Scripting Languages

Scripting Languages (prehistory)

- Scripting languages have ***always*** been important in computer systems
  - They are the glue that ties the different elements of the system together
  - Their origins go back to the days of card-based operating systems
    - JCL (OS360 JCL)
    - GEORGE II, GEORGE III
  - And they were much used in minicomputer operating systems
    - Data General's AOS
    - Unix

Scripting Languages (history)

- Scripting languages originate in systems which were used to join together programs (or tasks)
- Unix and other 1980's operating systems introduced powerful commands
  - And scripting languages could put these together to produce quite powerful tools quickly and easily.
  - Shell scripting is still much used particularly by system administrators.
- Later, the concept of scripting began to be used to describe inputs to programs that interpreted their input to produce desired results
  - So one might consider scripting to be a command language for an interpreter program: application-specific languages
- But there's no definition of scripting that really distinguishes them from mainstream programming languages.
- See http://en.wikipedia.org/wiki/Scripting_language

An alternative view

- Scripting is about producing simple very-high-level-languages that are friendly to the programmer who has has a life.
- Modern sophisticated HLLS
  - Java, C++, C#, etc. are extremely complex
  - (they have a nasty tendency to get bigger and bigger as designers add more and more useful facilities, and interface components, and bells and whistles, …)
  - Take a long time to learn to use (but are wonderful when you really understand them).
- Scripting languages are relatively simple, and often allow users
  - Who are not necessarily C.Eng programmers
- … to do complex things
- Hence: very high level programming systems
- See http://www.softpanorama.org/People/Scripting_giants/scripting_languages_as_vhll.shtml

Scripting now

- There are many scripting languages:
  - Python, perl, R, javascript, PHP, …
- Some scripting languages are used primarily in web systems.
  - Some are self-standing languages. However, others are embedded in HTML and used to enhance web pages.

- Here, we shall mainly look at how they can be used with the web.
  - N.B. This is not their only area of use.
    - See Matlab, for example. Or Octave, or R, …

Overview

- Some core features of programming languages:
    1. Basic (built-in) types.
    2. Strings (and string handling including pattern matching)
    3. Data structures (associative arrays, but also abstract data types).
    4. Control structures.
    5. Modular programming.
    6. Type regime.
    7. Operating environment.

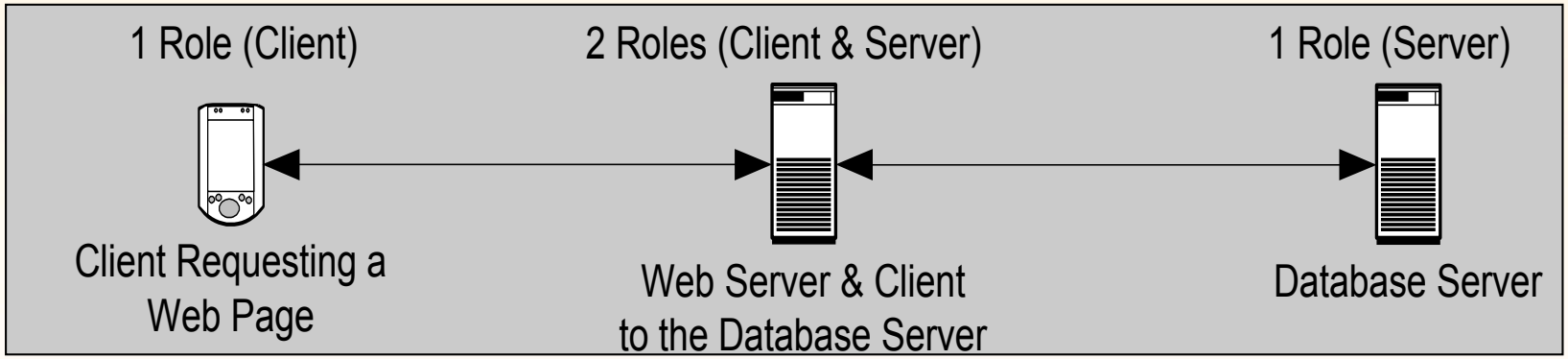
- These are present in scripting languages too. We start with an in-depth look at the www operating environment.

What is client-side and server-side?

- Any machine can play the role of either a client or a server
  - You could even have a machine being both

- Some languages, e.g. Javascript, are said to be ***client-side***.
  - Run on the user's browser/web client

- Other languages, e.g. PHP, are said to be ***server-side***.
  - Run on the server that is delivering content to the user



Servers

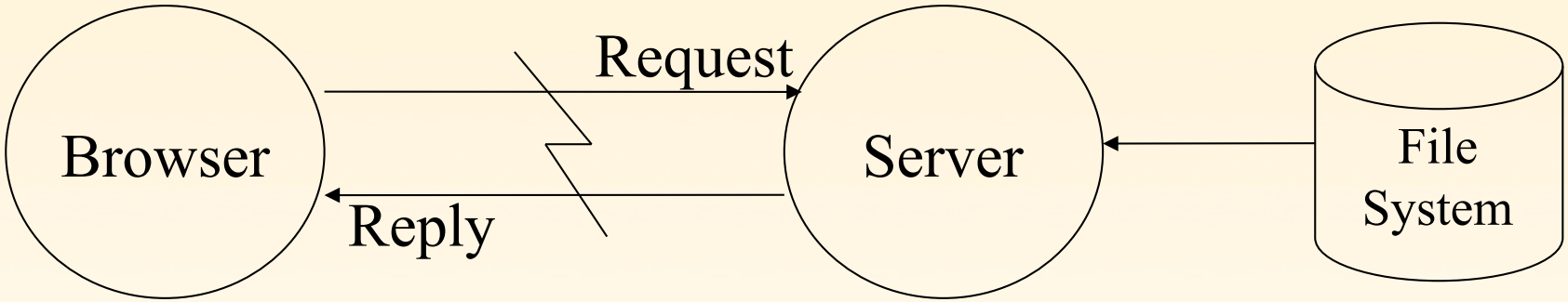
- Two important examples are:
  - Web Servers
  - Database Servers

- Some of these machines may be powerful computers dedicated to the task, i.e Database server

- A web server is a machine holding and delivering HTML pages that can be accessed by remote (client) machines over the Internet.

Refresher on Web Dynamics

- Static Web Model



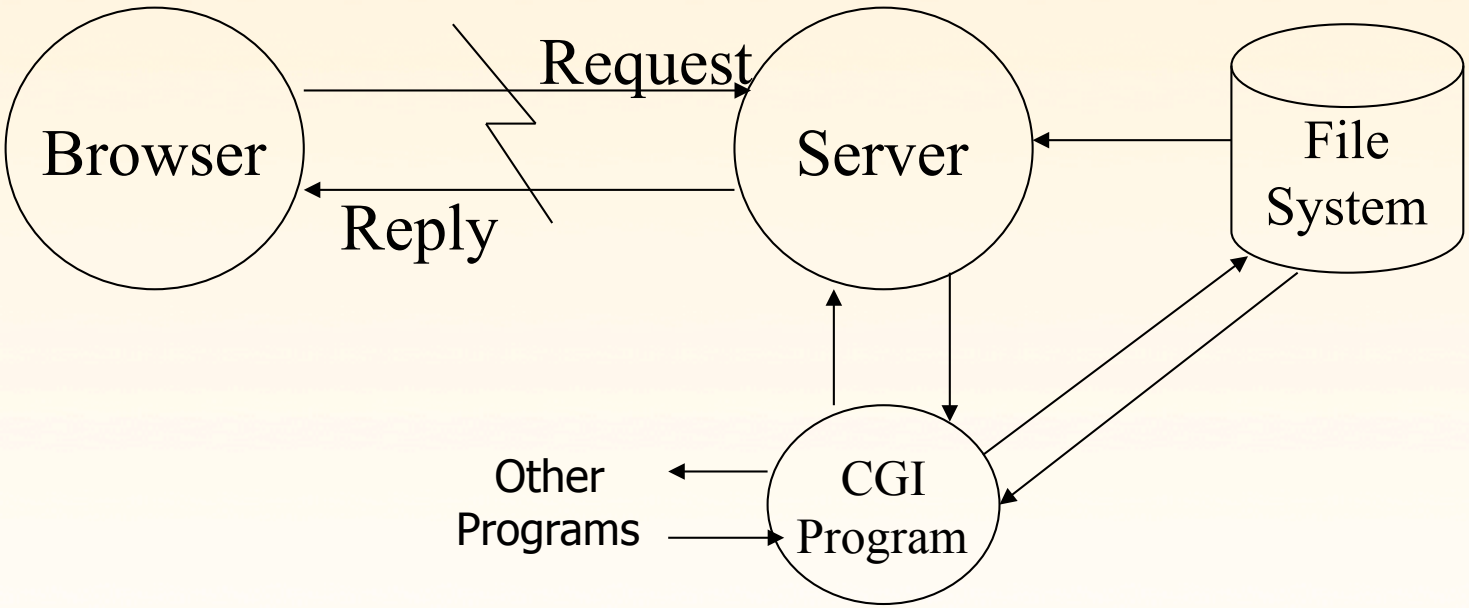
- You (the client) send a request to the server for a web page. The server looks up the web page using part of the URL you have sent it, then returns the HTML page which your browser subsequently displays on your machine.

More on Web Dynamics

- Let us now consider a more dynamic model.
  - You (the client) send a request to the server and it dynamically determines the HTML that is to be returned.
  - The dynamics of the reply is achieved through extending the ***web server*** with a program (script) that does some data processing and creates HTML output based on the data you sent (e.g. contents of a form).
  - The process of generating the HTML response is performed ***server-side***.

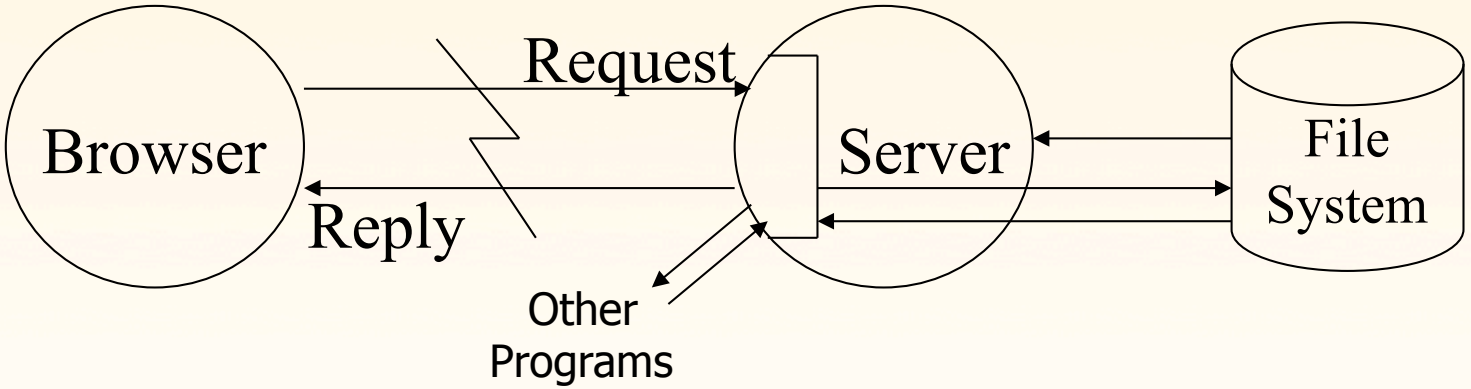
Server-side scripting

- Dynamic Web Model
  - One approach is the Common Gateway Interface (CGI) where we have a separate program that can be executed.



Server-side scripting

- Dynamic Web Model
  - An alternative is to have extra code in the HTML that can be executed on the ***server*** to determine the HTML that is to be returned.
  - That is how PHP works.



Client-Side Scripting

- The other (complementary) approach is to do the work on the client machine.
  - Again we have extra code in the HTML, but now it is executed by the user's browser (i.e. ***client-side***).
    - Most common client side script is Javascript.
  - An example of its use is when a web page has a form. We can use Javascript to validate the input data client-side before it is sent to a server.
- If we do the validation on the client, this reduces the work that the server has to do and reduces the time taken to respond to the user.
- HTML5 essentially includes Javascript elements to enhance its power.

Client-Side Scripting

- Javascript can also be used to create dynamic web page content. For example:
  - We could change the content based on the fact that you visited the web page before.
  - Time of day.
  - JavaScript popup menus.

Javascript

- Both Javascript and PHP are embedded within HTML code. Here is some Javascript, available at:

  http://www.cs.stir.ac.uk/courses/CSC9Y4/examples/y1.html
- (see also http://www.cs.stir.ac.uk/~lss/CSC941/javascript/main.html)

```
<html>
<head>
  <title>A First Program in JavaScript</title>
</head>
<body>
  <h1>Dynamic generation</h1>
  <script language = "JavaScript">
      document.writeln("<p>Welcome to
  JavaScript                          Programming!
  </p>");
  </script>
</body>
</html>
```

Javascript and PHP

- And here is some PHP held in:

  http://www.cs.stir.ac.uk/courses/CSC9Y4/examples/y1.php

```
<html>
<head>
  <title>A First Program in PHP</title>
</head>
<body>
  <h1>Dynamic generation</h1>
  <?php
  echo "<p>Welcome to PHP Programming</p>";
   ?>
</body>
</html>
```

Client and server

- Note that we have HTML in which we have an embedded script that is to be executed.
- There are clear similarities both in terms of the syntax and in how they are used.
- However, there is one ***important difference***:
  - the PHP script is executed on the ***web server***
  - the Javascript is executed by the ***browser*** on the client's machine.
- The page with Javascript goes in an ordinary `xx.html` file while the page with PHP goes in an `xx.php` file.

Viewing source

- When we view the source of the Javascript example, we see the code written on the slide.
  - That is because that is the HTML that is handled by the browser.
  - With the PHP example, on the other hand, the PHP is executed on the server and the result of that execution is sent to the browser.
  - Hence, when we view the source for the server-side executed web page, we do not see any of the PHP code.

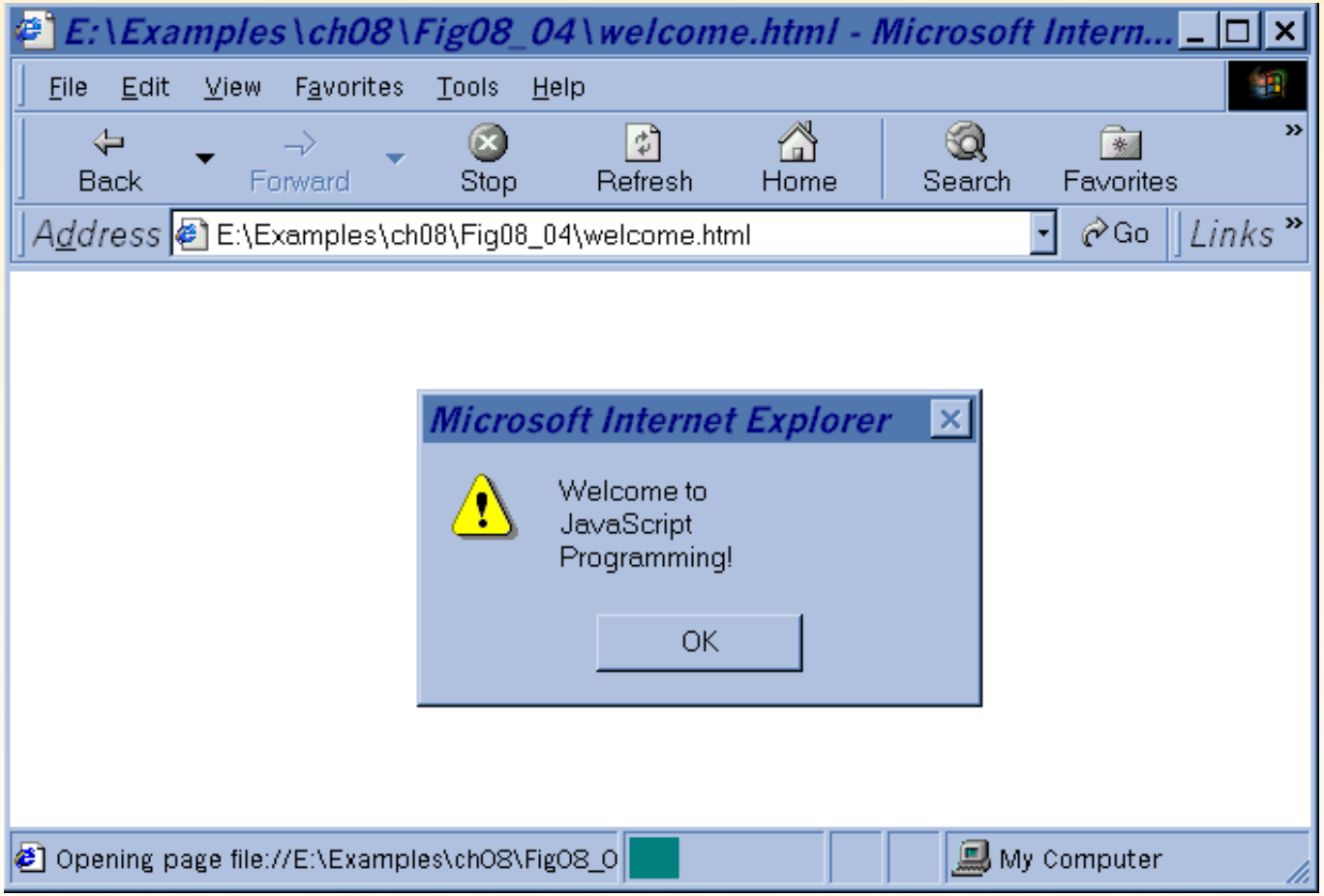
Another Javascript Example

Javascript is often used to create pop ups as in the following example held in:

www.cs.stir.ac.uk/courses/CSC9Y4/examples/y2.html

```
<html>
<head><title>An alert</title></head>
<body>
  <h1>Pop up example</h1>
  <script language = "JavaScript">
    window.alert("Welcome
          to\nJavaScript\nProgramming!");
  </script>
</body>
</html>
```

Alert box

- 
E:\Examples\ch08\Fig08_04\welcome.html - Microsoft Intern...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address E:\Examples\ch08\Fig08_04\welcome.html
Go Links
Microsoft Internet Explorer
Welcome to JavaScript Programming!
OK
Opening page file://E:\Examples\ch08\Fig08_0
My Computer

Server-side processing

- The following is the standard use of server-side processing:
  - Our browser is sent a static HTML page that contains a form. We type information into a textbox and press the submit button to send the information off to the server.
  - The server dynamically creates a new HTML page whose content depends on our input and returns this to us.
  - Search engines, for example, work in this way.

The form

- Let us look at how this would be done in PHP. Suppose that the original web page contains a form as in:

- `www.cs.stir.ac.uk/courses/CSC9Y4/examples/y3.html`

  - We have a textfield and a submit button.
  - An *action* field tells us where the PHP file is held that should receive the contents of the form.

The HTML code for a simple form

```
<html>
<head>
<title>Processing a form</title>
</head>
<body>
<h1>Processing a form</h1>
<form
action="http://www.cs.stir.ac.uk/courses/CSC9Y4/examples/form.php"
method="post"
>

What is your name? <input type="text" name="myname"><br>
<input value="Submit name" type="submit">

</form>
</body>
</html>
```

Execution

- If we type the name **`Jimmy`** into the text box and press the submit button then the following URL is sent:

```
http://www.cs.stir.ac.uk/courses/CSC9Y4/examples/form.php?myname=Jimmy
```

  - This causes the PHP program held in file **`form.php`** to be executed with the **`myname`** parameter having the value **`Jimmy`**.

  - Let us now look at the contents of the PHP file.

PHP program

```
<html>
<head>
  <title>Hello</title>
</head>
<body>
  <h1>
  <?php
    $name = $_POST["myname"];
    echo "Hello ";
    echo $name;
  ?>
  </h1>
</body>
</html>
```

PHP

- The value of the **`myname`** parameter is extracted from an array of Strings called **`$_POST`** and is saved in **`$name`**.
  - Note that we can access this array using a string label rather than an index number

- We then output a message that depends on the string that was sent. Hence a new web page is sent back that displays the string:

Hello Jimmy

The form

Let us now look at how this would be done using CGI and Perl.

http://www.cs.stir.ac.uk/courses/CSC9Y4/examples/y4.html

The original HTML is very similar although now the action field tells us where the CGI program is held.

```
<form action =
"http://www.cs.stir.ac.uk/cgi-bin/CSC9Y4/simple">
  What is your name? <input Name = "myname">
  <p>
  <input Type = submit Value = "Submit name">
  </p>
</form>
```

Execution

Again we type the name **`Jimmy`** into the text box and the following URL is sent:

http://www.cs.stir.ac.uk/cgi-bin/CSC9Y4/simple?myname=Jimmy

•This causes the CGI program held in file **`simple`** to be called with the **`myname`** parameter having the value **`Jimmy`**.

– The CGI program is held in a special folder.

•Here is a possible Perl program to deal with this. It creates a new web page that displays the string:

Hello Jimmy

Perl

```
#!/usr/bin/perl
use CGI ":standard";
$name = param("myname");
print <<FirstPart;
Content-type: text/html
  <html>
    <head>
    <title>Hello</title>
    </head>
    <body>
    <h1>Hello
FirstPart
print $name;
print <<Remaining;
    </h1>
    </body>
  </html>
Remaining
```

Perl

- The first line tells the system the file is in Perl and the second brings in the Perl CGI library.
  - The **`param`** operation extracts the value of the **`myname`** parameter which is saved in **`$name`**.
- We could now do lots of Perl processing.
- The rest of the program consists of three print statements.
  - Note the strange bracketing convention (here-document):

```
print <<SomeLabel;
...
SomeLabel
```

  - treats the parts between the brackets as a doubly-quoted string.

CGI and Perl

- The big difference is that with PHP, the code fragments are embedded in the HTML.

- With Perl, we have a stand-alone program and it uses print statements to generate the HTML.
  - As Perl is an ordinary programming language, it can be used for lots of tasks other than generating web pages.

Accessing databases

- These simple examples have shown how information can be sent from the client to the server and how the server can dynamically create a web page that is to be returned.
  - In practice, the server will do a lot more processing.
  - Typically, the server will access a database and will either update the database or retrieve information from it. That information is then used in the creation of the dynamic web page.

Client-side Scripting

•Let us now look at an example where all the work is done client-side.

- We want the browser to create a pop-up window where it will read in a value.
- A web page is then displayed where this value is displayed.
- The Javascript to do this is shown on the next slide.

www.cs.stir.ac.uk/courses/CSC9Y4/examples/y5.html

Javascript Example

```
<html>
<head>
  <title>A Program in JavaScript</title>
</head>
<body>
  <script language = "JavaScript">
        var thenumber;
        // Read in number from user
        thenumber = window.prompt("Enter number","0");
        // Display the number to the user
        document.writeln("<h1>The number is " +
         thenumber + "</h1>"  );
  </script>
  <p>Click Refresh (or Reload)to run the script again</p>
</body>
</html>
```

Client-side Execution

•The HTML code will have been sent from the server to the browser.

•The browser then:

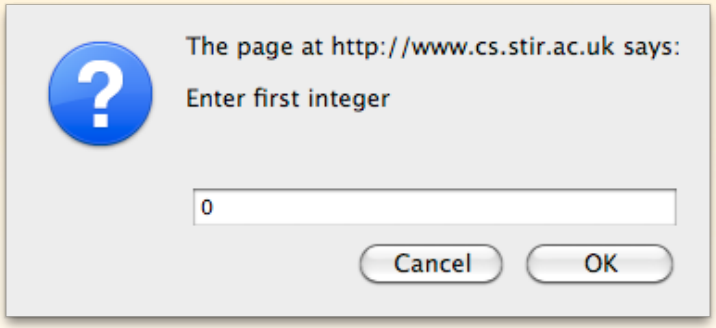
- creates the pop-up window,
- controls the reading of the input value
- displays the resulting web page.

•Note that this is done by your browser without going back to the server.

•When you click on refresh, the HTML code will be fetched again from the server.

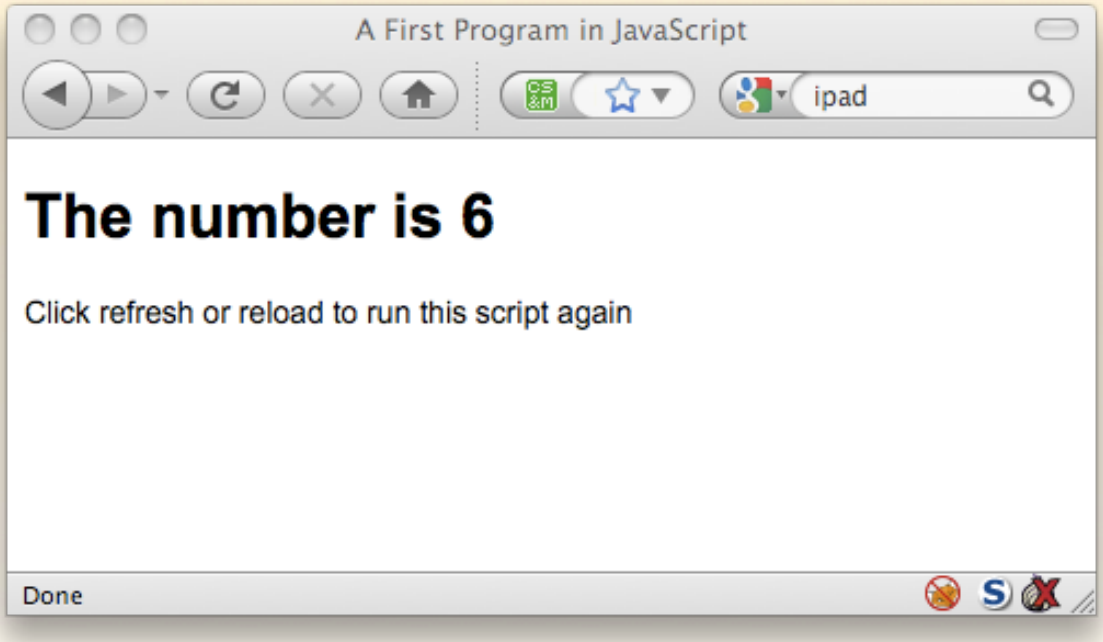
Input

•We get the following prompt:



New Page

•A new page is then generated by the browser:



Perl, Javascript, PHP & Python

•We are going to compare four scripting languages: Perl, Javascript, PHP, and Python.

- As we have seen, Javascript and PHP are embedded in HTML.
  - Javascript is a client-side language.
  - PHP is a server-side language.
- Perl and Python are self-standing but are also used in server-side CGI programming.
- The syntax of all four languages has much in common as, like Java, statements and expressions are all based on C.
- The order of execution of expressions in all these languages is the same as in C (and Java).

String Processing

- In scripting languages, strings are considered to be ***scalar types***.
- (In Java, they are objects and in C they are arrays of characters.)
- Processing strings is central to scripting languages and, as we will see later, they have many more string manipulation and pattern matching operators than we are used to in Java etc.
- (But in Python strings are objects too. Everything is an object/ reference type.)

Dynamic Typing

•As in other programming languages, we have variables and they are assigned values.

- However, scripting languages often differ from languages such as Java in that they are often ***dynamically typed***.
- In Javascript, for example, we do not give the type of a variable in its declaration (we need not declare it!) and the variable can be assigned values of differing type depending upon how it is used.
- Scripting languages usually have an operator (similar to *instanceof* in Java), e.g. *typeof* in Javascript, to find the current type of a variable.

- Dynamic typing *may* go with weak typing.
- Python is dynamically typed and strongly typed.

Static Typing

•Java has *static typing* – i.e. it is checked by the compiler at compile time.

- We must declare the types of all variables and this is fixed at compile time.
- A big advantage of static typing is that it can show up many logical errors at compile time and catch typos in variable names.
- These errors may otherwise go unnoticed and lead to incorrect results.
- Static typing also leads to more efficient code.

Dynamic Typing

•Dynamic typing is more flexible. As scripting languages are primarily dealing with strings, the need for static type checking is deemed less important.

- We have automatic type conversion between strings and numbers.
- Consider the Javascript:

```
a = 20;
b = “4”;
c = a + b;
```

  - The + operator is overloaded - it can mean addition or string concatenation.
- In the above example, ‘c’ will be given the string value 204, which is not necessarily what you expected or intended.

Dynamic Typing

•Consider:

```
c = a - b;
```

- In this case, the value of **`c`** is the integer **`16`**.
- What is the value of: **`a + (b - 0)`** ?

•Perl and PHP use the same conversion rules, but get round the problem of overloading the + operator by using . as the concatenation operator. This removes the ambiguity.

•Python will require a cast to be used.

Dynamic Typing

•The problem of dynamic typing is that it can lead to run-time errors, as in:

```
d = “hello”;
c = a - d;
```

– Perl and PHP get round this problem by defaulting to the value 0 when a string does not represent a number. However, that can lead to unintended results.

Dynamic Typing

- In PHP and Perl, we do not have to declare variables before we use them and they can be assigned values of differing types.
- However, declaring variables before use helps cut out errors and so a Perl programmer can add the line:

```
use strict;
```

- to ensure that more checking takes place and that variables have to be declared before use.

- Note that Perl uses three different prefixes:
  - $ for scalars
  - @ for arrays
  - % for hashes
- In a sense, this is a loose form of data typing.

Numbers

•In Perl, the only kind of scalar data is strings and numbers. We do not distinguish between integer and fractional numbers.

  – If possible, values are held as integers and if not as double length floating point numbers.

- Javascript also has only number type (no special float etc).
- In PHP on the other hand, we distinguish between integer and fractional numbers and use ***casts*** to convert from one to the other.
- Python is more like C, with int, long, float, and *complex*

Arrays

•In Perl, all scalar variables are prefixed with a $ symbol as in:

```
$sum = 0;
$day = “Monday”;
```

•Arrays are prefixed with an @ symbol as in:

```
@browser = (“Firefox”, “IE”, “Opera”);
```

•However, when we access array elements, we use a $ symbol as in:

```
$whichone = $browser[2];
```

Arrays

•PHP always prefixes variable names with a $ symbol. The **`browser`** array is defined as:

```
$browser = array("Firefox", "IE", "Opera");
```

•while the array access is the same as in Perl:

```
$whichone = $browser[2];
```

•These arrays are similar to **`ArrayList`** in Java in that elements can always be added to the end.

Arrays

Javascript is closer to what we expect in Java except that we do not have to state the type of the array elements.

```
var browser = new Array(3);
var whichone;
browser[0] = “Firefox”;
browser[1] = “IE”;
browser[2] = “Opera”;
whichone = browser[2];
```

Arrays

Python does not have arrays built in – but they are part of an "array" library.

Lists are the fundamental sequence data type for Python

```
browserlist = ['Firefox','IE','Opera'];
whichone = browserlist[2];
```

•or

```
browserlist = ["Firefox","IE","Opera"];
whichone = browserlist[2];
```

•Python also has tuples

```
browsertuple = ('Firefox','IE','Opera')
whichone = browsertuple[2];
```

•Tuples are like read-only lists.

Associative Arrays

•As string processing is a central activity in scripting languages, they have facilities for ***associative*** arrays where we can access an element using a string index.

- We refer to the accessing string as the ***key*** and the accessed string as the ***value***.
- They are extensible, i.e. we can add new ***key***, ***value*** pairs.

•Associative arrays are important in scripting languages as they give us a natural way of linking two strings (Java has a *Properties* class that provides very similar behaviour, and objective C has an NSDictionary/NSMutableDictionary class with similar properties).

•In the following example, we associate the name of a person with an office.

Associative Arrays

- In ***Javascript*** we can have:

```
var offices = new Array();
var where;
offices[”Smith"]="4B84";
offices["Shankland"]="4B62";
offices["DeptOffice"]="4B80";
where = offices["Smith"];
```

- ***PHP*** declares them as:

```
$offices = (”Smith" => "4B84", "Shankland" =>
"4B62", "Dept Office" => "4B80");
$where = $offices[“Smith"];
```

- In Perl, we refer to them as ***Hashes*** (a series of ***key***, ***value*** pairs):

```
%offices = ("Clark", "4B84", "Shankland",
  "4B62", "DeptOffice", "4B80");
$where = $offices{Clark};
```

Associative Arrays

- As an alternative in Perl, we can write:

```
%offices = (Clark => "4B84", Shankland =>
"4B62", DeptOffice => "4B80");
```

  - Note that we do not have to put the key in quotes.
  - Accessed in the same way
- In Python, we have ***dictionaries*** (key, value) pairs:

```
offices = {”Smith”:"4B84", "Shankland”:"4B62",
"Dept Office”:"4B80");
where = offices[“Smith"];
```

Reminder: Hash Tables

•With ordinary arrays, the elements are stored in sequence and so access is straightforward.

- Associative arrays are usually implemented as ***hash tables***.
- In a hash table, we have a set of ***key***, ***value*** pairs.
- Suppose that space is allocated for 200 such pairs.
  - To determine where a pair is to be stored, we apply a ***hash function*** to the ***key*** to get a number in the range `0` to `199`.
  - That tells us where in the table we are to store the pair.
  - To look up the table, we again take the key and again apply the hash function. That gives us where the pair is stored.
  - The problem is when two keys hash to the same location giving a ***collision***.
    - Handling collisions complicates matters and slows down access.

Associative Arrays

•Java has a **`Hashtable`** class that gives us the same effect as an associative array:

```
Hashtable offices = new Hashtable();
offices.put(”Smith", "4B84");
offices.put("Shankland", "4B62");
offices.put("Dept Office", "4B84");
```

•We then retrieve information by statements such as:

```
String where = (String) offices.get("Smith");
```

foreach

•Perl, PHP, Javascript and Python have the standard C control structures including the **`for`** loop. However we cannot use a **`for`** loop to visit all the elements of an associative array as they are not in any defined order in the hash table.

- In Java we have ***iterators*** that enable us to visit each element of a collection once, and a for-each loop to simplify access to collections.
- Scripting languages usually contain something like a **`foreach`** loop.

foreach

•Let us assume that we want to present a list of the people and their offices. In PHP, we can write:

```
foreach($offices as $key => $value) {
  echo “<p>Office of ”.$key.“ is at ” .$value.”</
  p>”;
}
```

•An alternative is:

```
while(list($key, $value) = each($offices)) {
  echo “<p>Office of ”.$key.“ is at ” .$value.”</
  p>”;
}
```

•In both cases, we are representing the associative array as a set of key, value pairs which we refer to respectively as $key and $value.

•In the body of the *foreach* loop, we can then refer to $key and to $value.

An Example

•Hence if we have the associative array:

$offices = (”Wang”=>"4B53", "Shankland”=>"4B62", "Dept Office”=>"4B80");

•We could print out the contents of the array (with suitable HTML formatting) as follows:

```
foreach($offices as $key => $value) {
  echo "<p>Office of ".$key." is at ”.$value."</p>";
}
```

foreach

- Perl has a **`foreach`** construct to get a list of either the keys or the values. To get both together, we write:

```
while (($key, $value) = each(%offices)) {
        print "<p> Office of ".$key." is at".$value."\n";
}
```

- To visit all the elements in Javascript, we write:

```
for (key in offices) {
   document.writeln("<p>Office of "+key+" is at "+
      $offices[key] + "</p>");
}
```

- To visit all the elements in Python, we write:

```
for (key in offices):
   print("<p>Office of "+key+" is at "+offices[key] + "</p>")
```

String Processing

•Processing strings is a central activity of scripting languages. In PHP and Perl, a single quoted string such as:

```
'Hello James,\n how are you?'
```

•is not interpreted, but treated literally. Double quoted strings are pre-processed and so **`\n`** is replaced by a newline.

String Processing

•Character sequences beginning with the backslash '\' character are replaced as follows:

\n is replaced by the newline character
\r is replaced by carriage-return
\t is replaced by tab
\$ is replaced by $
\" is replaced by a single double quote
\\ is replaced by a single backslash.

•Javascript supports both types of quotes and pre-processes the above character sequences in both.

PHP

•In PHP, if the value of **`$name`** is **`James`**, then the string:

```
"Hello $name, how are you?"
```

•is pre-processed to give:

```
"Hello James, how are you?"
```

•To get the same effect in Perl, we would write:

```
"Hello ${name}, how are you?"
```

Pattern Matching & Regular Expressions

•We are going to look at the facilities that PHP provides for processing strings. The facilities that other scripting languages provide are very similar. Java 1.4 introduced classes **`Pattern`** and **`Matcher`** for similar facilities (shows the power of an OO language - it really is extensible although added features can be less natural compared with when a feature is fully integrated in a language).

- Central to the approach is the use of ***regular expressions*** which are used to represent the required patterns.
- PHP has ***many*** string processing functions. We will only look at one in detail.

PHP Example – *preg_match*

•Suppose we want to find if a given string contains a pattern. Moreover, we want to get a list of the matches. We can call the function **`preg_match`** which has the following form:

```
int preg_match(string $p, string $t, array $r)
```

•It looks for the pattern **`$p`** in the string **`$t`** and puts the matches for sub-expressions in the array **`$r`** with the complete matched expression in **`$r[0]`**. If a match is found, the function returns **`1`**, **`0`** otherwise. The array parameter is optional.

•(There are some other optional parameters too, not covered here.)

PHP Example

•The ***meta-character*** . (dot) matches any character. Hence the result of:

```
$sentence = "The man cut his toe";
if (preg_match(”/t.e/", $sentence, $regs))
{
    echo "found";
}
else
{
    echo "not found";
}
```

•will be to output **`found`** and to set **`$regs[0]`** to **`"toe"`**.

Meta Characters

•In PHP (and Perl and Javascript), we enclose a regular expression in delimiters, as follows:

/pattern/

•Perl and Javascript use forward slashes, PHP is more flexible.

•Regular expressions make extensive use of meta-characters. The following use of meta-characters is true for Perl, Javascript and PHP.

- The symbols **^** and **$** act as ***anchors***.
  - The symbol **^** matches at the start of a string. Hence **^Hello** only matches **Hello** if it appears at the start of a string. (but see next)
  - The symbol **$** matches at the end of a string. The pattern **dog$** will only match if **dog** is the last word in the string.

Meta Characters

•A string of characters in ***square brackets*** represents ***one*** of the characters. Hence `[aeiou]` matches any single vowel.

•The pattern `[a-z]` matches any single lower case letter while `[a-zA-Z]` matches any single letter. With some meta-characters, their meaning depends on the context.

– Hence `[^a-z]` matches any character that is ***not*** a lower case letter. In this context, ^ means 'not'.

•The pattern `[0-9]+` matches ***one or more*** digits, `[0-9]*` matches ***zero or more*** digits while `[0-9]?` matches ***zero or one*** digit.

– **?** represents zero or one

– ***** represents zero or more

– **+** represents one or more

Meta Characters

- You can also use ordinary brackets and so:
  - `/(very )*big/` matches: **`big`**, **`very big`**, **`very very big`** etc.
  - `/(very )+big/` matches **`very big`**, **`very very big`** etc.
- You can give a range for the repetitions:
  - `/(very ){2}/` matches **`very very`**
  - `/(very ){1,2}/` matches **`very`** and **`very very`**
  - `/[0-9]{1,4}/` matches a string of digits of length between `1` and `4`.
- We use `|` to represent alternatives. Hence:
  - `/(The)|(the)|([aA])|(an)|(An)/` represents a definite or indefinite article.
- To represent a meta-character with its normal symbol, we precede it with `\`. For example to display a `[` we use `\[`

Modifiers

- Built-in modifiers for useful functionality:

  `/very/i` matches: **`very`**, **`VERY`**, **`VeRy`** etc.

  `/\bweb\b/` matches on whole word boundaries only, therefore matches **`web`** but not **`website`**.

Aside: Python

- Python regular expression language is very similar. Eg. usernames, passwords and emails:

```
/^[a-z0-9_-]{3,16}$/
```

- Starts and ends with 3-16 numbers, letters, underscores or hyphens (in any combination)

```
/^[a-z0-9_-]{6,18}$/
```

- Starts and ends with 6-18 letters, numbers, underscores, hyphens (in any combination)

```
/^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/
```

- Starts with a combination of letters, numbers, underscore, hyphen or dot (at least one character), followed by the at sign (snail, monkey tail, strudel) then two groups separated by a . and matching to the end where the first group has letters, numbers, hyphen, dot (at least one char) and the second group has 2-6 chars which are letters or dots.

PHP Example

- What is the effect of:

```
$sentence = "The man cut his toe";
if (preg_match(”/[tT].e/", $sentence, $regs))
  {
    echo "found";
} else {
    echo "not found";
}
```

PHP Program

- The following PHP program is at:
- http://www.cs.stir.ac.uk/courses/CSC9Y4/lectures/toe.php

```
<html><head>
 <title>A Program in PHP</title>
</head>
<body>
<h1>The toe</h1>
<?php
$sentence = "The man cut his toe";
if (preg_match (”/t.e/", $sentence, $regs)){
  echo "matched pattern is: ".$regs[0]."<br />";
}
else {
  echo "not found" . "<br />";
}
if (preg_match (”/[tT].e/", $sentence, $regs)) {
  echo "Matched pattern is: ”.$regs[0]."<br />”;
} else {
  echo "not found" . "<br />”;
}
?>
<h1>The end</h1></body></html>
```

PHP Processed by Server

•When we are developing an HTML program then you can just open the file in a browser. You do not need to have uploaded the file to your ISP. However if we just go to:

\\Fs0\courses\CSC9Y4\www\lectures\toe.php

•(where the file is stored) and open it in a browser then we only get:

The toe

The end

•as the PHP has to be processed by a server.

PHP Example 1

•Let us look at a pattern that will match an email address such as:

Fred_29.Smith@stir.ac.uk

•The pattern is:

```
/^[a-zA-Z0-9_\.]+@[a-zA-Z0-9_]+\.[a-zA-Z0-9_
  \.]+$/
```

•If we want to extract substrings then we can split the pattern up by using round brackets (). Each of the partial matches can be stored and used in further processing.

Substrings

- The code:

```
$p = ”/(^[a-zA-Z0-9_\.]+)@([a-zA-Z0-9_]+)\.([a-zA-Z0-9_
\.]+)$/";
$addr = "Fred_29.Smith@stir.ac.uk";
if (preg_match ($p, $addr, $regs)) {
  echo  $regs[0] . “<br />”;
  echo  $regs[1]."<br />".$regs[2]."<br />".$regs[3];
}
else {
  echo "Invalid address format: $addr";
}
```

- Will produce the output:

Fred_29.Smith@stir.ac.uk
Fred_29.Smith
stir
ac.uk

PHP Example 2

- The following code snippet takes a date in ISO format (YYYY-MM-DD) and prints it in DD.MM.YYYY format:

```
<?php
    $date = "2004-4-02";
  if (preg_match ("/([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})/",
$date,$regs)) {
        echo "preg matched part is: ".$regs[0]."<br />";
         echo "Transformed date is: ";
         echo "$regs[3].$regs[2].$regs[1]";
        }
        else
        {
        echo "Invalid date format:". $date;
}
?>
```

- The output string is: **02.4.2004**

PHP Example 3

- The following code snippet takes a string and checks if the password is strong (at least one upper case letter, one lower case letter, one digit, and 8 characters long):

```
<?php
$password = "Fyfjk34sdfjfsjq7";

if (preg_match(
"/^.*(?=.{8,})(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).*$/",
$password)) {
    echo "Your password is strong.";
} else {
    echo "Your password is weak.";
}
?>
```

Replacing

- Another standard operation is to replace one string by another.
- Suppose that we wanted to replace the value of the string **`$rude`** by the string `'%!@*'` wherever it appeared in the string **`$fback`**.

- This can be achieved by the following call:

```
$fback = str_replace($rude, '!@*', $fback);
```

Replacing

- Suppose now that you were monitoring a chatroom or course feedback and that there was a list of words that you wished to suppress.
- Instead of assigning **`$rude`** to a single string, it can be assigned to an array of strings!
- Similarly **`$fback`** can be an array of strings that you wish to be processed in this way and we can specify a list of replacements.

Using an Array

- In the following example, one word will be replaced.

```
$rude = "bother";
$sentence="Oh bother, I have forgotten my dashed key.";
$rep = str_replace($rude, "!@*", $sentence);
echo "old: $sentence<br />new: $rep";
```

- In this second example, both the offensive words `bother` and `dashed` will be replaced.

```
$rude = array("bother", "dashed");
$sentence="Oh bother, I have forgotten my dashed key.";
$rep = str_replace($rude, "!@*", $sentence);
echo "old: $sentence<br />new: $rep";
```

Dynamic Typing

•This shows the advantage of dynamic typing from the point of view of the user. We can make the call with a mixture of strings or arrays of strings and they will all be used appropriately.

•Of course, behind the scenes in the implementation of **`str_replace`**, the *implementer* had to write code to determine the current type of each parameter, but that does not have to bother the user of the language.

Static typing

- Let us now consider what would happen with static typing.
  - We could use overloading, but as each of the three parameters has two possibilities that means that we need 2x2x2 = 8 different overloaded methods!
  - If each parameter had three possibilities then we would need 27 overloaded methods.
  - In an OO language, inheritance might reduce the problem, but an array of strings is not a subclass/superclass of a string.

First class functions

•Functional features are finding their way into other paradigms. Eg. we can define functions in Javascript as follows:

```
<script LANGUAGE = "JavaScript">
        function square(x) { return x*x;}
  var y;
  y = square(3);
  document.writeln("<p>Square of 3: " + y + "</p>");
</script>
```

```
end: 4
Match found
text: foo3
start: 4
end: 8
```

• Note that the end is one beyond the matched character position in the string. We match the `foo2` and `foo3`, but not `foo4`.

Assigning functions

- However, we can also assign a function to a variable.

```
<script LANGUAGE = "JavaScript">
    var z = square;
    document.writeln("<p>Square of 2: "
            + z(2) + "</p>");
</script>
```

Anonymous functions

- We can assign a function literal directly to a variable.

```
var sq = function (x) {return x*x;};
var a = sq(5);
```

- Or even apply an anonymous function:

```
var b = (function(x) {return x*x;}) (6);
```

- We can assign the code of a function to a string as in:

```
var st = sq + "";
 document.writeln("Function is: " + st);
```

- The output is:

```
Function is: function (x) {return x*x;}
```

- The Javascript code is available at:
- http://www.cs.stir.ac.uk/courses/CSC9Y4/examples/ny10.html

Summary of Scripting Languages

- Scripting languages share core features of programming languages:
    1. Basic (built-in) types.
    2. Strings (and string handling including pattern matching)
    3. Data structures (associative arrays, but also abstract data types).
    4. Control structures.
    5. Modular programming.
    6. Type regime.
    7. Operating environment.

- What makes them special is the context in which they're used (often on the www, often interpreted) and the nature of tasks for which they're used (often string processing, so they have data types and operations to facilitate that).

Appendix: Matching in Java

•So, let us now look at how this is done in Java.

– We represent the pattern as a regular expression.

– The string representing the pattern can contain meta-characters in the same way as we saw with PHP.

– There are two classes that have to be imported:

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;
```

– A large number of methods are available with the `Matcher` class – some examples are given in the next slide. We can use brackets to capture sub-expressions in a similar way to PHP.

Appendix: Java Pattern Matching

- We can write Java code such as:

```
Pattern pattern = Pattern.compile("foo[1-3]");
Matcher matcher = pattern.matcher("foo2foo3foo4");
while (matcher.find()) {
  System.out.println("Match found");
  System.out.println("text:" + matcher.group());
  System.out.println("start:" + matcher.start());
  System.out.println("end:" + matcher.end());
}
```

Appendix: Java Pattern Matching

- This will give the result:

```
Match found
text: foo2
start: 0
```