CE212

Web Application Programming

Part 1

Recommended Reading

There is no required reading for this course; many resources are available on the web. Some books you may want to look at include

SEBESTA, .R.W., Programming the World Wide Web (7th edition, Pearson 2013, available as an e-book from www.coursesmart.co.uk)

FLANAGAN, D., JavaScript: The Definitive Guide (5th edition, O'Reilly 2006)

BERGSTEN, H., Java Server Pages (3rd edition, O'Reilly 2003)

HALL, M. AND BROWN, L., Core Servlets and JavaServer Pages, Vol. 1: Core Technologies (2nd edition, Sun 2003)

Assessment

- Two hour exam in May/June (70% of the module credit)
- Assignment 1 (10% of the module credit); to be handed out in week 17, for submission by Wednesday of week 20
- Assignment 2 (10% of the module credit); to be handed out in week 22, for submission by Monday week 25
- Multiple-choice test (10% of the module credit) in week 21

Introduction

Web application technology plays an increasing role in all aspects of computing.

In this module we will examine several different aspects of web application programming, involving client and server side technologies.

Web 1.0

In the traditional web client-server interaction model

- the user interacts with a web page
- the browser sends data to web server
- the server sends data back to browser
- the browser receives the data and updates the page
- page refresh may be intrusive and irritating

Web 2.0

In the newer model (commonly called Web 2.0) JavaScript is used extensively in the browser and single web-page applications are possible

- the entire page need not be refreshed
- the browser interacts with server by sending and receiving small chunks of data
- these update individual components or data items on a page
- this is also known as AJAX (a misleading but widely used term) – Asynchronous JavaScript and XML

There are two main advantages

- speed: fewer data items are exchanged
- a smoother experience for the user

Web Applications

The term ***web application*** is normally used to describe a dynamic interactive website with client-server communication. Such sites are widely used in e-commerce applications but also in many other types of application.

Web applications are usually

- easy to use (end-user focussed)
- familiar: frequent use of standard controls
- platform and machine independent
- easy to access (no installation required on client machine)
- easy to update (hit refresh!)
- hard to program??

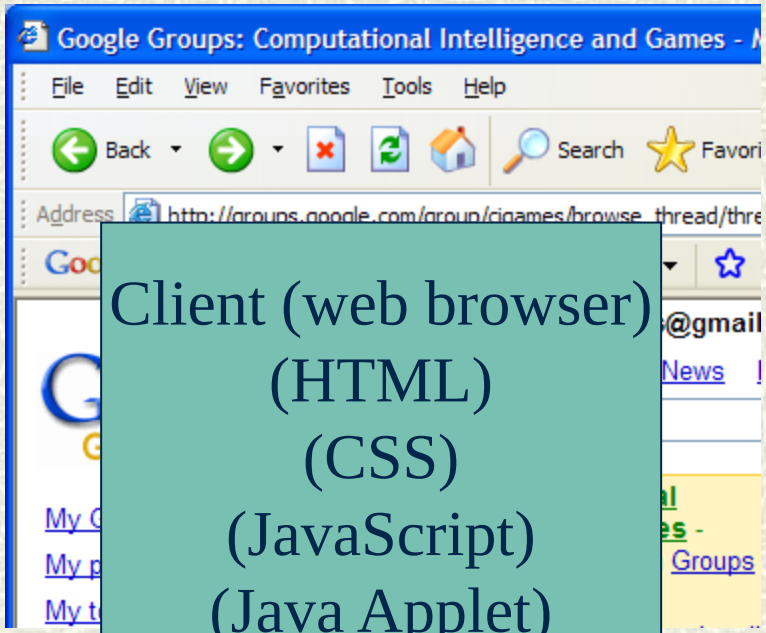
Web Application Architectures

Web applications use client-server architectures, with the client being a web browser. This is often referred to as thin-client computing – most of the processing is done on the server. Nowadays there is increasing use of JavaScript (or similar scripting languages) on the client for more dynamic web applications.

The server usually comprises a web server, an application server and a back-end database (for persistence of data).

We will use Tomcat as the web server and application server (JSP/Servlet engine) and mostly a relational database. We will also introduce an object-oriented database (db4o).

A Simple Web Application Architecture



Client (web browser)
(HTML)
(CSS)
(JavaScript)
(Java Applet)

Web Server +
Servlet/JSP
Engine
(e.g. Tomcat)

Database
(Live Data)

File System
HTML, CSS, JS

HTTP

Designing Web Applications

The design of web applications is challenging – there are many languages to learn

For client side design we must consider

- visual and logical layout
- validation
- JavaScript controls and content manipulation

For the server side we need to consider

- database design
- server-side language and design patterns
- security
- state management
- performance and reliability (24/7)

HTML 1

HTML (***hypertext markup language***) was responsible for transforming the internet into the Worldwide Web

It is used to specify page layout, with a simple hyperlink and page processing semantics

It is specified and used in conjunction with HTTP – the ***hypertext transmission protocol***.

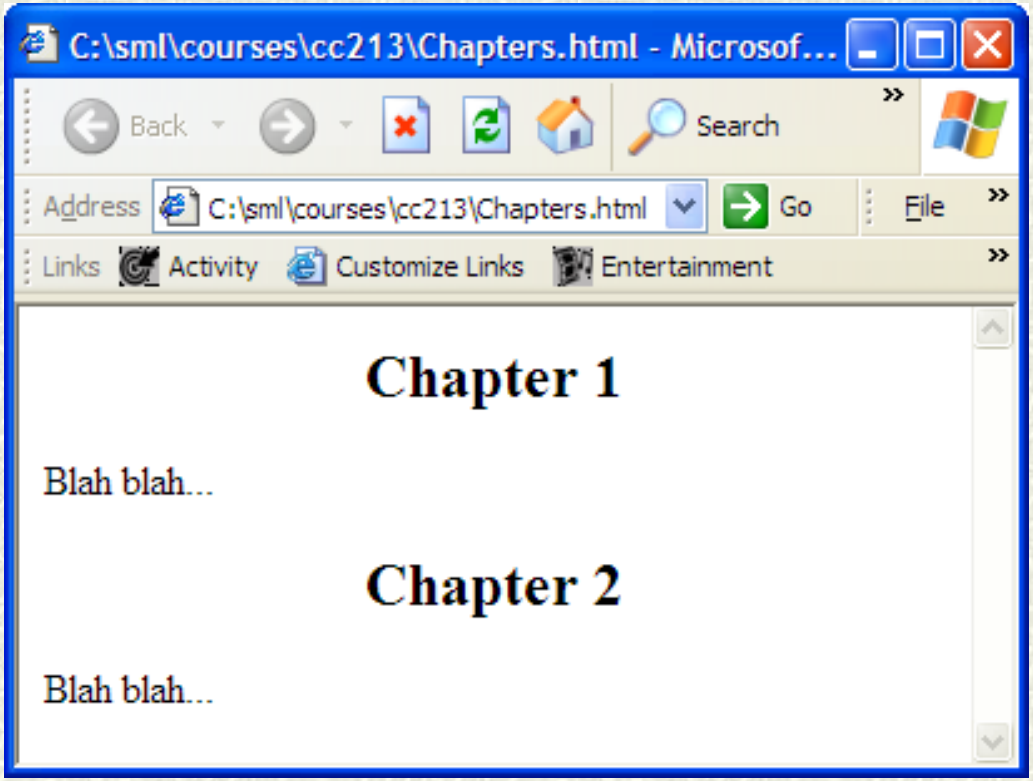
A fixed set of tags is used to specify logical document structure; these may have attributes to control appearance.

A simple example is shown on the next slide and its possible rendering on a browser in seen on the following slide.

HTML 2

```
<html>
  <body>
    <h2 align="center"> Chapter 1 </h2>
    <p>
      Blah blah...
    </p>
    <h2 align="center"> Chapter 2 </h2>
    <p>
      Blah blah...
    </p>
  </body>
</html>
```

HTML 3


C:\sml\courses\cc213\Chapters.html - Microsof...
Back
Search
Address
C:\sml\courses\cc213\Chapters.html
Go
File
Links
Activity
Customize Links
Entertainment
Chapter 1
Blah blah...
Chapter 2
Blah blah...

XHTML and HTML5

XHTML and HTML5 are the most modern versions of HTML.

They require that tags must be properly nested (e.g. `<b><i>…</b></i>` is not allowed) and are case-sensitive, and all attribute values must appear in string quotes.

In XHTML all tags must be closed (i.e. every tag must have a matching `/` tag). When a tag has no content it may be self-closing (e.g `<br/>`).

XHTML can be extended with user-defined mark-up.

JSP

Java Server Pages (JSP) is a server-side technology. It allows the embedding of JSP tags within HTML pages. These tags specify processing to be done on the server to produce a version of the HTML that is then sent to the client; hence the client never sees the JSP tags.

The use of JSP makes it relatively simple to build dynamic web pages – up-to-date data can be obtained from a database and embedded in the page that the client sees.

JSP should be used with caution – it should not be used for complex program logic; Java helper classes should be used instead for anything other than simple tasks.

CSS 1

CSS (***cascading stylesheets***) provides a means of separating formatting control from content by specifying attributes for tags separately from the tags themselves. This can ensure consistent style within an HTML page.

CSS code may be embedded within an HTML document, or may be placed in a separate file referenced from the document. The latter approach allows style to be reused, ensuring consistent presentation of web pages across a site, and also reduce file sizes.

The following slide shows how embedded CSS can be used to specify that all level-2 headings in our previous example should be centre-aligned.

CSS 2

```
<html>
  <head>
    <style type="text/css">
      h2 {text-align: center}
    </style>
  </head>
  <body>
    <h2> Chapter 1 </h2>
    <p> Blah blah... </p>
    <h2> Chapter 2 </h2>
    <p> Blah blah... </p>
  </body>
</html>
```

CSS 3

CSS defines the way that HTML elements should be presented:

- positioning (e.g. left, right or centred)
- font (family, size and weight)
- text decoration (e.g. underlined)
- borders (solid, dashed, invisible)
- image usage (e.g. for backgrounds and bullets)

CSS does not:

- re-order HTML (e.g. won’t sort a table)
- perform calculations (e.g. will not sum a shopping basket)
- filter (will not decide what to show

CSS 4

Styles in CSS are specified using a list of name/value attribute pairs, with the two elements of each pair separated by a colon, and the pairs separated by semicolons, e.g.

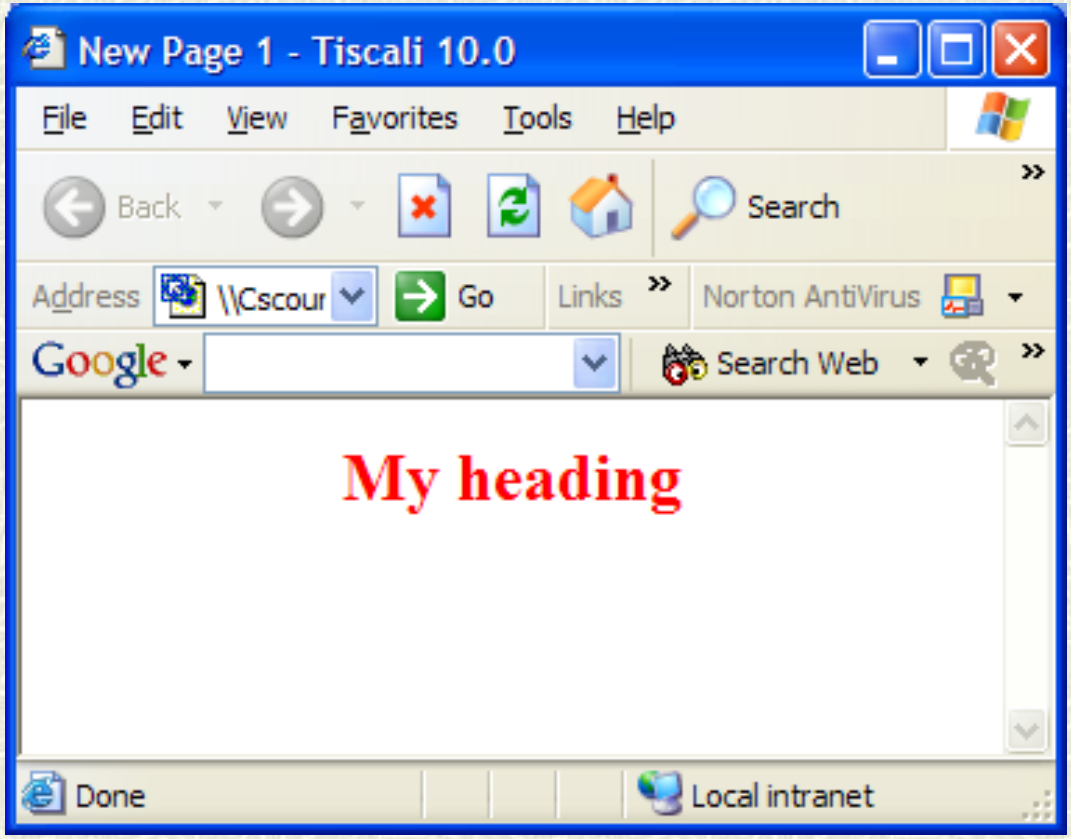
```
font-weight:bold; color:blue;
text-decoration:underline
```

The simplest way to apply a style is to use the `style` attribute in a tag, e.g.

```
<h2 style="text-align: center; color: red">
My heading
</h2>
```

The next slide shows how a browser nay render a page containing the above heading.

CSS 5


New Page 1 - Tiscali 10.0
File Edit View Favorites Tools Help
Back
Search
Address \\Cscour
Go
Links
Norton AntiVirus
Google
Search Web
My heading
Done
Local intranet

CSS 6

Since a major aim of the use of CSS is to separate document content and structure from presentation, the approach of giving `style` attributes to individual elements should not normally be used.

Instead, a ***stylesheet*** specifies rules indicating the styles of different types of elements. Each rule comprises a ***selector*** followed by, in braces, a style. The simplest form of selector is simply a tag or a list of tags e.g.

```
body { margin-left:20px; margin-right:10px;
       background-color:#ffffff }
h1,h2 { text-align:center }
```

Note that when the rule applies to more than one tag the tags are separated by commas.

CSS 7

A selector can also be a combinations of tags, e.g.

```
blockquote { font-style: italic }
blockquote i { font-style: normal }
```

The above indicates that text inside `<blockquote>` tags should be displayed in italic but when the `<i>` tag is used in a block quote normal style should be used instead.

CSS 8

Another kind of selector is used to specify a ***class*** of elements to which styles should be applied:

```
.highlight { font-weight: bold }
```

To associate an element with a class the HTML **`class`** attribute is used, e.g.

```
<title class="highlight"> …… </title>
```

It is possible to combine a tag name and a class in a selector, e.g.

```
p.highlight { color: red }
```

This indicates that when a paragraph has a **`class`** attribute with the value **`highlight`** that paragraph should be displayed in red.

CSS 9

Stylesheets can also contain rules that apply only to elements with a particular value for the `id` attribute:

```
#p1 { color: blue }
```

This is most commonly used with JavaScript code that changes the value of `id` attributes allowing, for example, the highlighting of items that the user has selected.

CSS 10

A stylesheet can be associated with a document by placing it between `<style type="text/css">` and `</style>` tags within the document header (i.e. between `<head>` and `</head>`):

```
<html>
  <head>
    <title> Document 1 </title>
    <style type="text/css">
      body { ..... }
      p { ..... }
    </style>
  </head>
  <body> ..... </body>
</html>
```

CSS – An Example 1

We now present an example where the `<pre>` tag is used to present code fragments – we wish to present Java and C++ in different styles so we introduce two classes.

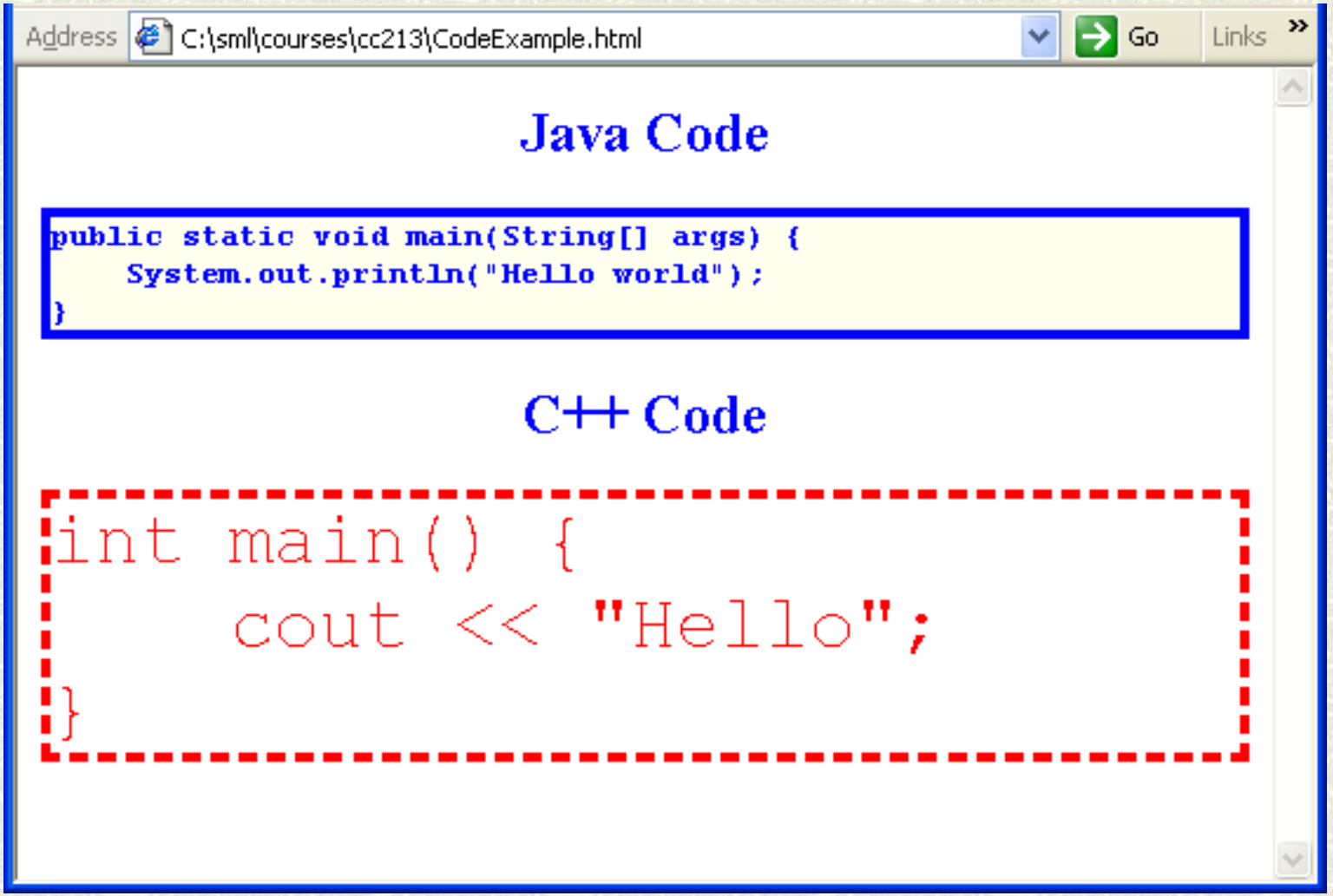
```
<html>
  <head>
    <style type="text/css">
        h2 { text-align: center; color: blue }
        pre.java
        { border: solid; background-color: #FFFFEE;
          color: blue; font-weight: bold }
        pre.cpp
        { font-size: 200%; border: dashed; color: red }
    </style>
  </head>
```

CSS – An Example 2

```
  <body>
    <h2>Java Code</h2>
    <pre class="java">
public static void main(String[] args) {
    System.out.println("Hello world");
}
    </pre>
    <h2>C++ Code</h2>
    <pre class="cpp">
int main() {
    cout << "Hello";
}
    </pre>
  </body>
</html>
```

A browser rendering of this example can be seen on the next slide.

CSS – An Example 3



External Stylesheets 1

When a stylesheet is to be used with more than one page it should be written is a separate file (without any enclosing tags). The **<link>** tag must be used to include an external stylesheet in a document:

```
<html>
  <head>
    <title> Document 1 </title>
    <link rel="stylesheet" href="styles.css"
          type="text/css"/>
  </head>
  <body> ..... </body>
</html>
```

External Stylesheets 2

If a page uses an external stylesheet and also some page-specific styles it is necessary to include the CSS file within the `<style>` tag in the header – this is done using the CSS `@import` directive:

```
<html>
  <head>
    <title> Document 2 </title>
    <style type="text/css">
      @import "styles.css"
      p { font-size:40px }
    </style>
  </head>
  <body> ..... </body>
</html>
```

Cascading 1

The style of an element can be specified in many places. A set of rules is used in order of most specific to most general to select which style to use in any particular case; this is called a ***cascade***.

A web browser typically has a default stylesheet and may allow the user to override the default styles with his own stylesheet.

The order of precedence (highest priority first) is

- inline styles
- internal document stylesheet
- external document stylesheet
- user stylesheet
- default browser stylesheet

Cascading 2

A stylesheet may also specify more than one style for an element. In this case the order of precedence is

- rules that specify an element id
- rules that specify a class
- rules that specify multiple tag names, e.g.

```
blockquote i { font-style: normal }
```

- rules that specify single tag names, e.g.

```
blockquote { font-style: italic }
```

Display and Visibility 1

It can sometimes be desirable for some content of web pages to be invisible; this allows nested content to be dynamically expanded (i.e. made visible at an appropriate time) and also allows switching between pre-loaded content

Two CSS attributes `display` and `visibility` control this; they can be used to determine how an element is displayed and whether it is visible on the display.

Display and Visibility 2

Possible values for the **`display`** attribute include **`inline`**, **`block`**, **`table`** and **`none`**. In the latter case the element is not displayed and occupies no space.

The possible values for **`visibility`** include **`visible`** and **`hidden`**. If an item has the value **`hidden`** it will not be displayed, the space it would have occupied being blank.

These attributes are normally used in class or id rules when it is necessary to change visibility dynamically by setting the value of the **`class`** or **`id`** attribute using JavaScript.

JavaScript 1

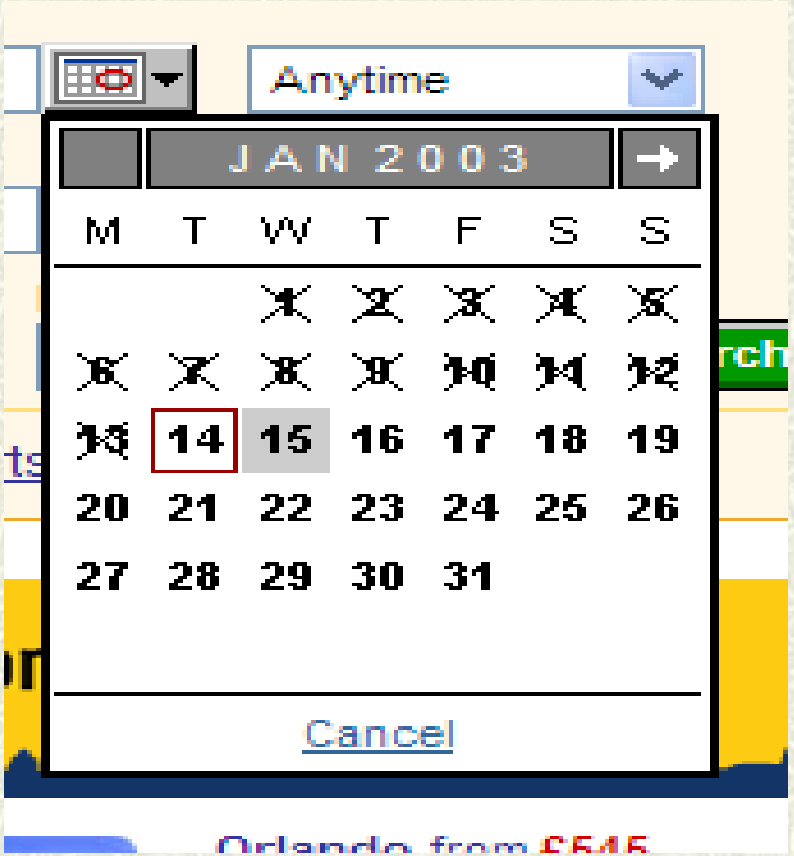
JavaScript (which is ***not*** the same as Java) is a scripting language used to specify client-side interactions such as

- form validation
- simple calculations (e.g. menu-pricing)
- better controls

An example of a component written using JavaScript can be seen on the next slide. It is a date selector which makes it easier to visualize dates, with no format confusion (e.g. is 12/11/2010 December 11th or November 12th?) and self-validation (e.g. the user cannot select February 30th).

[The diagram is a screenshot taken several years ago from the Expedia website.]

A JavaScript Date Control



The JavaScript Language

JavaScript is a powerful object oriented language with a syntax similar to Java. However its semantics is very different from that of Java (and similar languages such as C++ and C#) in several aspects; in particular the type system is much less rigid.

JavaScript is interpreted and dynamic; new functions can be defined at runtime.

There is built-in support for regular expressions.

Although JavaScript was designed for web browser interactions it can be used stand-alone as a powerful programming language.

JavaScript Syntax 1

JavaScript programs are written in Unicode so it is possible to include non-ASCII characters in programs (if your editor or development environment supports this).

Reserved words include most of those of Java and also `delete`, `function`, `in`, `typeof` , `var` , `let` (in recent versions only) and `with`.

Strings may be written in single or double quotes, e.g. `"hello"` or `'hello'`.

Numbers are written in the same way as in Java, although some implementations treat numbers beginning with 0 as octal and others do not.

JavaScript Syntax 2

As in Java white space in a source program is ignored but there is one significant exception to this rule.

Simple statements are normally terminated by semicolons, as in Java. However such semicolons may be omitted at the end of a line. This is not recommended and can lead to confusion. For example

```
return
   true;
```

and

```
return true;
```

mean different things!

JavaScript Types

JavaScript has a much less rigid typing system than Java. There are just three primitive types: boolean, number and string. (Note that unlike Java a string is not regarded as an object). As in Java there are wrapper classes (`Boolean`, `Number` and `String`) for use when primitive items need to be treated as objects.

Arrays are objects of type `Array` and can be used in the same way as in Java.

As in Java, all objects inherit from `Object` but the way that objects are declared differs from that of Java.

JavaScript Variables

Variables are declared using **`var`**, e.g.

```
var pi = 3.14, myAge, myString;
```

There is no type information so, as in Python, the same variable can be used to store data of different types at different times, e.g.

```
myAge = 14;
…
myAge = "fourteen";
```

JavaScript Functions 1

Functions are declared using **function**, e.g.

```
function ftoc(fahr)
{ return (fahr-32)*5/9;
}
```

No information is supplied about the argument types or the return type.

If there is more than one argument they are separated by commas, e.g.

```
function distance(x1, y1, x2, y2)
{ var dx = x2-x1, dy = y2-y1;
  return Math.sqrt(dx*dx + dy*dy);
}
```

JavaScript Functions 2

JavaScript supports ***higher-order*** functions – a function can be supplied as an argument to a function. The following example will apply its argument **`f`** to all of the values of the array **`a`** and output the results.

```
function map(a, f)
{ for (var i = 0; i<a.length; i++)
  print(f(a[i]));
}
```

JavaScript Functions 3

When calling a function it is permissible to supply fewer than the expected number of arguments. In this case all unsupplied arguments have the special value **`undefined`**.

If the number of arguments supplied is greater than the expected number the extra arguments are ignored.

Note that since there is no type-checking in JavaScript it is possible to try to access the value returned by a function that returns no result; if no result is explicitly returned (as in the example on the previous slide) the returned value will be **`undefined`**.

Type Conversions 1

Since JavaScript is loosely typed implicit type conversions can occur in many situations. For example, when a boolean value is needed (e.g. as the condition in an if statement or loop), if a number is supplied it will be regarded as true unless its value is 0 or the special value `Number.NaN` (not a number), if a string is supplied it will be regarded as true unless its length is 0, and if an object is supplied it will be regarded as true unless it is `null`. Note in particular that in code such as

```
var b = new Boolean(false);
if (b) {……}
```

the body of the loop will be executed since the value of `b` is not `null`.

Type Conversions 2

When a string is expected (e.g. as an argument of `+` when the other argument is a string) implicit conversion will take place as in Java, i.e. the `toString` function of the object (or its wrapper class in the case of a primitive data type) will be applied.

When a number is expected and a string is supplied an attempt will be made to convert the string to a number. If an object is supplied an attempt will be made to use ifs `valueOf` function; if this does not return a number the `toString` function will be called and an attempt then made to convert the string to a number. If this fails the value `Number.NaN` will be used.

JavaScript Operators 1

In addition to the operators of Java the operators of JavaScript include **`delete`**, **`typeof`**, **`in`**, **`===`** and **`!==`**.

Operator precedence is the same as in Java, with **`delete`** and **`typeof`** having the same precedence as the other prefix unary operators, **`in`** having the same precedence as **`instanceof`** and both **`===`** and **`!==`** having the same precedence as **`==`**.

Most operators behave in a similar way as in Java; however, the **`/`** operator always uses real-number division and all arithmetic operators give the value **`Number.NaN`** if any operand is **`Number.NaN`**.

JavaScript Operators 2

When the items being compared are of the same type the `==` and `!=` operators behave in the same way as in Java, with the exception that strings (being primitive) are compared by value, so it is not necessary to use `compareTo` to compare strings. However if the types differ and both are primitive attempts will be made to convert both to numbers, and when comparing an object with a primitive an attempt will first be made to convert the object to the appropriate primitive type.

`undefined` is regarded as being equal to `null`. `Number.NaN` is never equal to any value, including itself.

The `===` and `!==` operators do not attempt to perform type conversion.

JavaScript Operators 3

The `typeof` operator can be used to determine the type of a variable or expression. Its value is a string, which will be equal to `"number"`, `"boolean"`, `"string"`, `"object"` `"undefined"`, or `"function"`. (If the operand is of any object type, including an array or `null`, the value will be `"object"`; if the operand is the name of a variable that does not exist the value will be `"undefined"`.)

To determine what kind of object a variable holds it is necessary to use the `instanceof` operator.

JavaScript Objects 1

The members of objects (i.e. instance variables) are known as ***properties***.

Objects can be created using constructors as in Java, e.g.

```
var today = new Date(2010, 1, 28);
```

It is also possible to create an object by using an object literal as an initialisation in a variable declaration, e.g.

```
var bart = { name: "Bart Simpson", age: 10 };
```

Such an object is effectively an object of an anonymous inner class that extends the class `Object`.

JavaScript Objects 2

Objects can be viewed as ***associative arrays***, i.e. arrays whose subscripts are property names. For example we could access the `age` property of the object `bart` using `bart.age` or `bart["age"]`. This alternative means of access offers no benefits it the subscript is a string literal, but can be useful if it is a variable, e.g. we could access different properties of `bart` using `bart[s]` where `s` holds the desired property-name at any given time.

JavaScript Objects 3

There is no formal concept of a class in JavaScript. To create the equivalent of a class we simply write one or more constructors, e.g.

```
function Student(name, age, degree)
{ this.name = name; this.age = age;
  this.deg = degree;
}
```

To add a function to this pseudo-class we would use code such as

```
Student.prototype.incrementAge = function()
{ this.age++;
}
```

JavaScript Objects 4

The use of `prototype` indicates that the function must be applied to an object – omission of this word provides the equivalent of a static Java method, i.e. one that is applied to the class. (Note that `prototype` is not a reserved word; it is a property of the class `Function`; the syntax seen for declaring class functions is a shorthand way of invoking a constructor of the `Function` class.)

JavaScript Objects 5

When writing class functions it is necessary to prefix the properties with **this**, unless a **with** statement is used. Using such a statement we could rewrite the **incrementAge** function seen earlier with

```
Student.prototype.incrementAge = function()
{ with(this)
  { age++;
  }
}
```

Clearly this is only worthwhile if the function contains several accesses to properties.

JavaScript Objects 6

It is possible to add properties to an object at runtime. This is simply done by using a new property name, e.g.

```
bart.grade = "F";
```

A property can be removed from an object using the **delete** operator. e.g.

```
delete bart.grade;
```

The value of the **delete** operator is a boolean indicating whether the deletion is performed successfully – it could fail if the property does not exist or the argument is invalid

JavaScript Objects 7

As a consequence of the ability to add and delete properties it is sometimes necessary to be able to determine whether an object has a particular property. This can be done using the **in** operator, e.g.

```
if ("grade" in bart)
  print("Bart got grade " + bart.grade);
```

To print the value of a property whose name has supplied by a user, if it exists, we would need to use the in operator along with the associative array notation:

```
if (s in myObject)
  print(s +" : " + myObject[s]);
```

`RegExp` Objects 1

A ***regular expression*** is an object that describes a pattern of characters. Examples include

```
[acx]          // 'a', 'c', or 'x'
[^!]           // any character except '!'
a[bc]?         // 'a' optionally followed by 'b' or 'c'
a*b            // 0 or more occurrences of 'a' followed
               // by 'b'
a+!            // 1 or more occurrences of 'a' followed
               // by '!'
\d{3}          // exactly 3 digits
\d{2,4}        // between 2 and 4 digits
\s+java\s+     // "java" preceded by and followed by
               // one or more white space character
\w             // any alphanumeric character
```

`RegExp` Objects 2

In JavaScript regular expressions are objects of type **`RegExp`**. Such an object can be created using the **`RegExp`** constructor (with a string argument) or using a regular expression literal:

```
var re1 = new RegExep("[acx]");
var re2 = /[^!]'/
```

It is possible to check whether strings contain substrings that match regular expressions and also replace matches of regular expressions by applying the functions **`search`** and **`replace`** to **`String`** objects. The argument to these functions may be a string (which will be converted to a regular expression using the **`RegExp`** constructor) or a regular expression.

`RegExp` Objects 3

A regular expression literal may be followed by a flag to modify the behaviour of searches; for example `i` indicates that a search should be case-insensitive.

The `search` function returns the index of the first occurrence of a substring that matches the regular expression, or -1 if no match could be found.

```
var s = "JavaScript is fun";
s.search(/script/i);    // returns 4
s.search("a\w*p");      // returns 1
s.search(/x/);          // returns -1
```

Note that the function must be applied to a `String` object, so the wrapper class constructor is called implicitly.