Web Programming
COSC2413/2426

Week 10

Course Topics

- Week 1: HTML, XHTML
- Week 2: More on XHTML, and CSS
- Week 3: More on CSS
- Week 4: JavaScript
- Week 5: JavaScript
- Week 6: **PHP I - Basics**
  SEMESTER BREAK ***
- Week 7: PHP II – Files handling, Arrays
- Week 8: PHP III – Sessions and Cookies
- **Week 9: PHP IV – More on Sessions, Functions**
- Week 10: PHP V – Error handling, Testing, Security
- Week 11: A Sample PHP Application
- Week 12: Revision/Sample Exam (HTML 5 if time!)

RMIT UNIVERSITY

Course Experience Survey

HE

[redacted]

Course Code: 1350-2536-COSC1519
Class Number: 2536
Course Name: Introduction to Programming
School: Computer Science & Information Technology
Staff: Isaac Balbin (E23457)
Coordinator: Isaac Balbin (E23457)

Age:		Hours studying for course:			
20 or less:	33	0-2:	18	Full time:	52
21-24:	14	3-4:	28	Part time:	0
25-34:	6	5-6:	6	Local:	42
35-44:	0	7-10:	1	International:	11
>44:	0	>10:	0		

No of Students Who Submitted Surveys for this Course/Class Combination: 53
No of Students Who Answered GTS Questions for this Staff Member: 53

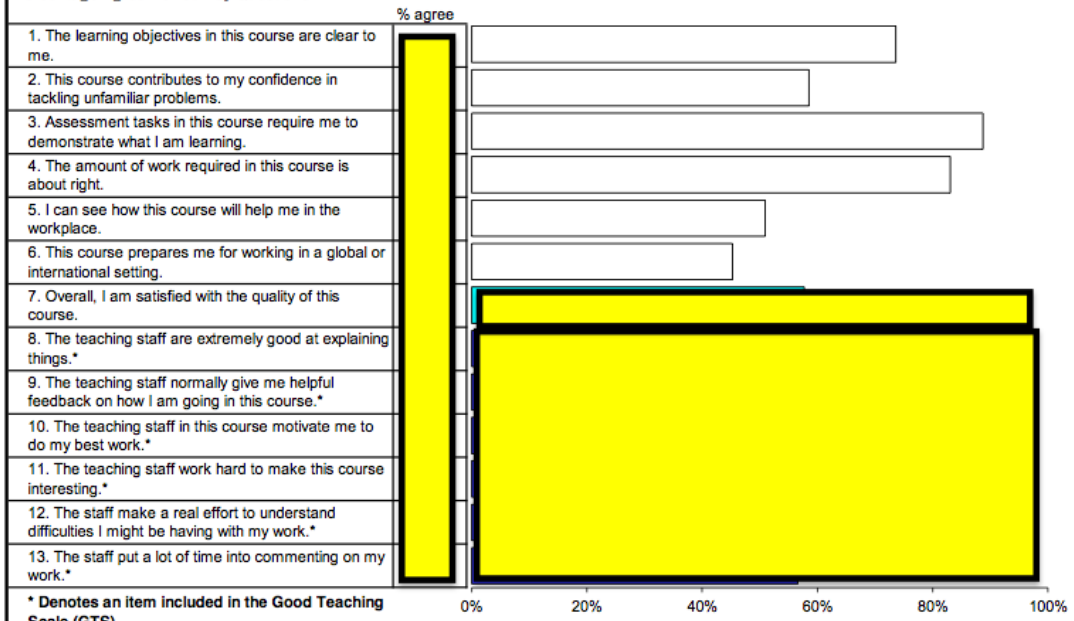
Survey Population *: 197

Demographic data shown come from submitted survey forms.

SUMMARY

[redacted]

Percentage Agree[b] for Survey Questions



*** Denotes an item included in the Good Teaching Scale (GTS)**

[redacted]

ongly agree" as a percentage of the number

RELIABILITY ASSESSMENT

With 53 responses from a survey population of 197
the data presented in this report are considered to be
Sufficient for use, including for academic expectations.
Number of responses needed to be considered sufficient: 18
Number of responses needed to be considered good: 56

Revision

- Why were sessions useful?
  - So we can store what we have done. HTTP is stateless, when the client talks to the server, the server doesn't remember between requests, or whether it is indeed the same user!
- Interaction between pages mean we have to pass things between pages/code on the pages.
- There are three (main) ways.
  - Sessions (PHP speciality),
  - Cookies (small file stored on the client side. Has a Key/Value pair). Web server can ask to store that.
  - Hidden input, where you have a value in the global array.

Hidden variables revision

- ```
  <?php $number_of_visits++;?>
  <input type="hidden" name="number_of_visits"
     value="<?php echo number_of_visits;?>"/>
  ```
- problem is that you have to use forms everywhere. Even to go to a new page, instead of a simple hyperlink click, you could use a form to move to the next page and thereby keep the hidden form data which is in the global variable $_POST array
  ```

Cookies revision

- Cookies are managed by the web server.
- When you return to a particular web server, some cookie data is sent as part of your http request.
- Server can be asked to look at the data in your cookie as stored by your browser
  - Name (of data=variable)
  - Expires (how long till it is not used)
  - Domain (the identity of the server that made the cookie)
  - Path --- which URLs in this domain/site use the cookie
  - Secure --- set this to encrypt the data via https rather than http protocol

- `setcookie:NAME=VALUE;[expires= date;] [path=path;] [domain=domain_name;] [secure]`
- In PHP

  `setcookie ( $name, $value, $expire, $path, $domain, $secure)`
  - only $name is compulsory
  - always send the cookie header before other http request headers
- **`$_COOKIE['mycookie']`**

Example from last week

```
<?php
session_start(); // always first line, creates or restores existing session.
                 // It is implemented with a cookie, and the cookie header
needs to always come first


if (empty($_SESSION['count'])) {
   $_SESSION['count'] = 1;
} else {
   $_SESSION['count']++;
}
?>
<p>
Hello visitor, you have seen this page <?php echo $_SESSION['count']; ?> times.
</p>
<p>
Your PHP SID is <?php echo session_id(); ?>.
</p>
<p>
To continue, <a href="page2.php?<?php echo htmlspecialchars(SID); ?>">click
here</a>.
</p>
```

How do ‘Sessions’ work?

- They are based on assigning each user a unique number, or **session id**. Even for extremely heavy use sites, this number can for all practical purposes can be regarded as **unique.**

e.g.

26fe536a534d3c7cde4297abb45e275a

How do ‘Sessions’ work?

- This **session id** is stored in a cookie, **or** passed in the URL between pages while the user browses.
- The **data** to be stored (e.g. name, log-in state, etc.) is stored **securely server-side in a PHP superglobal**, and referenced using the session id.

Starting or Resuming a Session

```
session_start();
```

PHP does all the work:

- It looks for a valid session id in the **`$_COOKIE`** or **`$_GET`** superglobals – if found it initializes the data.
- If none is found, a new session id is created.
- Note that like **`setcookie()`**, this function must be called before any echoed output to browser.

Storing Session Data

- The **$_SESSION** superglobal array can be used to store any session data.

  e.g.

```
$_SESSION['name'] = $name;
$_SESSION['age'] = $age;
```

Reading Session Data

- Data is simply read back from the **$_SESSION** superglobal array.

  e.g.

```
$name = $_SESSION['name'];
$age = $_SESSION['age'];
```

Session Propagation

- Sessions need to pass the session id between pages as a user browses to track the session.
- It can do this in two ways:
  - Cookie propagation
  - URL propagation

Cookie Propagation

- A cookie is stored on the user's computer containing the session id.
- It is read in whenever `session_start();` is called to initialise the session.
- Default behaviour is a cookie that expires when the browser is closed. Cookie properties can be modified with `session_set_cookie_params` if required.

URL Propagation

- The session id is propagated in the URL

```
(...some_folder/index.php?
            sid=26fe536a534d3c7cde4297abb45e275a)
```

- PHP provides a global constant to append the session id to any internal links, **`SID`**.

  e.g.

```
echo "<a href=\"nextpage.php?".SID."\\">Next
                        page</a>";
```

Which one..?

- The default setup of a PHP server is to use **both** methods.
  - It checks whether the user has cookies enabled.
  - If cookies are on, PHP uses cookie propagation. If cookies are off it uses URL propagation.

And this means..?

- That as developers, we must be aware that sessions can be propagated through URL, and append the constant `SID` to any internal links.
- If sessions **are** being propagated by cookies (your cookies are not turned off), the constant `SID` is an empty string, so the session id is not passed twice (it is retrieved from the cookie).

- To delete a single session value, you use the unset() function:

```php
<?php
session_start();
// delete the username value
unset($_SESSION["username"]);
```

- But this clears only data in the session not the session. If a user, say logs out, you want to **kill** the session.

```php
<?php
session_start();
// delete all session values
session_unset();
```

Destroying a Session

Sometimes not required, but if we want to destroy a session:

```
// clear all session variables
$_SESSION = array(); // array creation of null array

// delete the session cookie if there is one
if (isset($_COOKIE[session_name()])) {
   setcookie(session_name(),'',time()-42000,'/');
}

// destroy all data associated with session
session_destroy();
```

Session Expiry

- By default, PHP sessions expire:
  - After a certain length of inactivity (default 1440s, or 24 minutes), the PHP garbage collection processes deletes session variables. Important as most sessions will not be explicitly destroyed.
  - If propagated by cookies, default is to set a cookie that is destroyed when the browser is closed.
  - If URL propagated, session id is lost as soon as navigate *away* from the site.

Long-term Sessions

- Although it is possible to customise sessions so that they are maintained after the browser is closed, for most practical purposes PHP sessions can be regarded as short-term.
- Long-term session data (e.g. ‘remember me’ boxes) is usually maintained by explicitly setting and retrieving cookie data.

Session Hijacking

- A security issue: if a malicious user manages to get hold of an active session id that is not their own..

  e.g.
  - user 1 browsing site with cookies disabled (URL propagation).
  - user 1 logs in.
  - user 1 sends an interesting link to user 2 by email.. The URL copy and pasted contains his session id.
  - user 2 looks at the link before the session id is destroyed, and ‘hijacks’ user 1’s session.
  - user 2 is now logged in as user 1!!

... rule of thumb ...

If you are truly security conscious you should assume that a session propagated by URL may be compromised.

Propagation using cookies is more secure (but still not foolproof).

Login-Logout Example Using Sessions

- We will see a really practical use of Session Variables. We will set up a Login and Logout functionality using session variables.
- The following files have been used
  - login.php - Contains the HTML form to allow user login and calls the login function.
  - logout.php - Contains the HTML form to allow user logout and calls the logout function.
  - register.php - Contains the HTML form to allow a visitor to register and call the registration function.
  - common.php - Contains the main PHP function to separate the code and design.
  - index.php – is the secret page visible only to the logged users
- It has been hosted for your convenience at:

http://yallara.cs.rmit.edu.au/~e46991/login-system/login.php

Functions

Functions

- A function is a piece of code that achieves some well defined task. It may take parameters and return a value.
- PHP has many built-in functions and you can also define your own.

Using Functions

Functions are called or invoked:

```
$fp = fopen($name, $mode);
```

- fopen() is the function name.
- fopen() takes two parameters, the file name, and an opening mode.
- Functions can return a value. In this case fopen() returns a file handle, $fp.

For Contrast

```
phpinfo();
```

- This function is called **`phpinfo`**
- It reports information about the configuration of php on the system
- It does not take any parameters, and does not return a value.

Prototypes

- Manuals describe functions via prototypes
- Here is the prototype for fopen()

```
int fopen( string filename, string
mode, [int use_include_path] );
```

- The *type* at the front is the type of the data **returned** – file handles are actually integers.
- You can see that the fopen() function actually takes three parameters, but the third is optional and therefore is shown in **square** brackets.

A Side Note

- Function names are **not** case sensitive
- FOPEN() is the **same** as fopen()

- This is **different** from variables which *are* case sensitive
  - Yes it's silly in my view
- $fp is a different variable from $FP

Declaring Your Own Functions

- This is a very useful thing to do.
- There are many pieces of functionality you will use over and over again. You can write a function once, and then just call it when it's needed.
- Example:

```
function my_function()
{
    echo "My function was called";
}
```

Notes

- When you define a function you must begin with the word **`function`**
- This is followed by the **function name**, and a pair of brackets **`()`** enclosing any parameters.
- The **function body** is enclosed in curly braces **`{}`**
- **`my_function()`** function does not have any parameters, and does not return a value.

Calling Your Function

- After you have defined your function, you can call it with the name you gave it:

```
my_function();
```

as you would call a PHP built-in function.

Where Can I Use My Functions?

- The built in functions are available to all scripts, but your own functions are only available to the script(s) that they were declared in.
- It is common practice to have one file containing a set of related or commonly used functions.
- You could then have an `include('filename.php')` statement in all of your scripts to make your functions available.
- `require('filename.php')` is the same except that the script will STOP if there was an error. include will just give a warning. For security, require is better

Naming Your Functions

- Functions should have names that describe what they do, like **`sort()`** or **`fopen()`** or **`date()`**.
- Functions cannot have the same names as existing functions.
- Function names can consist of letters and underscores.
- They cannot begin with a digit.

Example 2

```
function nicer_fopen($name, $mode)
{
  @ $fp = fopen($name, $mode);
  if (!$fp)
  {
    echo "<p><strong> Oh No! I could not "
         ."open the file.</strong></p>";
    return false;
  }
  else
    return $fp;
}
```

revision

```php
<?php
for ($x=0; $x<=10; $x++)
  {
  echo "The number is: $x <br>";
  }
?>
```

```php
<?php
$x = array("a" => "red", "b" => "green");
?>
```

```php
<?php
$colors = array("red","green","blue","yellow");
foreach ($colors as $value)
  {
  echo "$value <br>";
  }
?>
```

Example 3

What does this function do?

```
function create_table($data) {
    echo "<table border = 1>";
    foreach ($data as $i => $value) {
        echo "<tr><td>$value</td></tr>";
    }
    echo "</table>";
}
```

Scope

- A variable's **scope** controls where that variable is **visible** and **useable**.
- Different programming languages have different rules.
- PHP's scope rules are very straightforward.

Scope Rules

1. Variables declared *inside* a function are in scope from the statement **where they are declared** to the **end of the function**. This is called **function scope**.

   These variables are called **local variables**.

2. Variables declared *outside* of functions are in scope from the statement **where they are declared** to the end of the file, but **not inside functions**! This is called global scope. These variables are called **global variables**.

The `global` Keyword

- This can be used to declare a variable **inside a function** as **global,** so that it will still exist when the function is finished
- It can also be used to **access** an **existing global variable** from inside a function.

```
function fn()
{
    global $var;
    // and so on...
}
```

What will this code print?

```
<?php
  function increment($fn_value)
  {
    $fn_value = $fn_value + 1;
  }

  $value = 10;
  increment($value);
  echo $value;
?>
```

Pass By Value

- When we pass a parameter, **a copy of the value is made**.

- The function **uses the copy and does not - cannot - change the original**.

What Will This Code Print?

```
function increment($value)
{
   $value = $value + 1;
}

$value = 10;
increment($value);
echo $value;
```

Global Versus Local

- The variable called **`$value`** inside the function is a different variable from the one called **`$value`** in the *main* program.

- One is **local**, the other is **global**.

How To Change The Value

- Technique 1: make the variable global.

```
function increment()
{
    global $value;
    $value = $value + 1;
}
```

- This will work, but what if we want to use the function on more than one variable?
- e.g `increment($something_else)` ?

How To Change The Value

- Technique 2: add a return statement to the function.

```
function increment($value)
{
        $value = $value + 1;
        return $value
}

$value = 10;
$value = increment($value);
```

Pass By Reference

- Technique 3:

```
function increment( &$fn_value )
{
  $fn_value = $fn_value + 1;
}
```

- Notice the & in the function argument list.
- This is called a **reference.**
- It means "**Use the original variable** passed in from the main program, **don't make a copy**."

Pass By Value (PBV)

- Pass by value **passes a copy and cannot change it.**
- Pass by reference **passes the original.**
- Only use pass by reference if you need to change the value.

Default Argument Values

- Sometimes it is convenient to supply a default value for a function argument.

```
<?php
function print_doc($papersize = “a4”)
{
    echo “The paper size is set
        to $papersize”;
}

print_doc()
?>
```