border

```
</p>
```



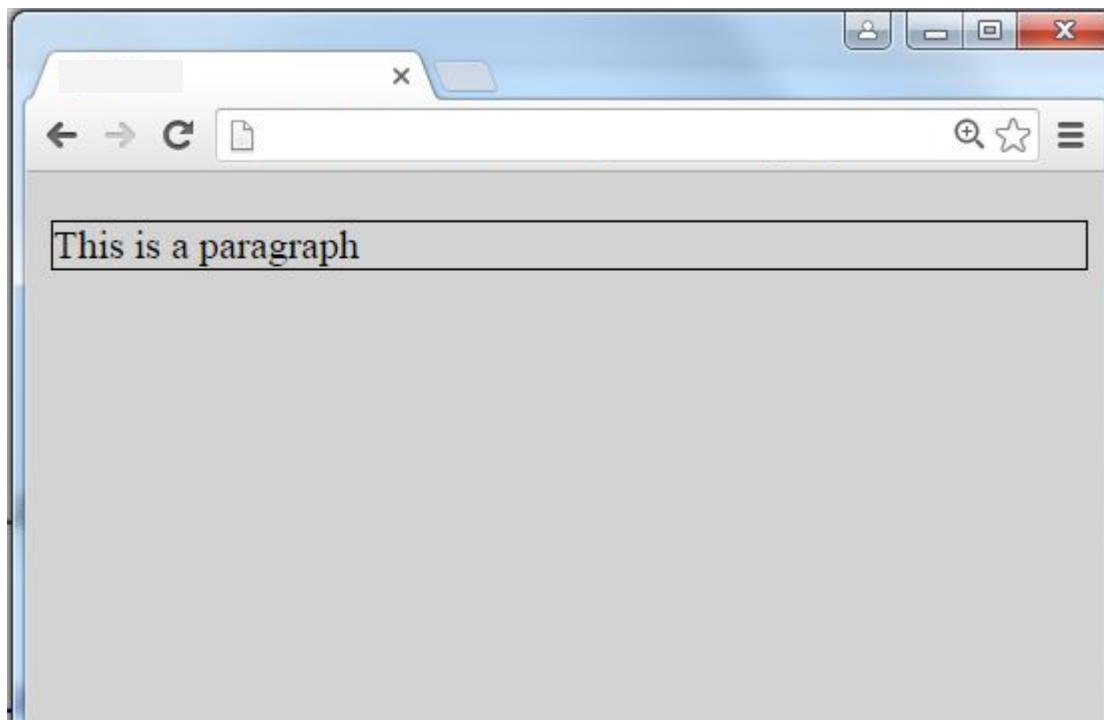
this is called a CSS **property**

Inline CSS

```
<p style="border:1px solid black;">
```

This is a paragraph with border

```
</p>
```

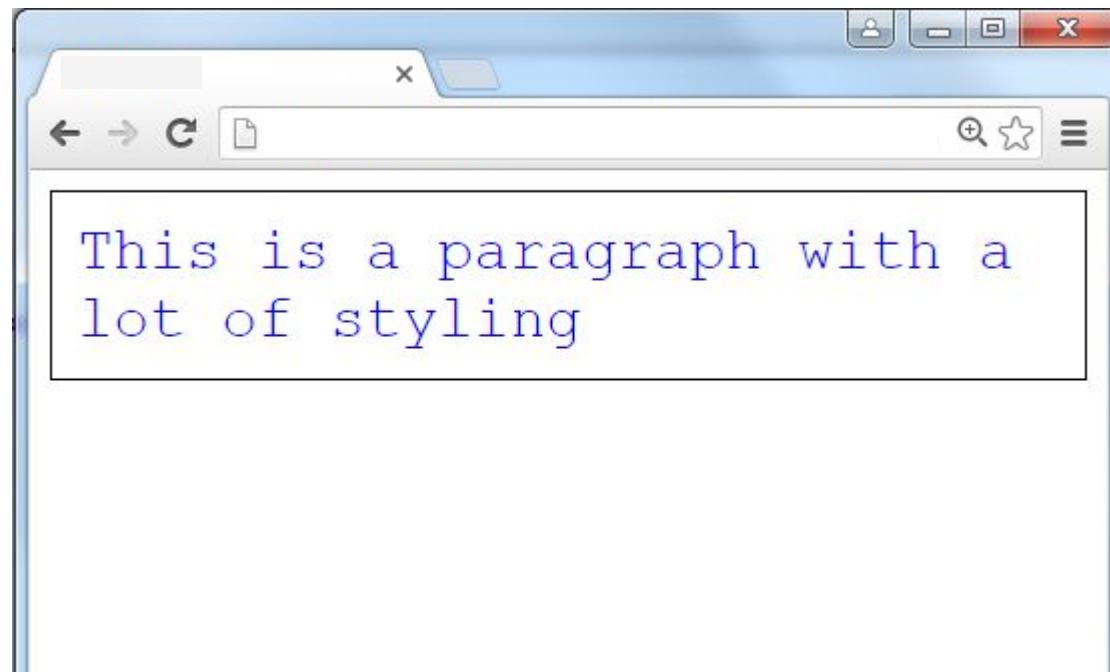


Inline CSS

```
<p style="border:1px solid black; padding:10px; color:blue;  
font-family:courier; font-size:150%;">
```

This is a paragraph with a lot of styling

```
</p>
```



Inline CSS

```
<p style="border:1px solid black; padding:10px; color:blue;  
font-family:courier; font-size:150%;">
```

This is a paragraph with a lot of styling

```
</p>
```

- A CSS style is specified with the following format

property:value

- We can specify more than one CSS property, separated by a semicolon (;)

```
style="border:1px solid black; padding:10px; color:blue;  
font-family:courier; font-size:150%;"
```

- A CSS property may have many values separated by space

border:1px solid black

Color

CSS supports 140 standard color names.

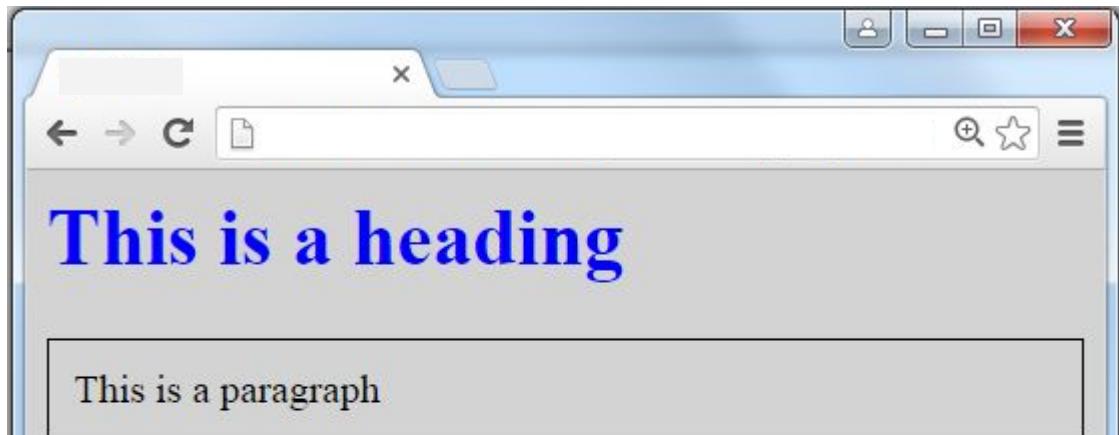
Color can also be specified by hex code.

```
<h1 style="color:lightgrey;">This is a Light Grey Heading</h1>
```

```
<h1 style="color:#D3D3D3;">This is a Light Grey Heading</h1>
```

Document CSS

```
<html>
<head>
<title>W3</title>
<style>
body {background-color:lightgrey;}
h1 {color:blue;}
p {border:1px solid black; padding:10px;}
</style>
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph</p>
</body>
</html>
```

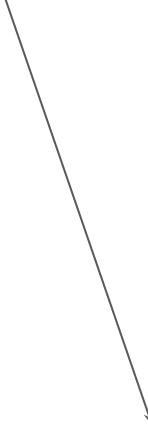


External CSS

```
<html>
<head>
<title>W3</title>

<link rel="stylesheet" href="path/to/mystyle.css">
```

```
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph</p>
</body>
</html>
```



mystyle.css

```
body {background-color:lightgrey;}
h1 {color:blue;}
p {border:1px solid black; padding:10px;}
```

Levels of CSS

- Inline CSS has precedence over document CSS
- Document CSS has precedence over external CSS
- Suppose an external CSS specifies a value for a particular property of a HTML element, then that value can be overridden by a document CSS, which in turn, can be overridden by an inline CSS.

CSS convention

This is a valid CSS

mystyle.css

```
body {background-color:lightgrey;}  
h1 {color:blue;}  
p {border:1px solid black; padding:10px;}
```

But for better clarity, we should use the following convention:

```
body {  
    background-color:lightgrey;  
}
```

```
h1 {  
    color:blue;  
}
```

```
p {  
    border:1px solid black;  
    padding:10px;  
}
```

each property on
a separate line

Simple selector

This is called a simple selector



```
p {  
    border:1px solid black;  
    padding:10px;  
}
```

We can also have this simple selector.



In this case, all `<h1>` and `<h2>` elements will be applied with this style.

```
h1, h2 {  
    border:1px solid black;  
    color:lightgrey;  
}
```

Class selector

```
<h1 class="userInfo">This is a heading 1</h1>
<p class="userInfo">This is a paragraph 1</p>
<h2 class="userInfo">This is a heading 2</h2>
<p class="userInfo">This is a paragraph 2</p>
```

```
<h1 class="eticket">This is a heading</h1>
<p class="eticket">This is a paragraph</p>
<h2 class="eticket">This is a heading</h2>
```

All `<p>` elements of `class userInfo` will be applied with this style.

```
-----> p.userInfo {
    border:1px solid black;
    padding:10px;
}
```

All `<h1>` and `<h2>` elements of `class userInfo` will be applied with this style.

```
-----> h1.userInfo, h2.userInfo {
    color:blue;
}
```

Class selector

```
<h1 class="userInfo">This is a heading 1</h1>
<p class="userInfo">This is a paragraph 1</p>
<h2 class="userInfo">This is a heading 2</h2>
<p class="userInfo">This is a paragraph 2</p>

<h1 class="eticket">This is a heading</h1>
<p class="eticket">This is a paragraph</p>
<h2 class="eticket">This is a heading</h2>
```

All elements of **class eticket**
will be applied with this style.



```
.eticket {
    color:green;
}
```

Id selector

```
<h1 id="userHeading">This is a heading 1</h1>
```

```
<p id="userDetails">This is a paragraph 1</p>
```

```
<h2 id="bankHeading">This is a heading 2</h2>
```

```
<p id="bankDetails">This is a paragraph 2</p>
```

The element with **id**
userHeading will be applied
with this style.

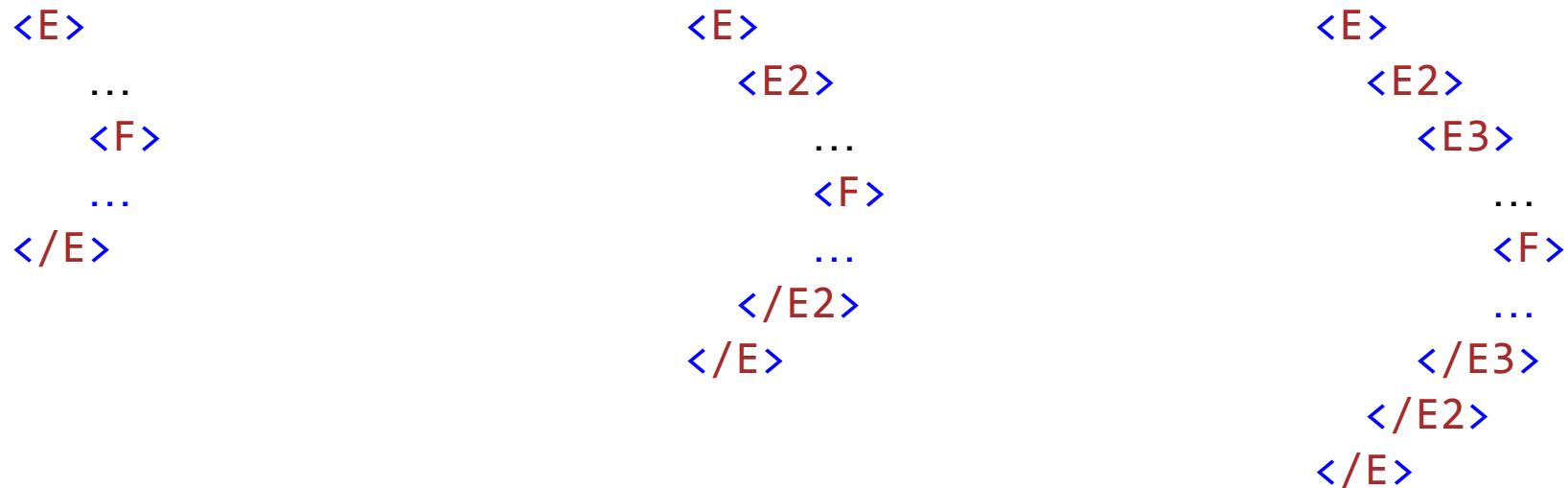


```
#userHeading {  
    color:blue;  
}
```

Note that each HTML element should have a unique id

Descendant-Ancestor

An element F is a *descendant* of element E if it appears in the content of E.
In this case, E is called an ancestor of F.



Descendant-Ancestor

Example:

What are the descendants
of this element `div` ?



`<div>`
Some text `<i>italic</i>` here.

`<p>`

Hi there `<i>italic again</i>`

`</p>`

`<div>`

This is the final `<i>italic</i>`.

`</div>`

`</div>`

Child-Parent

An element F is a *child* of element E if it is nested directly in the content of E. In this case, E is called a parent of F.

```
<E>
...
<F>
...
</E>
```

Of course, if F is a child of E then F is also a descendant of E.

Child-Parent

What are the children
of this element **div** ?



Example:

```
<div>
  Some text <i>italic</i> here.
  <p>
    Hi there <i>italic again</i>
  </p>
  <div>
    This is the final <i>italic</i>.
  </div>
</div>
```

Contextual Selector

Apply this style to every
descendant F of E



```
E F {  
  property:value  
...  
}
```

Apply this style to every
child F of E



```
E > F {  
  property:value  
...  
}
```

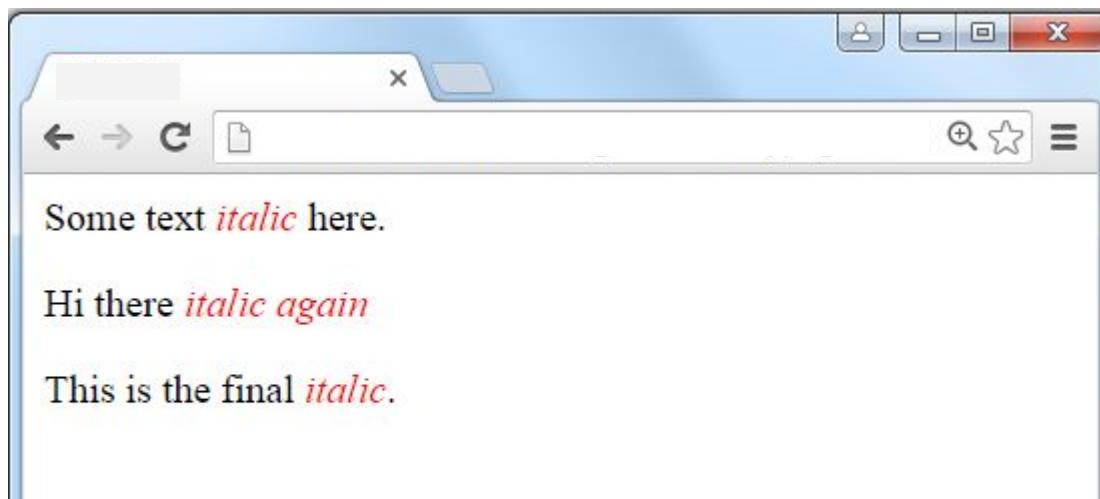
Contextual Selector

Example:

```
<div>
Some text <i>italic</i> here.

<p>
    Hi there <i>italic again</i>
</p>
<div>
    This is the final <i>italic</i>.
</div>
</div>
```

```
div i {
    color:red;
}
```



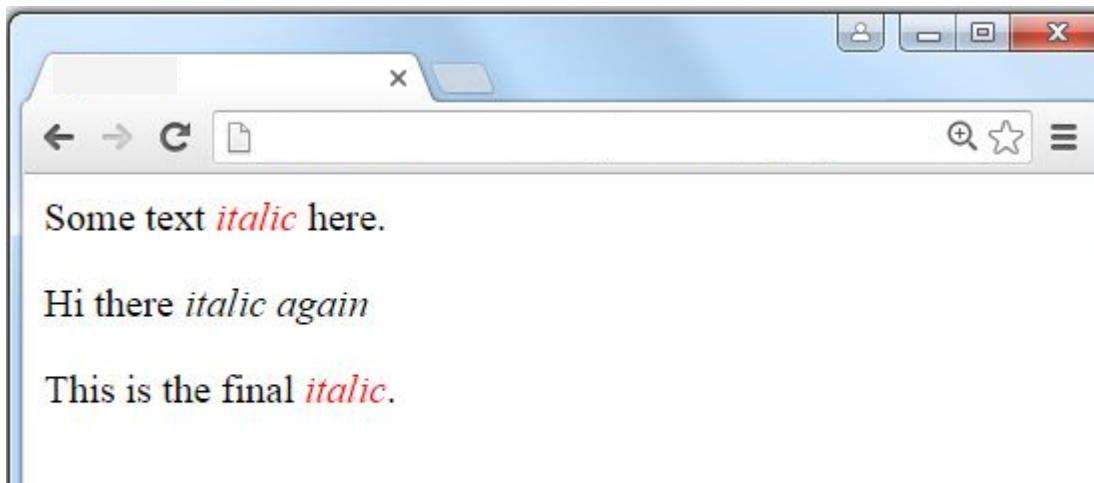
Contextual Selector

Example:

```
<div>
Some text <i>italic</i> here.

<p>
    Hi there <i>italic again</i>
</p>
<div>
    This is the final <i>italic</i>.
</div>
</div>
```

```
div > i {
    color:red;
}
```

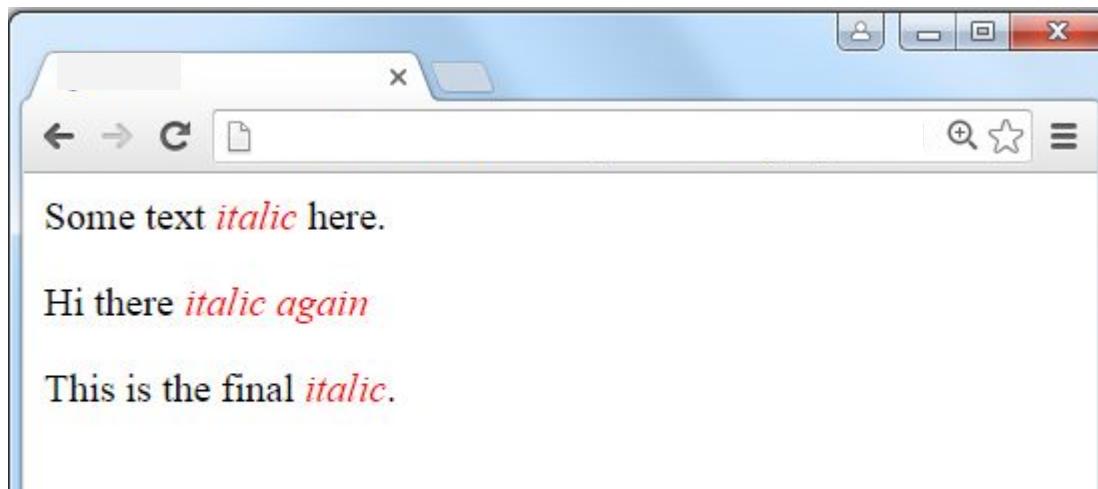


Contextual Selector

Example:

```
<div class="userInfo">  
Some text <i>italic</i> here.  
  
<p>  
    Hi there <i>italic again</i>  
</p>  
<div class="bankInfo">  
    This is the final <i>italic</i>.br/>  
</div>  
</div>
```

```
div.userInfo i {  
    color:red;  
}
```

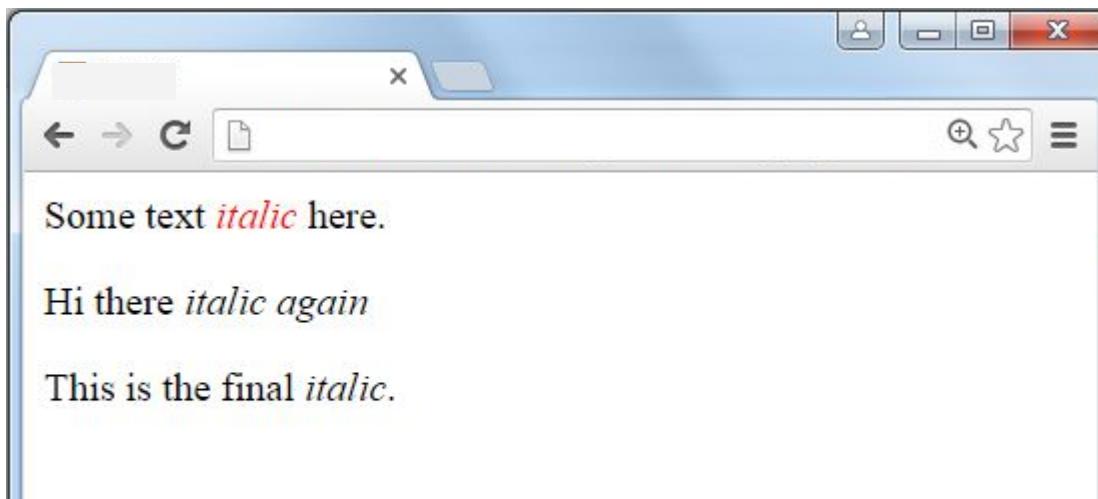


Contextual Selector

Example:

```
<div class="userInfo">  
  Some text <i>italic</i> here.  
  
  <p>  
    Hi there <i>italic again</i>  
  </p>  
  <div class="bankInfo">  
    This is the final <i>italic</i>.  
  </div>  
</div>
```

```
div.userInfo > i {  
  color:red;  
}
```

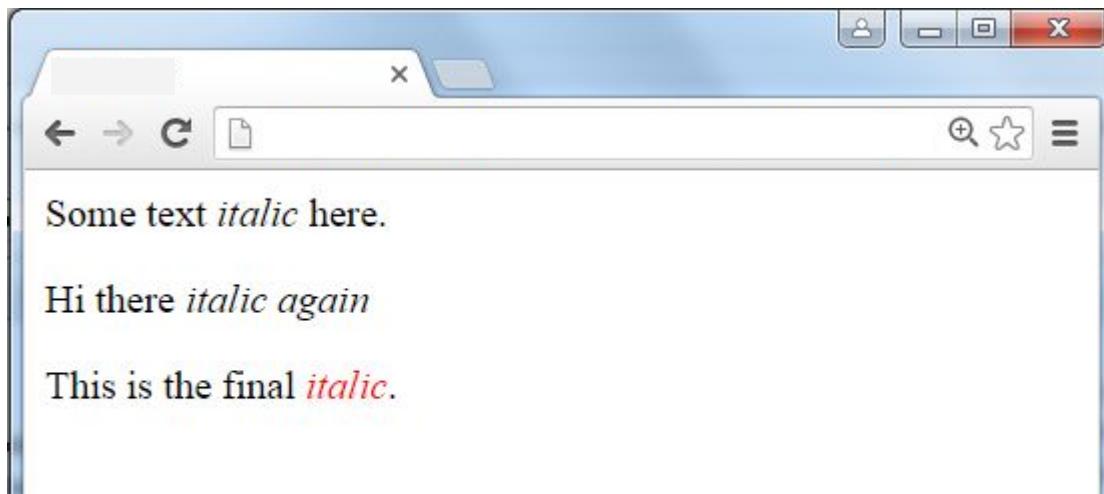


Contextual Selector

Example:

```
<div class="userInfo">  
Some text <i>italic</i> here.  
  
<p>  
    Hi there <i>italic again</i>  
</p>  
<div class="bankInfo">  
    This is the final <i>italic</i>.  
</div>  
</div>
```

```
div.userInfo i {  
    color:red;  
}  
div.bankInfo i {  
    color:red;  
}
```

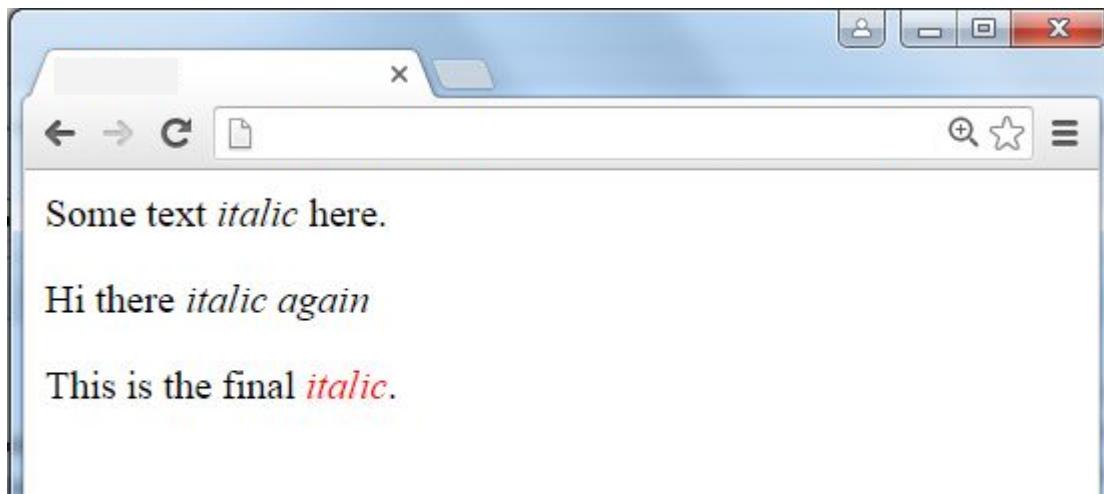


Contextual Selector

Example:

```
<div class="userInfo">  
Some text <i>italic</i> here.  
  
<p>  
    Hi there <i>italic again</i>  
</p>  
<div class="bankInfo">  
    This is the final <i>italic</i>.br/>  
</div>  
</div>
```

```
div.userInfo > i {  
    color:red;  
}
```



Pseudo class selector

```
<a href="http://www.uow.edu.au">UOW</a>
```

The **link** pseudo class is used to style a link that has not been selected.

The **visited** pseudo class is used to style a link that previously has been selected.

```
a:link {  
    color:red;  
}
```

```
a:visited {  
    color:green;  
}
```

```
h1:hover {  
    color:blue;  
}
```

```
<h1>A heading</h1>
```

Any time the mouse cursor is position over the **h1** element then the style will be applied.

List properties

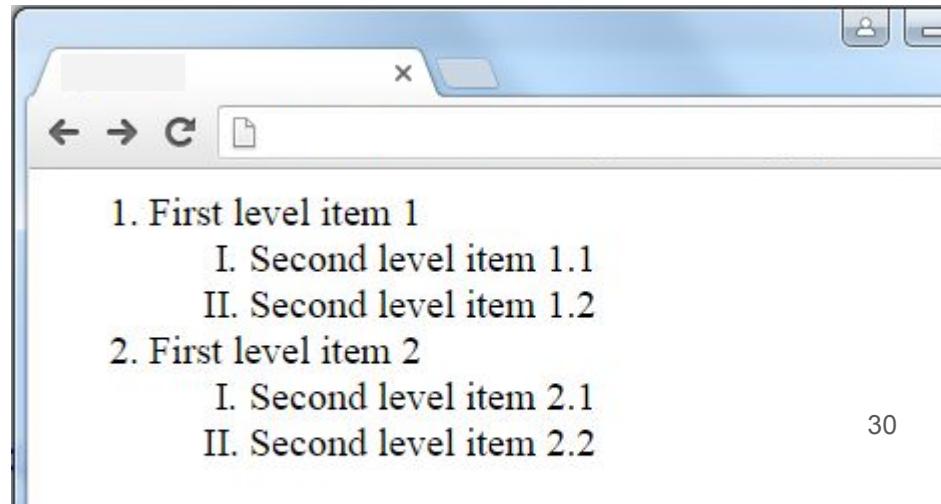
```
<ol>
  <li>First level item 1
    <ol>
      <li>Second level item 1.1</li>
      <li>Second level item 1.2</li>
    </ol>
  </li>

  <li>First level item 2
    <ol>
      <li>Second level item 2.1</li>
      <li>Second level item 2.2</li>
    </ol>
  </li>
</ol>
```

other values: `decimal-leading-zero`,
`lower-alpha`, `lower-latin`,
`lower-greek`, `disc`, `square`,
`circle`

```
ol {
  list-style-type:decimal;
}

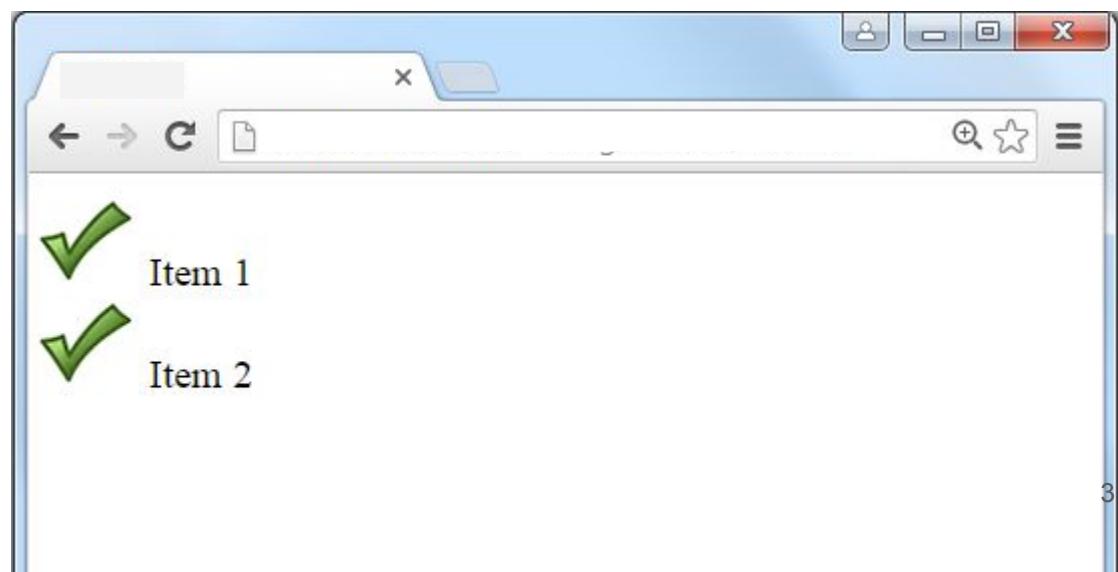
ol ol {
  list-style-type:upper-roman;
}
```



List properties

```
ol {  
    list-style-image:url(path/to/imagefile);  
}
```

```
<ol>  
    <li>Item 1</li>  
  
    <li>Item 2</li>  
</ol>
```

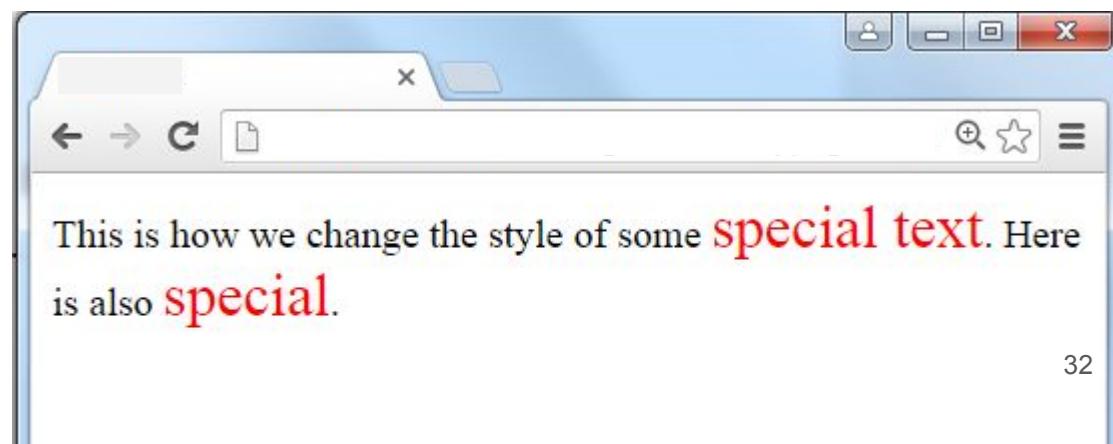


span

Sometimes it is useful to have a word or phrase in a line appear in a different style, we use `... ` for this purpose.

This is how we change the style of some
`special text`.
Here is also
`special`.

```
span.specialText {  
    color:red;  
    font-family:Ariel;  
    font-size:150%;  
}
```



div

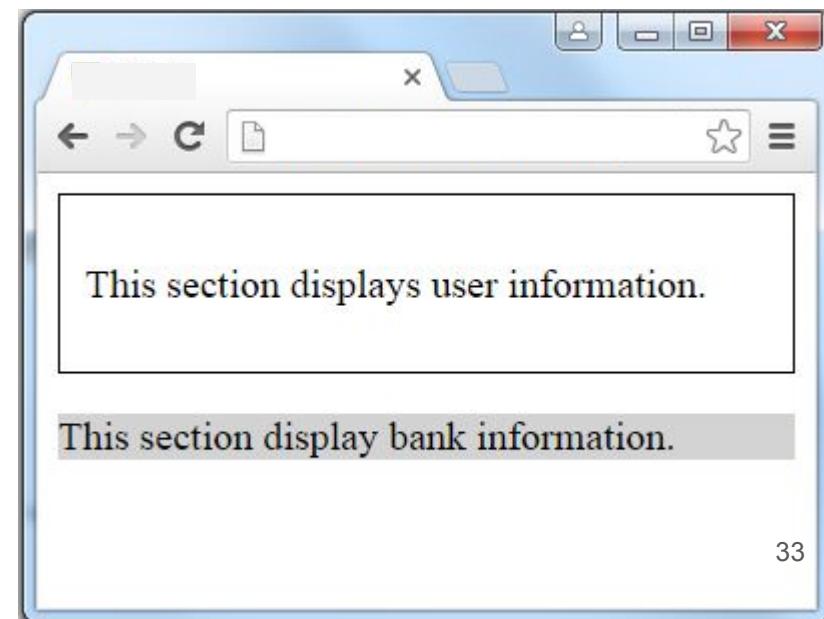
Sometimes we want to have different style at different section of the webpage, we use `<div>... </div>` for this purpose.

```
<div class="userInfo">  
  <p>This section displays user information.</p>  
</div>
```

```
<div class="bankInfo">  
  <p>This section display bank information.</p>  
</div>
```

```
div.userInfo {  
  border:1px solid black;  
  padding:10px;  
}
```

```
div.bankInfo {  
  background-color:lightgrey;  
}
```



Comments in CSS

A comment starts with `/*` and ends with `*/`

Comments can span over multiple lines.

```
p {  
    border:1px solid black;  
  
    /* This is a single-line comment */  
  
    color:blue;  
}  
  
/* This is  
a multi-line  
comment */
```

References

- <http://www.w3schools.com/css>
- https://en.wikipedia.org/wiki/Cascading_Style_Sheets
- <https://developer.mozilla.org/en-US/docs/Web/CSS/Reference>

Introduction to Web Technology

Basics of JavaScript

Joseph Tonien

School of Computing and Information Technology
University of Wollongong

JavaScript

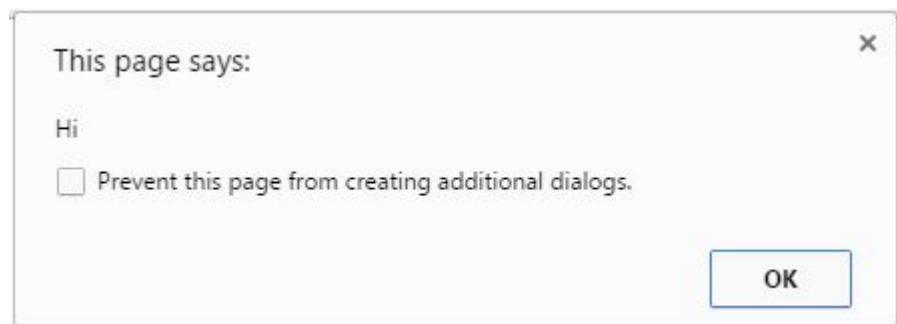
Objectives:

- learn basic JavaScript programming language syntax
- use JavaScript to make your website interactive

My First JavaScript

```
<button onClick="alert('Hi');">  
Click me  
</button>
```

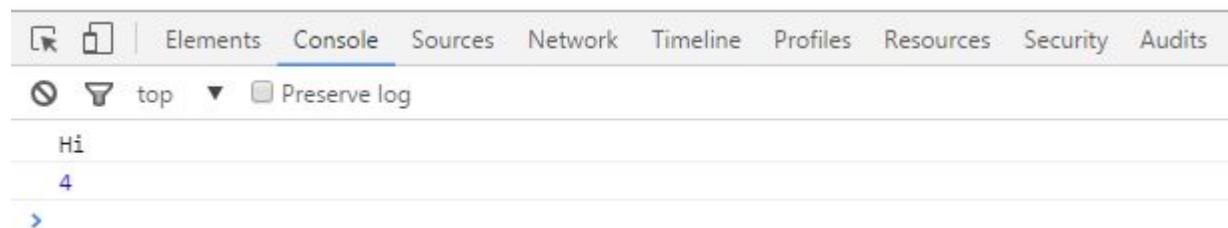
```
<button onClick="alert(1+1);">  
Click me  
</button>
```



My First JavaScript

```
<button onClick="console.log('Hi');>  
Click me  
</button>
```

```
<button onClick="console.log(2+2);>  
Click me  
</button>
```



My First JavaScript

```
<button onClick="alert('Hi') ; console.log(2+2) ;">  
Click me  
</button>
```

We better put the code inside a function to make it clearer!

My First JavaScript

```
<button onClick="sayHi () ;">  
Click me  
</button>  
  
<script>  
  
function sayHi () {  
    alert ("Hi") ;  
}  
  
</script>
```

Where to include JavaScript

We can put JavaScript code anywhere in the HTML file.

Common practice:

- In the head
- At the end of body

```
<script>

function sayHi () {
    alert("Hi");
}

</script>
```

Where to include JavaScript

In the head

<head>

```
<title>JavaScript Example</title>
```

<script>

```
function sayHi () {  
    alert("Hi");  
}
```

</script>

</head>

Where to include JavaScript

At the end of body (just before the closing body tag)

...

<script>

```
function sayHi () {  
    alert("Hi");  
}
```

</script>

</body>

</html>

External JavaScript

Instead of putting javascript code inside the html file

```
<script>

function sayHi () {
    alert("Hi");
}

</script>
```

we can specify an external javascript file:

```
<script type="text/javascript" src="js/myscript.js"></script>
```

Basic JavaScript syntax

JavaScript statements are separated by semicolons

```
function silly() {  
    alert('Hi');  
    console.log(2+2);  
}
```

Basic JavaScript syntax

JavaScript Comments

Code after double slashes `//` or between `/*` and `*/` is treated as a comment.

Comments are ignored, and will not be executed.

```
/*
this function does a few silly things
*/
function silly(){
    // display an alert box
    alert('Hi');

    // print out the number 4 on the console
    console.log(2+2);

}
```

Basic JavaScript syntax

JavaScript uses the **var** keyword to declare variables.

```
var studentName = "John";  
  
var x, y;  
  
x = 5;  
  
y = x + 2;
```

All JavaScript identifiers are **case sensitive**.

- The variables **studentName** and **StudentName** are two different variables.
- The variables **x** and **X** are two different variables.

Basic JavaScript syntax

Variable naming: two common conventions

underscore:

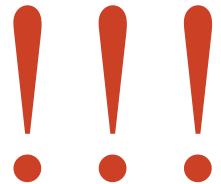
student_name, student_id, first_name, last_name

camel case:

studentName, studentId, firstName, lastName

Basic JavaScript syntax

JavaScript has dynamic types.



This means that the same variable can be used as **different types**:

```
var x; // x is undefined
```

```
alert(x);
```

```
var x = 2016; // x is a number
```

```
alert(x);
```

```
var x = "Wollongong"; // x is a string
```

```
alert(x);
```

A variable declared without a value will have the value **undefined**.

Basic JavaScript syntax

JavaScript data type: number

```
var age = 19;  
var pi = 3.14;
```

Arithmetic operators are used to perform arithmetic on numbers

- + Addition
- Subtraction
- * Multiplication
- / Division
- % Modulus

Basic JavaScript syntax

JavaScript data type: string

```
var age = "19";
```

```
var name = 'John';
```

Basic JavaScript syntax

Strings are text, written within double or single quotes:

```
var firstName, lastName, fullName;  
  
firstName = "John";           // using double quotes  
lastName = 'Lee';            // using single quotes  
  
fullName = firstName + " " + lastName;  
alert(fullName);
```

Use **+** for string concatenation

Basic JavaScript syntax

Mixing between double or single quotes:

```
var x;
```

```
x = "I'm John";           //single quote inside double quotes  
alert(x);
```

```
x = "My name is 'John'"; //single quotes inside double quotes  
alert(x);
```

```
x = 'My name is "John"'; //double quotes inside single quotes  
alert(x);
```

Basic JavaScript syntax

Change string to number

```
var ageString = "19";  
  
var age = Number(ageString); // age is the number 19
```

Change number to string

```
var age = 19;  
  
var ageString = age.toString(); // ageString is the string "19"
```

Basic JavaScript syntax

JavaScript evaluates expressions from left to right

```
var x;  
  
x = 2016 + "Wollongong";           //2016Wollongong  
alert(x);
```

```
x = 2016 + 1 + "Wollongong";     //2017Wollongong  
alert(x);
```

```
x = "Wollongong" + 2016;         //Wollongong2016  
alert(x);
```

```
x = "Wollongong" + 2016 + 1;    //Wollongong20161  
alert(x);
```

Basic JavaScript syntax

JavaScript data type: boolean

```
var authenticated = false;  
var isReturningUser = true;
```

```
var x = 5;  
var positive = (x > 0);      //true  
  
if(positive) {  
    alert("x is positive");  
}
```

Basic JavaScript syntax

Comparison and Logical Operators

`==` equal to

`!=` not equal

`>` greater than

`<` less than

`>=` greater than or equal to

`<=` less than or equal to

Basic JavaScript syntax

```
var x = 5;
```

```
var y = 6;
```

```
if(x == y) {
```

```
    alert("x and y are equal");
```

```
}else{
```

```
    alert("x and y NOT are equal");
```

```
}
```

```
var x = 5;
```

```
var y = 6;
```

```
if(x != y) {
```

```
    alert("x and y are not equal");
```

```
}else{
```

```
    alert("x and y are equal");
```

```
}
```

Basic JavaScript syntax

```
var mark = 75;  
  
if (mark > 85) {  
    alert("Grade A");  
}  
else if (mark > 65) {  
    alert("Grade B");  
}  
else if (mark > 50) {  
    alert("Grade C");  
}  
else {  
    alert("Grade D");  
}
```

For-Loop statement:

```
for (var i = 0; i < 5; i++) {  
    alert(i);  
}
```

Useful tags for dynamic content:

- The <div> tag defines a generic section container
- The tag defines a generic inline container

Change content by JavaScript

- **Step 1:** give the HTML element that we want to change an **ID**
- **Step 2:** use the function
`var e = document.getElementById("the-id");`
to get the HTML element that we want to change
- **Step 3:** change the content of the HTML element

for span, div, etc.:

```
e.innerHTML = "the-new-content";
```

for input text field:

```
e.value = "the-new-value";
```

for image:

```
e.src = "the-new-image-src";
```

Cat & Dog 1

The web page displays **2 buttons**: “Cat” and “Dog”.

If the user clicks the “Cat” button, a meao-meao message is displayed, and if the user clicks the “Dog” button, a woof-woof message is displayed.



Woof woof woof!



Meao meao meao!

Cat & Dog 1

```
<button onClick="cat () " >Cat</button>  
  
<button onClick="dog () " >Dog</button>  
  
<br /> <br />  
  
<span id="display"></span>
```



Cat & Dog 1

```
function dog() {  
    // get the span element  
  
    // show dog message  
  
}
```

Cat & Dog 1

```
function dog() {  
    // get the span element  
    var displaySpan = document.getElementById("display");  
  
    // show dog message  
}  
}
```



```
<span id="display"></span>
```



Cat & Dog 1

```
function dog() {  
    // get the span element  
    var displaySpan = document.getElementById("display");  
  
    // show dog message  
    displaySpan.innerHTML = "Woof woof woof!";  
}
```



Woof woof woof!

Cat & Dog 1

```
function cat() {  
    // get the span element  
    var displaySpan = document.getElementById("display");  
  
    // show cat message  
    displaySpan.innerHTML = "Meao meao meao!";  
}
```



Meao meao meao!



Change content by JavaScript

- **Step 1:** give the HTML element that we want to change an **ID**
- **Step 2:** use the function
`var e = document.getElementById("the-id");`
to get the HTML element that we want to change
- **Step 3:** change the content of the HTML element

for span, div, etc.:

```
e.innerHTML = "the-new-content";
```

for input text field:

```
e.value = "the-new-value";
```

for image:

```
e.src = "the-new-image-src";
```

Cat & Dog 2

The web page displays **2 buttons**: “Cat” and “Dog”, and a **text field**.

If the user clicks the “Cat” button, a meao-meao message is displayed inside a text field, and if the user clicks the “Dog” button, a woof-woof message is displayed in a text field.

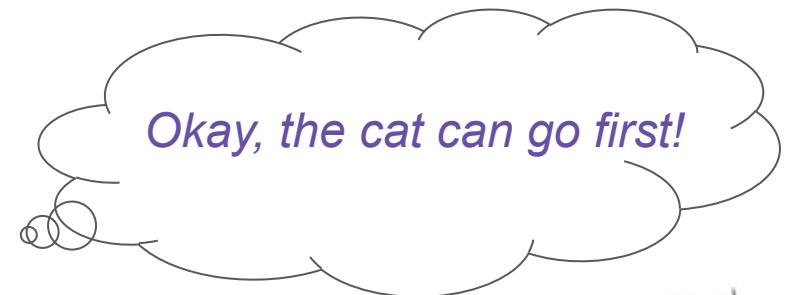
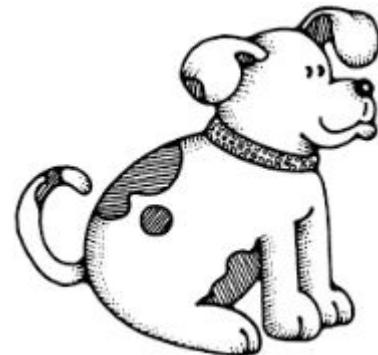
Cat & Dog 2

```
<button onClick="cat () " >Cat</button>  
  
<button onClick="dog () " >Dog</button>  
  
<br /> <br />  
  
<input type="text" id="display" />
```

Cat Dog

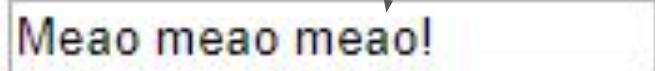
Cat & Dog 2

```
function cat() {  
    // get the text field element  
  
    // show cat message  
  
}
```



Cat & Dog 2

```
function cat() {  
    // get the text field element  
    var displayField = document.getElementById("display");  
  
    // show cat message  
    displayField.value = "Meao meao meao!";  
}
```



```
<input type="text" id="display" />
```

Cat & Dog 2

```
function dog() {  
    // get the text field element  
    var displayField = document.getElementById("display");  
  
    // show cat message  
    displayField.value = "Woof woof woof!";  
}
```



Woof woof woof!

```
<input type="text" id="display" />
```

Change content by JavaScript

- **Step 1:** give the HTML element that we want to change an **ID**
- **Step 2:** use the function
`var e = document.getElementById("the-id");`
to get the HTML element that we want to change
- **Step 3:** change the content of the HTML element

for span, div, etc.:

```
e.innerHTML = "the-new-content";
```

for input text field:

```
e.value = "the-new-value";
```

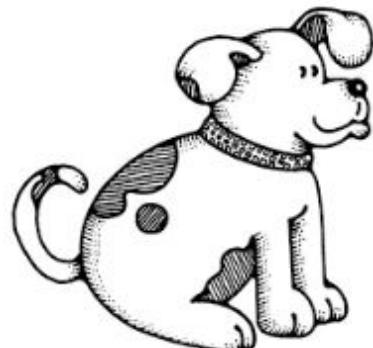
for image:

```
e.src = "the-new-image-src";
```

Cat & Dog 3

The web page displays **2 buttons**: “Cat” and “Dog”.

If the user clicks the “Cat” button, a cat picture is displayed, and if the user clicks the “Dog” button, a dog picture is displayed.



Cat & Dog 3

```
<button onClick="cat () " >Cat</button>  
  
<button onClick="dog () " >Dog</button>  
  
<br /> <br />  
  
<img id="display" />
```



(empty image: no src)

Cat & Dog 3

```
function cat() {  
    // get the image element  
  
    // show cat picture  
}
```

Cat & Dog 3

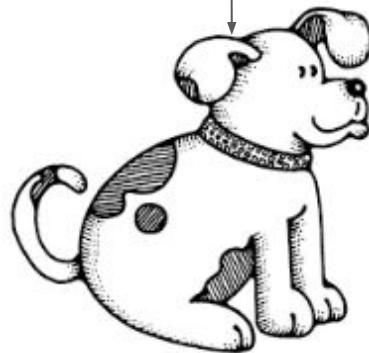
```
function cat() {  
    // get the image element  
    var image = document.getElementById("display");  
  
    // show cat picture  
    image.src = "cat.png";  
}
```



Cat & Dog 3

```
function dog() {  
    // get the image element  
    var image = document.getElementById("display");  
  
    // show dog picture  
    image.src = "dog.png";  
}
```

Cat Dog



```
<img id="display" />
```

Using variables to save state information

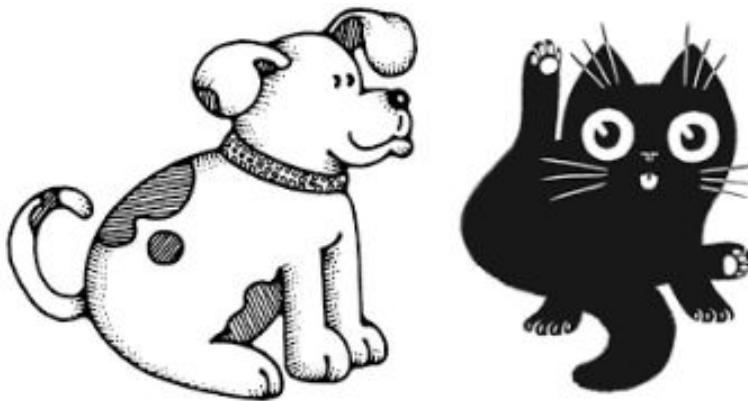
Sometime we use variables to save the **current status** of the page.

Cat & Dog 4

The web page displays **2 images**: “Cat” and “Dog”, and **2 click counters**.

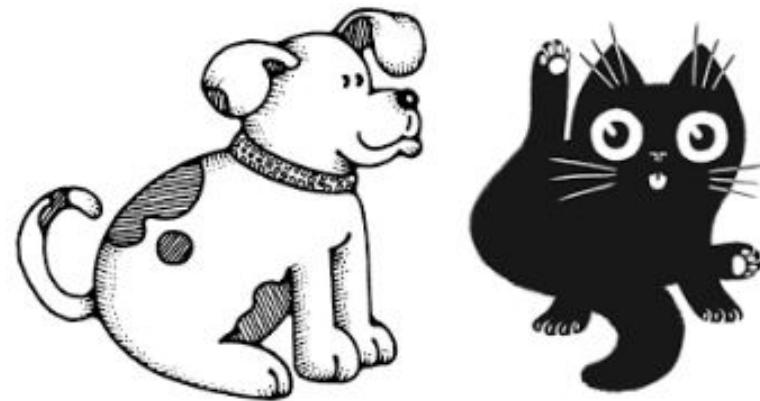
If the user clicks the “Cat” image, then the click counter for cat is increased.

If the user clicks the “Dog” image, then the click counter for dog is increased.



Dog click count: 0

Cat click count: 0



Dog click count: 3

Cat click count: 7

Cat & Dog 4

```

```

```

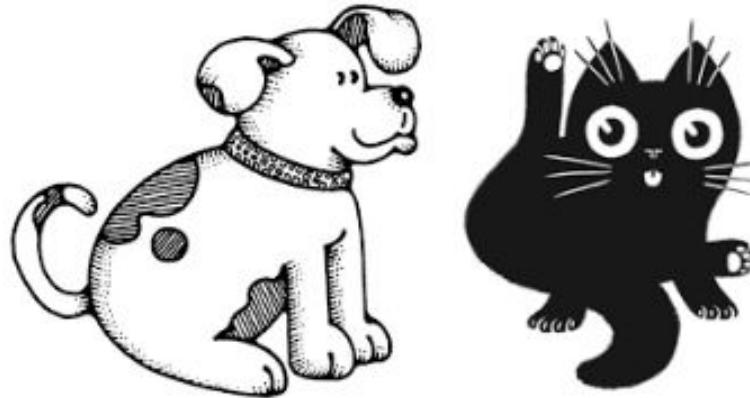
```

```
<br /> <br />
```

Dog click count: 0

```
<br /> <br />
```

Cat click count: 0



Dog click count: 0

Cat click count: 0

Cat & Dog 4

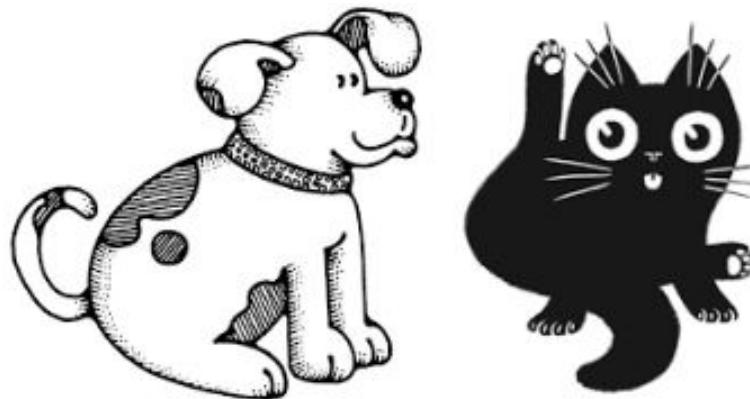
```
// variable to save the number of dog clicks
```

```
var dogClick = 0; _____
```

```
// variable to save the number of cat clicks
```

```
var catClick = 0; _____
```

We use **variables** to save the current number of **dog-clicks** and **cat-clicks**.

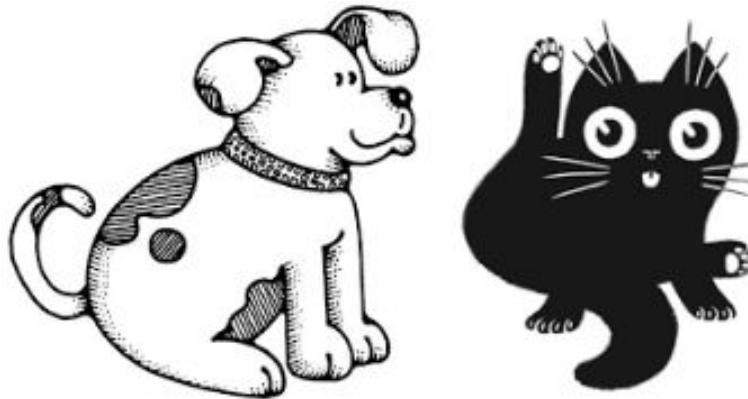


Dog click count: 0 ←

Cat click count: 0 ←

Cat & Dog 4

```
// variable to save the number of dog clicks  
var dogClick = 0;  
  
function dog() {  
  // increase the number of dog clicks by 1  
  
  // display the number of dog clicks  
}
```

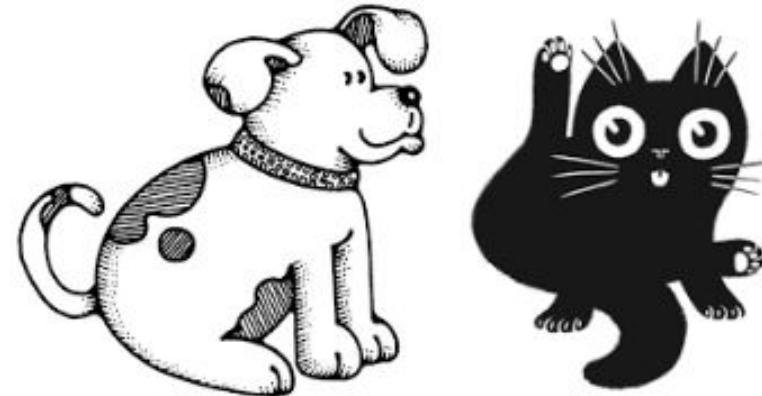


Dog click count: 0

Cat click count: 0

Cat & Dog 4

```
// variable to save the number of dog clicks  
var dogClick = 0;  
  
function dog() {  
    // increase the number of dog clicks by 1  
    dogClick = dogClick + 1;  
  
    // display the number of dog clicks  
    var dogSpan = document.getElementById("dogDisplay");  
    dogSpan.innerHTML = dogClick;  
}  
  
<span id="dogDisplay">0</span>
```



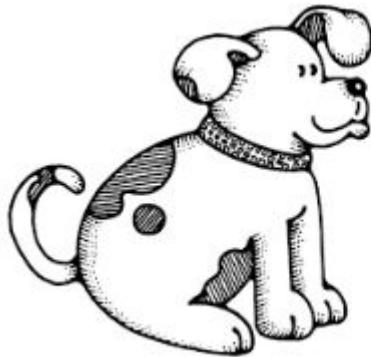
Dog click count: 0

Cat click count: 0

Cat & Dog 5

The web page displays **2 images**: “Dog” on the **left**, “Cat” on the **right**, and a **button** “Switch”.

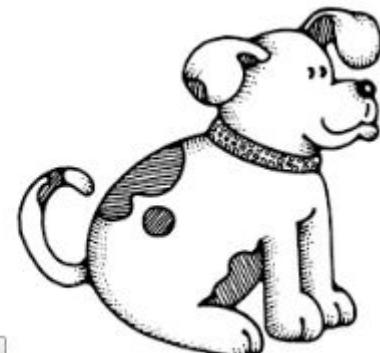
If the user clicks the “Switch” button, then the two images switch their places.



Switch



Switch



Cat & Dog 5

```
  
<button onClick="switchImage()">  
Switch  
</button>  

```

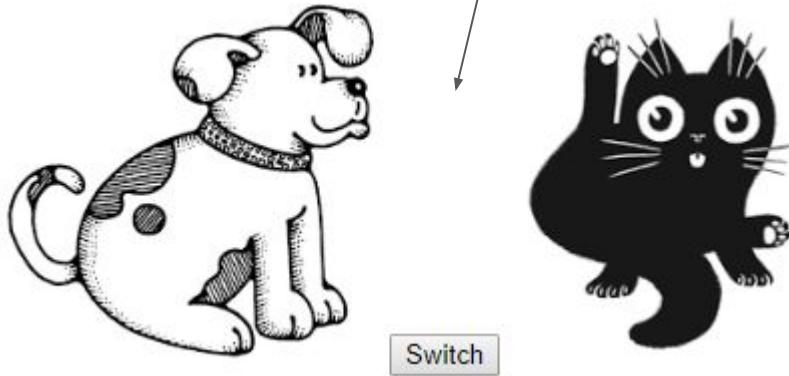


Switch

Cat & Dog 5

```
// variable to save the position of dog and cat images  
// two values: "dog-cat" or "cat-dog"  
// original position is "dog-cat"  
var position = "dog-cat";
```

We use a **variable** to save the current position of the images



Cat & Dog 5

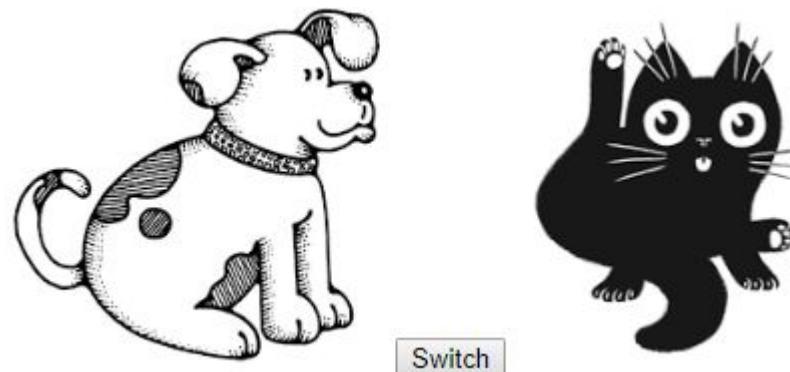
```
var position = "dog-cat";  
  
function switchImage() {  
    // check what is the current position, then switch it  
    // change position variable  
    // change the images  
  
}
```



Switch

Cat & Dog 5

```
if(position == "dog-cat") {  
    // change position variable  
position = "cat-dog";  
  
    // change the images  
  
    var leftImage = document.getElementById("left");  
leftImage.src = "cat.png";  
  
    var rightImage = document.getElementById("right");  
rightImage.src = "dog.png";  
  
}else...  
}
```



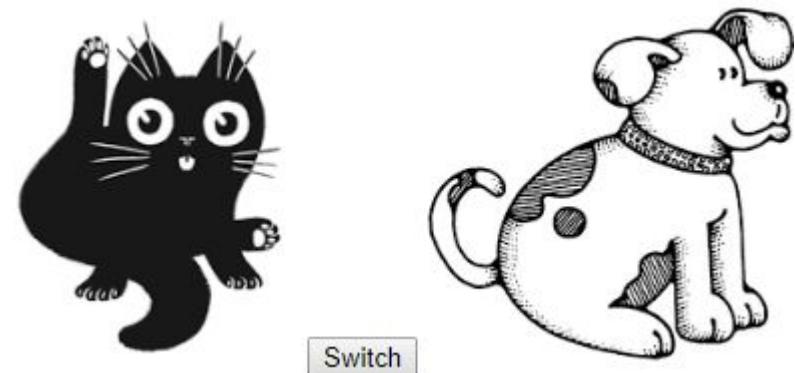
Current position is dog-cat

Cat & Dog 5

```
else{
    // change position variable
position = "dog-cat";

    // change the images
var leftImage = document.getElementById("left");
leftImage.src = "dog.png";

var rightImage = document.getElementById("right");
rightImage.src = "cat.png";
}
```



Current position is cat-dog

Cat & Dog 6

The web page displays a “Dog” picture.

If the user clicks the “Dog” picture, then it turns into a “Cat” picture.

If the user clicks the “Cat” picture, then it turns back to the “Dog” picture.



Cat & Dog 6

```

```



Cat & Dog 6

```
// variable to save the current displayed animal  
// two values: "dog" or "cat"  
// original value is "dog"  
var animal = "dog";
```

We use a **variable** to save the current displayed animal



Cat & Dog 6

```
var animal = "dog";  
  
function changeImage() {  
    // check what is the current animal, then change it  
    // change animal variable  
    // change the image  
  
}
```



Cat & Dog 6

```
if(animal == "dog") {  
    // change animal variable  
animal = "cat";  
  
    // change the image  
var image = document.getElementById("animal");  
image.src = "cat.png";  
}  
else...  
}
```



Current animal is dog

Cat & Dog 6

```
else{
    // change animal variable
animal = "dog";

    // change the image
var image = document.getElementById("animal");
image.src = "dog.png";
}
```



Current animal is cat

String

```
var text = "One Fish, Two Fish, Red Fish, Blue Fish";  
  
var textLength = text.length;  
    → 39  
  
var upper = text.toUpperCase();  
    → ONE FISH, TWO FISH, RED FISH, BLUE FISH  
  
var lower = text.toLowerCase();  
    → one fish, two fish, red fish, blue fish  
  
var fishIndex = text.indexOf("Fish");           → 4  
var catIndex = text.indexOf("cat");            → -1  
  
var redFound = text.includes("Red");          → true  
var greenFound = text.includes("Green");       → false
```

String

```
var text = "One Fish, Two Fish, Red Fish, Blue Fish";
```

```
var s1 = text.slice(10, 12);      → Tw
```

```
var s2 = text.slice(10);          → Two Fish, Red Fish, Blue Fish
```

```
var s3 = text.slice(-9, -6);     → Blu
```

```
var s4 = text.slice(-9);         → Blue Fish
```

Date

There are several ways to create a **Date** object.

```
var d = new Date(); //current date & time  
var d = new Date(millisec);  
var d = new Date(dateString);  
var d = new Date(year, month, day, hour, min, sec, millisec);
```

Date

```
var d = new Date(millisec);
```

Dates are calculated in milliseconds from 01 January, 1970 00:00:00 Universal Time (UTC). One day contains 86,400,000 millisecond.

```
var d = new Date(86400000);
alert(d);      //02 Jan 1970 00:00:00 UTC
```

Date

```
var d = new Date(dateString);  
  
//using YYYY-MM-DD format  
var d = new Date("2000-01-30");  
alert(d);  
  
//using YYYY-MM-DDTHH:MI:SS  
var d = new Date("2000-01-30T10:00:00");  
alert(d);
```

Date

```
var d = new Date(year, month, day, hour, min, sec, millisec);
```

The last 4 parameters can be omitted.

Months count from 0 to 11. January is 0. December is 11.

```
var d = new Date(2000, 0, 1);    // 01 Jan 2000  
alert(d);
```

Date

getDate()	Get the day as a number (1-31)
getDay()	Get the weekday as a number (0-6) <i>Sunday is 0, Saturday is 6</i>
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11) <i>January is 0, December is 11</i>
getSeconds()	Get the seconds (0-59)
getTime()	Get the milliseconds since 01/Jan/1970

Date

```
var now = new Date();  
alert("now is " + now);  
alert("getDate returns " + now.getDate());  
alert("getDay returns " + now.getDay());  
alert("getFullYear returns " + now.getFullYear());  
alert("getHours returns " + now.getHours());  
alert("getMilliseconds returns " + now.getMilliseconds());  
alert("getMinutes returns " + now.getMinutes());  
alert("getMonth returns " + now.getMonth());  
alert("getSeconds returns " + now.getSeconds());  
alert("getTime returns " + now.getTime());
```

Date

setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the milliseconds since 01/Jan/1970

Date

```
var now = new Date();  
alert(now);
```

```
var tomorrow = new Date();  
tomorrow.setDate(now.getDate() + 1);  
alert(tomorrow);
```

```
var hundredDayAgo = new Date();  
hundredDayAgo.setDate(now.getDate() - 100);  
alert(hundredDayAgo);
```

References

- <http://www.w3schools.com/js>
- <http://developer.mozilla.org/en-US/docs/Web/JavaScript>

Introduction to Web Technology

Web Form

Joseph Tonien

School of Computing and Information Technology
University of Wollongong

Web form

Objectives:

- design a web form;
- understand the web form HTML structure;
- use a web form to submit data to the server.

Form

- HTML forms are used to collect user input.
- HTML forms contain **form elements**.
- Different types of form elements: text field, textarea, checkboxes, radio buttons, option selection, submit buttons, ...

SOLS
including Moodle

USERNAME

PASSWORD

LOGIN

SOLS HELP

Search Train Timetables (T)

T1 North Shore & Northern Line Which line is my station on?

Berowra to City via Gordon, Hornsby to City via Macc

Leave after Arrive before

Today (Tue) 16 05

Remember me View timetable

Form

- When user clicks a submit button, user input will be sent to the server to process



- The front-end form elements must match up with the server back-end program's parameters.

Example: Whosville's library book search

The server back-end program is running at: <http://library.whosville/bsearch>

The search engine accepts the following parameters:

- **author**: this parameter is to specify the author of the book.
- **year**: this parameter is to specify the publication year.
- **sub**: this parameter is to specify the subjects for the search, the search engine can accept **multiple** values. The acceptable values are:
 - **mth** (for mathematics),
 - **cs** (for computer science),
 - **bio** (for biology),
 - **phy** (for physics), and
 - **chem** (for chemistry).

Example: Whosville's library book search

The server back-end program is running at: <http://library.whosville/bsearch>

The search engine accepts the following parameters:

- **author**: this parameter is to specify the author of the book.
- **year**: this parameter is to specify the publication year.
- **sub**: this parameter is to specify the subjects for the search, the search engine can accept **multiple** values. The acceptable values are:
 - **mth** (for mathematics),
 - **cs** (for computer science),
 - **bio** (for biology),
 - **phy** (for physics), and
 - **chem** (for chemistry).

Whosville library book search

Author name:

Publication year:

Subject:

Mathematics
 Computer Science
 Biology
 Physics
 Chemistry

Example: Whosville's library book search

Whosville library book search

Author name:

Publication year:

Subject:

- Mathematics
- Computer Science
- Biology
- Physics
- Chemistry

When the form is submitted, we can see the following appears in the URL:

`http://library.whosville/bsearch?author=tonien&year=2000&sub=cs&sub=phy`

form's action

match up with the back-end
program's parameters

Example: Whosville's library book search

```
<form action="http://library.whosville/bsearch" method="get">
```

Whosville library book search

Author name: ← <input type="text" name="author"/>

Publication year: ← <input type="text" name="year"/>

Subject:

Mathematics
 Computer Science
 Biology
 Physics
 Chemistry

```
<input type="checkbox" name="sub" value="mth"/>  
<input type="checkbox" name="sub" value="cs"/>  
<input type="checkbox" name="sub" value="bio"/>  
<input type="checkbox" name="sub" value="phy"/>  
<input type="checkbox" name="sub" value="chem"/>
```

The front-end form elements must match up with the server back-end program's parameters:

- Use the attribute **name** to specify the back-end's parameter

Example: Whosville's library book search

```
<form action="http://library.whosville/bsearch" method="get">
```

Whosville library book search

Author name: ← <input type="text" name="author"/>
Publication year: ← <input type="text" name="year"/>

Subject:

- Mathematics
- Computer Science
- Biology
- Physics
- Chemistry

```
<input type="checkbox" name="sub" value="mth"/>
<input type="checkbox" name="sub" value="cs"/>
<input type="checkbox" name="sub" value="bio"/>
<input type="checkbox" name="sub" value="phy"/>
<input type="checkbox" name="sub" value="chem"/>
```

The front-end form elements must match up with the server back-end program's parameters:

- Use the attribute **value** to specify the data sent back to the back-end

Example: Whosville's library book search

```
<form action="http://library.whosville/bsearch" method="get">
```

Whosville library book search

Author name: ← <input type="text" name="author"/>

Publication year: ← <input type="text" name="year"/>

Subject:

Mathematics

Computer Science

Biology

Physics

Chemistry

```
<input type="checkbox" name="sub" value="mth"/>
<input type="checkbox" name="sub" value="cs"/>
<input type="checkbox" name="sub" value="bio"/>
<input type="checkbox" name="sub" value="phy"/>
<input type="checkbox" name="sub" value="chem"/>
```

The front-end form elements must match up with the server back-end program's parameters:

- Sometimes the data sent back to the back-end is **different** from the data displayed to the user.

Class discussion

Go to some webforms on the Internet:

- Look at the source code of the form
- Look at the attribute **name** used to specified the back-end parameters
- Submit the form and look at data displayed on the URL

<http://www.ebay.com/>

<http://www.amazon.com.au>

Form - type text

First name:

<input type="text" name="firstname" size="30"/>

Last name:

<input type="text" name="lastname" size="30"/>

First name:

Last name:

Form - type password

Username:


```
<input type="text" name="username" size="30"/><br />
```

Password:


```
<input type="password" name="password" size="30"/><br />
```

Username:

Password:

Form - type checkbox

This is the value that sent to server

Choose journals to subscribe:

<input type="checkbox" name="journal" value="AMM"/>American
Mathematical Monthly

<input type="checkbox" name="journal" value="CMJ"/>College
Mathematics Journal

<input type="checkbox" name="journal" value="MM"/>Mathematics
Magazine

This is the value that get displayed

Choose journals to subscribe:

- American Mathematical Monthly
- College Mathematics Journal
- Mathematics Magazine

Form - type checkbox

Choose subscription method:


```
<input type="checkbox" name="subscription"  
value="e" checked="checked" />eJournal <br />
```

```
<input type="checkbox" name="subscription"  
value="paper" />Hard copy
```

This should be pre-selected (checked)
when the page loads

Choose subscription method:
 eJournal
 Hard copy

Form - type radio

This is the value that sent to server

Select student type:


```
<input type="radio" name="studentType" value="u"/>Undergraduate  
<input type="radio" name="studentType" value="p"/>Postgraduate  
<input type="radio" name="studentType" value="other"/>Other
```

This is the value that get displayed

Select student type:

- Undergraduate
- Postgraduate
- Other

Form - type radio

Select student type:


```
<input type="radio" name="studentType" value="u"/>Undergraduate
```

```
<input type="radio" name="studentType" value="p"/>Postgraduate
```

```
<input type="radio" name="studentType"  
      value="other" checked="checked"/>Other
```

This should be pre-selected (checked)
when the page loads

Select student type:

- Undergraduate
- Postgraduate
- Other

Form - select

This is the value that sent to server

Select day:


```
<select name="day">
  <option value="mon">Monday</option>
  <option value="tue">Tuesday</option>
  <option value="wed">Wednesday</option>
  <option value="thu">Thursday</option>
  <option value="fri">Friday</option>
</select>
```

This is the value that get displayed

Select day:

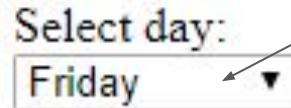
Monday ▾

Form - select

Select day:


```
<select name="day">
  <option value="mon">Monday</option>
  <option value="tue">Tuesday</option>
  <option value="wed">Wednesday</option>
  <option value="thu">Thursday</option>
  <option value="fri" selected="selected">Friday</option>
</select>
```

This should be pre-selected
when the page loads



Form - select multiple

Select day:
/ >

```
<select name="day" multiple>
  <option value="mon">Monday</option>
  <option value="tue">Tuesday</option>
  <option value="wed">Wednesday</option>
  <option value="thu">Thursday</option>
  <option value="fri">Friday</option>
</select>
```



Form - textarea

Enter your comment:
/br />

```
<textarea name="comment" rows="5" cols="30">  
</textarea>
```

Enter your comment:

Form - submit

this is the program in the server that processes the form

```
<form action="handle_login" method="post">
    Username:<br />
    <input type="text" name="username" size="30"/><br />
    Password:<br />
    <input type="password" name="password" size="30"/><br /><br />

    <input type="submit" value="Login"/>
    <input type="reset" value="Reset form"/>

</form>
```

Username:

Password:

Login

Reset form

Form method

```
<form action="handle_login" method="post">
```

	method="get"	method="post"
Visibility	Data is visible in the URL	Data is not displayed in the URL
History	Parameters remain in the browser history	Parameters are not saved in browser history
Bookmarked	Can be bookmarked	Cannot be bookmarked
Security	get is less secure compared to post because data sent in part of the URL. Never use get when sending passwords or other sensitive information.	
Back button/ Reload	Harmless	Data will be re-submitted. Browser should alert the user about resubmission.

Form method

```
<form action="handle_login" method="post">
```

	method="get"	method="post"
Restrictions on data length	Yes, when sending data, the <code>get</code> method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed.

Form - reset

```
<form action="handle_login" method="post">
    Username:<br />
    <input type="text" name="username" size="30"/><br />
    Password:<br />
    <input type="password" name="password" size="30"/><br /><br />

    <input type="submit" value="Login"/>
    <input type="reset" value="Reset form"/>

</form>
```

When reset button is clicked
all input in the form will be
cleared

Username:

Password:

Form

```
<form ...>
    First name:<br />
    <input type="text" name="firstname" size="30"/><br />
    Last name:<br />
    <input type="text" name="lastname" size="30"/><br />

    Select day:<br />
    <select name="day">
        <option value="mon">Monday</option>
        <option value="tue">Tuesday</option>
        <option value="wed">Wednesday</option>
        <option value="thu">Thursday</option>
        <option value="fri">Friday</option>
    </select>
    ...
</form>
```

**Finally, an important question:
What is the purpose of the attribute `name` ?**

References

- <http://www.w3schools.com/html>
- http://www.w3schools.com/html/html_forms.asp
- <http://www.w3.org/TR/html5/forms.html>

Introduction to Web Technology

Dynamic Documents with Javascript, Animation

Joseph Tonien

School of Computing and Information Technology
University of Wollongong

Objectives:

- use JavaScript to build website with dynamic content;
- use JavaScript to generate animation on your web site.

REVIEW: Change content by JavaScript

- **Step 1:** give the HTML element that we want to change an **ID**
- **Step 2:** use the function
`var e = document.getElementById("the-id");`
to get the HTML element that we want to change
- **Step 3:** change the content of the HTML element

for span, div, etc.:

```
e.innerHTML = "the-new-content";
```

for input text field:

```
e.value = "the-new-value";
```

for image:

```
e.src = "the-new-image-src";
```

Useful tags for dynamic content:

- The <div> tag defines a generic section container
- The tag defines a generic inline container

Say Hi 1

The web page displays **2 text fields**: *first name* and *last name*, and a **button** "Say Hi". If the user enters John in the first name text field and Smith in the last name text field, and clicks the "Say Hi" button, then a greeting message is displayed on the page: "*Hi John Smith!*".

First name:

Last name:

Hi John Smith!

Say Hi 1

```
First name: <input type="text" id="firstname" /> <br /> —  
Last name: <input type="text" id="lastname" /> <br /> —  
<button onClick="sayHi()">Say Hi</button> <br /> —  
<span id="greeting"></span> ——————
```

First name:

Last name:

Say Hi

Hi John Smith!

Say Hi 1

```
function sayHi () {  
    // get the first name  
  
    // get the last name  
  
    // construct the greeting message  
  
    // display the greeting message  
}
```

Say Hi 1

```
// get the first name  
  
var firstnameInput = document.getElementById("firstname");  
  
var firstname = firstnameInput.value;
```

```
<input type="text" id="firstname" />
```

First name: John

Last name: Smith

Say Hi

Hi John Smith!

Say Hi 1

```
// get the last name  
  
var lastnameInput = document.getElementById("lastname");  
  
var lastname = lastnameInput.value;
```

```
<input type="text" id="lastname" />
```

The diagram illustrates the state of the DOM after the JavaScript code has run. It shows an input field with the ID "lastname" containing the value "Smith". Two arrows point from the variable names in the code to their corresponding elements in the UI: one arrow points from "lastnameInput" to the input field, and another points from "lastname" to the value "Smith" within the input field.

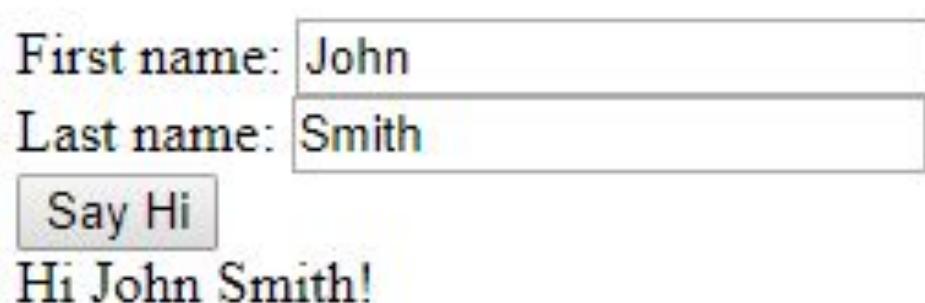
First name:	John
Last name:	Smith

Say Hi

Hi John Smith!

Say Hi 1

```
// get the first name  
  
var firstnameInput = document.getElementById("firstname");  
var firstname = firstnameInput.value;  
  
// get the last name  
  
var lastnameInput = document.getElementById("lastname");  
var lastname = lastnameInput.value;  
  
// construct the greeting message  
  
var greetingMessage = "Hi " + firstname + " " + lastname + "!";
```

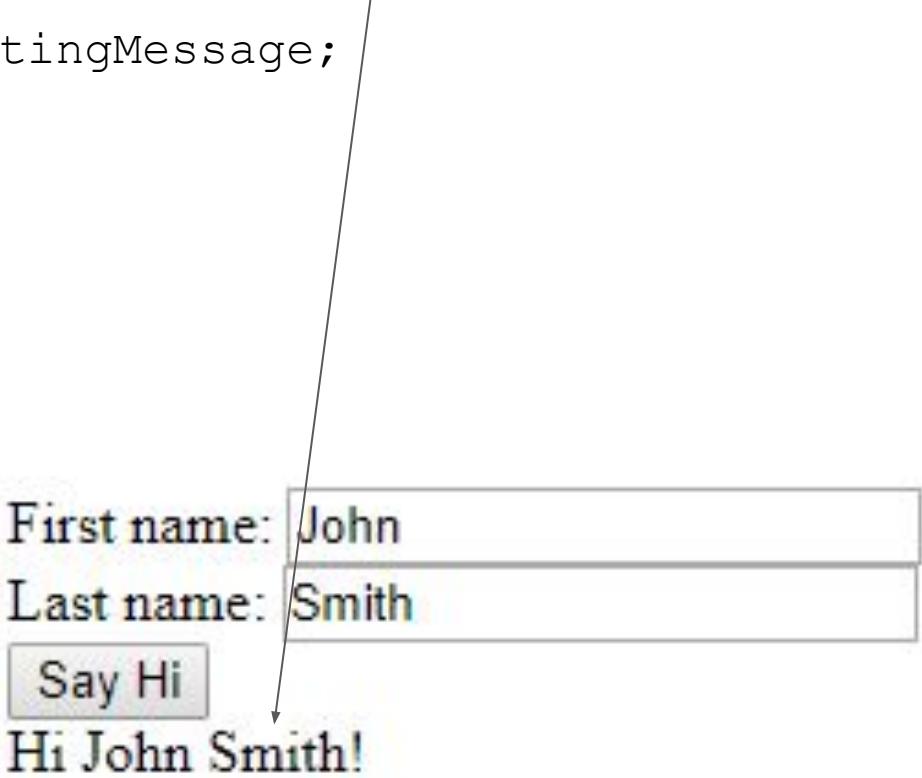


The screenshot shows a simple web interface for greeting someone. At the top, there are two input fields: 'First name:' containing 'John' and 'Last name:' containing 'Smith'. Below these fields is a button labeled 'Say Hi'. Underneath the button, the resulting greeting 'Hi John Smith!' is displayed.

First name:	John
Last name:	Smith
Say Hi	
Hi John Smith!	

Say Hi 1

```
// construct the greeting message  
  
var greetingMessage = "Hi " + firstname + " " + lastname + "!";  
  
// display the greeting message  
  
var greetingSpan = document.getElementById("greeting");  
greetingSpan.innerHTML = greetingMessage;
```



The diagram illustrates the flow of data from user input to output. It shows two input fields: 'First name:' containing 'John' and 'Last name:' containing 'Smith'. A button labeled 'Say Hi' is positioned below them. An arrow points from the 'Say Hi' button to a span element with the id 'greeting'. This span contains the text 'Hi John Smith!', which is the result of concatenating the first and last names.

```
<span id="greeting"></span>
```

Say Hi 2

Adding a button “Clear”.

If the user clicks the “Clear” button then the texts in the input fields and the greeting message are removed.

First name:

Last name:

Say Hi 2

```
First name: <input type="text" id="firstname" /> <br /><br />
Last name: <input type="text" id="lastname" /> <br /><br />
<button onClick="sayHi ()">Say Hi</button>
<button onClick="clearPage ()">Clear</button> <br /><br />
<span id="greeting"></span>
```

First name:

Last name:

Say Hi 2

```
function clearPage() {  
    // clear the firstname text field  
  
    // clear the lastname text field  
  
    // clear the greeting message  
}
```

First name:

Last name:

Say Hi 2

```
// clear the firstname text field  
  
var firstnameInput = document.getElementById("firstname");  
firstnameInput.value = "";  
  
// clear the lastname text field  
  
var lastnameInput = document.getElementById("lastname");  
lastnameInput.value = "";  
  
// clear the greeting message  
  
var greetingSpan = document.getElementById("greeting");  
greetingSpan.innerHTML = "";
```

First name:

Last name:

¹⁵

Math Question

The user enters 2 numbers into the two text fields, then selects the operation (+, - , x) from the drop-down list, then clicks the “=” button, then answer will be generated.

3	+ ▾	5	=	8	16
---	-----	---	---	---	----

Math Question

```
<input type="text" id="input1" />  
  
<select id="operationSelect">  
  <option value="add">+</option>  
  <option value="subtract">-</option>  
  <option value="multiply">x</option>  
</select>  
  
<input type="text" id="input2" />  
  
<button onClick="answer()">  
 =  
</button>  
  
<input type="text" id="result" />
```

3

+ ▾

5

=

8

Math Question

```
function answer() {  
    // get the 1st number  
    // get the 2nd number  
    // get the operation  
    // calculate the result  
    // display the result  
}
```

3

+ ▾

5

=

8

Math Question

```
// get the 1st number  
  
var inputField1 = document.getElementById("input1");  
var number1 = Number(inputField1.value);  
  
  
// get the 2nd number  
  
var inputField2 = document.getElementById("input2");  
var number2 = Number(inputField2.value);  
  
  
// get the operation  
  
var operationSelect = document.getElementById("operationSelect");  
var operation = operationSelect.value;
```

3	+ ▾	5	=	8	19
---	-----	---	---	---	----

Math Question

```
// calculate the result  
  
var result;  
  
if(operation == "add") {  
  
    result = number1 + number2;  
  
}else if(operation == "subtract") {  
  
    result = number1 - number2;  
  
}else if(operation == "multiply") {  
  
    result = number1 * number2;  
  
}  
  
  
// display the result  
  
var resultField = document.getElementById("result");  
resultField.value = result;
```

3

+ ▾

5

=

8

Change style

```
<button onClick="changeHelloWorldStyle();">  
Click me to change the style of the text  
</button>  
  
<span id="hello">Hello world</span>  
  
<script>  
  
function changeHelloWorldStyle() {  
  
    var helloSpan = document.getElementById("hello");  
  
    helloSpan.style.color = "orange";  
  
    helloSpan.style.fontSize = "30px";  
  
    helloSpan.style.fontStyle = "italic";  
  
}  
  
</script>
```

Click me to change the style of the text Hello world

Click me to change the style of the text

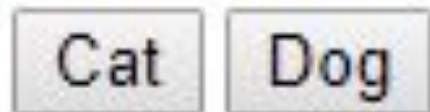
Hello world

Passing parameters to function

Sometimes, it is useful to pass **parameters** to the **function**

Cat & Dog 1

```
<button onClick="cat()">Cat</button>
<button onClick="dog()">Dog</button>
<br /> <br />
<span id="display"></span>
```



Old solution: using two functions

Woof woof woof!

New solution: using one function with parameter

Cat & Dog 1B

```
<button onClick="showMessage('Meao meao meao! ')>Cat</button>  
<button onClick="showMessage('Woof woof woof! ')>Dog</button>  
<br /> <br />  
<span id="display"></span>
```



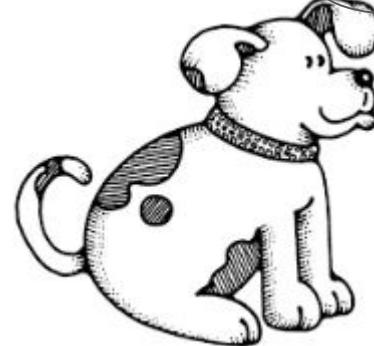
Woof woof woof!

Cat & Dog 1B

```
function showMessage(message) {  
    // get the span element  
  
    var displaySpan = document.getElementById("display");  
  
    // show the message  
    displaySpan.innerHTML = message;  
}
```



Woof woof woof!



Cat & Dog 2B

```
<button onClick="showMessage('Meao meao meao! ')>Cat</button>
<button onClick="showMessage('Woof woof woof! ')>Dog</button>
<br /> <br />

<input type="text" id="display" />

function showMessage(message) {
    // get the text field element
    var displayField = document.getElementById("display");

    // show the message
    displayField.value = message;
}
```

Cat Dog

Woof woof woof!

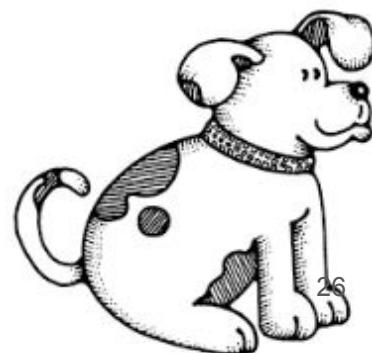
Cat & Dog 3B

```
<button onClick="showImage('cat.png')">Cat</button>
<button onClick="showImage('dog.png')">Dog</button>
<br /> <br />
<img id="display" />

function showImage(imageFile) {
    // get the image element
    var image = document.getElementById("display");

    // show the animal picture
    image.src = imageFile;
}
```

Cat Dog

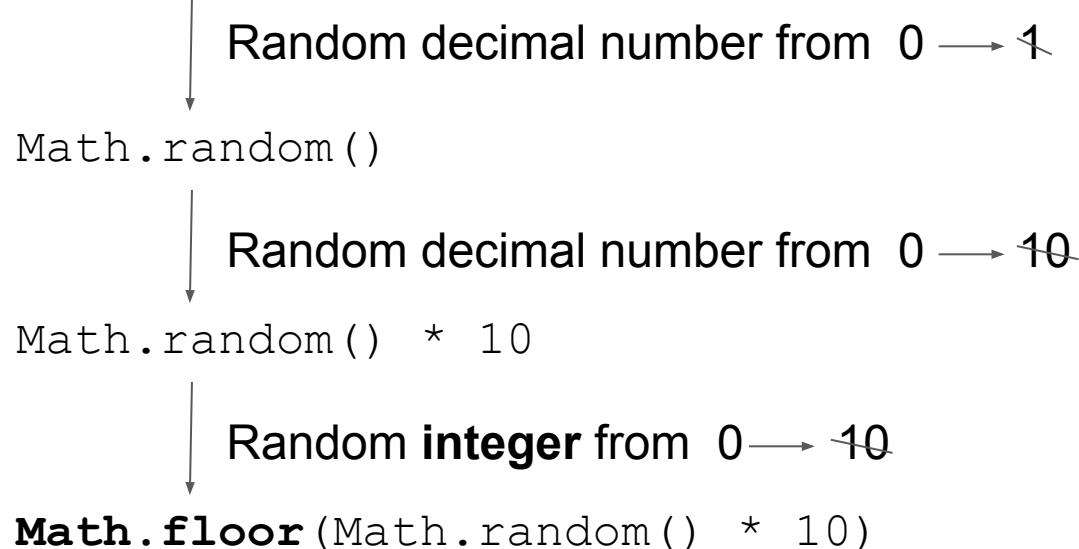


Random

To get a random (decimal) number from 0 (**inclusive**) to 1 (**exclusive**):

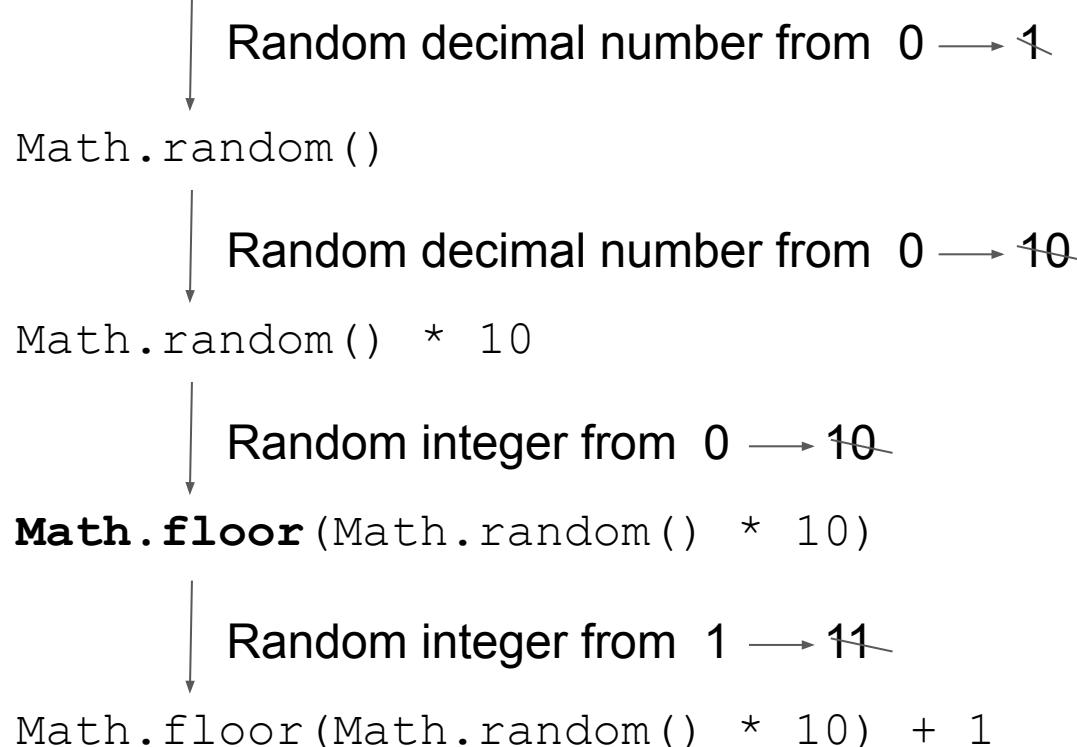
```
var x = Math.random(); // random decimal number 0 → 1
```

Random



```
//generate a random integer: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
var x = Math.floor(Math.random() * 10);
```

Random



```
//generate a random integer: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
var x = Math.floor(Math.random() * 10) + 1;
```

Dice 1

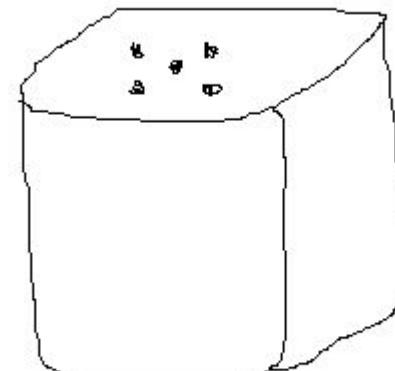
When the button is clicked, a random dice is displayed.

Generate a random integer from 1 to 6:

```
Math.floor(Math.random() * 6) + 1
```

Roll the dice

Roll the dice



Dice 1

```
<btton onClick="rollDice()">
```

Roll the dice

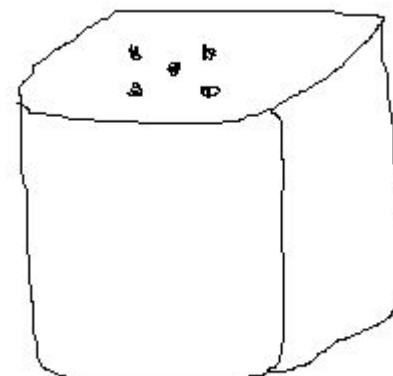
```
</button>
```

```
<br />
```

```
<img id="dice" />
```

Roll the dice

Roll the dice

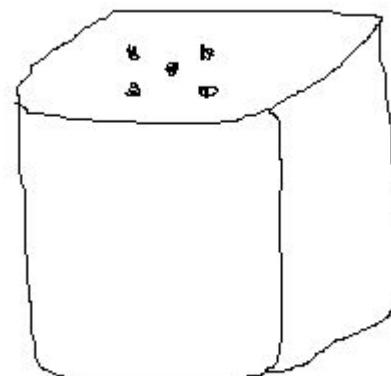


Dice 1

```
function rollDice() {  
    // generate a random dice value from 1 to 6  
    var diceValue = Math.floor(Math.random() * 6) + 1;  
  
    // get image file name for this dice value  
    var imageFile = "dice" + diceValue + ".png";  
  
    // show the image  
    var diceImage = document.getElementById("dice");  
    diceImage.src = imageFile;  
}
```

Roll the dice

dice5.png



Confirm box

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel".

If the user clicks "OK", the box returns true.

If the user clicks "Cancel", the box returns false.

```
var ok = confirm("Do you want to proceed with the order?");  
if(ok){  
    alert("User clicked OK");  
}else{  
    alert("User clicked Cancel.");  
}
```

Prompt box

When a prompt box pops up, the user will have to click either "OK" or "Cancel".

If the user clicks "OK" the box returns the input value.

If the user clicks "Cancel" the box returns null.

We can also specify the default text in the input box:

```
prompt("sometext", "defaultText");
```

```
var name = prompt("Please enter your name", "cat in the hat");
if(name != null) {
    alert("Hello " + name);
}
```

Add subject 1

When the button is clicked, a prompt box appears asking the user to enter a subject code. Then the subject is added to the page.

[Click here to add subject](#)

MATH 111

CSCI 230

Add subject 1

```
<button onClick="addSubject()">  
Click here to add subject  
</button>  
  
<div id="subjectList">  
</div>
```

Click here to add subject

MATH 111

CSCI 230

Add subject 1

```
function addSubject() {  
    // ask user for a subject code  
    var subject = prompt("Enter subject code");  
  
    if(subject != null) {  
        // create a new paragraph holding the subject code  
        var para = document.createElement("p");  
        var subjectText = document.createTextNode(subject);  
        para.appendChild(subjectText);  
  
        // add the new paragraph element to the subject div  
        var subjectDiv = document.getElementById("subjectList");  
        subjectDiv.appendChild(para);  
    }  
}
```

[Click here to add subject](#)

```
<div id="subjectList">  
    <p>MATH 111</p>  
</div>
```

MATH 111

CSCI 230

Add subject 2

When the button is clicked, a prompt box appears asking the user to enter a subject code. Then the subject is added to the page in an **unordered list**.

[Click here to add subject](#)

- MATH 111
- CSCI 230

Add subject 2

```
<button onClick="addSubject()">  
Click here to add subject  
</button>  
  
<ul id="subjectList">  
</ul>
```

Click here to add subject

- MATH 111
- CSCI 230

Add subject 2

```
function addSubject() {  
    // ask user for a subject code  
    var subject = prompt("Enter subject code");  
  
    if(subject != null) {  
        // create a new list item holding the subject code  
        var li = document.createElement("li");  
        var subjectText = document.createTextNode(subject);  
        li.appendChild(subjectText);  
  
        // add the new list item element to the unordered list  
        var subjectUL = document.getElementById("subjectList");  
        subjectUL.appendChild(li);  
    }  
}
```

[Click here to add subject](#)

- MATH 111
- CSCI 230

Animation

Start an animation

```
var animationSchedule = setInterval(animationFunction, milisecs);
```

specify 2 things:

- ***what need to be done***: write a function to do the task
- ***how often***: how often this task need to be done

Stop the animation

```
clearInterval(animationSchedule);
```

Counter animation

When the “Start Counter” button is clicked, the counter animation displays a number increasing every 1 second.

When the “Stop Counter” button is clicked, the number is stop increasing.

```
initially counter = 0
```

```
start animation
```

1000 milisec	showCounter()	counter = 1
1000 milisec	showCounter()	counter = 2
1000 milisec	showCounter()	counter = 3
1000 milisec	showCounter()	counter = 4
1000 milisec	showCounter()	counter = 5

```
.....
```

Start counter

Stop counter

```
stop animation
```

Counter animation

```
<button onClick="startCounterAnimation()">  
Start counter  
</button>  
  
<button onClick="stopCounterAnimation()">  
Stop counter  
</button>  
  
<br /><br />  
  
<font size="7">  
<span id="counter"></span>  
</font>
```

Start counter Stop counter

Counter animation

```
var counter = 0;  
  
var counterSchedule;  
  
function startCounterAnimation() {  
  
    // start the counter animation  
    counterSchedule = setInterval(showCounter, 1000);  
  
}
```

Start an animation

- **what need to be done:** write a function to do the task
- **how often:** how often this task need to be done

[Start counter](#)

[Stop counter](#)

Counter animation

what need to be done:

write a function to do the task of the animation

```
function showCounter() {  
  
    // increase the counter by 1  
    counter = counter + 1;  
  
    // show the counter  
    var counterSpan = document.getElementById("counter");  
    counterSpan.innerHTML = counter;  
  
}
```

Counter animation

```
function stopCounterAnimation() {  
    clearInterval(counterSchedule);  
}  
});
```

Start counter **Stop counter**

Dice 2 - animation

When the button is clicked, within 1 second, a flash of 10 random dice images are displayed, and then it stops.

start animation

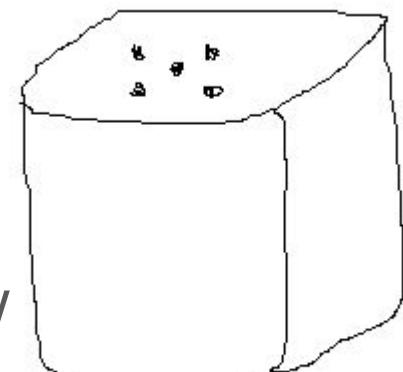
100 milisec	rollDice()	counter = 0
100 milisec	rollDice()	counter = 1
100 milisec	rollDice()	counter = 2
100 milisec	rollDice()	counter = 3
100 milisec	rollDice()	counter = 4
100 milisec	rollDice()	counter = 5
100 milisec	rollDice()	counter = 6
100 milisec	rollDice()	counter = 7
100 milisec	rollDice()	counter = 8
100 milisec	rollDice()	counter = 9
100 milisec	rollDice()	counter = 10

stop animation

Roll the dice



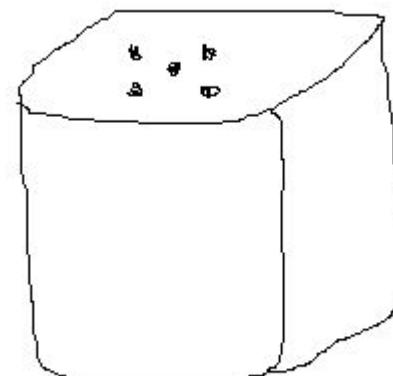
Use a **counter variable** to know when to stop the animation



Dice 2 - animation

```
<button onClick="rollDiceAnimation()">  
  Roll the dice  
</button>  
  
<br />  
  
<img id="dice" />
```

Roll the dice



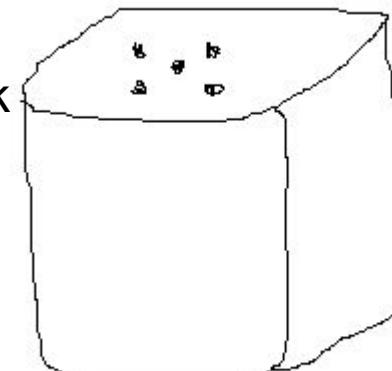
Dice 2 - animation

```
var rollDiceSchedule;  
var rollDiceCounter;  
  
function rollDiceAnimation() {  
    // set the roll dice counter to 0  
    rollDiceCounter = 0;  
  
    // start the roll dice animation  
    rollDiceSchedule = setInterval(rollDice, 100);  
}  
}
```

Roll the dice

Start an animation

- **what need to be done**: write a function to do the task
- **how often**: how often this task need to be done

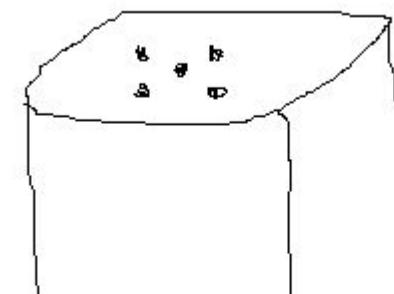


what need to be done:

write a function to do the task of the animation

Dice 2 - animation

```
function rollDice() {  
    // generate a random dice value from 1 to 6  
    var diceValue = Math.floor(Math.random() * 6) + 1;  
  
    // get image file name for this dice value  
    var imageFile = "dice" + diceValue + ".png";  
  
    // show the image  
    var diceImage = document.getElementById("dice");  
    diceImage.src = imageFile;  
  
    // increase the roll dice counter  
    rollDiceCounter = rollDiceCounter + 1;  
  
    // if the roll dice counter reaches 10 then stop the animation  
    if(rollDiceCounter == 10){  
        clearInterval(rollDiceSchedule);  
    }  
}
```



References

- <http://www.w3schools.com/js>
- <http://developer.mozilla.org/en-US/docs/Web/JavaScript>

Introduction to Web Technology

XML and DTD

Joseph Tonien

School of Computing and Information Technology
University of Wollongong

XML and DTD

Objectives:

- use XML to store and transport data over the Internet
- learn DTD language to define the structure of an XML document

XML

EXtensible Markup Language

- XML is a markup language much like HTML
- XML is a software- and hardware-independent tool for storing and transporting data.
- XML separates data from presentation.
- File extension is .xml

```
<?xml version="1.0" ?>
<student>
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <email>jsmith@gmail.com</email>
    <mobile>0211223344</mobile>
</student>
```

XML

- HTML tags are predefined.
- XML tags are defined by user.
- Using **XML Document Type Definition (DTD)**, or **XML Schema Definition (XSD)**, different parties can agree on a standard XML format for interchanging data.
- Another popular format for interchanging data is **JavaScript Object Notation (JSON)**

```
{  
  "firstName": "John",  
  "lastName": "Smith",  
  "email": "jsmith@gmail.com",  
  "mobile": "0211223344"  
}
```

- In most web applications, XML and JSON are used to store or transport data, while HTML and XSLT are used to transform and display the data.

XML:

The first example of XML:

```
<?xml version="1.0" ?>
<student>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <email>jsmith@gmail.com</email>
  <mobile>0211223344</mobile>
</student>
```

XML: XML declaration

```
<?xml version="1.0" ?> ← XML declaration  
<student>  
  <firstName>John</firstName>  
  <lastName>Smith</lastName>  
  <email>jsmith@gmail.com</email>  
  <mobile>0211223344</mobile>  
</student>
```

- The **XML declaration** is optional and it **must come first in the document**.
- The XML declaration identifies the document as being XML. Even though it is optional, all XML documents should begin with an XML declaration.
- The XML declaration must be situated at the first position of the first line in the XML document.
 - **Do not start an XML file with a blank line!!!**
- Syntax for the XML declaration:

```
<?xml version="version_number"  
encoding="encoding_declaration"  
standalone="standalone_status" ?>
```

XML: root element

```
<?xml version="1.0" encoding="UTF-8" ?>
<student> ← root element
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <email>jsmith@gmail.com</email>
  <mobile>0211223344</mobile>
</student>
```

- An XML document **must contain one root element** that is the parent of all other elements

```
<rootElement>
  <child>
    <subchild>.....</subchild>
  </child>
</rootElement>
```

XML: root element

This is NOT a well-formed XML document because it has no root element

```
<?xml version="1.0" encoding="UTF-8"?>
<student>
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <email>jsmith@gmail.com</email>
</student>
<student>
    <firstName>Mary</firstName>
    <lastName>Jane</lastName>
    <email>mjane@gmail.com</email>
</student>
```

XML: root element

This is a well-formed XML document because it has a root element

```
<?xml version="1.0" encoding="UTF-8"?>
<b><studentList></b>
<student>
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <email>jsmith@gmail.com</email>
</student>
<student>
    <firstName>Mary</firstName>
    <lastName>Jane</lastName>
    <email>mjane@gmail.com</email>
</student>
</b></studentList>
```

XML: element

```
<tag attribute1="..." attribute2="...">  
  </tag>
```

- An **XML element** is everything from (including) the element's start tag to (including) the element's end tag.

```
<?xml version="1.0" encoding="UTF-8"?>  
<dailyTransaction date="24/02/2015">  
  <person staffDbId="103" operation="update">  
    <firstName>John</firstName>  
    <lastName>Smith</lastName>  
    <mobile>0211223344</mobile>  
  </person>  
  <person staffDbId="-1" operation="add">  
    <firstName>Mary</firstName>  
    <lastName>Jane</lastName>  
    <mobile>0244556677</mobile>  
  </person>  
</dailyTransaction>
```

Where is the dailyTransaction element?

Where is a person element?

Where is a mobile element?

XML: element

XML tags are **case sensitive**.

The tag <student> is different from the tag <STUDENT>

Common **naming convention** for XML tags

```
<student_list>
...
</student_list>
```

or

```
<studentList>
...
</studentList>
```

XML: attribute

```
<tag attribute1="..." attribute2="...">  
  </tag>
```

- **XML attributes** are used to describe XML elements, or to provide additional information about elements.

```
<?xml version="1.0" encoding="UTF-8"?>  
<dailyTransaction date="24/02/2015">  
  <person staffDbId="103" operation="update">  
    <firstName>John</firstName>  
    <lastName>Smith</lastName>  
    <mobile>0211223344</mobile>  
  </person>  
  <person staffDbId="-1" operation="add">  
    <firstName>Mary</firstName>  
    <lastName>Jane</lastName>  
    <mobile>0244556677</mobile>  
  </person>  
</dailyTransaction>
```

Does the dailyTransaction element has attributes?

Does a person element has attributes?

Does a mobile element has attributes?

XML: attribute

In XML, the attribute values must always be quoted (either by single quote or double quote):

```
<dailyTransaction date='24/02/2015'>
  <person staffDbId="103" operation="update">
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <mobile>0211223344</mobile>
  </person>
</dailyTransaction>
```

XML: relationship between elements

```
<parent>
  <child>
    <subchild>.....</subchild>
  </child>
</parent>
```

- An XML tree starts at a root element and branches from the root to child elements.
- The terms parent, child, and sibling are used to describe the relationships between elements.
 - Parent have children. Children have parents.
 - Siblings are children on the same level

XML: attribute vs child element

Any attribute can be defined as a child element.

For example, instead of using `gender` as an attribute

```
<person gender="M"
```

we can define `gender` as a child element of `person`

```
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <email>jsmith@gmail.com</email>
  <gender>M</gender>
</person>
```

This contains the same information.

XML: attribute vs child element

Any attribute can be defined as a child element.

For example, attributes `staffDbId` and `operation`

```
<person staffDbId="103" operation="update">
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <mobile>0211223344</mobile>
</person>
```

can become child elements

```
<person>
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <mobile>0211223344</mobile>
    <staffDbId>103</staffDbId>
    <operation>update</operation>
</person>
```

This contains the same information.

XML: attribute vs child element

Any attribute can be defined as a child element, **so when should we use attribute and when should we use element?**

Metadata (data about data) should be stored as attributes, and the data itself should be stored as elements.

```
<person gender="M">
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <email>jsmith@gmail.com</email>
</person>
```

```
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <email>jsmith@gmail.com</email>
  <gender>M</gender>
</person>
```

this is better

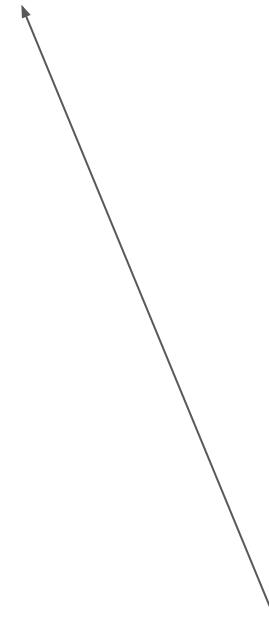
XML: attribute vs child element

Any attribute can be defined as a child element, so when should we use attribute and when should we use element?

Metadata (data about data) should be stored as attributes, and the data itself should be stored as elements.

```
<person staffDbId="103" operation="update">  
    <firstName>John</firstName>  
    <lastName>Smith</lastName>  
    <mobile>0211223344</mobile>  
</person>
```

```
<person>  
    <firstName>John</firstName>  
    <lastName>Smith</lastName>  
    <mobile>0211223344</mobile>  
    <staffDbId>103</staffDbId>  
    <operation>update</operation>  
</person>
```



this is better

XML: empty element and self-closing tag

In HTML, some elements might work well, even with a missing closing tag:

```
<br>
<hr>
<p>
<input ...>
```

In XML, all elements **must** have a closing tag:

```
<student>
...
</student>
```

An element with no content is called an **empty element**:

```
<emptyElement></emptyElement>
```

We can use **self-closing tag** for an empty element:

```
<emptyElement />
```

XML: nested rule

In HTML, some elements might not be nested properly:

```
<b><i>This text is bold and italic</b></i>
```

In XML, all elements **must** be properly nested:

```
<student>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <email>jsmith@gmail.com</email>
</student>
```

XML: entity reference

If we place a character like < inside an XML element, it will generate an error.
In this case, we need to use the entity reference <

Entity references

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

XML: comments

Comments in XML:

```
<!-- this is a comment -->
```

DTD

- XML Document Type Definition commonly known as DTD is a way to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes.
- Using a DTD, different parties can agree on a standard XML format for interchanging data.
- We can check whether an XML document conforms to a DTD or not.
- File extension is .dtd

DTD

The DTD can be declared inside the XML file, or it can be defined in a separate file:

- Internal DTD
- External DTD

DTD: internal DTD

The following DTD is declared inside the XML file:

```
<?xml version="1.0" standalone="yes" ?>
<!DOCTYPE student [
    <!ELEMENT student (firstName,lastName,email,mobile)>
    <!ELEMENT firstName (#PCDATA)>
    <!ELEMENT lastName (#PCDATA)>
    <!ELEMENT email (#PCDATA)>
    <!ELEMENT mobile (#PCDATA)>
]>
<student>
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <email>jsmith@gmail.com</email>
    <mobile>0211223344</mobile>
</student>
```

DTD: external DTD

DTD is declared outside the XML file:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE student SYSTEM "student.dtd">
<student>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <email>jsmith@gmail.com</email>
  <mobile>0211223344</mobile>
</student>
```

The content of `student.dtd`

```
<!ELEMENT student (firstName,lastName,email,mobile)>
<!ELEMENT firstName (#PCDATA)>
<!ELEMENT lastName (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT mobile (#PCDATA)>
```

DTD: internal DTD

The following DTD is declared inside the XML file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE studentList [
    <!ELEMENT studentList (student*)>
    <!ELEMENT student (firstName,lastName,email)>
    <!ELEMENT firstName (#PCDATA)>
    <!ELEMENT lastName (#PCDATA)>
    <!ELEMENT email (#PCDATA)>
]>
<studentList>
    <student>
        <firstName>John</firstName>
        <lastName>Smith</lastName>
        <email>jsmith@gmail.com</email>
    </student>
    <student>
        <firstName>Mary</firstName>
        <lastName>Jane</lastName>
        <email>mjane@gmail.com</email>
    </student>
</studentList>
```

DTD: external DTD

DTD is declared outside the XML file:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE studentList SYSTEM "studentList.dtd">
<studentList>
  <student>
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <email>jsmith@gmail.com</email>
  </student>
  <student>
    <firstName>Mary</firstName>
    <lastName>Jane</lastName>
    <email>mjane@gmail.com</email>
  </student>
</studentList>
```

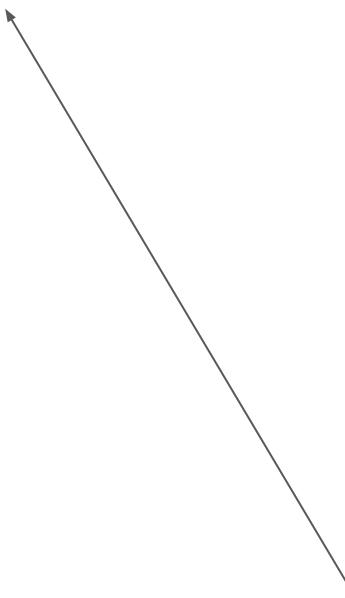
The content of **studentList.dtd**

```
<!ELEMENT studentList (student*)>
<!ELEMENT student (firstName,lastName,email)>
<!ELEMENT firstName (#PCDATA)>
<!ELEMENT lastName (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

DTD: external DTD

DTD is declared outside the XML file:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE studentList SYSTEM "studentList.dtd">
<studentList>
  <student>
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <email>jsmith@gmail.com</email>
  </student>
  <student>
    <firstName>Mary</firstName>
    <lastName>Jane</lastName>
    <email>mjane@gmail.com</email>
  </student>
</studentList>
```



To reference it as external DTD, `standalone` attribute in the XML declaration must be set as `no`. This means, declaration includes information from the external source.

DTD: Element declaration

XML elements are building blocks of an XML document.

An element is everything from the element's start tag to the element's end tag:

```
<firstName>John</firstName>
<lastName>Smith</lastName>
```

In DTD, we declare element as follows:

```
<!ELEMENT firstName (#PCDATA)>
<!ELEMENT lastName (#PCDATA)>
```

Here PCDATA stands for parsed character data.

DTD: Element declaration

An element can contain other elements

```
<student>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <email>jsmith@gmail.com</email>
</student>
```

In DTD, we declare as follows:

```
<!ELEMENT student (firstName,lastName,email)>
```

It means, the element **student** contains elements **firstName**, **lastName** and **email**.

DTD: Element declaration

An element can contain other elements

```
<studentList>
  <student>
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <email>jsmith@gmail.com</email>
  </student>
  <student>
    <firstName>Mary</firstName>
    <lastName>Jane</lastName>
    <email>mjane@gmail.com</email>
  </student>
</studentList>
```

In DTD, we declare as follows:

```
<!ELEMENT studentList (student*)>
```

It means, the element **studentList** contains zero or more elements **student**.

DTD: Element declaration

This is the general form of element declaration:

```
<!ELEMENT elementName (content)>
```

- **elementName** is the element name that you are defining.
- **content** defines what content (if any) can go within the element

DTD: Element declaration

Element content:

```
<!ELEMENT elementName (child1, child2,...)>
```

Example:

```
<!ELEMENT studentList (student*)>
```

```
<!ELEMENT student (firstName,lastName,email)>
```

<pre><!ELEMENT elementName (child+)></pre>	child element can occur one or more times inside parent element
<pre><!ELEMENT elementName (child*)></pre>	child element can occur zero or more times inside parent element
<pre><!ELEMENT elementName (child?)></pre>	child element can occur zero or one time inside parent element
<pre><!ELEMENT elementName (child1 child2)></pre>	either of child1 or child2 must occur in inside parent element
<pre><!ELEMENT elementName (child1,child2,child3,...)></pre>	Parent element must have child1, child2, child3, ... appear in this order

DTD: Attribute declaration

This is the general form of attribute declaration:

```
<!ATTLIST elementName attributeName attributeType attributeValue>
```

- **elementName** specifies the name of the element to which the attribute applies,
- **attributeName** specifies the name of the attribute,
- **attributeType** defines the type of attributes
- **attributeValue** defines the attribute value

DTD: Attribute declaration

```
<!ATTLIST elementName attributeName attributeType attributeValue>  
attributeValue
```

- can have a default value

```
<!ATTLIST elementName attributeName attributeType "default-value">
```

- can have a fixed value

```
<!ATTLIST elementName attributeName attributeType #FIXED "value">
```

- is required

```
<!ATTLIST elementName attributeName attributeType #REQUIRED>
```

- is implied: if the attribute has no default value, has no fixed value, and is not required, then it must be declared as implied

```
<!ATTLIST elementName attributeName attributeType #IMPLIED>
```

DTD: Attribute declaration

```
<?xml version="1.0" ?>
<dailyTransaction date="24/02/2015">
  <person staffDbId="103" operation="update">
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <mobile>0211223344</mobile>
  </person>
  <person staffDbId="-1" operation="add">
    <firstName>Mary</firstName>
    <lastName>Jane</lastName>
    <mobile>0244556677</mobile>
  </person>
</dailyTransaction>
```

```
<!ELEMENT dailyTransaction (person*)>
<!ATTLIST dailyTransaction date CDATA #REQUIRED>
<!ELEMENT person (firstName,lastName,mobile)>
<!ATTLIST person staffDbId CDATA #REQUIRED>
<!ATTLIST person operation CDATA #REQUIRED>
<!ELEMENT firstName (#PCDATA)>
<!ELEMENT lastName (#PCDATA)>
<!ELEMENT mobile (#PCDATA)>
```

References

- XML:

<https://developer.mozilla.org/en-US/docs/Web/XML>

- DTD:

[https://msdn.microsoft.com/en-us/library
/ms256469\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/ms256469(v=vs.110).aspx)

Introduction to Web Technology

XSD

Joseph Tonien
School of Computing and Information Technology
University of Wollongong

XSD

Objective:

- learn XSD language to define the structure of an XML document

XSD

- **XML Schema Definition (XSD)** is another way to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes.
- Using a XSD, different parties can agree on a standard XML format for interchanging data.
- We can check whether an XML document conforms to a XSD or not.
- File extension is .xsd

XSD: student example

XML file:

```
<?xml version="1.0" ?>
<student xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=" student.xsd">
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <email>jsmith@gmail.com</email>
    <mobile>0211223344</mobile>
</student>
```

XSD file **student.xsd**:

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xsd:element name="student">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="firstName" type="xsd:string"/>
                <xsd:element name="lastName" type="xsd:string"/>
                <xsd:element name="email" type="xsd:string"/>
                <xsd:element name="mobile" type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

XSD: student example

XML file:

```
<?xml version="1.0" ?>
<student xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="student.xsd">
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <email>jsmith@gmail.com</email>
  <mobile>0211223344</mobile>
</student>
```

elements and data types used in the schema
come from the namespace
<http://www.w3.org/2001/XMLSchema>



XSD file **student.xsd**:

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="student">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="firstName" type="xsd:string"/>
        <xsd:element name="lastName" type="xsd:string"/>
        <xsd:element name="email" type="xsd:string"/>
        <xsd:element name="mobile" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

XSD: student example

XML file:

```
<?xml version="1.0" ?>
<student xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="student.xsd">
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <email>jsmith@gmail.com</email>
  <mobile>0211223344</mobile>
</student>
```

the elements and data types that come from
the namespace
<http://www.w3.org/2001/XMLSchema>
should be prefixed with **xsd**

XSD file **student.xsd**:

```
<?xml version="1.0" ?>
<xsd: schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd: element name="student">
    <xsd: complexType>
      <xsd: sequence>
        <xsd: element name="firstName" type="xsd:string"/>
        <xsd: element name="lastName" type="xsd:string"/>
        <xsd: element name="email" type="xsd:string"/>
        <xsd: element name="mobile" type="xsd:string"/>
      </xsd: sequence>
    </xsd: complexType>
  </xsd: element>

</xsd: schema>
```

XSD: student example

XML file:

```
<?xml version="1.0" ?>
<student xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="student.xsd">
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <email>jsmith@gmail.com</email>
  <mobile>0211223344</mobile>
</student>
```

XSD file **student.xsd**:

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

Complex type

```
  <xsd:element name="student">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="firstName" type="xsd:string"/>
        <xsd:element name="lastName" type="xsd:string"/>
        <xsd:element name="email" type="xsd:string"/>
        <xsd:element name="mobile" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

Simple type

```
</xsd:schema>
```

XSD: element

XML element can be defined in XSD as 2 types:

- simpleType
 - complexType
-
- Element contains other elements → complexType
 - Element contains attributes → complexType
 - Element contains NO attributes, NO elements → simpleType

XSD: complex type containing element

- Element contains other elements → complexType

```
<result>
  <mark>85</mark>
  <grade>A</grade>
</result>
```

```
<xsd:element name="result">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="mark" type="xsd:integer"/>
      <xsd:element name="grade" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

XSD: complex type containing element and attribute

- Element contains other elements and attributes → complexType

```
<scan schedule="hourly">
  <start>2018-06-20T13:00:00</start>
  <finish>2018-06-20T13:01:47</finish>
  <virusFound>true</virusFound>
</scan>
```

*The attribute declarations
must always come last*

```
<xsd:element name="scan">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="start" type="xsd:dateTime" />
      <xsd:element name="finish" type="xsd:dateTime" />
      <xsd:element name="virusFound" type="xsd:boolean" />
    </xsd:sequence>
    <xsd:attribute name="schedule" type="xsd:string" />
  </xsd:complexType>
</xsd:element>
```

XSD: complex type containing attributes only

- Text-only element contains attributes (*does not contain elements*)
→ complexType

```
<price promotionCode="FAMILYDEAL">39.50</price>
```

```
<xsd:element name="price">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:decimal">
        <xsd:attribute name="promotionCode" type="xsd:string" />
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

XSD: simple type containing no element, no attribute

- Element contains no elements, no attributes → simpleType

```
<website>http://www.uow.edu.au/student</website>
```

```
<lastDayToEnrol>2000-03-24</lastDayToEnrol>
```

```
<favouriteColor>blue</favouriteColor>
```

```
<xsd:element name="website" type="xsd:anyURI" />
```

```
<xsd:element name="lastDayToEnrol" type="xsd:date" />
```

```
<xsd:element name="favouriteColor" type="xsd:string" />
```

XSD: simple type with restriction

Grade can have 4 values: A, B, C, D

```
<grade>B</grade>
```

Without restriction:

```
<xsd:element name="grade" type="xsd:string" />
```

With restriction:

```
<xsd:element name="grade">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="A"/>
      <xsd:enumeration value="B"/>
      <xsd:enumeration value="C"/>
      <xsd:enumeration value="D"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

XSD: simple type with restriction

Mark can have values between 0-100

```
<mark>84</mark>
```

Without restriction:

```
<xsd:element name="mark" type="xsd:integer" />
```

With restriction:

```
<xsd:element name="mark">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="100"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

XSD: studentList example

```
<?xml version="1.0" ?>
<studentList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="studentList.xsd">
    <student>
        <firstName>John</firstName>
        <lastName>Smith</lastName>
        <email>jsmith@gmail.com</email>
    </student>
    <student>
        <firstName>Mary</firstName>
        <lastName>Jane</lastName>
        <email>mjane@gmail.com</email>
    </student>
</studentList>

<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="studentList">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="student" minOccurs="0" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="firstName" type="xsd:string"/>
                            <xsd:element name="lastName" type="xsd:string"/>
                            <xsd:element name="email" type="xsd:string"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

XSD: studentList example

```
<?xml version="1.0" ?>
<studentList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="studentList.xsd">
  <student>
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <email>jsmith@gmail.com</email>
  </student>
  <student>
    <firstName>Mary</firstName>
    <lastName>Jane</lastName>
    <email>mjane@gmail.com</email>
  </student>
</studentList>
```

Let's start with the root element `studentList`

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="studentList">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="student" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="firstName" type="xsd:string"/>
              <xsd:element name="lastName" type="xsd:string"/>
              <xsd:element name="email" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XSD: studentList example

```
<?xml version="1.0" ?>
<studentList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="studentList.xsd">
  <student>
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <email>jsmith@gmail.com</email>
  </student>
  <student>
    <firstName>Mary</firstName>
    <lastName>Jane</lastName>
    <email>mjane@gmail.com</email>
  </student>
</studentList>
```

Let's start with the root element `studentList`

- it is a complex type

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="studentList">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="student" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="firstName" type="xsd:string"/>
              <xsd:element name="lastName" type="xsd:string"/>
              <xsd:element name="email" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XSD: studentList example

```
<?xml version="1.0" ?>
<studentList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="studentList.xsd">
  <student>
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <email>jsmith@gmail.com</email>
  </student>
  <student>
    <firstName>Mary</firstName>
    <lastName>Jane</lastName>
    <email>mjane@gmail.com</email>
  </student>
</studentList>
```

Let's start with the root element `studentList`

- it is a complex type
- which contains a sequence of `student` elements

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="studentList">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="student" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="firstName" type="xsd:string"/>
              <xsd:element name="lastName" type="xsd:string"/>
              <xsd:element name="email" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XSD: studentList example

```
<?xml version="1.0" ?>
<studentList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="studentList.xsd">
  <student>
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <email>jsmith@gmail.com</email>
  </student>
  <student>
    <firstName>Mary</firstName>
    <lastName>Jane</lastName>
    <email>mjane@gmail.com</email>
  </student>
</studentList>
```

Let's start with the root element `studentList`

- it is a complex type
- which contains a sequence of `student` elements
- `studentList` contains zero or unlimited number of `student` elements

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="studentList">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="student" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="firstName" type="xsd:string"/>
              <xsd:element name="lastName" type="xsd:string"/>
              <xsd:element name="email" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XSD: studentList example

```
<?xml version="1.0" ?>
<studentList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="studentList.xsd">
<b><student></b>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <email>jsmith@gmail.com</email>
</b><b></student></b>
<student>
  <firstName>Mary</firstName>
  <lastName>Jane</lastName>
  <email>mjane@gmail.com</email>
</student>
</studentList>
```

The element **student** is also a complex type

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="studentList">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="student" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="firstName" type="xsd:string"/>
              <xsd:element name="lastName" type="xsd:string"/>
              <xsd:element name="email" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XSD: studentList example

```
<?xml version="1.0" ?>
<studentList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="studentList.xsd">
<b><student></b>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <email>jsmith@gmail.com</email>
</b><b></student></b>
<student>
  <firstName>Mary</firstName>
  <lastName>Jane</lastName>
  <email>mjane@gmail.com</email>
</student>
</studentList>
```

The element **student** is also a complex type

- which contains a sequence of elements

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="studentList">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="student" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="firstName" type="xsd:string"/>
              <xsd:element name="lastName" type="xsd:string"/>
              <xsd:element name="email" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XSD: studentList example

```
<?xml version="1.0" ?>
<studentList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="studentList.xsd">
<b><student></b>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <email>jsmith@gmail.com</email>
</b><b></student></b>
<student>
  <firstName>Mary</firstName>
  <lastName>Jane</lastName>
  <email>mjane@gmail.com</email>
</student>
</studentList>
```

The element **student** is also a complex type

- which contains a sequence of elements:
firstName, lastName, email

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="studentList">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="student" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="firstName" type="xsd:string"/>
              <xsd:element name="lastName" type="xsd:string"/>
              <xsd:element name="email" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XSD: studentList example

```
<?xml version="1.0" ?>
<studentList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="studentList.xsd">
  <b><student></student></b>
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <email>jsmith@gmail.com</email>
  </b><student>
    <firstName>Mary</firstName>
    <lastName>Jane</lastName>
    <email>mjane@gmail.com</email>
  </student>
</studentList>
```

firstName, lastName, email elements are all simple type

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="studentList">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="student" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="firstName" type="xsd:string"/>
              <xsd:element name="lastName" type="xsd:string"/>
              <xsd:element name="email" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XSD: dailyTransaction example

```
<?xml version="1.0" ?>
<dailyTransaction date="24/02/2015">
  <person staffDbId="103" operation="update">
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <mobile>0211223344</mobile>
  </person>
  <person staffDbId="-1" operation="add">
    <firstName>Mary</firstName>
    <lastName>Jane</lastName>
    <mobile>0244556677</mobile>
  </person>
</dailyTransaction>
```

complexType: dailyTransaction, person

simpleType: firstName, lastName, mobile

XSD: dailyTransaction example

```
<dailyTransaction date="24/02/2015">
  <person staffDbId="103" operation="update">
    ...
  </person>
  <person staffDbId="-1" operation="add">
    ...
  </person>
</dailyTransaction>
```

Start with the root element dailyTransaction:

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="dailyTransaction">
    <xsd:complexType>
      ...
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XSD: dailyTransaction example

```
<dailyTransaction date="24/02/2015">
  <person staffDbId="103" operation="update">
    ...
  </person>
  <person staffDbId="-1" operation="add">
    ...
  </person>
</dailyTransaction>
```

The root element `dailyTransaction` contains a sequence of `person` elements and has attribute `date`

```
<xsd:element name="dailyTransaction">
  <xsd:complexType>
    <xsd:sequence>
      ...
    </xsd:sequence>
    <xsd:attribute name="date" type="xsd:string" />
  </xsd:complexType>
</xsd:element>
```

XSD: dailyTransaction example

```
<dailyTransaction date="24/02/2015">
  <person staffDbId="103" operation="update">
    ...
  </person>
  <person staffDbId="-1" operation="add">
    ...
  </person>
</dailyTransaction>
```

The root element `dailyTransaction` contains a sequence of `person` elements and has attribute `date`

```
<xsd:element name="dailyTransaction">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="person" minOccurs="0" maxOccurs="unbounded">
        ...
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="date" type="xsd:string" />
  </xsd:complexType>
</xsd:element>
```

XSD: dailyTransaction example

```
<person staffDbId="103" operation="update">
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <mobile>0211223344</mobile>
</person>
```

The element person contains:

- elements: firstName, lastName, mobile
- attributes: staffDbId, operation

```
<xsd:element name="person" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
        <xsd:sequence>
            ...
        </xsd:sequence>
        <xsd:attribute name="staffDbId" type="xsd:integer" />
        <xsd:attribute name="operation" type="xsd:string" />
    </xsd:complexType>
</xsd:element>
```

XSD: dailyTransaction example

```
<person staffDbId="103" operation="update">
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <mobile>0211223344</mobile>
</person>
```

The element person contains:

- elements: firstName, lastName, mobile
- attributes: staffDbId, operation

```
<xsd:element name="person" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="firstName" type="xsd:string"/>
            <xsd:element name="lastName" type="xsd:string"/>
            <xsd:element name="mobile" type="xsd:string"/>
        </xsd:sequence>
        <xsd:attribute name="staffDbId" type="xsd:integer" />
        <xsd:attribute name="operation" type="xsd:string" />
    </xsd:complexType>
</xsd:element>
```

References

- XSD: https://www.w3schools.com/xml/schema_intro.asp
- XSD: [https://msdn.microsoft.com/en-us/library/ms256235\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/ms256235(v=vs.110).aspx)

Introduction to Web Technology

XSLT

Joseph Tonien

School of Computing and Information Technology
University of Wollongong

XSLT

EXtensible Stylesheet Language Transformation (XSLT) is an XML language for transforming XML documents

- file extension is .xsl
- used to transform XML file into other file formats, such as HTML
- describes how the XML elements should be displayed.

The content of the stylesheet XSL file looks like the following:

```
<?xml version="1.0"?>
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/1999/xhtml">

    ... xslt code here ...

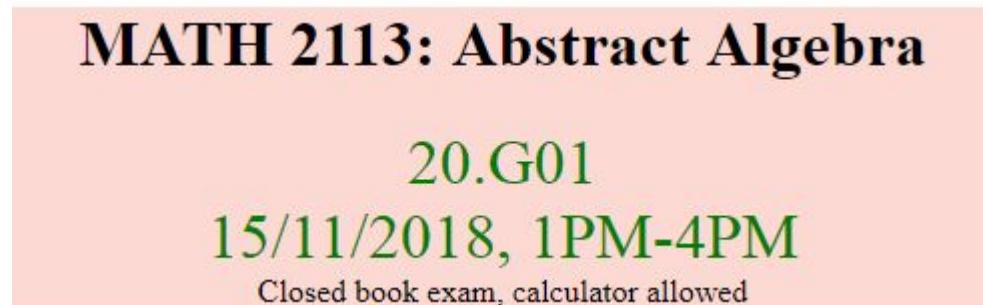
</xsl:stylesheet>
```

Example: Exam timetable

This is an XML data representing exam information for a particular subject:

```
<exam subject="MATH 2113">
  <title>Abstract Algebra</title>
  <venue>20.G01</venue>
  <date>15/11/2018</date>
  <time>1PM-4PM</time>
  <note>Closed book exam, calculator allowed</note>
</exam>
```

We would like to transform this XML content into the following HTML display:



Example: Exam timetable

exam.xml

```
<?xml version="1.0" ?>
<exam subject="MATH 2113">
    <title>Abstract Algebra</title>
    <venue>20.G01</venue>
    <date>15/11/2018</date>
    <time>1PM-4PM</time>
    <note>Closed book exam, calculator allowed</note>
</exam>
```

exam-with-style.xml

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="style-exam.xsl"?>
<exam subject="MATH 2113">
    <title>Abstract Algebra</title>
    <venue>20.G01</venue>
    <date>15/11/2018</date>
    <time>1PM-4PM</time>
    <note>Closed book exam, calculator allowed</note>
</exam>
```

These two XML files have almost the same content.

The 2nd one has a reference to a **stylesheet**. Let's look at them on a browser.

Example: Exam timetable

`exam.xml`

```
▼<exam subject="MATH 2113">
  <title>Abstract Algebra</title>
  <venue>20.G01</venue>
  <date>15/11/2018</date>
  <time>1PM-4PM</time>
  <note>Closed book exam, calculator allowed</note>
</exam>
```

`exam-with-style.xml`

MATH 2113: Abstract Algebra

20.G01

15/11/2018, 1PM-4PM

Closed book exam, calculator allowed

On web browser, the two XML files display differently.

This is because the second XML uses a **stylesheet**.

```
<?xml-stylesheet type="text/xsl" href="style-exam.xsl"?>
```



*Let's have a look at the **stylesheet** file*

Example: Exam timetable

Let's have a look at the stylesheet file: **style-exam.xsl**

```
<?xml version="1.0"?>
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/1999/xhtml">

    <xsl:output method="xml" indent="yes" encoding="UTF-8"/>

    <xsl:template match="/exam">

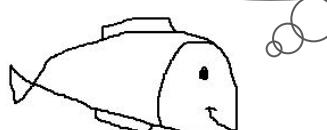
        <html>

            <head>
                <title>XSLT example</title>
            </head>

            <body>
                ...
            </body>

        </html>
    </xsl:template>
</xsl:stylesheet>
```

*What do you see in this stylesheet file:
style-exam.xsl*



Example: Exam timetable

Stylesheet file: style-exam.xsl

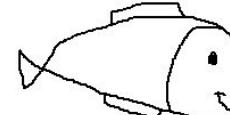
```
<?xml version="1.0"?>
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/1999/xhtml">

    <xsl:output method="xml" indent="yes" encoding="UTF-8"/>

    <xsl:template match="/exam">
        <html>
            <head>
                <title>XSLT example</title>
            </head>

            <body>
                ...
            </body>

        </html>
    </xsl:template>
</xsl:stylesheet>
```



this is the root element of the XML file

```
<?xml version="1.0" ?>
<exam subject="MATH 2113">
    <title>Abstract Algebra</title>
    <venue>20.G01</venue>
    <date>15/11/2018</date>
    <time>1PM-4PM</time>
    <note>Closed book exam, calculator allowed</note>
</exam>
```

Example: Exam timetable

Stylesheet file: style-exam.xsl

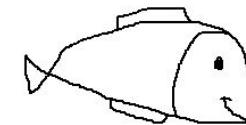
```
<?xml version="1.0"?>
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/1999/xhtml">

    <xsl:output method="xml" indent="yes" encoding="UTF-8"/>

    <xsl:template match="/exam">
        <html>
            <head>
                <title>XSLT example</title>
            </head>

            <body>
                ...
            </body>

        </html>
    </xsl:template>
</xsl:stylesheet>
```



this looks like HTML code,

that's why the XML file is displayed on the browser as HTML

MATH 2113: Abstract Algebra

20.G01

15/11/2018, 1PM-4PM

Closed book exam, calculator allowed

Example: Exam timetable

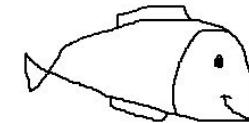
Stylesheet file: style-exam.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/1999/xhtml">

    <xsl:output method="xml" indent="yes" encoding="UTF-8"/>

    <xsl:template match="/exam">

        <html>
            <head>
                <title>XSLT example</title>
            </head>
            <body>
                ...
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```



Let's have a look at this code in more detail

MATH 2113: Abstract Algebra

20.G01

15/11/2018, 1PM-4PM

Closed book exam, calculator allowed

Example: Exam timetable

Stylesheet file: style-exam.xsl

...

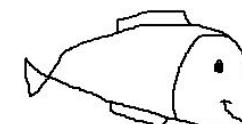
<body>

```
<div align="center" style="background-color:#f0371930">

<h1>
  <xsl:value-of select="@subject" />
  <xsl:text>: </xsl:text>
  <xsl:value-of select="title" />
</h1>

<font size="6" color="green">
  <xsl:value-of select="venue" />
  <br />
  <xsl:value-of select="date" />
  <xsl:text>, </xsl:text>
  <xsl:value-of select="time" />
</font>
<br />

<xsl:value-of select="note" />
</div>
</body>
```



MATH 2113: Abstract Algebra

20.G01

15/11/2018, 1PM-4PM

Closed book exam, calculator allowed

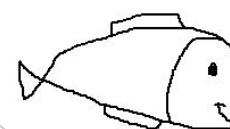
Example: Exam timetable

Stylesheet file: style-exam.xsl

```
...
<body>
  <div align="center" style="background-color:#f0371930">
    <h1>
      <xsl:value-of select="@subject" />
      <xsl:text>: </xsl:text>
      <xsl:value-of select="title" />
    </h1>

    <font size="6" color="green">
      <xsl:value-of select="venue" />
      <br />
      <xsl:value-of select="date" />
      <xsl:text>, </xsl:text>
      <xsl:value-of select="time" />
    </font>
    <br />

    <xsl:value-of select="note" />
  </div>
</body>
...
```



MATH 2113: Abstract Algebra
20.G01
15/11/2018, 1PM-4PM
Closed book exam, calculator allowed

Example: Exam timetable

Stylesheet file: style-exam.xsl

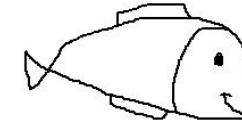
```
...
<xsl:value-of select="@subject" /> ----- MATH 2113
<xsl:text>: </xsl:text>
<xsl:value-of select="title" />

<xsl:value-of select="venue" />

<xsl:value-of select="date" />
<xsl:text>, </xsl:text>
<xsl:value-of select="time" />

<xsl:value-of select="note" />
...

```



Get value from attribute

```
<?xml version="1.0" ?>
<exam subject="MATH 2113">
  <title>Abstract Algebra</title>
  <venue>20.G01</venue>
  <date>15/11/2018</date>
  <time>1PM-4PM</time>
  <note>Closed book exam, calculator allowed</note>
</exam>
```

MATH 2113: Abstract Algebra
20.G01
15/11/2018, 1PM-4PM
Closed book exam, calculator allowed

Example: Exam timetable

Stylesheet file: style-exam.xsl

...
<xsl:value-of select="@subject" />
<xsl:text>: </xsl:text>
<xsl:value-of select="title" />

→ Abstract Algebra

<xsl:value-of select="venue" />

→ 20.G01

<xsl:value-of select="date" />

→ 15/11/2018

<xsl:text>, </xsl:text>

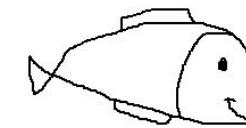
<xsl:value-of select="time" />

→ 1PM-4PM

<xsl:value-of select="note" />

→ Closed book exam,
calculator allowed

...
<?xml version="1.0" ?>
<exam subject="MATH 2113">
 <title>Abstract Algebra</ title>
 <venue>20.G01</ venue>
 <date>15/11/2018</ date>
 <time>1PM-4PM</ time>
 <note>Closed book exam, calculator allowed</ note>
</exam>



Get value from element

MATH 2113: Abstract Algebra

20.G01

15/11/2018, 1PM-4PM

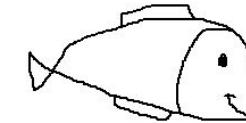
Closed book exam, calculator allowed

Example: Exam timetable

Stylesheet file: style-exam.xsl

```
...  
<xsl:value-of select="@subject" />  
<xsl:text>: </xsl:text> → Write literal text  
<xsl:value-of select="title" />  
  
<xsl:value-of select="venue" />  
  
<xsl:value-of select="date" />  
<xsl:text>, </xsl:text>  
<xsl:value-of select="time" />  
  
<xsl:value-of select="note" />  
...
```

```
<?xml version="1.0" ?>  
<exam subject="MATH 2113">  
  <title>Abstract Algebra</title>  
  <venue>20.G01</venue>  
  <date>15/11/2018</date>  
  <time>1PM-4PM</time>  
  <note>Closed book exam, calculator allowed</note>  
</exam>
```



Write literal text

MATH 2113: Abstract Algebra

20.G01

15/11/2018, 1PM-4PM

Closed book exam, calculator allowed

Example: Exam timetable

Stylesheet file: style-exam.xsl

...

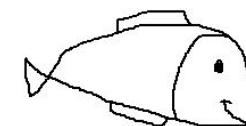
<body>

```
<div align="center" style="background-color:#f0371930">

<h1>
  <xsl:value-of select="@subject" />
  <xsl:text>: </xsl:text>
  <xsl:value-of select="title" />
</h1>

<font size="6" color="green">
  <xsl:value-of select="venue" />
  <br />
  <xsl:value-of select="date" />
  <xsl:text>, </xsl:text>
  <xsl:value-of select="time" />
</font>
<br />

<xsl:value-of select="note" />
</div>
</body>
...
```



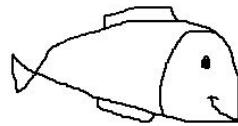
MATH 2113: Abstract Algebra

20.G01

15/11/2018, 1PM-4PM

Closed book exam, calculator allowed

Summary



Get attribute value:

```
<xsl:value-of select="@attribute-name" />
```

Get element value:

```
<xsl:value-of select="element-name" />
```

Literal text:

```
<xsl:text>some text here ...</xsl:text>
```

Example: Transaction v0

transaction0.xml uses stylesheet **transaction-style0.xsl**

```
<?xml version="1.0" ?>
<?xmlstylesheet type="text/xsl" href="transaction-style0.xsl"?>
<dailyTransaction date="24/02/2015">
    <person staffDbId="103" operation="update">
        <firstName>John</firstName>
        <lastName>Smith</lastName>
        <mobile>0211223344</mobile>
    </person>
    <person staffDbId="-1" operation="add">
        <firstName>Mary</firstName>
        <lastName>Jane</lastName>
        <mobile>0244556677</mobile>
    </person>
    ...
</dailyTransaction>
```

Example: Transaction v0

Have a look at the XML file `transaction0.xml` in the browser

24/02/2015

John
Smith
0211223344
103
update

Mary
Jane
0244556677
-1
add

Matt
Brown

Example: Transaction v0

Let's look at the xml stylesheet: **transaction-style0.xsl**

```
...
<xsl:template match="/dailyTransaction">
  <html>
    <head>
      <title>XSLT example</title>
    </head>
    <body>
      ...
      </body>
    </html>
</xsl:template>
...

```

```

dailyTransaction date="24/02/2015">
  <person staffDbId="103" operation="update">
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <mobile>0211223344</mobile>
  </person>
  <person staffDbId="-1" operation="add">
    <firstName>Mary</firstName>
    <lastName>Jane</lastName>
    <mobile>0244556677</mobile>
  </person>
  ...
</dailyTransaction>
```

Example: Transaction v0

Let's look at the xml stylesheet: **transaction-style0.xsl**

```
...
<body>
  <xsl:value-of select="@date" /> <br /><br />

  <xsl:for-each select="person">
    <xsl:value-of select="firstName" />
    <br />

    <xsl:value-of select="lastName" />
    <br />

    <xsl:value-of select="mobile" />
    <br />

    <xsl:value-of select="@staffDbId" />
    <br />

    <xsl:value-of select="@operation" />
    <br />
    <br />

  </xsl:for-each>
</body>
```

John
Smith
0211223344
103
update

Mary
Jane
0244556677
-1
add

Matt
Brown

```
<dailyTransaction date="24/02/2015">
  <person staffDbId="103" operation="up
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <mobile>0211223344</mobile>
  </person>
  <person staffDbId="-1" operation="add
    <firstName>Mary</firstName>
    <lastName>Jane</lastName>
    <mobile>0244556677</mobile>
  </person>
  ...
</dailyTransaction>
```

Example: Transaction v0

Let's look at the xml stylesheet: **transaction-style0.xsl**

```
...
<body>
  <xsl:value-of select="@date" /> <br /><br />

  <xsl:for-each select="person"> ←
    <xsl:value-of select="firstName" />
    <br />

    <xsl:value-of select="lastName" />
    <br />

    <xsl:value-of select="mobile" />
    <br />

    <xsl:value-of select="@staffDbId" />
    <br />

    <xsl:value-of select="@operation" />
    <br />
    <br />

  </xsl:for-each>
</body>
...

```

FOR LOOP

John
Smith
0211223344
103

update

Mary
Jane
0244556677
-1

add

Matt
Brown

```
<dailyTransaction date="24/02/2015">
  <person staffDbId="103" operation="u">
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <mobile>0211223344</mobile>
  </person>
  <person staffDbId="-1" operation="a">
    <firstName>Mary</firstName>
    <lastName>Jane</lastName>
    <mobile>0244556677</mobile>
  </person>
  ...
</dailyTransaction>
```

Example: Transaction v0

Let's look at the xml stylesheet: **transaction-style0.xsl**

```
...
<body>
  <xsl:value-of select="@date" /> <br /><br />

  <xsl:for-each select="person">

    <xsl:value-of select="firstName" />
    <br />

    <xsl:value-of select="lastName" />
    <br />

    <xsl:value-of select="mobile" />
    <br />

    <xsl:value-of select="@staffDbId" />
    <br />

    <xsl:value-of select="@operation" />
    <br />
    <br />

  </xsl:for-each>
</body>
...
```

John Smith 0211223344 103 update	Mary Jane 0244556677 -1 add	Matt Brown
<dailyTransaction date="24/02/2015"> <person staffDbId="103" operation="up"> <firstName>John</firstName> <lastName>Smith</lastName> <mobile>0211223344</mobile> </person> <person staffDbId="-1" operation="add"> <firstName>Mary</firstName> <lastName>Jane</lastName> <mobile>0244556677</mobile> </person> ... </dailyTransaction>		

Example: Transaction v0

Let's look at the xml stylesheet: **transaction-style0.xsl**

```
...
<body>
  <xsl:value-of select="@date" /> <br /><br />

  <xsl:for-each select="person">

    <xsl:value-of select="firstName" />
    <br />

    <xsl:value-of select="lastName" />
    <br />

    <xsl:value-of select="mobile" />
    <br />

    <xsl:value-of select="@staffDbId" />
    <br />

    <xsl:value-of select="@operation" />
    <br />
    <br />

  </xsl:for-each>
</body>
...
```

<code>John</code> <code>Smith</code> <code>0211223344</code> <code>103</code> <code>update</code>	<code>Mary</code> <code>Jane</code> <code>0244556677</code> <code>-1</code> <code>add</code>	<code>Matt</code> <code>Brown</code>
<dailyTransaction date="24/02/2015"> <person staffDbId="103" operation="update"> <firstName>John</firstName> <lastName>Smith</lastName> <mobile>0211223344</mobile> </person> <person staffDbId="-1" operation="add"> <firstName>Mary</firstName> <lastName>Jane</lastName> <mobile>0244556677</mobile> </person> ... </dailyTransaction>		

Example: Transaction v1

transaction1.xml uses stylesheet **transaction-style1.xsl**

```
<?xml version="1.0" encoding="UTF-8" ?>
<?xmlstylesheet type="text/xsl" href="transaction-style1.xsl"?>
<dailyTransaction date="24/02/2015">
    <person staffDbId="103" operation="update">
        <firstName>John</firstName>
        <lastName>Smith</lastName>
        <mobile>0211223344</mobile>
    </person>
    <person staffDbId="-1" operation="add">
        <firstName>Mary</firstName>
        <lastName>Jane</lastName>
        <mobile>0244556677</mobile>
    </person>
    ...
</dailyTransaction>
```

Example: Transaction v1

Have a look at the XML file in the browser

Daily transaction 24/02/2015

- John Smith, 0211223344, 103, update
- Mary Jane, 0244556677, -1, add
- Matt Brown, 0413273345, 104, remove
- Jack Patel, 0211223344, 105, update
- Frank Jones, 0244556677, -1, add
- James Williams, 0413273345, 106, remove

Example: Transaction v1

Let's look at the xml stylesheet: **transaction-style1.xsl**

```
...
<h1>Daily transaction <xsl:value-of select="@date" /> </h1>
<ul>
  <xsl:for-each select="person">
    <li>
      <xsl:value-of select="firstName" />
      <xsl:text> </xsl:text>
      <xsl:value-of select="lastName" />
      <xsl:text>, </xsl:text>
      <xsl:value-of select="mobile" />
      <xsl:text>, </xsl:text>
      <xsl:value-of select="@staffDbId" />
      <xsl:text>, </xsl:text>
      <xsl:value-of select="@operation" />
    </li>
  </xsl:for-each>
</ul>
...

```

Daily transaction 24/02/2015

- John Smith, 0211223344, 103, update
- Mary Jane, 0244556677, -1, add
- Matt Brown, 0413273345, 104, remove
- Jack Patel, 0211223344, 105, update
- Frank Jones, 0244556677, -1, add
- James Williams, 0413273345, 106, remove

Example: Transaction v2

Version 2 is exactly like version 1 except that we only display staffDbId if it is a positive number.

```
...
<xsl:if test="@staffDbId > 0">           ←———— IF statement
  <xsl:text>, </xsl:text>
  <xsl:value-of select="@staffDbId" />
</xsl:if>
```

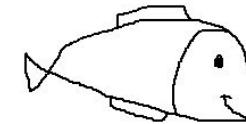
...

Daily transaction 24/02/2015

- John Smith, 0211223344, 103, update
- Mary Jane, 0244556677, add
- Matt Brown, 0413273345, 104, remove
- Jack Patel, 0211223344, 105, update
- Frank Jones, 0244556677, add
- James Williams, 0413273345, 106, remove

Example: Transaction v2

```
...
<xsl:if test="@staffDbId > 0">
  <xsl:text>, </xsl:text>
  <xsl:value-of select="@staffDbId" />
</xsl:if>
...
...
```



Danger Stranger!!!
There is no IF-ELSE

*IF statement will be applied when a specified condition is true.
Use CHOOSE-WHEN-OTHERWISE statement to express multiple conditional tests.*

Daily transaction 24/02/2015

- John Smith, 0211223344, 103, update
- Mary Jane, 0244556677, add
- Matt Brown, 0413273345, 104, remove
- Jack Patel, 0211223344, 105, update
- Frank Jones, 0244556677, add
- James Williams, 0413273345, 106, remove

Conditional statement examples

Mark = 49:

mark = 49

Mark not equal to 49:

mark != 49

Student type = 'U':

type = 'U'

Student type not equal to 'U':

type != 'U'

Mark > 35:

mark > 35

Mark >= 85:

mark >= 85

Mark < 35:

mark < 35

Mark <= 85:

mark <= 85

```
<xsl:if test="@staffDbId > 0">  
    <xsl:text>, </xsl:text>  
    <xsl:value-of select="@staffDbId" />  
</xsl:if>
```

Mark NOT equal to 49:

not (mark = 49)

Student type = 'U' or 'P':

(type = 'U') or (type = 'P')

Mark >= 75 and mark < 85:

(mark >= 75) and (mark < 85)

Example: Transaction v3

Now look at `transaction3.xml`, it uses an xml stylesheet:

transaction-style3.xsl

```
<?xml version="1.0"?>
<?xmlstylesheet type="text/xsl" href="transaction-style3.xsl"?>
<dailyTransaction date="24/02/2015">
  <person staffDbId="103" operation="update">
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <mobile>0211223344</mobile>
  </person>
  <person staffDbId="-1" operation="add">
    <firstName>Mary</firstName>
    <lastName>Jane</lastName>
    <mobile>0244556677</mobile>
  </person>
  ...
</dailyTransaction>
```

Example: Transaction v3

View the source of the xml stylesheet: `transaction-style3.xsl`
we can see that it displays a table and the data is sorted.

Daily transaction 24/02/2015

Sorted by lastName

Name	Mobile	Staff Id	Operation
Matt Brown	0413273345	3	remove
Mary Jane	0244556677	-1	add
Frank Jones	0244556677	-1	add
Jack Patel	0211223344	10	update
John Smith	0211223344	103	update
James Williams	0413273345	105	remove

Example: Transaction v3

```
<xsl:for-each select="person">  
  <xsl:sort select="lastName"/>  
  ...  
</xsl:for-each>
```

Daily transaction 24/02/2015

Sorted by lastName

Name	Mobile	Staff Id	Operation
Matt Brown	0413273345	3	remove
Mary Jane	0244556677	-1	add
Frank Jones	0244556677	-1	add
Jack Patel	0211223344	10	update
John Smith	0211223344	103	update
James Williams	0413273345	105	remove

Example: Transaction v3

```
<xsl:for-each select="person">  
    <xsl:sort select="@operation"/>  
    . . .  
</xsl:for-each>
```

Sorted by operation

Name	Mobile	Staff Id	Operation
Mary Jane	0244556677	-1	add
Frank Jones	0244556677	-1	add
Matt Brown	0413273345	3	remove
James Williams	0413273345	105	remove
John Smith	0211223344	103	update
Jack Patel	0211223344	10	update

Example: Transaction v3

```
<xsl:for-each select="person">
    <xsl:sort select="@staffDbId"/>
    ...
</xsl:for-each>

<xsl:for-each select="person">
    <xsl:sort select="@staffDbId" data-type="number"/>
    ...
</xsl:for-each>
```

Sorted by staffDbId as string data type

Name	Mobile	Staff Id	Operation
Mary Jane	0244556677	-1	add
Frank Jones	0244556677	-1	add
Jack Patel	0211223344	10	update
John Smith	0211223344	103	update
James Williams	0413273345	105	remove
Matt Brown	0413273345	3	remove

Sorted by staffDbId as number data type

Name	Mobile	Staff Id	Operation
Mary Jane	0244556677	-1	add
Frank Jones	0244556677	-1	add
Matt Brown	0413273345	3	remove
Jack Patel	0211223344	10	update
John Smith	0211223344	103	update
James Williams	0413273345	105	remove

Example: Transaction v4

Now look at transaction4.xml

Daily transaction 24/02/2015

Name	Mobile	Staff Id	Operation
Matt Brown	0413273345	104	remove
Mary Jane	0244556677	-1	add
Frank Jones	0244556677	-1	add
Jack Patel	0211223344	105	update
John Smith	0211223344	103	update
James Williams	0413273345	106	remove

Example: Transaction v4

Now look at transaction-style4.xsl

```
<xsl:choose>
```

```
  <xsl:when test="@operation = 'remove'">
    <td bgcolor="#ffe6e6">
      <xsl:value-of select="@operation" />
    </td>
  </xsl:when>
```

```
  <xsl:when test="@operation = 'add'">
    <td bgcolor="#ffffe6">
      <xsl:value-of select="@operation" />
    </td>
  </xsl:when>
```

```
  <xsl:otherwise>
    <td bgcolor="#d6f5d6">
      <xsl:value-of select="@operation" />
    </td>
  </xsl:otherwise>
```

```
</xsl:choose>
```

Example: Transaction v5

Now look at transaction5.xml

Daily transaction 24/02/2015

Name	Mobile	Staff Id	Operation
Matt Brown		104	remove
Mary Jane	0244556677	-1	add
Frank Jones	0244556677	-1	add
Jack Patel	0211223344	105	update
John Smith		103	update
James Williams	0413273345	106	remove

Example: Transaction v5

Now look at transaction-style5.xsl

```
<xsl:choose>

<xsl:when test="mobile = ''">
    <td bgcolor="#ffe6e6"> </td>
</xsl:when>

<xsl:otherwise>
    <td> <xsl:value-of select="mobile" /> </td>
</xsl:otherwise>

</xsl:choose>
```

Example: Transaction v6

Now look at transaction6.xml

Daily transaction 24/02/2015

New records

Name	Mobile	Staff Id	Operation
Mary Jane	0244556677	-1	add
Frank Jones	0244556677	-1	add

Updated records

Name	Mobile	Staff Id	Operation
Jack Patel	0211223344	105	update
John Smith	0211223344	103	update

Removed records

Name	Mobile	Staff Id	Operation
Matt Brown	0413273345	104	remove
James Williams	0413273345	106	remove

Example: Transaction v6

Now look at transaction-style6.xsl

```
<xsl:for-each select="person[@operation='add']">
  <xsl:sort select="lastName"/>
  <tr>
    <td>
      <xsl:value-of select="firstName" />
      <xsl:text> </xsl:text>
      <xsl:value-of select="lastName" />
    </td>
    <td>
      <xsl:value-of select="mobile" />
    </td>
    <td>
      <xsl:value-of select="@staffDbId" />
    </td>
    <td>
      <xsl:value-of select="@operation" />
    </td>
  </tr>
</xsl:for-each>
```

Example: Transaction v7

Now look at transaction7.xml

Daily transaction 24/02/2015

Name	Mobile	Staff Id	Operation
Matt Brown	0413273345	104	
Mary Jane	0244556677	-1	
Frank Jones	0244556677	-1	
Jack Patel	0211223344	105	
John Smith	0211223344	103	
James Williams	0413273345	106	

Example: Transaction v7

Now look at transaction-style7.xsl

```
...
<td>
  <xsl:value-of select="@staffDbId" />
</td>

<td align="center">
  <img>
    <xsl:attribute name="src">
      <xsl:text>images/</xsl:text>
      <xsl:value-of select="@operation"/>
      <xsl:text>.png</xsl:text>
    </xsl:attribute>

    <xsl:attribute name="width">
      <xsl:text>30px</xsl:text>
    </xsl:attribute>
  </img>
</td>
...
```

Example: Transaction v8

Now look at transaction8.xml

Daily transaction 24/02/2015

Name	Mobile	Staff Id	Operation
Matt Brown	0413273345	104	remove
Mary Jane	0244556677	-1	add
Frank Jones	0244556677	-1	add
Jack Patel	0211223344	105	update
John Smith	0211223344	103	update
James Williams	0413273345	106	remove

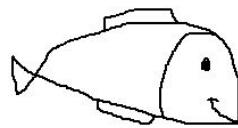
Example: Transaction v8

Now look at `transaction-style8.xsl`

```
...
<td>
<xsl:choose>
  <xsl:when test="@staffDbId < 0">
    <span style="color:red">
      <xsl:value-of select="@staffDbId" />
    </span>
  </xsl:when>

  <xsl:otherwise>
    <span style="color:green">
      <xsl:value-of select="@staffDbId" />
    </span>
  </xsl:otherwise>
</xsl:choose>
</td>
...
```

Summary



Get attribute value:

```
<xsl:value-of select="@attribute-name" />
```

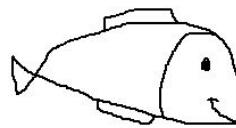
Get element value:

```
<xsl:value-of select="element-name" />
```

Literal text:

```
<xsl:text>some text here ...</xsl:text>
```

Summary



FOR loop:

```
<xsl:for-each select="element-name">  
    ...  
</xsl:for-each>
```

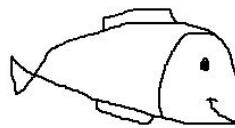
FOR loop with sort:

```
<xsl:for-each select="element-name">  
    <xsl:sort select="field-to-be-sorted"/>  
    ...  
</xsl:for-each>
```

FOR loop with filter:

```
<xsl:for-each select="element-name[filter-condition]">  
    ...  
</xsl:for-each>
```

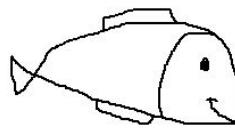
Summary



IF statement: (note, there is no IF-ELSE)

```
<xsl:if test="the-if-condition">  
    ...  
</xsl:if>
```

Summary



CHOOSE-WHEN-OTHERWISE statement:

```
<xsl:choose>

  <xsl:when test="condition1">
    ...
  </xsl:when>

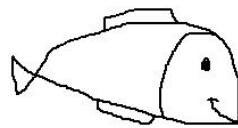
  <xsl:when test="condition2">
    ...
  </xsl:when>

  ...
  ...

  <xsl:otherwise>
    ...
  </xsl:otherwise>

</xsl:choose>
```

Summary



Mark = 49:

```
mark = 49
```

Mark not equal to 49:

```
mark != 49
```

Student type = 'U':

```
type = 'U'
```

Student type not equal to 'U':

```
type != 'U'
```

Mark > 35:

```
mark > 35
```

Mark >= 85:

```
mark >= 85
```

Mark < 35:

```
mark < 35
```

Mark <= 85:

```
mark <= 85
```

Mark NOT equal to 49:

```
not (mark = 49)
```

Student type = 'U' or 'P':

```
(type = 'U') or (type = 'P')
```

Mark >= 75 and mark < 85:

```
(mark >= 75) and (mark < 85)
```

References

- https://www.w3schools.com/xml/xsl_intro.asp
- <https://developer.mozilla.org/en-US/docs/Web/XSLT>

Introduction to Web Technology

JSON

Joseph Tonien

School of Computing and Information Technology
University of Wollongong

JavaScript: Array

```
var arrayName = [item0, item1, ...];  
  
var emptyArray = [];  
  
  
  
  
var subjectList = ["ISIT206", "MATH121", "CSCI301"];  
subjectList[1] = "LOGIC101";  
subjectList[6] = "LAW201"; // this will create holes in array  
  
  
  
  
// loop through an array  
  
for(var i = 0; i < subjectList.length; i++) {  
    alert(subjectList[i]);  
}  
}
```

JavaScript: Object

Object is defined by a list of `property:value`

```
var objectName = {property1:value1, property2:value2, ...};
```

```
var emptyObject = {};
```

```
var info = {  
    name: "John",  
    dob: new Date("1996-01-20"),  
    year: 2  
};
```

Object values can be obtained by **two ways**:

`obj.property`

`obj["property"]`

```
// two ways:  
info.year  
info["year"]
```

JavaScript: Array vs Object

```
var arrayName = [ item0, item1, ... ] ;
```

```
var objectName = { property1:value1, property2:value2, ... } ;
```

Arrays use numbered index:

```
arrayName[0] = "LOGIC101";
```

```
arrayName[1] = "CSCI111";
```

Objects use named index:

```
objectName["firstName"] = "John";
```

```
objectName.lastName = "Lee";
```

JavaScript Object Notation (JSON)

- In most web applications, XML and JSON are used to store or transport data
- JSON is "self-describing" and easy to understand

This is an example of a JSON describing a student object:

```
{  
  "fullname": "John Smith",  
  "studentNumber": "U1234567",  
  "age": 20,  
  "csMajor": true  
}
```

JSON

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects

```
{  
  "fullname": "John Smith",  
  "studentNumber": "U1234567",  
  "age": 20,  
  "csMajor": true  
}
```

JSON

Square brackets hold arrays

```
[  
  {  
    "firstName": "John",  
    "lastName": "Smith"  
  },  
  {  
    "firstName": "Kate",  
    "lastName": "Williams"  
  }]  
]
```

JSON

- Curly braces hold objects
- Square brackets hold arrays

```
{  
  "firstName": "John",  
  "lastName": "Smith",  
  "subjectList": [  
    {  
      "code": "MATH101",  
      "title": "Algebra"  
    },  
    {  
      "code": "CSIT122",  
      "title": "C programming"  
    }  
  ]  
}
```

JSON

Translate from Javascript object to JSON string

```
objJSON = JSON.stringify(obj);
```

Translate from JSON string to javascript object

```
obj = JSON.parse(objJSON);
```

JSON

OBJECT

```
{  
  fullname: "John Smith",  
  studentNumber: "U1234567",  
  age: 20,  
  csMajor: true  
}
```

~~JSON.stringify~~

~~JSON.parse~~

JSON

```
{  
  "fullname": "John Smith",  
  "studentNumber": "U1234567",  
  "age": 20,  
  "csMajor": true  
}
```

JSON.stringify

The **JSON.stringify** method converts a JavaScript value to a JSON string.

Syntax: `JSON.stringify(jsvalue, replacer, space)`

- `jsvalue`: the javascript value to convert to a JSON string.
- `replacer` (Optional): selecting/filtering which properties of the object to be included in the JSON string. If the `replacer` is null or not provided, all properties of the object are included in the resulting JSON string.
- `space` (Optional): use for indentation, specifying white spaces in the output JSON string for readability purposes.

JSON.stringify function demo

Enter information to construct a student object:

Full name

Student number

Age

CompSci major

Click View buttons to see JSON string of the student object.

View `JSON.stringify(studentObj)`

```
{"fullname": "John Smith", "studentNumber": "U1234567", "age": 20, "csMajor": false}
```

View `JSON.stringify(studentObj, null, 2)`

```
{
  "fullname": "John Smith",
  "studentNumber": "U1234567",
  "age": 20,
  "csMajor": false
}
```

View `JSON.stringify(studentObj, ["studentNumber", "csMajor"]);`

```
{"studentNumber": "U1234567", "csMajor": false}
```

View `JSON.stringify(studentObj, ["studentNumber", "csMajor"], 2)`

```
{
  "studentNumber": "U1234567",
  "csMajor": false
}
```

JSON.stringify

```
var studentObj = {  
    fullname: "John Smith",  
    studentNumber: "U1234567",  
    age: 20,  
    csMajor: false  
};
```

JSON.stringify(studentObj)

```
{"fullname": "John Smith", "studentNumber": "U1234567", "age": 20,  
"csMajor": false}
```

output JSON sticks together
make it hard to read

JSON.stringify

```
var studentObj = {  
    fullname: "John Smith",  
    studentNumber: "U1234567",  
    age: 20,  
    csMajor: false  
};
```

`JSON.stringify(studentObj, null, 2)`

using 2 spaces indentation

```
{  
    "fullname": "John Smith",  
    "studentNumber": "U1234567",  
    "age": 20,  
    "csMajor": false  
}
```

JSON.stringify

```
var studentObj = {  
    fullname: "John Smith",  
    studentNumber: "U1234567",  
    age: 20,  
    csMajor: false  
};
```

JSON.stringify(studentObj, ["studentNumber", "csMajor"])

only output the student number
and compsci major

```
{"studentNumber": "U1234567", "csMajor": false}
```

JSON.stringify

```
var studentObj = {  
    fullname: "John Smith",  
    studentNumber: "U1234567",  
    age: 20,  
    csMajor: false  
};
```

```
JSON.stringify(studentObj, ["studentNumber", "csMajor"], 2)
```

only output the student number
and compsci major, using 2
spaces indentation

```
{  
    "studentNumber": "U1234567",  
    "csMajor": false  
}
```

Example 1: JSON.stringify

```
function showObjectJSON() {  
    //create a student object  
    var studentObj = {};  
    studentObj.fullname = "John Smith";  
    studentObj.studentNumber = "U1234567";  
    studentObj.age = 20;  
    studentObj.csMajor = true;  
  
    //get JSON string from the javascript object  
    var studentJSON = JSON.stringify(studentObj);  
  
    //print the JSON string to the console  
    console.log(studentJSON);  
}  
  
<button onClick="showObjectJSON()">  
Click here to see JSON string  
</button>
```

Example 2: JSON.parse

```
function showObject() {
    //JSON string
    var studentJSON = '{"fullname":"John Smith","studentNumber":
    "U1234567","age":20,"csMajor":true}';

    //get javascript object from JSON string
    var studentObj = JSON.parse(studentJSON);

    //print the object to the console
    console.log(studentObj);
    console.log("Full name is " + studentObj.fullname);
}

<button onClick="showObject()">
Click here to see object from JSON
</button>
```

Example 3: JSON.stringify

```
function showArrayJSON() {  
    var user1 = {};  
    user1.firstName = "John";  
    user1.lastName = "Smith";  
  
    var user2 = {};  
    user2.firstName = "Kate";  
    user2.lastName = "Williams";  
  
    //create an array of user objects  
    var userList = [user1, user2];  
  
    //get JSON string from the javascript array  
    var userListJSON = JSON.stringify(userList);  
  
    //print the JSON string to the console  
    console.log(userListJSON);  
}  
  
                                <button onClick="showArrayJSON()">  
                                Click here to see JSON string  
                                </button>
```

Example 4: JSON.parse

```
function showArray() {  
    //JSON string  
    var userListJSON = '[{"firstName":"John","lastName":"Smith"},  
                        {"firstName":"Kate","lastName":"Williams"}]';  
  
    //get javascript array from JSON string  
    var userList = JSON.parse(userListJSON);  
  
    //print the object to the console  
    console.log(userList);  
    console.log("There are " + userList.length + " users");  
}  
  
<button onClick="showArray()">  
Click here to see array from JSON  
</button>
```

Example 5: JSON.stringify

```
function showObjectJSON() {  
    var studentObj = {} ; //create a student object  
    studentObj.firstName = "John";  
    studentObj.lastName = "Smith";  
    studentObj.subjectList = [] ; //empty array to hold subjects  
  
    var subjectObj1 = {} ;  
    subjectObj1.code = "MATH101";  
    subjectObj1.title = "Algebra";  
    //add subject into array  
    studentObj.subjectList.push(subjectObj1);  
  
    var subjectObj2 = {} ;  
    subjectObj2.code = "CSIT122";  
    subjectObj2.title = "C programming";  
    //add subject into array  
    studentObj.subjectList.push(subjectObj2);  
  
    //get JSON string from obj and print it on console  
    var studentJSON = JSON.stringify(studentObj, null, 2);  
    console.log(studentJSON);  
}
```

Example 5: JSON.stringify

```
{  
    "firstName": "John",  
    "lastName": "Smith",  
    "subjectList": [  
        {  
            "code": "MATH101",  
            "title": "Algebra"  
        },  
        {  
            "code": "CSIT122",  
            "title": "C programming"  
        }  
    ]  
}
```

References

- <http://www.w3schools.com/json>
- <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON>

Introduction to Web Technology

AJAX

Joseph Tonien

School of Computing and Information Technology
University of Wollongong

AJAX: asynchronous JavaScript and XML

Consider the following scenario:

Suppose we want to build a website about Wollongong.
We want to display information about

- Accommodation
- Attractions
- Events
- Restaurants
- Timetable
- Weather

AJAX

Wollongong



Restaurants



PEPE's on the beach



Coconut Thai Restaurant



Outback Steakhouse

Wollongong NSW

Friday 12:00 pm
Sunny



23 °C | °F

Precipitation: 0%
Humidity: 47%
Wind: 13 km/h

Temperature Precipitation Wind



Accommodation



Adina Apartment Hotel
Wollongong

From \$140 per night



Austinmer Gardens Bed
and Breakfast

From \$108 per night



Austinmer Sur La Mer B&B

From \$175 per night

Events

Station details

Wollongong timetable

Address: Lowden Square, Wollongong
Telephone: 4223 5517

Lines serviced:

- South Coast Line
- Southern Highlands Line



AJAX

Wollongong



Restaurants

loading restaurants information...

loading weather information...

Accommodation

loading accommodation information...

loading train timetable...

Events

loading events information...

AJAX

if we use synchronous calls to load informations

- loading info 1...
- loading info 2...
- loading info 3...
- ...

then the webpage will froze and is not responsive during the loading.

What happen if one of these calls fails?

AJAX

asynchronous allows us to send all the requests simultaneously and register **callback functions**

- sending request 1... if success then do this callback1
- sending request 2... if success then do this callback2
- sending request 3... if success then do this callback3
- ...
- request 2 success -> evoke callback2 function
- request 3 success -> evoke callback3 function
- request 1 success -> evoke callback1 function
- ...

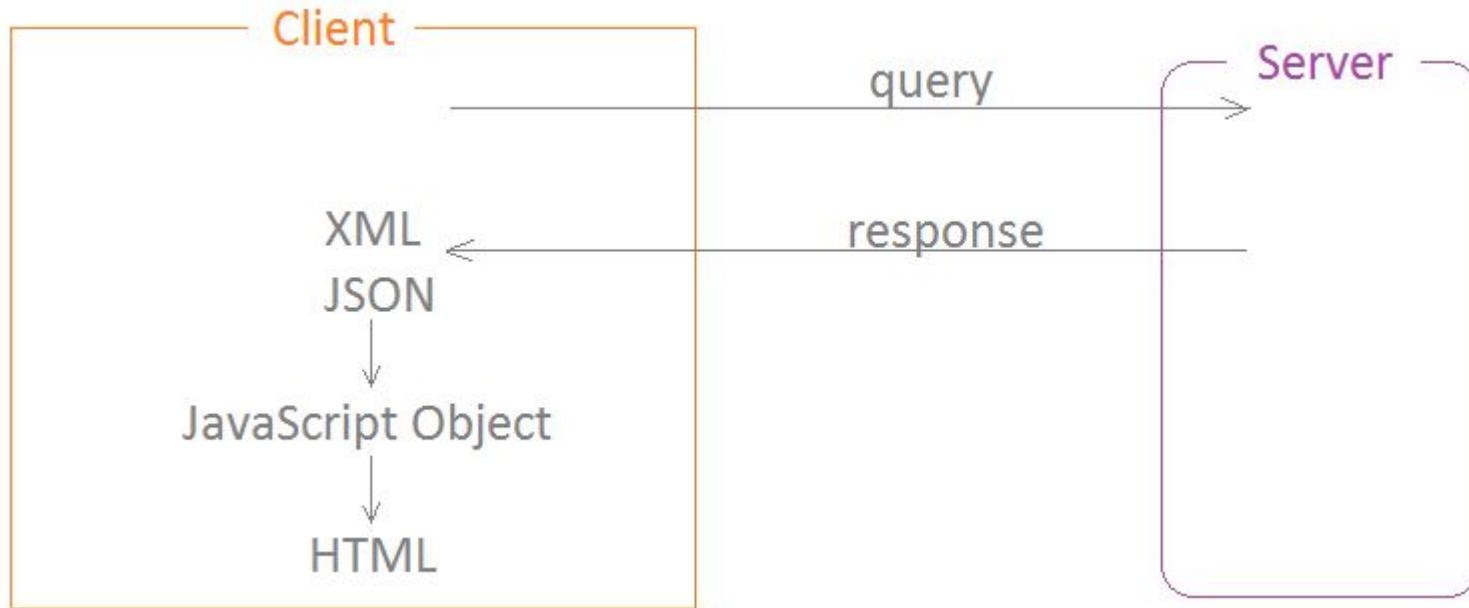
AJAX

With Ajax we can

- update a web page without reloading the page
- request data from a server - after the page has loaded
- receive data from a server - after the page has loaded
- send data to a server - in the background

Despite the name, the use of XML is not required, we can also use JSON as an alternative.

AJAX



Writing AJAX/JSON application:

- **Step 1:** Make the query
- **Step 2:** Get the response JSON
- **Step 3:** Parse the JSON response into a JavaScript object
- **Step 4:** Display the JavaScript object in a HTML page

A sample AJAX/JSON program

A sample AJAX/JSON program

This is the main function:

Step 1: Make the query

```
function makeAjaxQuery() {  
    // create an XMLHttpRequest  
    var xhttp = new XMLHttpRequest();  
  
    // create a handler for the readyState change  
    xhttp.onreadystatechange = function() {  
        readyStateChangeHandler(xhttp);  
    };  
  
    // making query by async call  
    xhttp.open("GET", "url-to-query-the-server", true);  
    xhttp.send();  
}  
  
// handler for the readyState change  
function readyStateChangeHandler(xhttp) { ... }
```

A sample AJAX/JSON program

This is the callback function:

```
// handler for the readyState change
function readyStateChangeHandler(xhr) {
    if (xhr.readyState == 4) {
        // readyState = 4 means DONE
        if(xhr.status == 200) {
            // status = 200 means OK
            handleStatusSuccess(xhr);
        }else{
            // status is NOT OK
            handleStatusFailure(xhr);
        }
    }
}

// XMLHttpRequest failed
function handleStatusFailure(xhr) { ... }

// XMLHttpRequest success
function handleStatusSuccess(xhr) { ... }
```

A sample AJAX/JSON program

```
// XMLHttpRequest success
function handleStatusSuccess ( xhttp ) {

    var jsonText = xhttp.responseText; ← Step 2: Get the response JSON

    // parse the json into an object
    var obj = JSON.parse(jsonText); ← Step 3: Parse the JSON response  
into a JavaScript object

    // display the object on the page
    display(obj); ← Step 4: Display the object  
in a HTML page
}
```

A sample AJAX/JSON program

```
// parse the json into an object  
var obj = JSON.parse(jsonText);
```

Step 3: Parse the JSON response into a JavaScript object.

Note that this step is done by an easy function call `JSON.parse()`

```
// display the object on the page  
function display(obj) {  
    // construct HTML code to display the object  
    ...  
}
```

Step 4: Display the object in a HTML page

*The main job the AJAX/JSON program is to write the function: **display***

AJAX/JSON Example:

Weather Forecast

This example emulates an application where a server allows the user to retrieve current weather forecast for a queried location.

[Get Weather JSON](#)

Wollongong

Mostly Cloudy

21 °C

Humidity: 66%
Wind speed: 18 km/h

AJAX/JSON Example: Weather Forecast

The purpose of this example is

- to show how to distinguish between a failed request and a successful request
- when the request is failed, display an error message
- when the request is successfully then display the weather information:
 1. parse the JSON response to a JavaScript weather object;
 2. display the weather object on the web page.

AJAX/JSON Example: Weather Forecast

Get Weather JSON

Wollongong

Mostly Cloudy

21 °C

Humidity: 66%

Wind speed: 18 km/h

```
<button onClick="makeAjaxQueryWeather () ">
    Get Weather JSON
</button>

<br /><br />

<div id="display">
</div>

function makeAjaxQueryWeather () {
    // create an XMLHttpRequest
    var xhttp = new XMLHttpRequest () ;
    // create a handler for the readyState change
    xhttp.onreadystatechange = function()  {
        readyStateChangeHandler(xhttp);
    };
    // get JSON file by making async call
    xhttp.open("GET", "weather.json", true);
    xhttp.send();
}
```

AJAX/JSON Example: Weather Forecast

```
// handler for the readyState change

function readyStateChangeHandler(xhr) {
    if (xhr.readyState == 4) {
        // readyState = 4 means DONE
        if(xhr.status == 200) {
            // status = 200 means OK
            handleStatusSuccess(xhr);
        } else{
            // status is NOT OK
            handleStatusFailure(xhr);
        }
    }
}

function handleStatusFailure(xhr) { ... }

function handleStatusSuccess(xhr) { ... }
```

AJAX/JSON Example: Weather Forecast

When the request is failed, display an error message

```
// XMLHttpRequest failed
function handleStatusFailure(xhr) {

    // display error message

    var displayDiv = document.getElementById("display");

    displayDiv.innerHTML = "XMLHttpRequest failed: status " + xhr.status;
}
```

AJAX/JSON Example: Weather Forecast

When the request is successful

```
// XMLHttpRequest success
function handleStatusSuccess(xhttp) {
    var jsonText = xhttp.responseText; ← Get the response JSON
    // parse the json into an object
    var weatherObj = JSON.parse(jsonText); ← Parse the JSON response  
into a JavaScript object
    // display the object on the page
    displayWeather(weatherObj); ← Display the object in a  
HTML page
}
```

AJAX/JSON Example: Weather Forecast

```
// parse the json into an object  
var weatherObj = JSON.parse(jsonText);
```

What is the weatherObj look like?

```
{  
  "queryLocation": "Wollongong",  
  "forecast": "Mostly Cloudy",  
  "temperature": {  
    "degree": "21",  
    "scale": "C"  
  },  
  "humidity": "66%",  
  "windSpeed": "18 km/h"  
}
```



```
weatherObj {  
  queryLocation: "Wollongong",  
  forecast: "Mostly Cloudy",  
  temperature: {  
    degree: "21",  
    scale: "C"  
  },  
  humidity: "66%",  
  windSpeed: "18 km/h"  
}
```

parse the JSON response into a JavaScript object

AJAX/JSON Example: Weather Forecast

```
// display the weather object on the page
function displayWeather(weatherObj) {
    ...
}
```

```
weatherObj {
    queryLocation: "Wollongong",
    forecast: "Mostly Cloudy",
    temperature
        degree: "21",
        scale: "C"
    },
    humidity: "66%",
    windSpeed: "18 km/h"
}
```

Wollongong

Mostly Cloudy

21°C

Humidity: 66%

Wind speed: 18 km/h

We need to construct the
following **HTML code** to
display the weather
information

```
<h1>Wollongong</h1>
<font size='5' color='gray'>Mostly Cloudy</font>
<br /><br />

<font size='7'>21</font>
&deg; C
<br /><br />

<i>Humidity: 66%</i>
<br />

<i>Wind speed: 18 km/h</i>
```

AJAX/JSON Example: Weather Forecast

```
// display the weather object on the page
function displayWeather(weatherObj) {
    ...
}
```

```
weatherObj {
    queryLocation: "Wollongong",
    forecast: "Mostly Cloudy",
    temperature
        degree: "21",
        scale: "C"
    },
    humidity: "66%",
    windSpeed: "18 km/h"
}
```

Wollongong
Mostly Cloudy
21°C
Humidity: 66%
Wind speed: 18 km/h

Q: How to we get the query location?

A:

weatherObj.queryLocation

```
<h1>Wollongong</h1>
<font size='5' color='gray'>Mostly Cloudy</font>
<br /><br />

<font size='7'>21</font>
&deg; C
<br /><br />

<i>Humidity: 66%</i>
<br />

<i>Wind speed: 18 km/h</i>
```

AJAX/JSON Example: Weather Forecast

```
// display the weather object on the page
function displayWeather(weatherObj) {
    ...
}
```

```
weatherObj {
    queryLocation: "Wollongong",
    forecast: "Mostly Cloudy",
    temperature: {
        degree: "21",
        scale: "C"
    },
    humidity: "66%",
    windSpeed: "18 km/h"
}
```

Wollongong
Mostly Cloudy
21°C
Humidity: 66%
Wind speed: 18 km/h

Q: How to we get the temperature scale?

A:

`weatherObj.temperature.scale`

```
<h1>Wollongong</h1>
<font size='5' color='gray'>Mostly Cloudy</font>
<br /><br />

<font size='7'>21</font>
&deg; C
<br /><br />

<i>Humidity: 66%</i>
<br />

<i>Wind speed: 18 km/h</i>
```

AJAX/JSON Example: Weather Forecast

```
// display the weather object on the page
function displayWeather(weatherObj) {
    // construct HTML code to display weather information
    var html = "<h1>" + weatherObj.queryLocation + "</h1>";

    html = html + "<font size='5' color='gray'>" + weatherObj.forecast + "</font>";
    html = html + "<br /><br />";

    html = html + "<font size='7'>" + weatherObj.temperature.degree + "</font>";
    html = html + "&deg;" + weatherObj.temperature.scale;
    html = html + "<br /><br />";

    html = html + "<i>Humidity: " + weatherObj.humidity + "</i>";
    html = html + "<br />";

    html = html + "<i>Wind speed: " + weatherObj.windSpeed + "</i>";

    // show the constructed HTML code in the display div
    var displayDiv = document.getElementById("display");
    displayDiv.innerHTML = html;
}
```

Wollongong

Mostly Cloudy

21 °C

Humidity: 66%
Wind speed: 18 km/h

AJAX/JSON Example:

Stock Market

This example emulates an application where a server allows the user to retrieve stock market information.

AJAX/JSON Example: Stock Market

Assume that there is a JSON file, called `market.json`. Write HTML and JavaScript codes that do the following:

There is a button “Click here to view Stock Market Activity”. When the user clicks on this button, make an Ajax call to get the stock information from the json file and display them in a table.

[Click here to view Stock Market Activity](#)

Stock Market Activity 24/02/2015 11:30:00

Stock	Value	Change	Net / %
NASDAQ	4725.64	-37.58▼	0.79%
NASDAQ-100 (NDX)	4312.01	-29.38▼	0.68%
Pre-Market (NDX)	4316.29	-25.1▼	0.58%
After Hours (NDX)	4320.61	8.6▲	0.2%
DJIA	17651.26	-99.65▼	0.56%
S&P 500	2051.12	-12.25▼	0.59%
Russell 2000	1113.13	-8.62▼	0.77%

AJAX/JSON Example: Stock Market

This is the content of the JSON file `market.json`

```
{  
  "queryTime": "24/02/2015 11:30:00",  
  "stockList": [  
    {  
      "name": "NASDAQ",  
      "value": 4725.64,  
      "change": -37.58,  
      "netpct": 0.79  
    },  
    {  
      "name": "NASDAQ-100 (NDX)",  
      "value": 4312.01,  
      "change": -29.38,  
      "netpct": 0.68  
    },  
    ....  
    {  
      "name": "Russell 2000",  
      "value": 1113.13,  
      "change": -8.62,  
      "netpct": 0.77  
    }  
  ]  
}
```

Version 0 - plain display

```
// display the market object on the page
function displayMarket(marketObj) {
    // construct HTML code to display market information
    var html = "";

    html += "queryTime: " + marketObj.queryTime;
    html += "<br /><br />";

    for(var i=0; i < marketObj.stockList.length; i++) {
        var stockObj = marketObj.stockList[i];

        html += "name: " + stockObj.name;
        html += "<br />";

        html += "value: " + stockObj.value;
        html += "<br />";

        html += "change: " + stockObj.change;
        html += "<br />";

        html += "netpct: " + stockObj.netpct;
        html += "<br /><br />";
    }

    // show the constructed HTML code in the display div
    var displayDiv = document.getElementById("display");
    displayDiv.innerHTML = html;
}
```

```
marketObj {
    queryTime: "24/02/2015 11:30:00",
    stockList: [
        {
            name: "NASDAQ",
            value: 4725.64,
            change: -37.58,
            netpct: 0.79
        },
        {
            name: "NASDAQ-100 (NDX)",
            value: 4312.01,
            change: -29.38,
            netpct: 0.68
        },
        ...
        {
            name: "Russell 2000",
            value: 1113.13,
            change: -8.62,
            netpct: 0.77
        }
    ]
}

queryTime: 24/02/2015 11:30:00
name: NASDAQ
value: 4725.64
change: -37.58
netpct: 0.79
name: NASDAQ-100 (NDX)
value: 4312.01
change: -29.38
netpct: 0.68
...
name: Russell 2000
value: 1113.13
change: -8.62
netpct: 0.77
```

Version 1 - table display

```
// display the object on the page
function displayMarket(marketObj) {
    ...
}
```

```
<h2>Stock Market Activity 24/02/2015 11:30:00</h2>
```

```
<table border='1'>

<tr> <th>Stock</th> <th>Value</th> <th>Change</th> <th>Net / %</th> </tr>

<tr>
    <td><b>NASDAQ</b></td>
    <td align='right'> 4725.64</td>
    <td style='color:red' align='right'>
        -37.58
        <img src='stockDown.png' />
    </td>
    <td align='right'> 0.79%</td>
</tr>

<tr>
    <td><b>After Hours (NDX)</b></td>
    <td align='right'> 4320.61</td>
    <td style='color:green' align='right'>
        8.6
        <img src='stockUp.png' />
    </td>
    <td align='right'> 0.2%</td>
</tr>
</table>
```

```
marketObj{
    queryTime: "24/02/2015 11:30:00",
    stockList: [
        {
            name: "NASDAQ",
            value: 4725.64,
            change: -37.58,
            netpct: 0.79
        },
        {
            name: "NASDAQ-100 (NDX)",
            value: 4312.01,
            change: -29.38,
            netpct: 0.68
        },
        ...
    ]
}
```

We need to construct the following HTML code to display the stock market information

Stock Market Activity 24/02/2015 11:30:00

Index	Value	Change	Net / %
NASDAQ	4725.64	-37.58▼	0.79%
NASDAQ-100 (NDX)	4312.01	-29.38▼	0.68%
Pre-Market (NDX)	4316.29	-25.1▼	0.58%
After Hours (NDX)	4320.61	8.6▲	0.2%
DJIA	17651.26	-99.65▼	0.56%
S&P 500	2051.12	-12.25▼	0.59%
Russell 2000	1113.13	-8.62▼	0.77%

Version 1 - table display

```
// display the market object on the page
function displayMarket(marketObj) {
    // construct HTML code to display market information
    var html = "<h2>Stock Market Activity " + marketObj.queryTime + "</h2>";
    html += "<table border='1'>";
    html += "<tr><th>Stock</th><th>Value</th><th>Change</th><th>Net / %</th></tr>";
    for(var i=0; i < marketObj.stockList.length; i++) {
        var stockObj = marketObj.stockList[i];
        html += "<tr>";
        html += "<td><b>" + stockObj.name + "</b></td>";
        html += "<td align='right'>" + stockObj.value + "</td>";
        if(stockObj.change < 0) {
            html += "<td style='color:red' align='right'>";
            html += stockObj.change;
            html += "<img src='stockDown.png' />";
            html += "</td>";
        } else{
            html += "<td style='color:green' align='right'>";
            html += stockObj.change;
            html += "<img src='stockUp.png' />";
            html += "</td>";
        }
        html += "<td align='right'>" + stockObj.netpct + "%</td>";
        html += "</tr>";
    }
    html += "</table>";
    // show the constructed HTML code in the display div
    var displayDiv = document.getElementById("display");
    displayDiv.innerHTML = html;
}
```

```
marketObj {
    queryTime: "24/02/2015 11:30:00",
    stockList: [
        {
            name: "NASDAQ",
            value: 4725.64,
            change: -37.58,
            netpct: 0.79
        },
        {
            name: "NASDAQ-100 (NDX)",
            value: 4312.01,
            change: -29.38,
            netpct: 0.68
        },
        ...
        {
            name: "Russell 2000",
            value: 1113.13,
            change: -8.62,
            netpct: 0.77
        }
    ]
}
```

Stock Market Activity 24/02/2015 11:30:00

Index	Value	Change	Net / %
NASDAQ	4725.64	-37.58	0.79%
NASDAQ-100 (NDX)	4312.01	-29.38	0.68%
Pre-Market (NDX)	4316.29	-25.1	0.58%
After Hours (NDX)	4320.61	8.6	0.2%
DJIA	17651.26	-99.65	0.56%
S&P 500	2051.12	-12.25	0.59%
Russell 2000	1113.13	-8.62	0.77%

References

- <http://www.w3schools.com/json>
- Robert W. Sebesta, *Programming the World Wide Web*, Pearson.

Introduction to Web Technology

HTML5: Graphic Canvas, Drag and Drop

Joseph Tonien
School of Computing and Information Technology
University of Wollongong

HTML 5

Canvas

- First introduced in WebKit by Apple for the OS X Dashboard, Graphic Canvas has since been implemented in other major browsers.
- Canvas is used to draw graphics, such as paths, boxes, circles, text, and images, on the fly, via JavaScript.

HTML 5

Drag and Drop

- Drag and Drop enable applications to use drag and drop features in browsers.
- The user can select draggable elements with a mouse, drag the elements to a droppable element, and drop those elements by releasing the mouse button.

Canvas

The <canvas> element is used to draw graphics on a web page.

```
<canvas id="mycanvas" width="1000" height="500"  
style="border:1px solid black;">  
Your browser does not support canvas.  
</canvas>
```

Canvas

The <canvas> element is used to draw graphics on a web page.

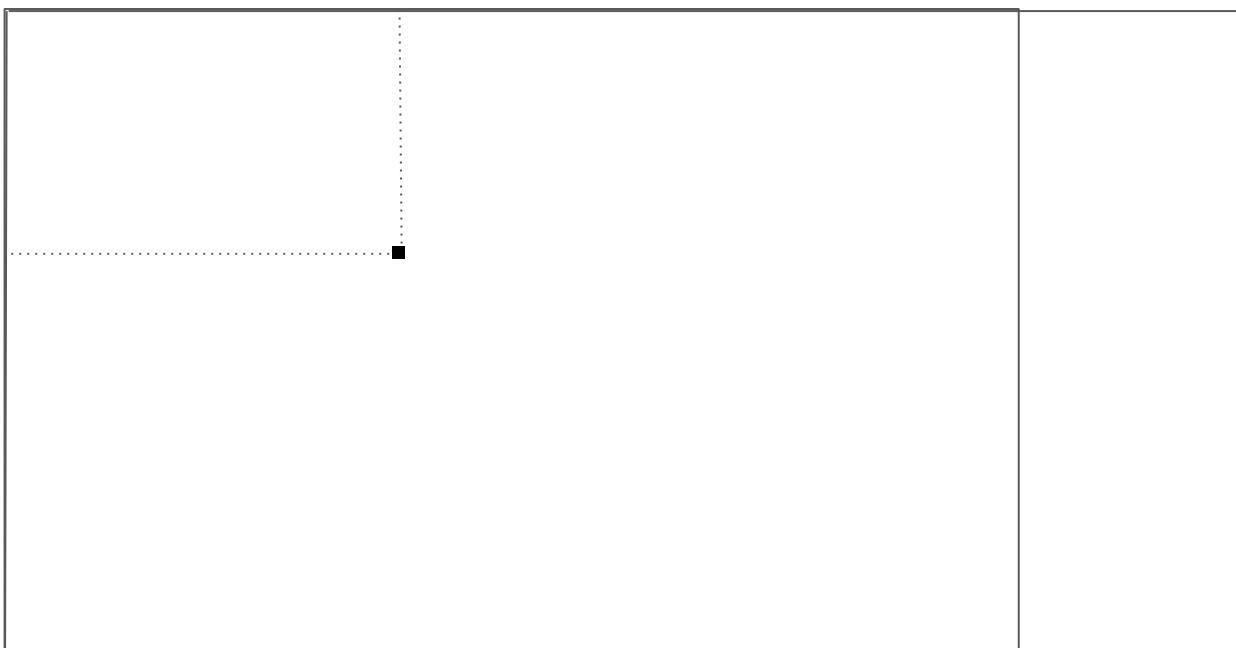
```
<canvas id="mycanvas" width="1000" height="500"  
style="border:1px solid black;">  
Your browser does not support canvas.  
</canvas>
```

The <canvas> element is only a container for the graphics. We must use JavaScript to actually draw the graphics content.

Canvas

(0,0)

x axis



y axis

Each point on the canvas has a coordinate (x,y)

Canvas

CanvasRenderingContext2D is used for drawing text, images, shapes and other objects onto the canvas element. It provides the 2D rendering context for the drawing surface of a canvas element.

```
// get the canvas's 2d context  
var canvas = document.getElementById("the-canvas-id");  
var context = canvas.getContext("2d");
```

There are other rendering contexts for canvas that are not covered in this subject:

WebGLRenderingContext,
WebGL2RenderingContext

Hello World

HELLO WORLD

Hello World

Start

Hello World

HELLO WORLD

```
<canvas id="canvas" width="1300" height="500"  
       style="border:1px solid black;">  
He // // () // // // () // - // () //  
Your browser does not support canvas.  
</canvas>
```

```
<br /><br /> 
```

```
<button onClick="drawTextHello()">  
Start  
</button>
```

Hello World

HELLO WORLD

Hello World

```
function drawTextHello() {  
    // get the canvas's 2d context  
    // fillText  
  
    // strokeText  
}
```

Hello World

```
<canvas id="canvas" width="1300" height="500"  
        style="border:1px solid black;">  
HELLO WYour browser does not support canvas.  
</canvas>
```

Hello World

Start

```
// get the canvas's 2d context  
  
var canvas = document.getElementById("canvas");  
  
var context = canvas.getContext("2d");
```

Hello World

HELLO WORLD

Hello World

```
// fillText
```

```
context.font = "italic small-caps bold 50px Arial";
```

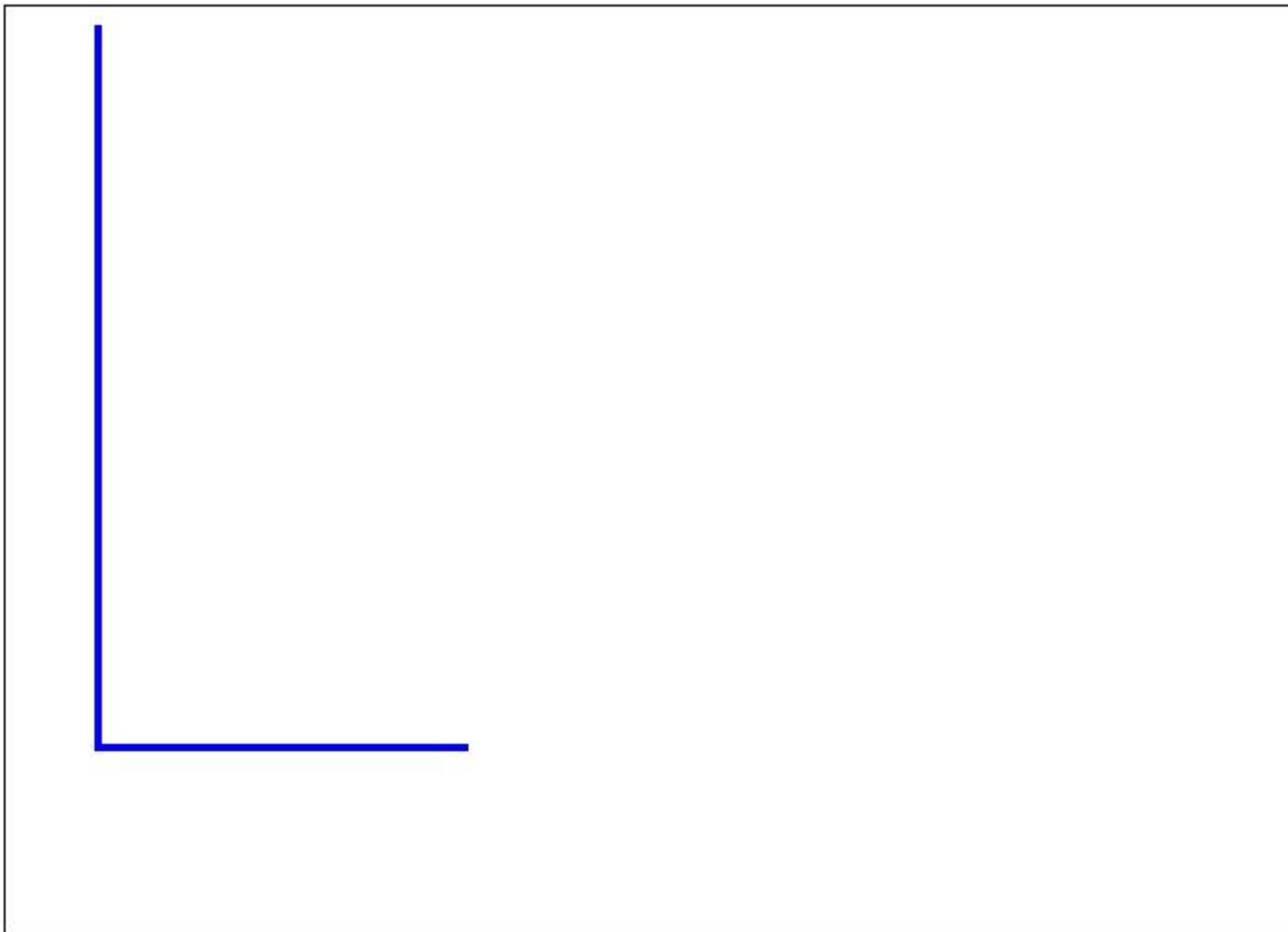
```
context.fillText("Hello World", 200, 100);
```

```
// strokeText
```

```
context.font = "oblique 100px Courier New";
```

```
context.strokeText("Hello World", 250, 300);
```

Stroke Demo 1



Start

Stroke Demo 1

```
<canvas id="canvas" width="700" height="500"  
style="border:1px solid black;">  
Your browser does not support canvas.  
</canvas>  
  
<br /><br />  
  
<button onClick="strokeDemo ()">  
Start  
</button>
```

Start

Stroke Demo 1

```
function strokeDemo () {  
    // get the canvas's 2d context  
  
    // specify the path  
  
    // make the stroke along the path  
}
```

Start

Stroke Demo 1

```
// get the canvas's 2d context
var canvas = document.getElementById("canvas");
var context = canvas.getContext("2d");

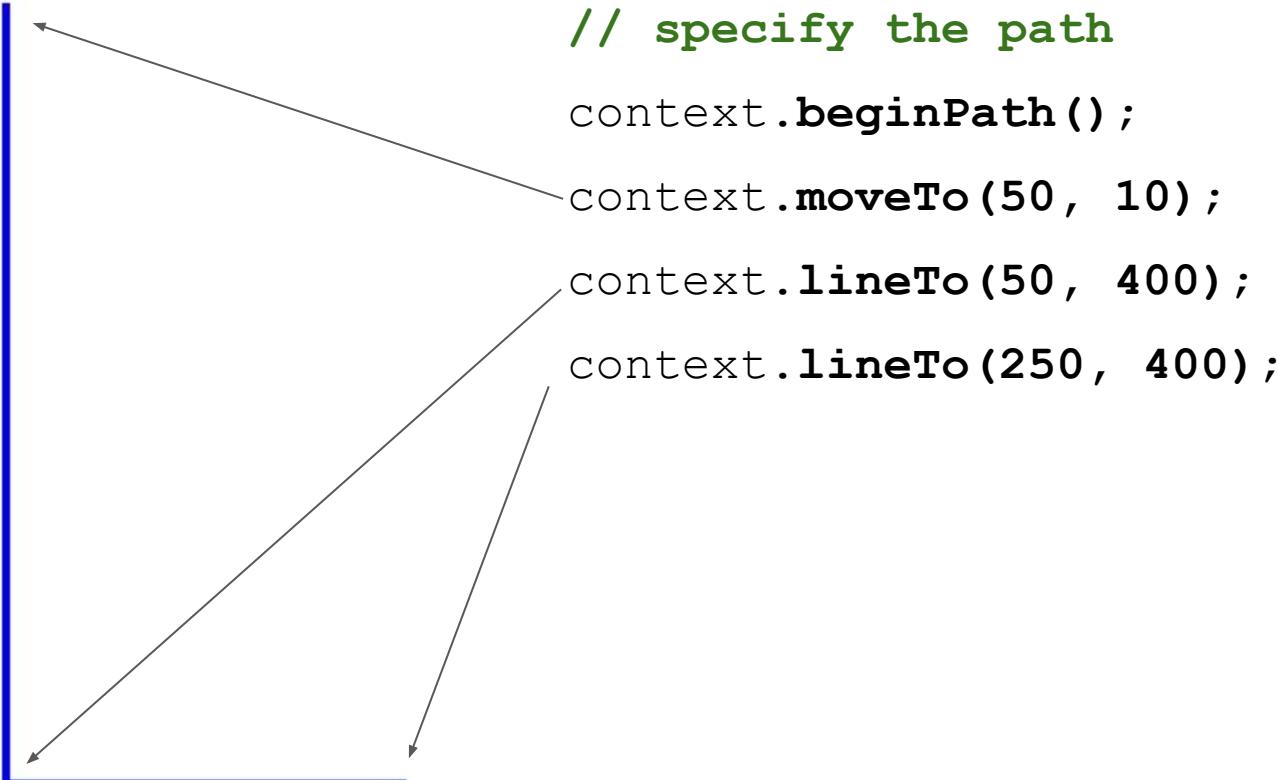
<canvas id="canvas" width="700" height="500"
style="border:1px solid black;">
Your browser does not support canvas.
</canvas>
```

Start

Stroke Demo 1

(0,0)

X



```
// specify the path
context.beginPath();
context.moveTo(50, 10);
context.lineTo(50, 400);
context.lineTo(250, 400);
```

Y

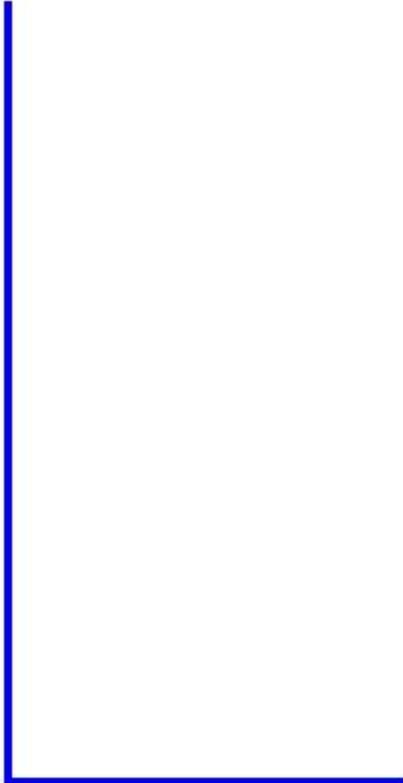
Start

Stroke Demo 1



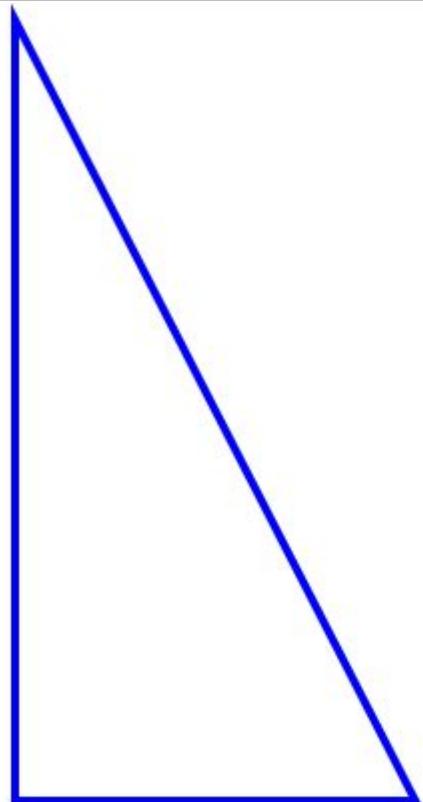
```
// specify the path
context.beginPath();
context.moveTo(50, 10);
context.lineTo(50, 400);
context.lineTo(250, 400);

// make the stroke along the path
context.strokeStyle = "blue";
context.lineWidth = "4";
context.stroke();
```



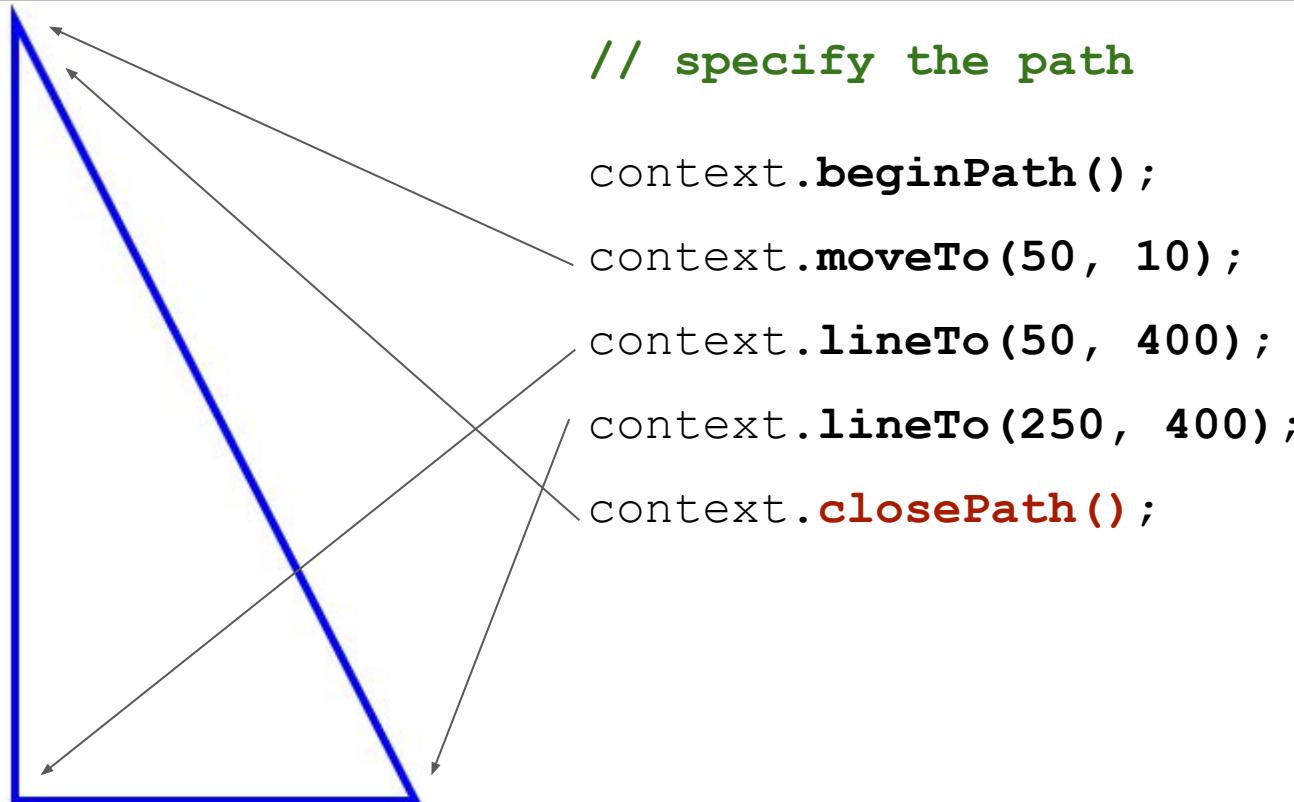
Start

Stroke Demo 2



Start

Stroke Demo 2



Start

Fill Demo 1



Start

Fill Demo 1

```
function fillDemo() {  
    // get the canvas's 2d context  
  
    // specify the path  
  
    // make the fill of the region enclosed by the path  
}
```

Start

Fill Demo 1

```
// get the canvas's 2d context  
var canvas = document.getElementById("canvas");  
var context = canvas.getContext("2d");
```

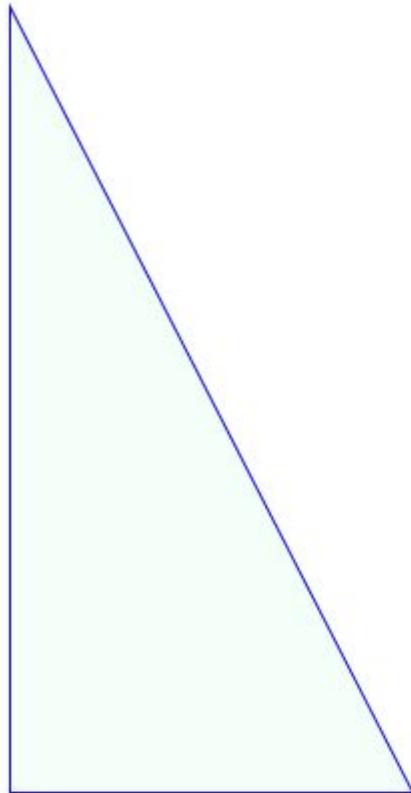
Start

Fill Demo 1

```
// specify the path  
  
context.beginPath();  
  
context.moveTo(50, 10);  
  
context.lineTo(50, 400);  
  
context.lineTo(250, 400);  
  
context.closePath();  
  
// make the fill of the region enclosed by the path  
  
context.fillStyle="#F5FFFA";  
  
context.fill();
```

Start

Fill Demo 2



```
// specify the path
context.beginPath();
context.moveTo(50, 10);
context.lineTo(50, 400);
context.lineTo(250, 400);
context.closePath();

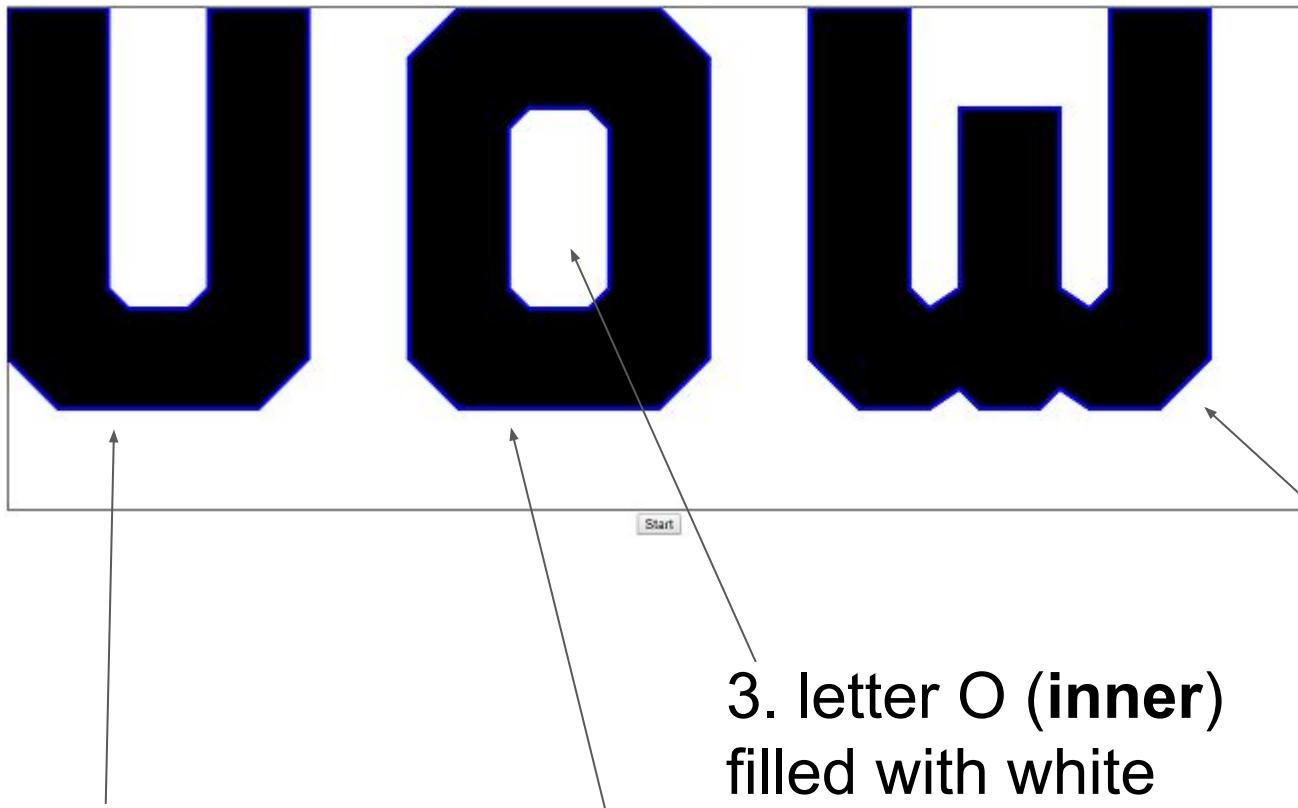
// make the stroke along the path
context.strokeStyle = "blue";
context.lineWidth = "2";
context.stroke();

// make the fill of the region enclosed by the path
context.fillStyle="#F5FFFA";
context.fill();
```

UOW



UOW



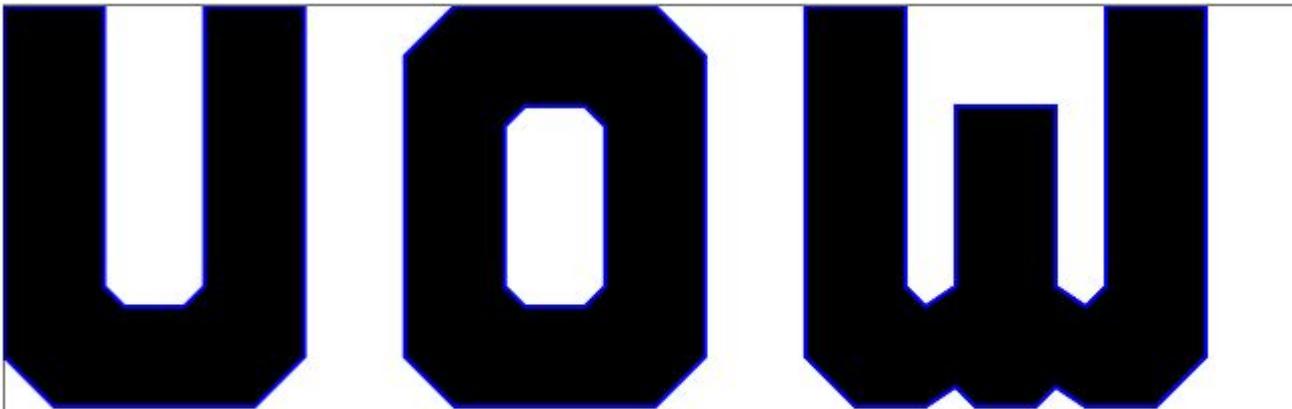
1. letter U
filled with black

2. letter O (**outer**)
filled with black

3. letter O (**inner**)
filled with white

4. letter W
filled with black

UOW

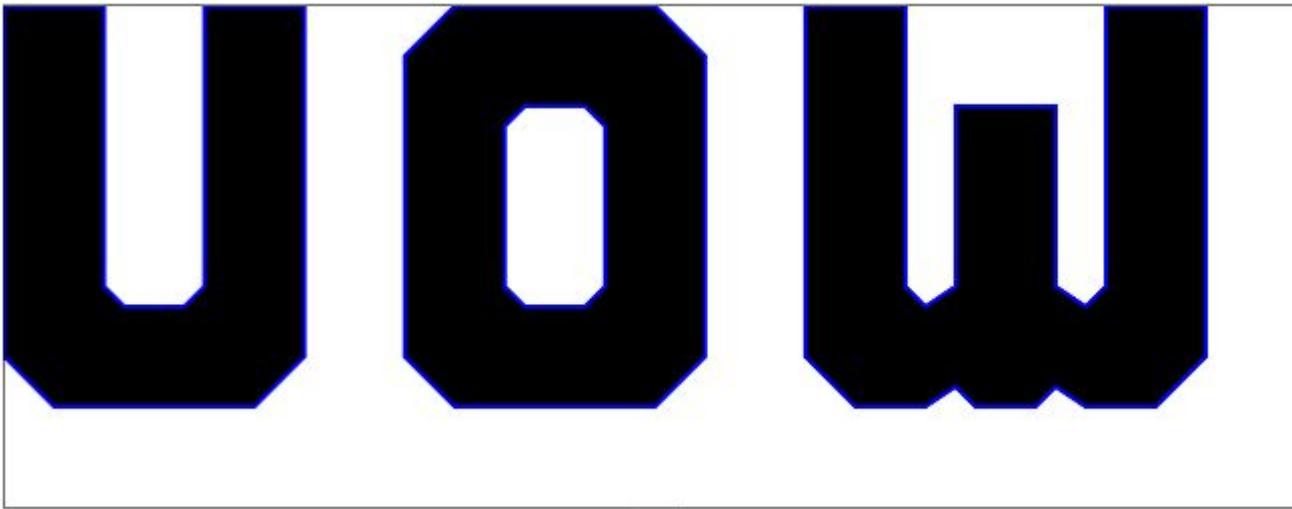


```
<canvas id="canvas" width="1300"
        height="500" style="border:1px solid
        black;">
    Your browser does not support canvas.
</canvas>

<br /><br />

<button onClick="drawUOW()">
    Start
</button>
```

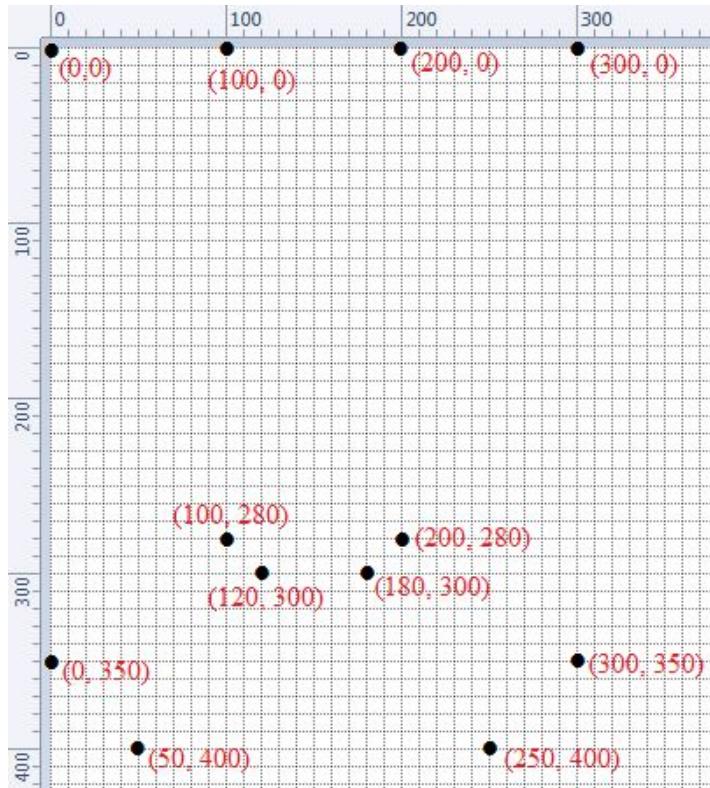
UOW



Start

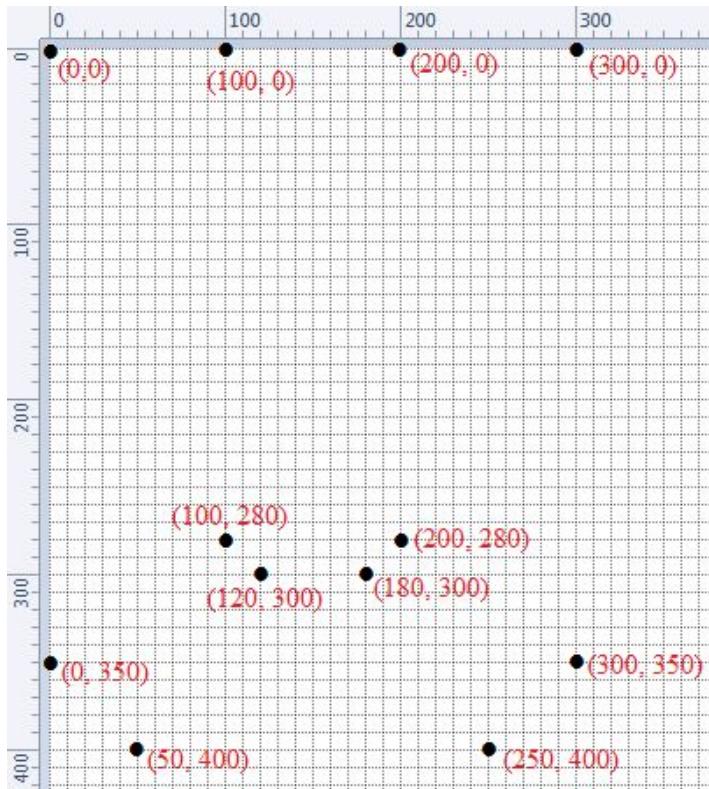
```
function drawUOW() {  
    // get the canvas's 2d context  
    // letter U  
    // letter O (outer)  
    // letter O (inner)  
    // letter W  
}
```

UOW



```
// letter U  
  
context.beginPath();  
  
context.moveTo(0, 0);  
  
context.lineTo(0, 350);  
  
context.lineTo(50, 400);  
  
context.lineTo(250, 400);  
  
context.lineTo(300, 350);  
  
context.lineTo(300, 0);  
  
context.lineTo(200, 0);  
  
context.lineTo(200, 280);  
  
context.lineTo(180, 300);  
  
context.lineTo(120, 300);  
  
context.lineTo(100, 280);  
  
context.lineTo(100, 0);  
  
context.closePath();
```

UOW



```
// letter U  
context.beginPath();  
context.moveTo(0, 0);  
...  
context.lineTo(100, 0);  
context.closePath();  
  
context.fillStyle="black";  
context.fill();  
  
context.strokeStyle="blue";  
context.lineWidth = "4";  
context.stroke();
```

Drag and Drop

Need to specify 2 types of elements:

- ***Draggable elements***: *elements that we can be dragged*
- ***Droppable elements***: *elements that can be dropped on*

The user can select **draggable elements** with a mouse, drag the elements to a **droppable element**, and drop those elements by releasing the mouse button.

Drag and Drop

Need to specify 2 types of elements:

- **Draggable elements**: *elements that we can be dragged*
- **Droppable elements**: *elements that can be dropped on*

```
<element id="drag-id" draggable="true"
onDragStart="dragStart(event)">draggable
element</element>
```

```
<element id="drop-id" onDrop="drop(event)"
onDragOver="dragOver(event)">droppable element</element>
```

Drag and Drop

Draggable elements: elements that we can be dragged

```
<element id="drag-id" draggable="true"
onDragStart="dragStart(event)" >draggable
element</element>
```

```
function dragStart(event) {
    // get the dragged element ID
    var dragId = event.target.id;

    // store the dragged element ID into the
    //dataTransfer object
    event.dataTransfer.setData("dragId", dragId);
}
```

dragStart event is fired when
the user starts dragging an
element

Drag and Drop

Draggable elements: elements that we can be dragged

```
<element id="drag-id" draggable="true"
onDragStart="dragStart(event)" >draggable
element</element>
```

```
function dragStart(event) {
    // get the dragged element ID
    var dragId = event.target.id;
    // store the dragged element ID into the dataTransfer object
    event.dataTransfer.setData("dragId", dragId);
}
```

We need to know what object we are dragging

The DataTransfer object is used to hold the data that is being dragged during a drag and drop operation.

Drag and Drop

Droppable elements: elements that can be dropped on

```
<element id="drop-id" onDrop="drop(event)"  
onDragOver="dragOver(event)">droppable element</element>
```

```
function drop(event) {  
    // get the drop element ID  
  
    var dropId = event.target.id;  
  
    // retrieve the dragged element ID from the dataTransfer object  
  
    var dragId = event.dataTransfer.getData("dragId");  
  
    // do the dropping logic  
  
}
```

The **drop** event is fired when an element is dropped on a valid drop target.

Drag and Drop

Droppable elements: elements that can be dropped on

```
<element id="drop-id" onDrop="drop(event)"  
onDragOver="dragOver(event)">droppable element</element>
```

What is the **dragOver** event for?

Calling the `preventDefault()` method during a **dragOver** event will indicate that a drop is allowed at that location.

```
function dragOver(event) {  
  
    event.preventDefault();  
  
}
```

Drag and Drop: Hello World

Drag an orange word and drop it on a red word.

hello hi bonjour salut

web maze earth world

When “hello” is dropped on “world”, the page displays “hello world”.

hello hi bonjour salut

web maze earth world

hello world

Drag and Drop: Hello World

Drag an **orange** word and drop it on a **red** word.

hello hi bonjour salut

web maze earth world

draggable elements:
elements that we can be
dragged

droppable elements:
elements that can be
dropped on

Drag and Drop: Hello World

Drag an **orange** word and drop it on a **red** word.

hello hi bonjour salut

web maze earth world

draggable elements:
elements that we can drag

```
<span id="hello" draggable="true"
onDragStart="dragStart(event)">hello</span>
```

```
<span id="hi" draggable="true"
onDragStart="dragStart(event)">hi</span>
```

```
<span id="bonjour" draggable="true"
onDragStart="dragStart(event)">bonjour</span>
```

...

Drag and Drop: Hello World

Drag an **orange** word and drop it on a **red** word.

hello hi bonjour salut

web maze earth world

droppable elements:
*elements that can be
dropped on*

```
<span id="web" onDrop="drop(event)"  
onDragOver="dragOver(event)">web</span>
```

```
<span id="maze" onDrop="drop(event)"  
onDragOver="dragOver(event)">maze</span>
```

```
<span id="earth" onDrop="drop(event)"  
onDragOver="dragOver(event)">earth</span>
```

...

Drag and Drop: Hello World

Drag an orange hello onDragStart="dragStart(event)" >hello

web maze earth world

dragStart event is fired when
the user starts dragging an
element

```
function dragStart(event) {  
  
    // get the dragged element ID  
  
    var dragId = event.target.id;  
  
    // store the dragged element ID into the dataTransfer object  
  
    event.dataTransfer.setData("dragId", dragId);  
  
}
```

Drag and Drop: Hello World

```
Drag an orange <span id="hello" draggable="true">  
hello </span>
```

web maze earth world

```
function dragStart(event) {  
    // get the dragged element ID  
    var dragId = event.target.id;  
  
    // store the dragged element ID into the dataTransfer object  
    event.dataTransfer.setData("dragId", dragId);  
}
```

If hello is dragged, then
event.target.id = "hello"
and we store "hello" into the
dataTransfer object

Drag and Drop: Hello World

Drag an orange word and drop it on a red word.

hello hi horizon sunset
`<span id="world" onDrop="drop(event)"
onDragOver="dragOver(event)">world`

web maze earth world

```
function drop(event) {  
    // get the drop element ID  
    var dropId = event.target.id;  
  
    // retrieve the dragged element ID from the dataTransfer object  
    var dragId = event.dataTransfer.getData("dragId");  
  
    // display the message  
    var messageSpan = document.getElementById("message");  
  
    messageSpan.innerHTML = dragId + " " + dropId;  
}
```

The **drop** event is fired when an element is dropped on a valid drop target.

Drag and Drop: Hello World

Drag an **orange** word and drop it on a **red** word.

h~~ell~~ e l i **orange** ~~o~~ut
`<span id="world" onDrop="drop(event)"
onDragOver="dragOver(event)">world`

web maze **earth** **world**

*What is the **dragOver** event for?*

*Calling the `preventDefault()` method during a **dragOver** event will indicate that a drop is allowed at that location.*

```
function dragOver(event) {  
  
    event.preventDefault();  
  
}
```

References

- https://www.w3schools.com/html/html5_canvas.asp
- https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial
- https://www.w3schools.com/html/html5_draganddrop.asp
- https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API

Introduction to Web Technology

HTML5: Client-Side Storage

Joseph Tonien

School of Computing and Information Technology
University of Wollongong

Client-Side Web Storage

- Store data on the client side, instead of the server
- Make the web application available offline
- The storage is per origin (protocol + domain + port)
- Simple storage: data is stored in name/value pair

2 types of storage:

- **localStorage**: a single persistent object which stores data with no expiration date;
- **sessionStorage**: stores data for one session only, data is cleared when the browser tab is closed.

Client-Side Web Storage

Checking if the browser supports web storage or not:

```
// return true if local storage is supported  
// otherwise return false  
  
function storageSupported() {  
    if (typeof(Storage) !== "undefined") {  
        return true;  
    } else {  
        return false;  
    }  
}
```

Client-Side Web Storage

Storing and retrieving data from Web Storage:

```
// storing data to the localStorage  
localStorage.setItem("the-key", "the-value");  
  
// get data from localStorage  
var the-value = localStorage.getItem("the-key");
```

Removing data from Web Storage:

```
// removing data to the localStorage  
localStorage.removeItem("the-key");
```

Example: To-Do-List

We want to create a web application where the user can create a to-do-list and save it to the local storage.

We will store the JSON of the task list into the local storage.

Task: Urgency:

feed the dog	X
go shopping	X
cook lunch	X

Example: To-Do-List

design the HTML elements

Task: Urgency:

feed the dog 

go shopping 

cook lunch 

a text field for user to enter task description

`<input type="text" id="task" />`

Example: To-Do-List

design the HTML elements

Task:

Urgency:

feed the dog 

go shopping 

cook lunch 

a selection for user to select task urgency

```
<select id="urgency">  
    <option value="High">High</option>  
    <option value="Medium">Medium</option>  
    <option value="Low" selected="selected">Low</option>  
</select>
```

Example: To-Do-List

design the HTML elements

Task:

Urgency:

feed the dog 

go shopping 

cook lunch 

button to add task

```
<button onClick="addTask()">  
    Add  
</button>
```

Example: To-Do-List

design the HTML elements

Task: Urgency:

feed the dog 

a div to display all the tasks

go shopping 

```
<div id="taskDisplay">  
</div>
```

cook lunch 

Example: To-Do-List

design the data structure

Task: Urgency:

feed the dog 

{
id:12345xxx...,
task:"feed the dog",
urgency:"High"

go shopping 

{
id:yyy...,
task:"go shopping",
urgency:"Medium"

cook lunch 

{
id:zzz...,
task:"cook lunch",
urgency:"Low"

each task is an object

Example: To-Do-List

design the data structure

Task: Urgency:

feed the dog  *store all tasks into an array variable*

go shopping  → [{ id:12345xxx..., task:"feed the dog", urgency:"High" }, { id:yyy..., task:"go shopping", urgency:"Medium" }]

cook lunch  [{ id:zzz..., task:"cook lunch", urgency:"Low" }]

Example: To-Do-List

design the data structure

Task:

Urgency:

feed the dog 

store all tasks into an array variable

go shopping 

```
// list of tasks
// each task is an object that contains:
// {
//   id: the task id (the time when task created)
//   task: the task name
//   urgency: the task urgency (High, Medium, or Low)
// }
var toDoList = [];
```

cook lunch 

Example: To-Do-List

design the local storage

Task: Urgency:

translate task array into JSON string

toDoList

```
[  
  {  
    id:12345xxx...,  
    task:"feed the dog",  
    urgency:"High"  
  },  
  {  
    id:yyy...,  
    task:"go shopping",  
    urgency:"Medium"  
  },  
  {  
    id:zzz...,  
    task:"cook lunch",  
    urgency:"Low"  
  }]  
]
```

toDoListJSON

```
[  
  {  
    "id":12345xxx...,  
    "task":"feed the dog",  
    "urgency":"High"  
  },  
  {  
    "id":yyy...,  
    "task":"go shopping",  
    "urgency":"Medium"  
  },  
  {  
    "id":zzz...,  
    "task":"cook lunch",  
    "urgency":"Low"  
  }]  
]
```

Example: To-Do-List

design the local storage

Task:

Urgency:

Low



Add

store JSON string into local storage

ToDoListJSON

```
[  
  {  
    "id":12345xxx...,  
    "task":"feed the dog",  
    "urgency":"High"  
  },  
  {  
    "id":yyy...,  
    "task":"go shopping",  
    "urgency":"Medium"  
  },  
  {  
    "id":zzz...,  
    "task":"cook lunch",  
    "urgency":"Low"  
  }  
]
```

Key	Value
ToDoListJSON	[{"id":1605572931427, "task":"feed the dog", "urgency":"High"}, ...]

Example: To-Do-List

Function: Add a task

Task: **feed the dog**

Urgency: **High**  **Add**



get task info from user and create task object



toDoList

add task object to the list of tasks



```
<div id="taskDisplay">  
/</div>
```

display updated list of tasks



store updated list of tasks into local storage

Key	Value
toDoListJSON	[{"id":1605572931427, "task":"feed the dog", "urgency":"High"}, ...]

Example: To-Do-List

Function: Add a task

Task:

Urgency:

```
function addTask() {
  // get task info from user and create task object
  var taskObj = createTask();

  // add task object to the list of tasks
  toDoList.push(taskObj);

  // display updated list of tasks
  displayTasks();

  // store updated list of tasks into local storage
  saveTasksToLocal();
}
```

Example: To-Do-List

Function: Add a task

Task: **feed the dog**

Urgency: **High** ▾ **Add**

```
function createTask() {
    // get task info from user
    var taskTf = document.getElementById( "task" );
    var taskName = taskTf.value;

    var urgencySelect = document.getElementById( "urgency" );
    var taskUrgency = urgencySelect.value;

    // create task object
    var taskObj = {};
    var currentDate = new Date();
    taskObj.id = currentDate.getTime();
    taskObj.task = taskName;
    taskObj.urgency = taskUrgency;

    // return task object
    return taskObj;
}
```

Example: To-Do-List

Function: Display tasks

Task: **feed the dog**

Urgency: **High** ▾ **Add**

```
function displayTasks() {
    // construct the html contains all the tasks
    var html = "";

    // use for loop to go through all the tasks
    for(var i=0; i < toDoList.length; i++) {
        var taskObj = toDoList[i];
        var taskHTML = getTaskHTML(taskObj);
        html = html + taskHTML;
    }

    // display tasks in the DIV
    var displayDiv = document.getElementById( "taskDisplay" );
    displayDiv.innerHTML = html;
}
```

Example: To-Do-List

Function: Display tasks

Task: Urgency:

feed the dog 

```
function getTaskHTML(taskObj) {
    // construct the html for displaying the task
    var html = "<p>";

    // task description in color
    var taskDesc = getTaskDescriptionHTML(taskObj);
    html += taskDesc;

    // task button
    var taskButton = getTaskDeleteButtonHTML(taskObj);
    html += taskButton;

    html += "</p>";

    return html;
}
```

Example: To-Do-List

Function: Display tasks

Task: Urgency:

feed the dog 

```
function getTaskDescriptionHTML(taskObj) {  
    // using different color for the urgency  
    var desc = "";  
    if (taskObj.urgency == "High") {  
        desc = "<span style='color:red;'>"  
            + taskObj.task  
            + "</span>";  
    }  
    else if (taskObj.urgency == "Medium") {  
        desc = "<span style='color:orange;'>"  
            + taskObj.task  
            + "</span>";  
    }  
    else if (taskObj.urgency == "Low") {  
        desc = "<span style='color:green;'>"  
            + taskObj.task  
            + "</span>";  
    }  
    return desc;  
}
```

Example: To-Do-List

Function: Display tasks

Task: Urgency:

feed the dog 

```
function getTaskDeleteButtonHTML(taskObj) {
    var deleteEmoji = "\ud83d\udcfa";
    var deleteButton = "<span onClick='deleteTask(" + taskObj.id + ")" + "')'>" +
                      deleteEmoji +
                      "</span>";
    return deleteButton;
}

function deleteTask(taskId) {
    ...
}
```

Example: To-Do-List

Function: Delete a task

Task:

Urgency:

feed the dog 

toDoList

remove task object from the list of tasks

```
<div id="taskDisplay">  
/</div>
```

display updated list of tasks

store updated list of tasks into local storage

Key	Value
toDoListJSON	[{"id":1605572931624, "task":"cook lunch", "urgency":"Low"}, ...]

Example: To-Do-List

Function: Delete a task

Task:

Urgency: ▾

feed the dog 

```
function deleteTask(taskId) {
    // remove task object from the list of tasks
    // search for task id
    for(var i=0; i < ToDoList.length; i++) {
        var taskObj = ToDoList[i];
        if (taskObj.id == taskId) {
            // delete task out of the task array
            ToDoList.splice(i, 1);
        }
    }

    // display updated list of tasks
    displayTasks();

    // store updated list of tasks into local storage
    saveTasksToLocal();
}
```

Example: To-Do-List

Function: save tasks to local storage

Task:

Urgency:

feed the dog 

go shopping 

cook lunch 

toDoList

translate task array into JSON string

toDoListJSON

store JSON string into local storage

Key	Value
toDoListJSON	[{"id":1605572931235, "task":"feed the dog", "urgency":"High"},...]

Example: To-Do-List

Function: save tasks to local storage

Task: Urgency:

feed the dog 

go shopping 

cook lunch 

```
function saveTasksToLocal() {
    // check if local storage supported
    if(storageSupported()){
        // translate task array into JSON string
        var ToDoListJSON = JSON.stringify(ToDoList);

        // store JSON string into local storage
        localStorage.setItem("ToDoListJSON", ToDoListJSON);
    }
}
```

Example: To-Do-List

Function: save tasks to local storage

Task: Urgency:

feed the dog 

go shopping 

cook lunch 

```
// return true if local storage is supported
// otherwise return false
function storageSupported() {
    if (typeof(Storage) !== "undefined") {
        return true;
    } else {
        return false;
    }
}
```

Example: To-Do-List

Function: load tasks

Task: Urgency:

feed the dog 

go shopping 

cook lunch 

What happens when the user closes the website and then comes back on another day?

Example: To-Do-List

Function: load tasks

Task:

Urgency:

feed the dog 

What happen when the user closes the website and then comes back on another day?

go shopping 

When the website loaded, we need to read the local storage for the saved list of tasks and then we need to display this saved list of tasks.

cook lunch 

Example: To-Do-List

Function: load tasks

Task:

Urgency:

feed the dog 

go shopping 

cook lunch 

Key	Value
ToDoListJSON	[{"id":1605572931235, "task":"feed the dog", "urgency":"High"}, ...]

read saved JSON from local storage

ToDoListJSON

translate JSON string to task array

ToDoList

display list of tasks

```
<div id="taskDisplay">  
/</div>
```

Example: To-Do-List

Function: load tasks

Task:

Urgency:

feed the dog 

```
<body onLoad="loadTasks () ">
```

go shopping 

cook lunch 

```
function loadTasks () {
    // check if local storage supported
    if(storageSupported ()) {
        // read saved JSON from local storage
        var ToDoListJSON = localStorage.getItem( "ToDoListJSON" );

        // translate JSON string to task array
        if(toDoListJSON != null) {
            ToDoList = JSON.parse(toDoListJSON);
        }

        // display list of tasks
        displayTasks ();
    }
}
```

References

- <https://www.w3.org/TR/webstorage/>
- https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API