B211 Internet Computing

Server-Side Scripting with PHP

Learning Objectives

1. Learn how to access and use PHP scripts from the web browser.

2. Learn how PHP scripts operate by following some examples.

3. Understand how PHP relates to HTTP communications between the web client and web server.

Understanding PHP

- Don’t worry too much about understanding the details of the PHP scripts I present in this lecture. They are mainly here so that you can understand some basic concepts in server-side scripting.
    - We will be going through the syntax for PHP in the next lecture.
    - Also, I give the complete scripts here so that you can copy them to do your PHP programming exercises next week.

- You will get a picture of what PHP scripts looks like in this lecture. Return to them after you have learnt the syntax for Perl - they will make much more sense then.

Trying out the scripts

- While reading these lecture notes, start up your browser and try out the scripts as indicated by the screen dumps. All those scripts really do exist at URLs as indicated.

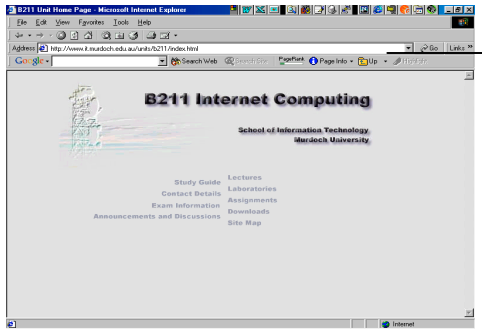
    - All scripts are at the location http://phoenix.murdoch.edu.au/~s12345678/examples/

PHP

- PHP stands for "***PHP: Hypertext Preprocessor***"
    - The definition is recursive, so don't ask what the "PHP" in "PHP: Hypertext Preprocessor" stands for.

- PHP is "*a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML*."

How Server-side Scripting Works



1. Browser requests URL

```
...HTML...
...script code...
...HTML...
...script code...
```

2. Web server invokes interpreter to turn the script code into HTML

```
...HTML...
...HTML...
...HTML...
...HTML...
```

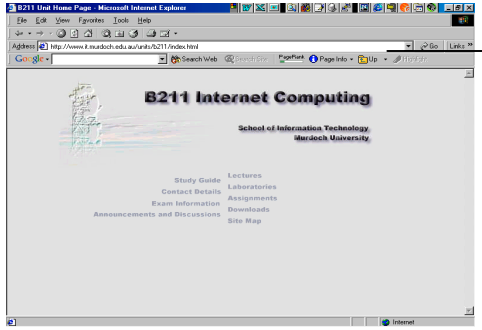
3. Web server sends HTML back to browser

4. Browser reads and displays HTML

Client-side

Server-side

Compared to Client-side Scripting



1. Browser requests URL

```
...HTML...
...script code...
...HTML...
...script code...
```

2. Web server sends HTML and script code back to browser

3. Browser interprets the script code and displays appropriately

Client-side

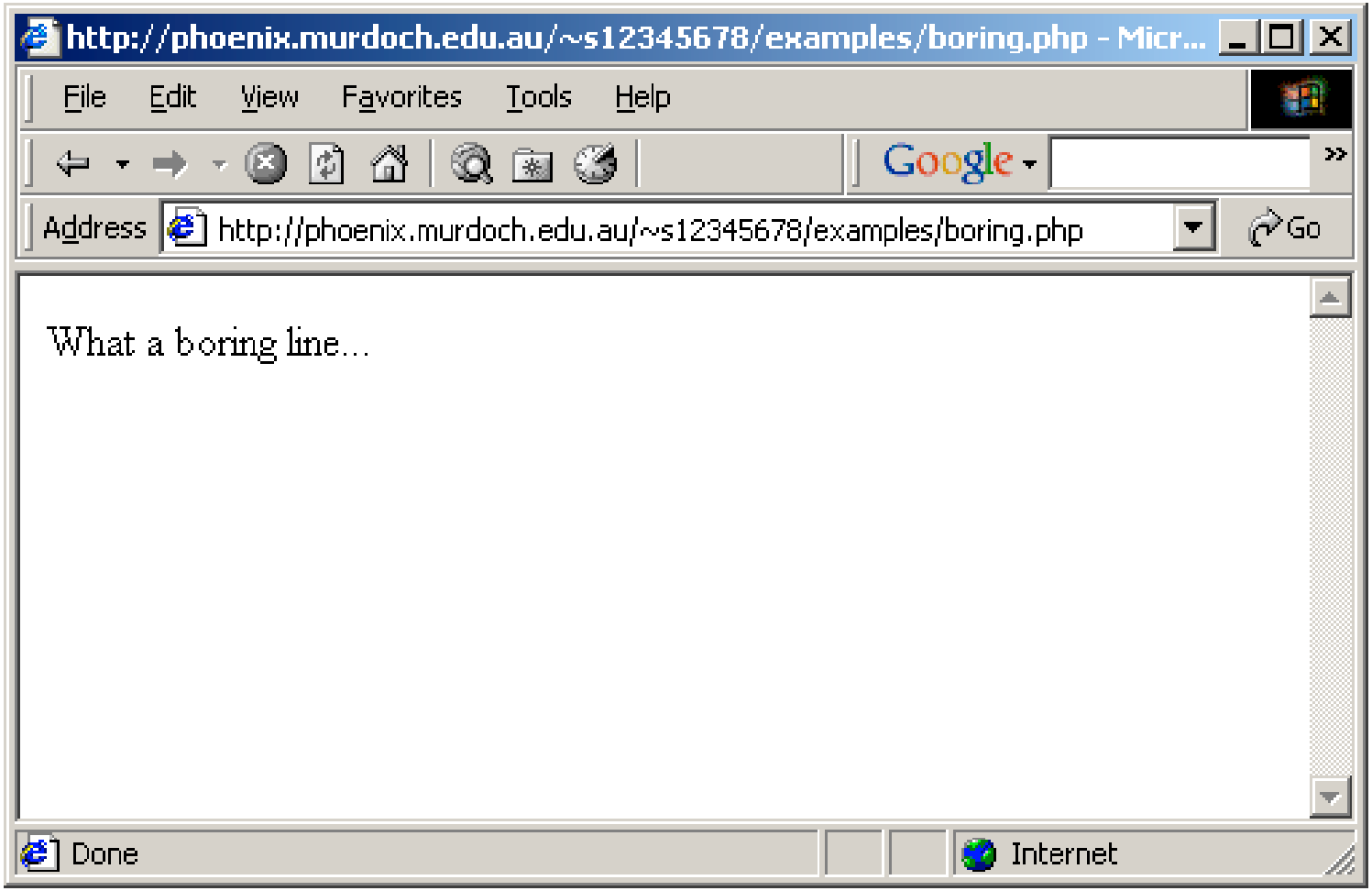
Server-side

A simple PHP script

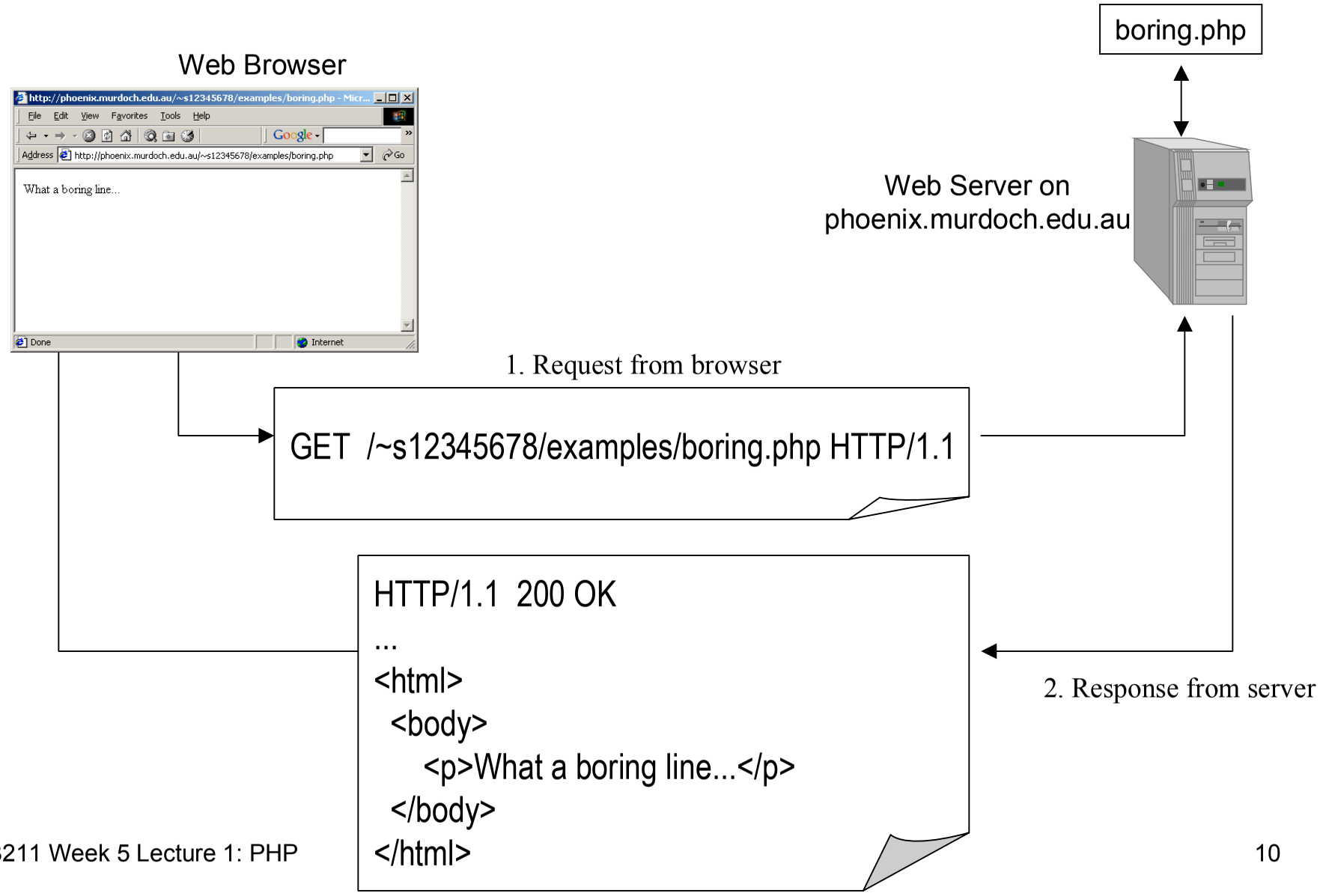
- All it does is print the word in a browser “Hello!”

```
<html>
  <body>
     <? echo "<p>What a boring line...</p>" ; ?>
  </body>
</html>
```

Accessing the script from a browser



What actually happens:



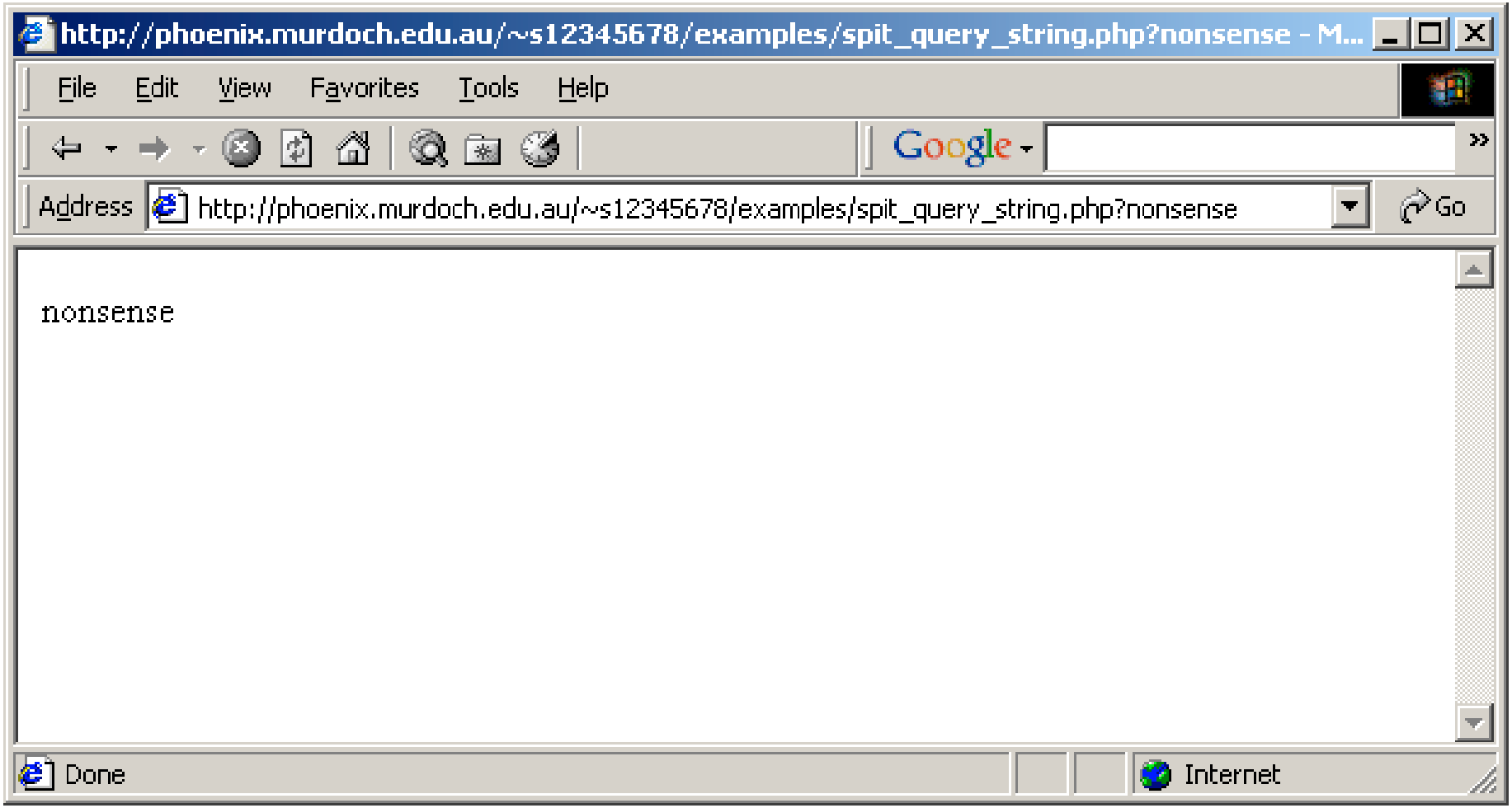
The Script's Query String

spit_query_string.php

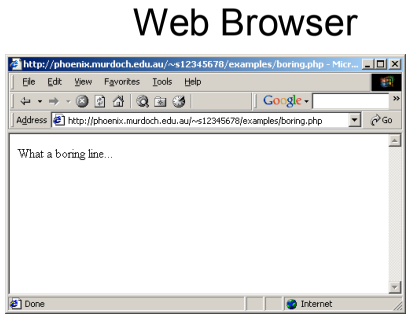
```
<html>
  <body>
    <p>
      <? echo $_SERVER['QUERY_STRING'] ; ?>
    </p>
  </body>
</html>
```

- Whatever parameters this script receives, it sends back to the browser.

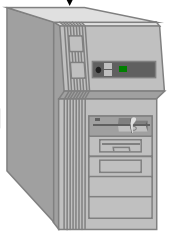
Accessing the script from a browser



What actually happens:



spit_query_string.php



Web Server on
phoenix.murdoch.edu.au

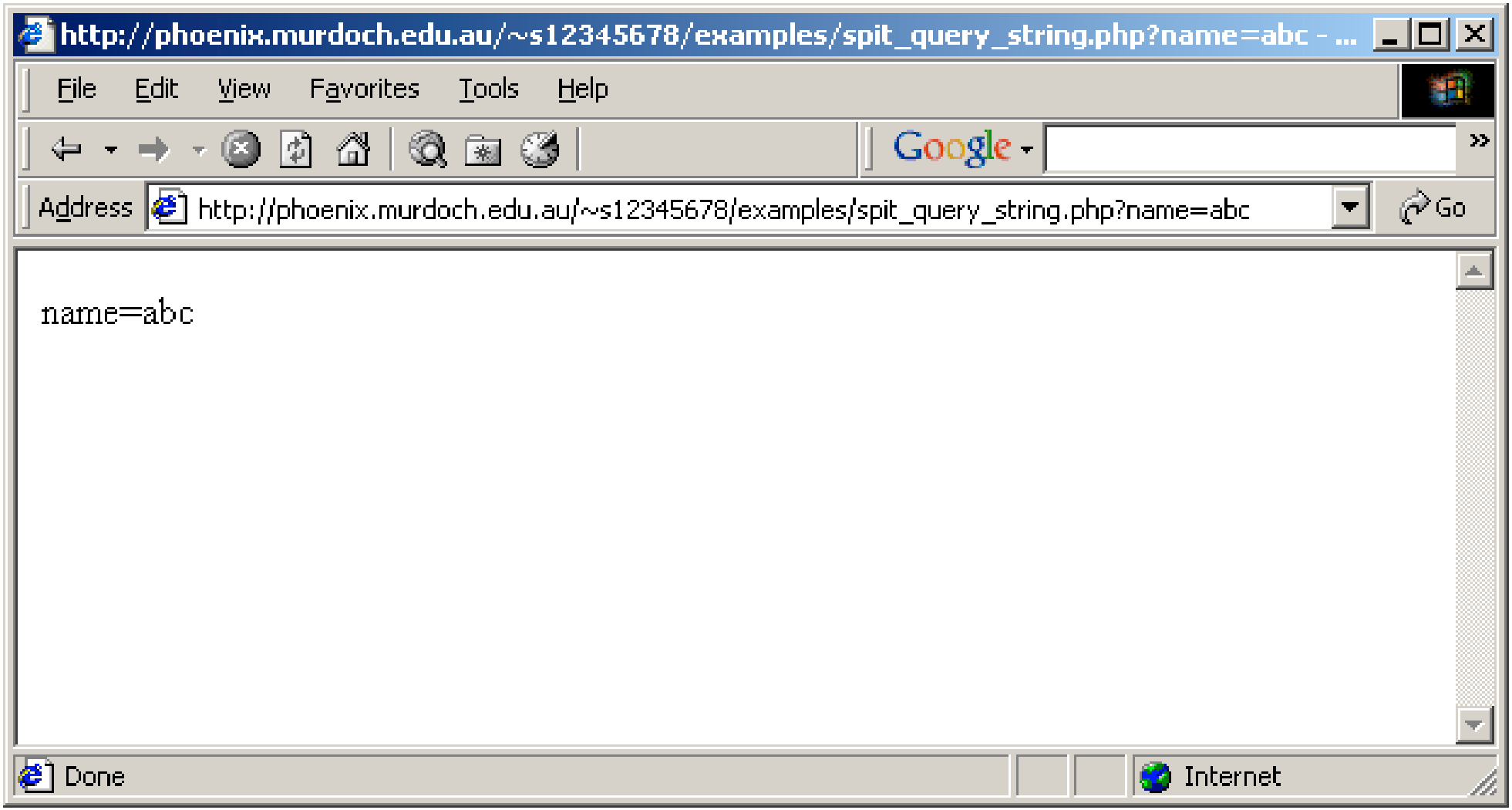
1. Request from browser

```
GET  /~s12345678/examples/spit_query_string.php?nonsense HTTP/1.1
```

```
HTTP/1.1  200 OK
...
<html>
  <body>
     <p>
       nonsense
     </p>
  </body>
</html>
```

2. Response from server

Another example of spit_query_string.php

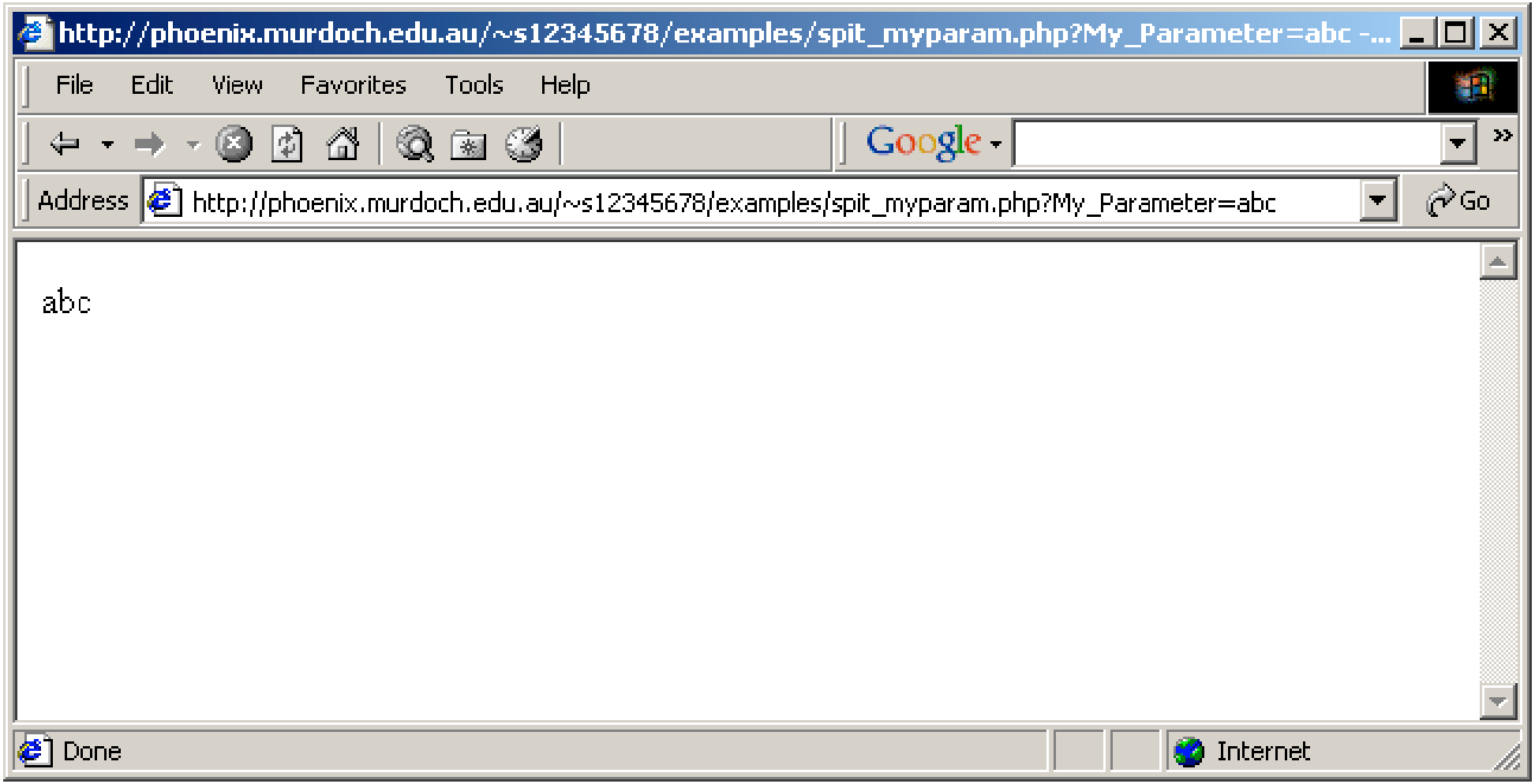


Script Parameter Values

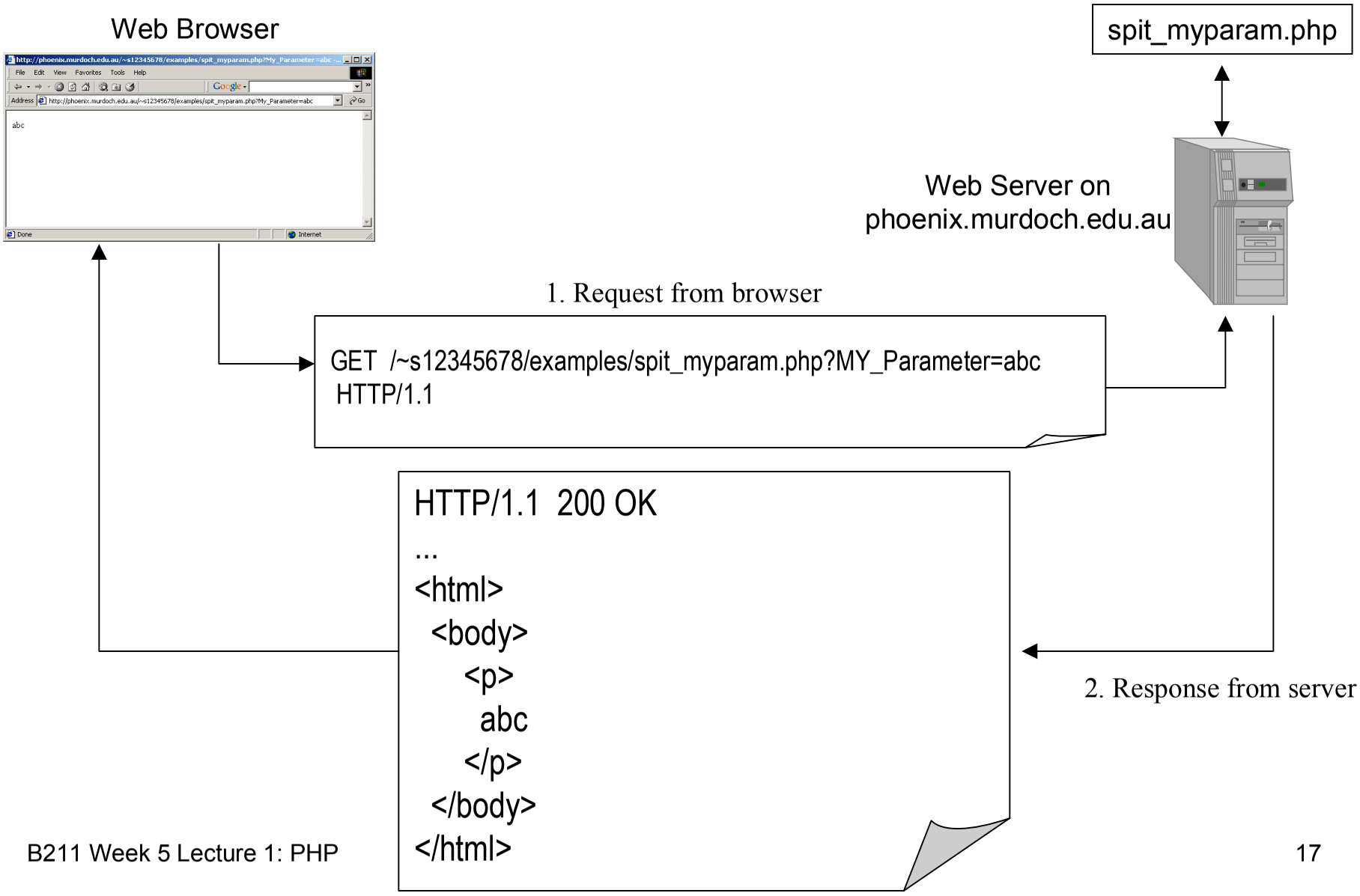
spit_myparam.php

```
<html>
  <body>
    <p>
      <? echo $My_Parameter ; ?>
    </p>
  </body>
</html>
```

Accessing the script from a browser



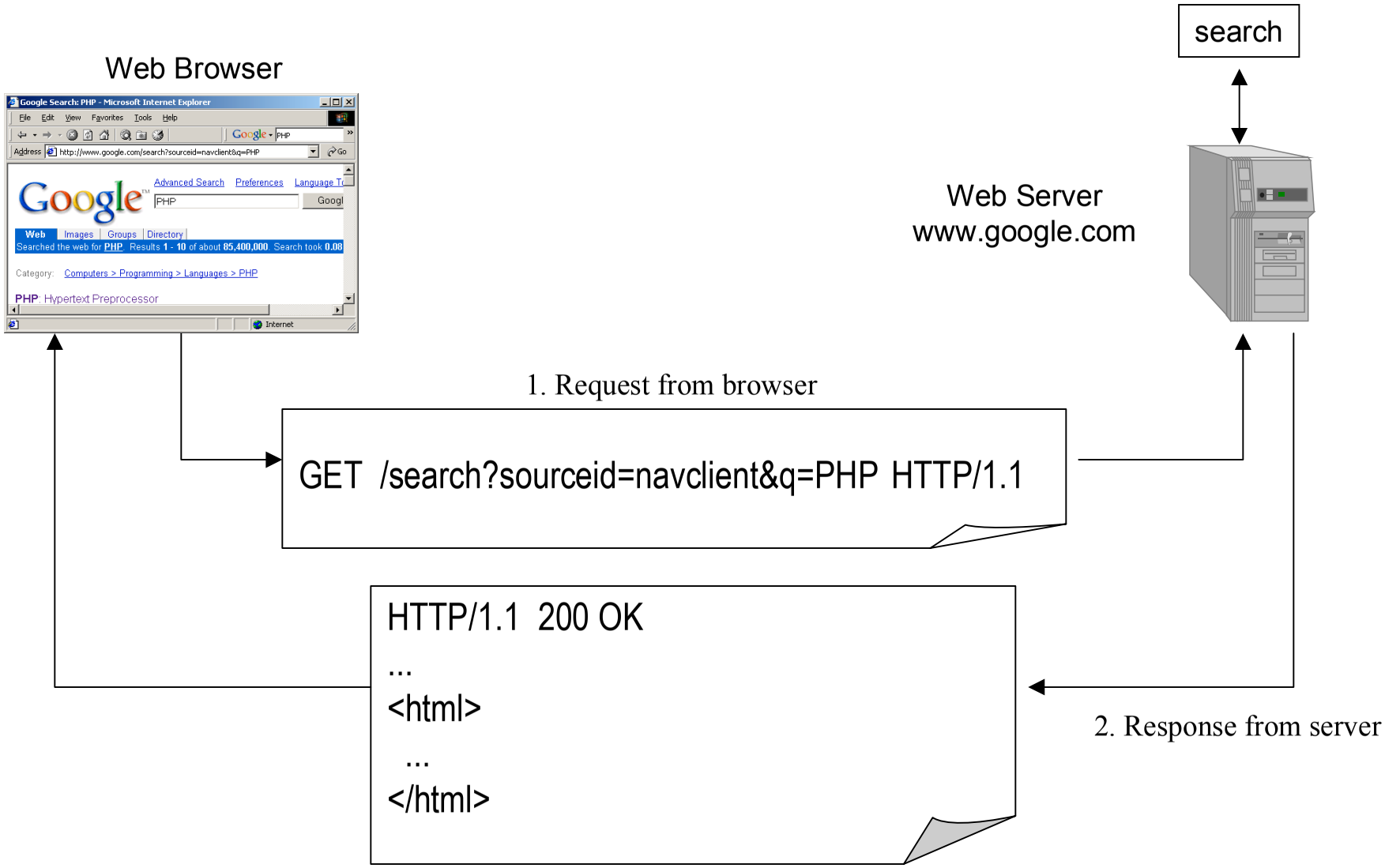
What actually happens:



Using Query Strings

A lot of web applications use query strings to receive data from browsers.

What actually happens:



Query String Format

- A query string of the format “sourceid=navclient&q=PHP” basically means:
    - There are two fields of data, separated by “&”
    - The first data field name is “sourceid”, with value “navclient”.
    - The second data field name is “q”, with value “PHP”.
- Data from HTML forms with method “GET” are sent to server scripts as query strings.

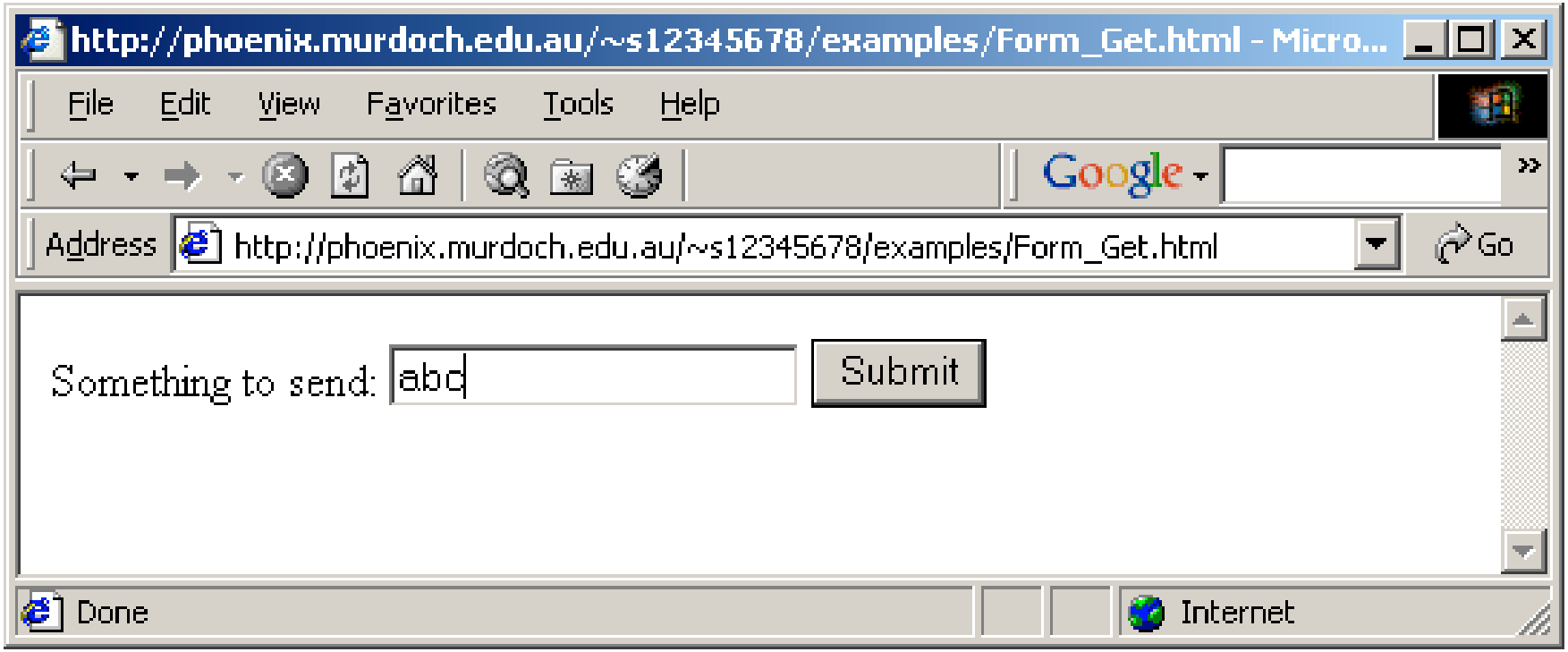
Sending Data from an HTML form

Form_Get.html

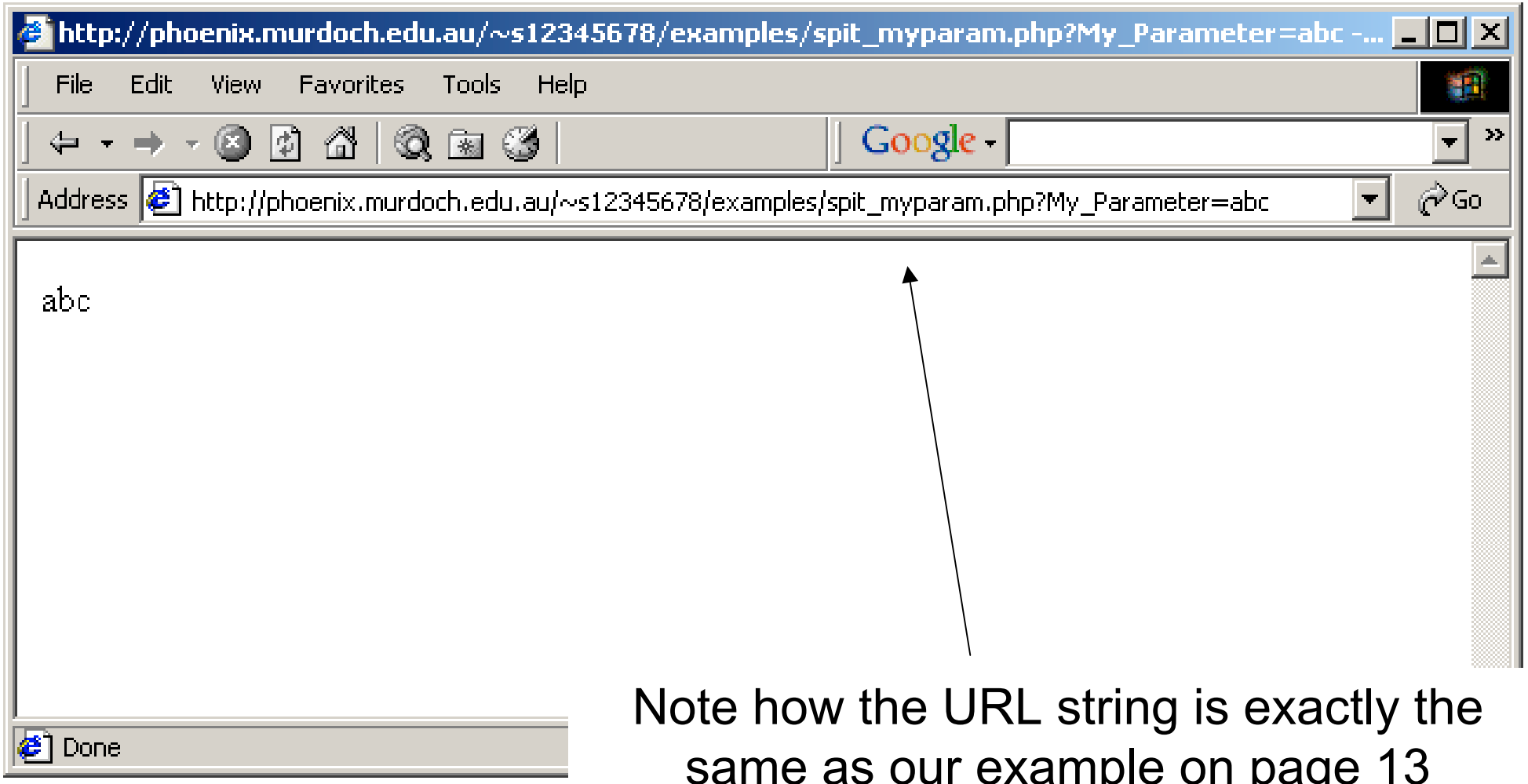
```
<html>
  <body>
    <form method="GET"
     action="http://phoenix.murdoch.edu.au/~s12345678/examples/spit_myparam.php">
      Something to send: <input type="text" name="My_Parameter">
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

- Accessing spit_myparams.php from a form instead of typing it directly in the browser's URL box.

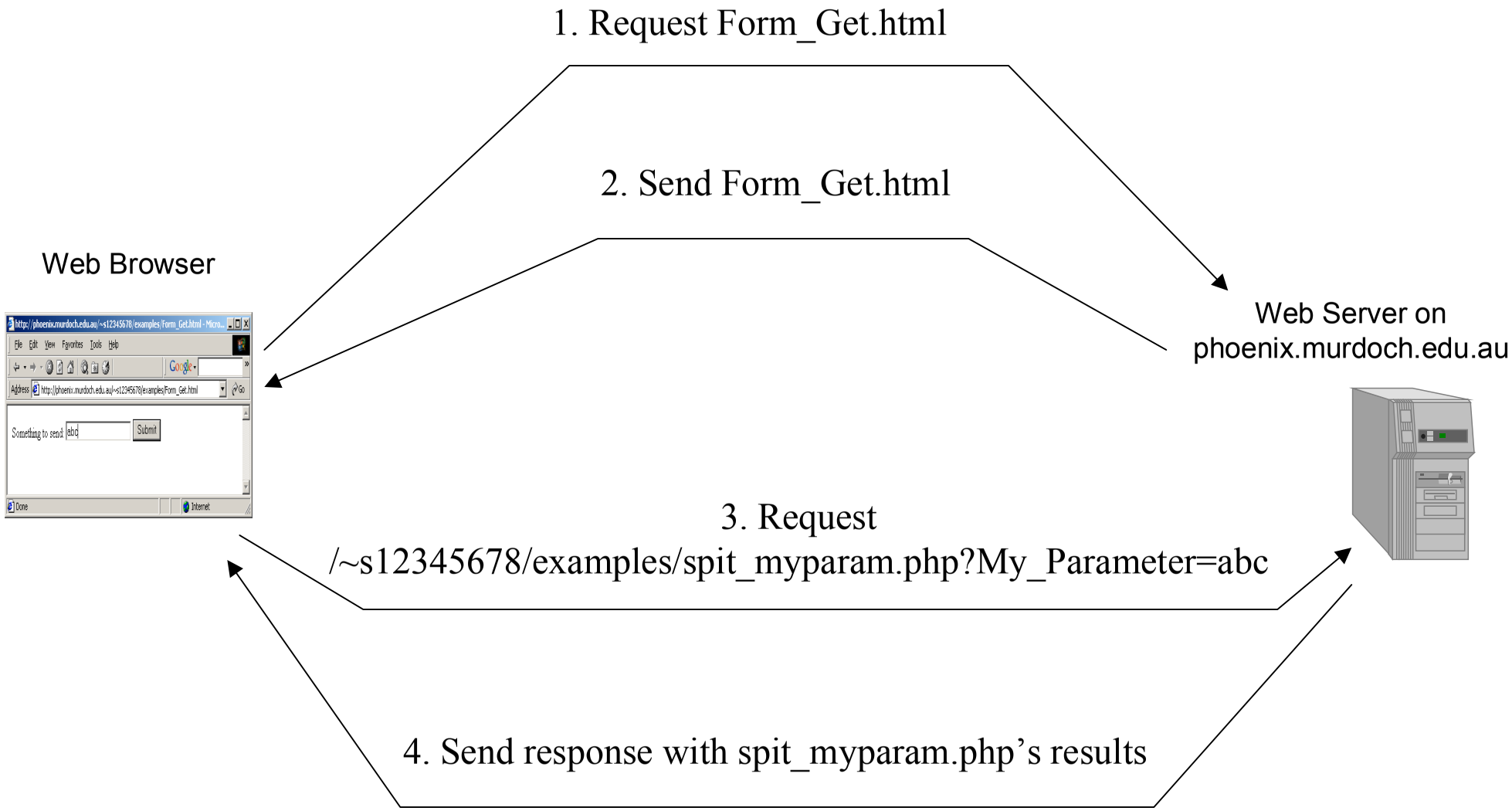
The browser display before submission



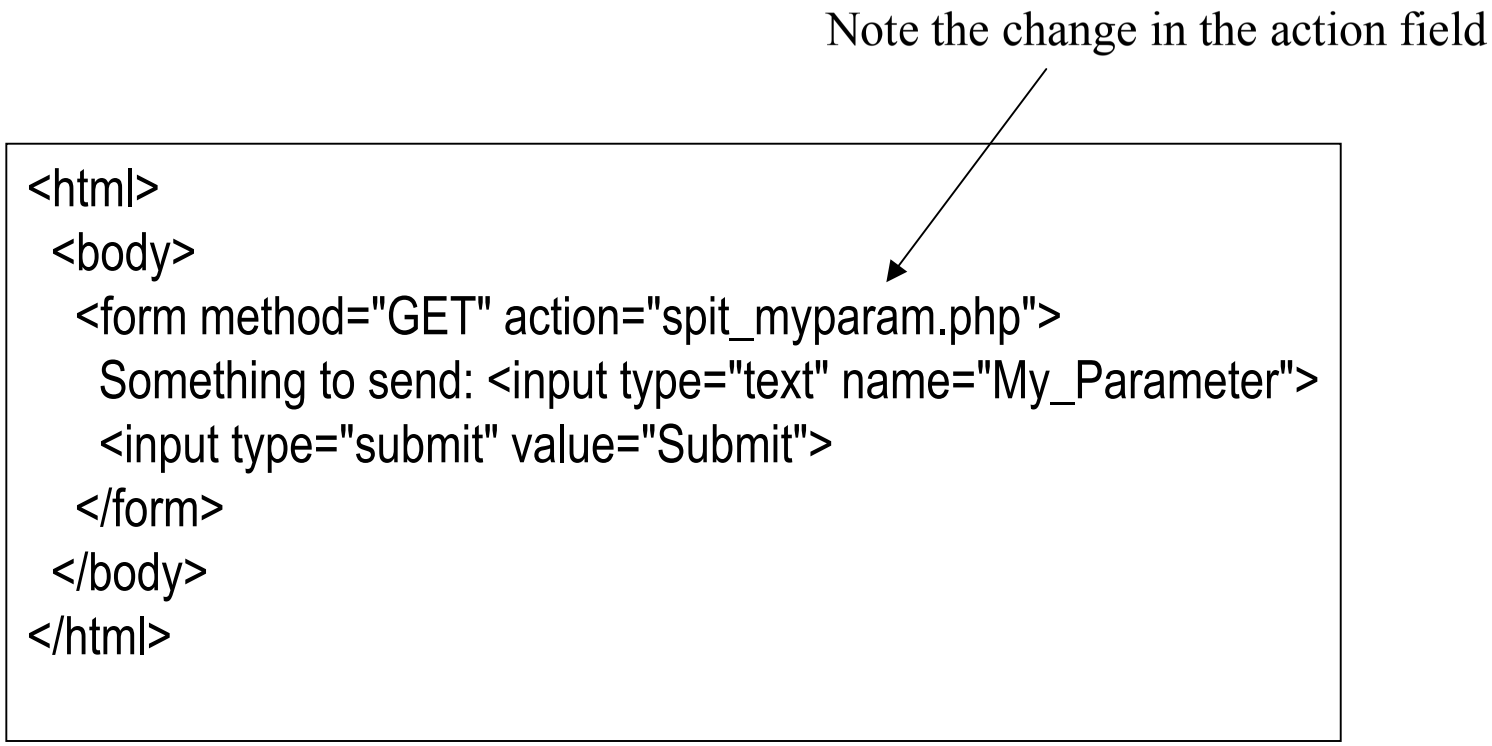
The result after pressing “Submit”



Note how the URL string is exactly the same as our example on page 13


1. Request Form_Get.html
2. Send Form_Get.html
Web Browser
Web Server on
phoenix.murdoch.edu.au
Something to send
abc
Submit
3. Request
/~s12345678/examples/spit_myparam.php?My_Parameter=abc
4. Send response with spit_myparam.php's results

Scripts and Forms in the same directory



Form_Get.html

Scripts and Forms in the same directory

- If the PHP script is in the same directory as the HTML form document, we can put the relative path action="spit_myparam.php" instead of the absolute URL action="http://phoenix.murdoch.edu.au/~s12345678/examples/spit_myparam.php"

- This is better in cases when the script and form will be moved to different directories together.

    - Please use relative paths in your assignment 2, as your files will be copied to an unknown directory for assessment.

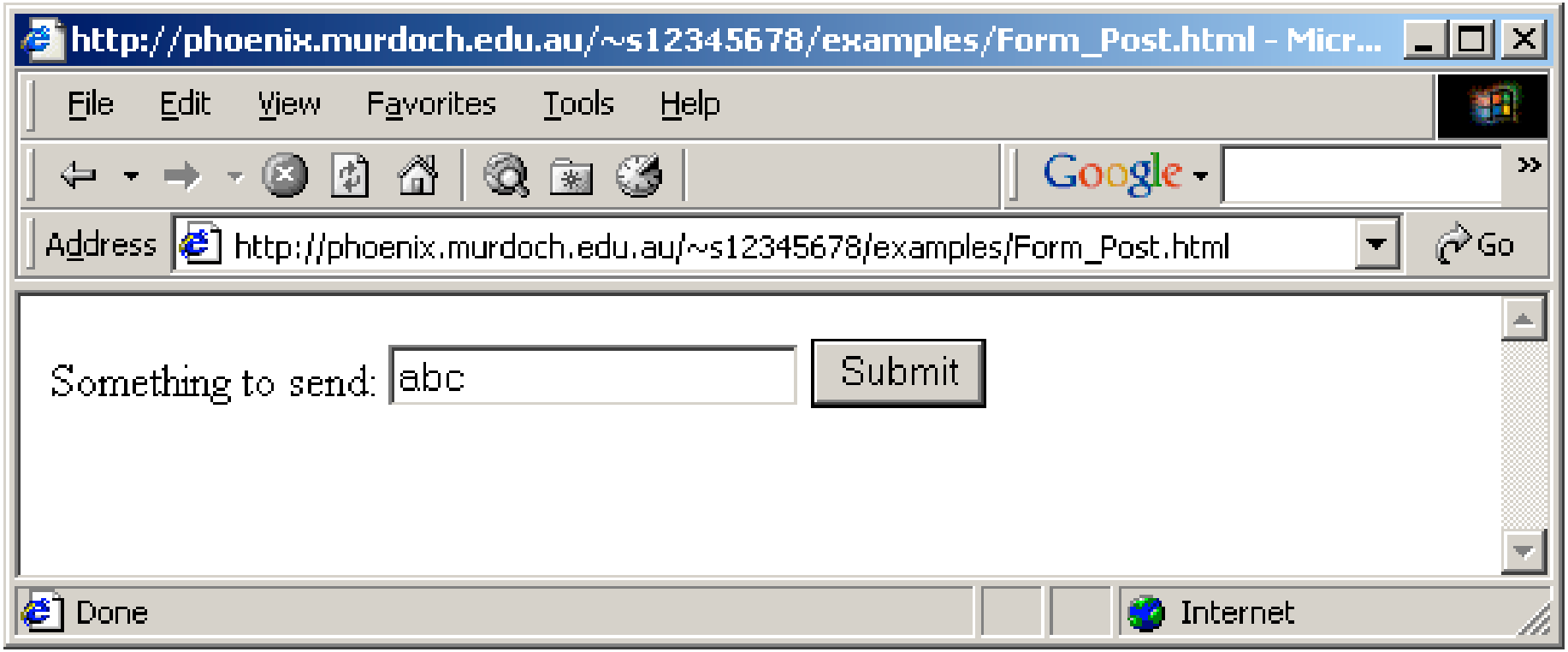
Using POST instead of GET

Form_Post.html

```
<html>
  <body>
    <form method="POST" action="spit_myparam.php">
      Something to send: <input type="text" name="My_Parameter">
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

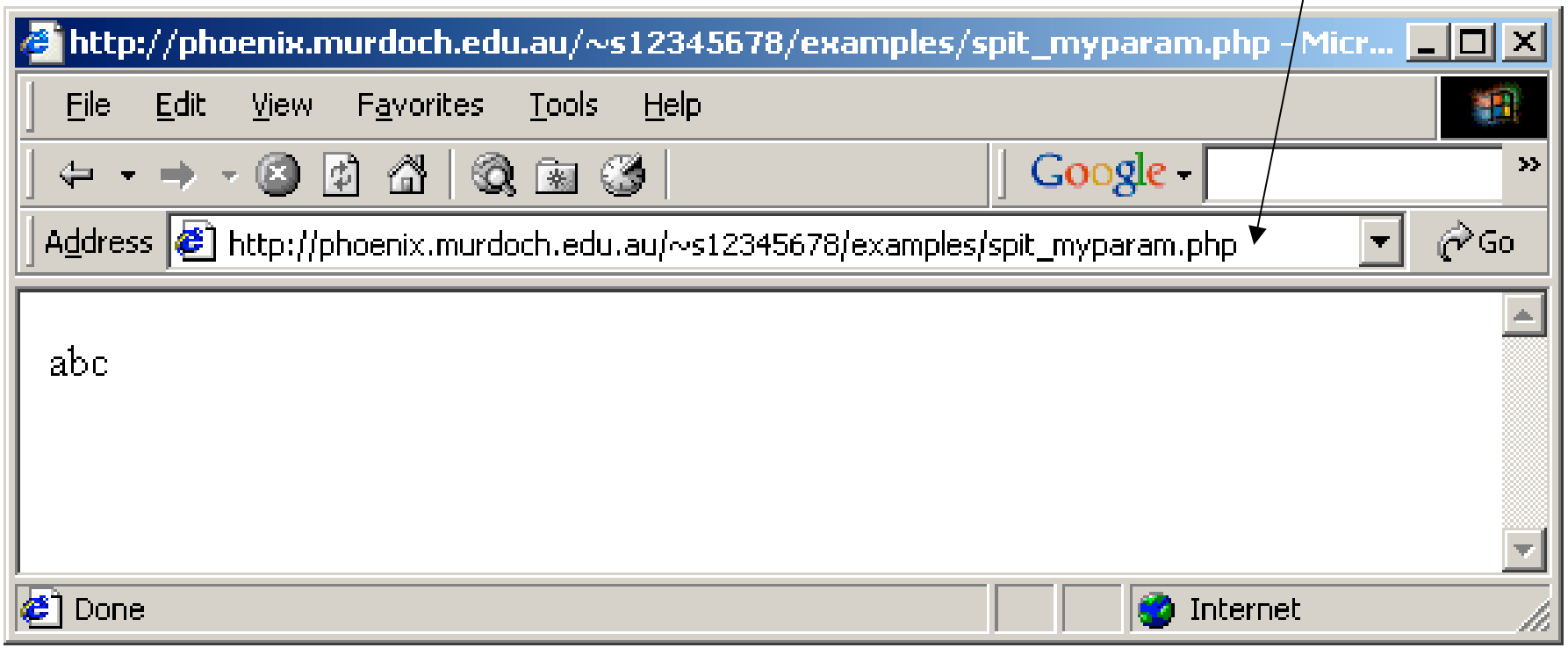
- Note the change from `method="GET"` to `method="POST"`

The form before submission

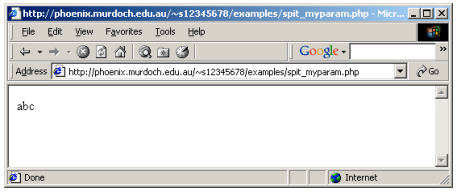


The result after pressing “Submit”

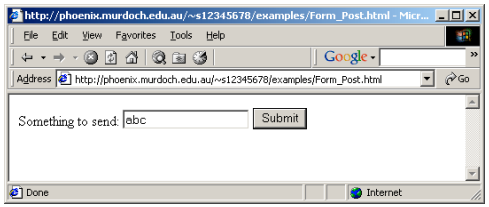
Note how the data are not part of the URL, unlike Form_Get.html. This is useful for passing data you do not want to appear on the screen.



What actually happens:



Web Browser



spit_myparam.php

Web Server on
phoenix.murdoch.edu.au

1. Request from browser, with data in the request body instead of the query string

```
POST  /~s12345678/examples/spit_myparam.php HTTP/1.1
...
My_Parameter=abc
```

```
HTTP/1.1  200 OK
...
<html>
  ...
</html>
```

2. Response from server

Returning Another Form

- Because your PHP result is in HTML, there is no reason why you can't return ***another form*** to the user to fill out, that in turn accesses ***the same or another script***.

Example

Form_Post_Name.html

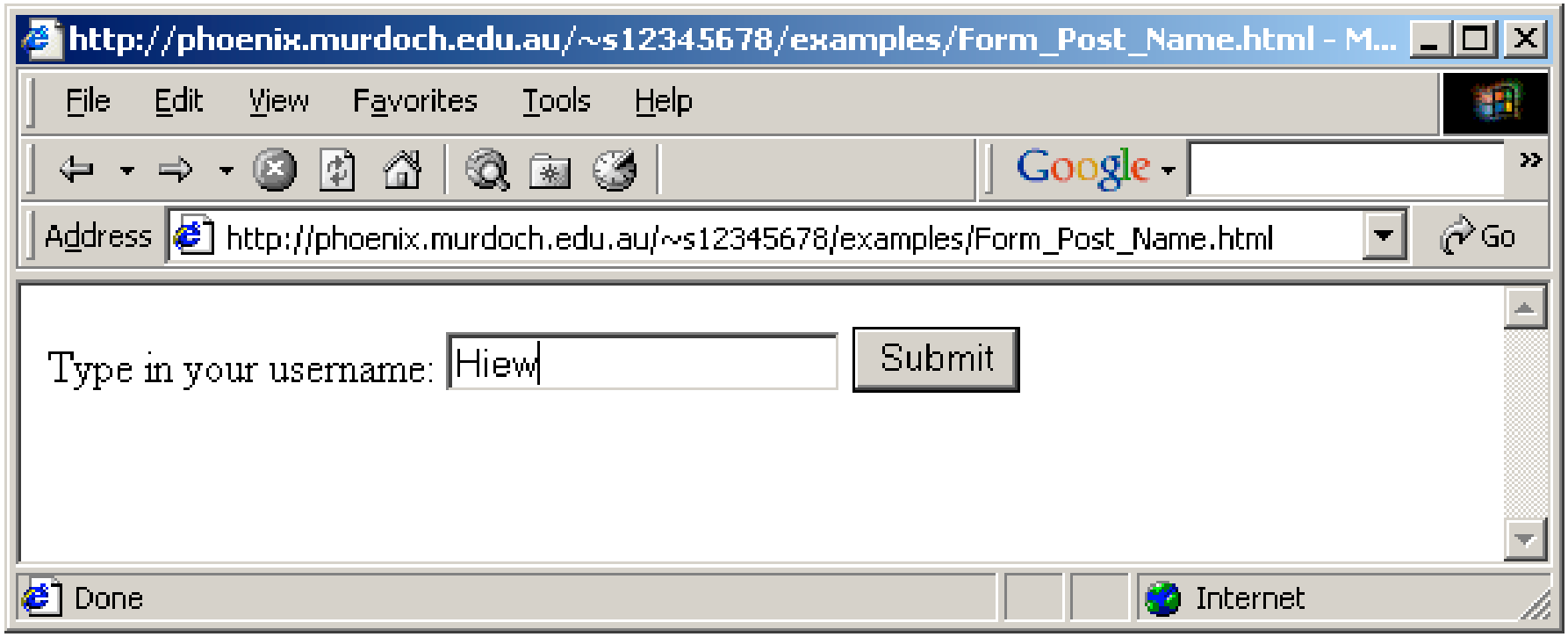
```
<html>
 <body>
  <form method="POST" action="handle_name.php">
   Type in your username: <input type="text" name="Username">
   <input type="submit" value="Submit">
  </form>
 </body>
</html>
```

Example

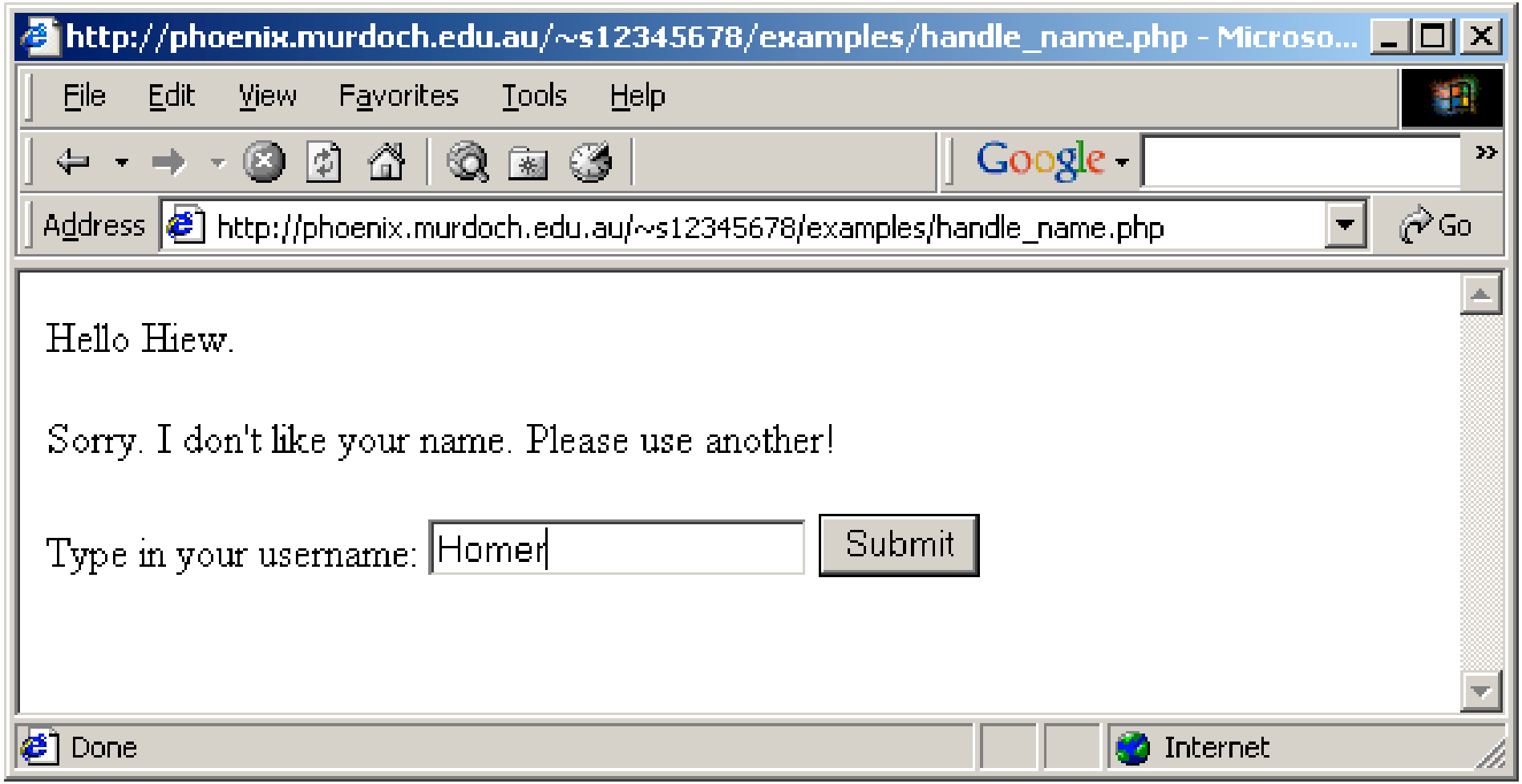
handle_name.php

```
<html>
 <body>
  <?
    echo "<p>Hello $Username.</p>" ;
    if ($Username == "Homer") {
     echo "<p>How are you?</p>" ;
    } else {
     echo "<p>Sorry. I don't like your name. Please use another!</p>" ;
     echo "<form method=POST action=#>" ;
     echo "   Type in your username: <input type=text name=Username>" ;
     echo "   <input type=submit value=Submit>" ;
     echo "</form>" ;
    }
  ?>
 </body>
</html>
```

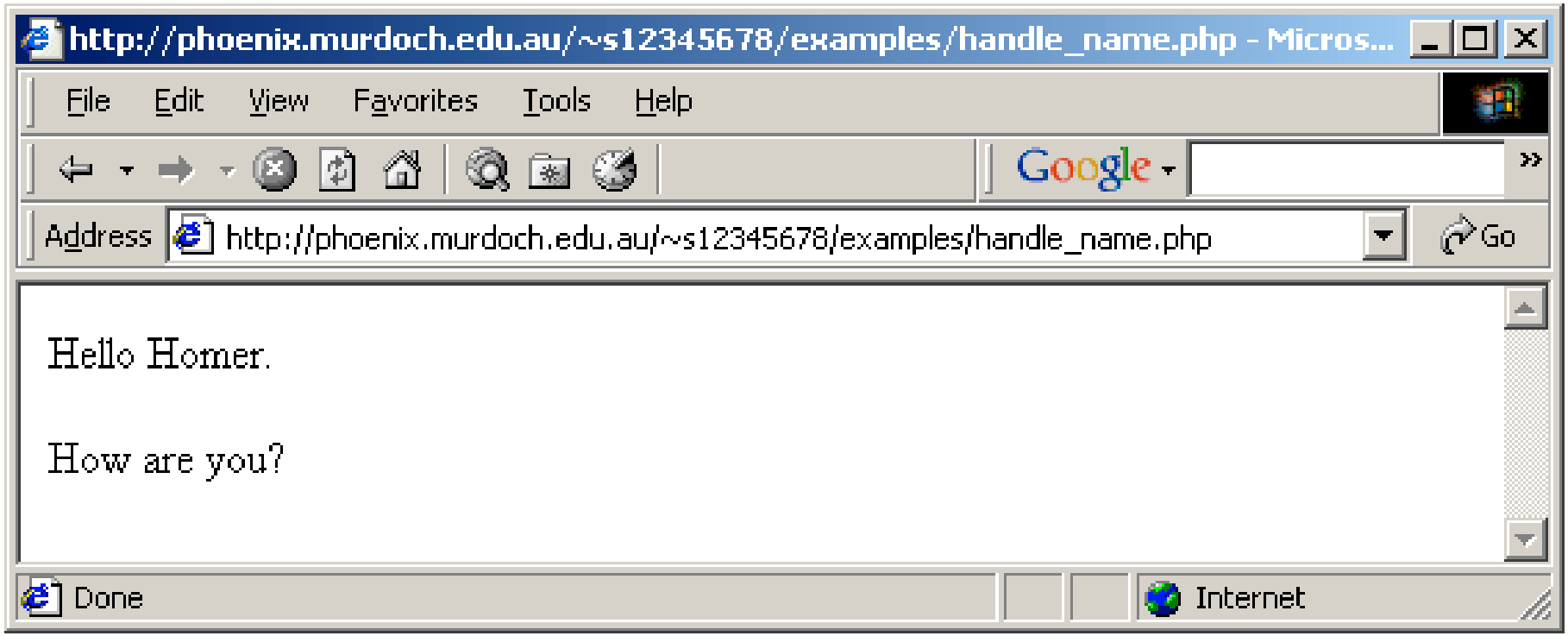
The first form...



After submitting, we get another form...



Submitting again with different data...



Why Server-side Scripting?

- HTML is static.

- JavaScript dynamic but only on the client-side
  - It can't get access to the resources on a server (eg. a database)
  - There are certain information which we cannot transfer down to the client to process (eg. passwords).

- Server-side technologies such as PHP allows us to give users access to services, but have the services processed by the server.

In labs week 7-10...

- In your lab exercises on week 7-10, you will be given instructions on how to get the above scripts to work in your own directory on red.

- What it essentially involves is:
  - Create script files above using a text editor (eg. Notepad) on your local machine.
  - FTP files into the `public_html` directory in your account on phoenix.
  - Access the scripts using a browser and the appropriate URLs.