Processing Multiple Selections with PHP

- The following HTML fragment creates a Select input in which multiple car types may be selected by the user:

```
<select name="carBrands" size=5 multiple>
<option value="Ford" SELECTED>Ford Motor Company
<option value="GM">General Motors
<option value="Honda">Honda Motor Company
<ottion value="Toyota">Toyota Motor Company
<option value="Ford">Jaguar
<option value="Mazda">Mazda
<option value="Volvo">Volvo
</select>
```

- Until now we have only concerned ourselves with data which only holds one value
- In this case there is the potential for multiple data values to be associated with a single form element.
- The way PHP handles this is to place the multiple selections in an array

Processing Multiple Selections with PHP

- Before that happens, however, we need to make a small modification to our declaration of the HTML Select element.

```
<select name="carBrands[]" size=5 multiple>
<option value="Ford" SELECTED>Ford Motor Company
<option value="GM">General Motors
<option value="Honda">Honda Motor Company
<ottion value="Toyota">Toyota Motor Company
<option value="Ford">Jaguar
<option value="Mazda">Mazda
<option value="Volvo">Volvo
</select>
```

- All we need to do to turn this data into an array is to place [] after the element name:
- Now, in our server side PHP script we can extract the selected items from this array:

```
print_r($_POST['carBrands']);
```

Create a PHP Array

- Arrays are created using the array() function.
- The array() function takes zero or more arguments
  - returns the new array which is assigned to a variable using the assignment operator (=)
- If arguments are provided they are used to initialize the array with data.
- PHP arrays grow and shrink dynamically as items are added and removed
  - it is not necessary to specify the array size at creation time

```
<?php
      $colorArray = array();
?>
```

Create a PHP Array

- Alternatively, we can create an array pre-initialized with values by providing the values as arguments to the array() function:

```
<?php
      $colorArray = array("Red", "Yellow", "Green", "Blue", "Indigo");
?>
```

Accessing Elements in a PHP Array

- The elements in a PHP numerical key type array are accessed by referencing the variable containing the array
  - followed by the index into array of the required element enclosed in square brackets ([])

```
<?php
   $colorArray = array("Red", "Yellow", "Green", "Blue", "Indigo");

   echo $colorArray[1];
?>
```

  - The above echo command will output the value in index position 1 of the array
    - in this case the string "Yellow".

Creating an Associative Array

- An **associative** array **assigns names** to **positions** in the array.
- This provides a **more human friendly** way to access array elements
- Once again, the **array()** function is used to create the array
  - with optional arguments to pre-initialize the array elements.
- The arguments are of the form **key=>value**
  - where key is the name by which the element will be referenced
  - value is the value to be stored in that position in the array

```
<?php
$customerArray = array('customerName'=>'John Smith',
'customerAddress'=>'1 The Street', 'accountNumber'=>'123456789');
?>
```

Accessing Elements of an Associative Array

- We can use those names to access the corresponding array values.
- We can, therefore, extend our previous example to extract the customer name from our $customerArray:

```
<?php
$customerArray = array('customerName'=>'John Smith',
'customerAddress'=>'1 The Street', 'accountNumber'=>'123456789');

echo $customerArray['customerName'];
?>
```

Creating Multidimensional PHP Arrays

- A multidimensional PHP array is nothing more than an array
  - in which each array element is itself an array.
- A multidimensional array can, therefore, be thought of as a table
  - where each element in the parent array represents a row of the table
  - the elements of each child array represent the columns of the row.

```
<?php
   $books = array();
   $books[0] = array('title'=>'JavaScript in 24 Hours',
   'author'=>'Moncur');
   $books[1] = array('title'=>'JavaScript Unleashed',
   'author'=>'Wyke');
   $books[2] = array('title'=>'Network+ Second Edition',
   'author'=>'Harwood');
?>
```

Accessing Elements in a Multidimensional PHP Array

- We need to first specify the array row that we wish to access.
  - we need to specify the column in that row
- To access an element, therefore we specify the array name
  - and then follow it with the desired row and column of the array
  - each enclosed in square brackets ([]).

```
<?php
   $books = array();
   $books[0] = array('title'=>'JavaScript in 24 Hours',
   'author'=>'Moncur');
   $books[1] = array('title'=>'JavaScript Unleashed',
   'author'=>'Wyke');
   $books[2] = array('title'=>'Network+ Second Edition',
   'author'=>'Harwood');

   echo $books[1]['author'];
?>
```

Using PHP Array Pointers

- PHP arrays maintain an internal pointer that records the current element.
- This pointer can be changed using the next, previous, reset and end functions.
  - The reset and end functions move the pointer to the first and last elements of the array respectively
  - The next function moves the pointer on to the next array element.
  - The prev moves the pointer to the previous array element.
  - The next and prev functions return false when it is not possible to move any further in the corresponding direction.
- Each function takes the name of the array in which the pointer adjustment is to take place as an argument

Using PHP Array Pointers

```
<?php
      $colorArray = array("Red", "Yellow", "Green", "Blue", "Indigo");

      echo 'The last element is ' . end($colorArray) . '<br>';
      echo 'The previous element is ' .  prev($colorArray) . '<br>';
      echo 'The first element is ' . reset($colorArray) . '<br>';
      echo 'The next element is ' . next($colorArray) . '<br>';
?>
```

Changing, Adding and Removing PHP Array Elements

- n array element can be changed by assigning a new value to it
  - using the appropriate index key

```
$colorArray = array("Red", "Yellow", "Green", "Blue", "Indigo");

$colorArray[0] = "Orange";
```

- A new item can be added to the end of an array using the array_push() function
- This function takes two arguments
  - the first being the name of the array
  - the second the value to be added

```
$colorArray = array("Red", "Yellow", "Green", "Blue", "Indigo");

array_push($colorArray, "White");
```

Changing, Adding and Removing PHP Array Elements

- A new element can be inserted at the start of the array
  - using the array_shift() function which takes the array name and the value to be added as arguments

```
$colorArray = array("Red", "Yellow", "Green", "Blue", "Indigo");

array_shift($colorArray, "White"); // Add White to start of array
```

- The last item added to an array can be removed from the array
  - using the array_pop() function.

```
$colorArray = array("Red", "Yellow", "Green", "Blue", "Indigo");

array_push($colorArray, "White"); // Add White to end of array

array_pop($colorArray); // Remove White from the end of the array
```

Changing, Adding and Removing PHP Array Elements

- The first item in the array can be removed using the array_unshift() function:

```
$colorArray = array("Red", "Yellow", "Green", "Blue", "Indigo");

array_shift($colorArray, "White"); // Add White to start of array

array_unshift($colorArray) // Remove White from the start of the array
```

Looping through PHP Array Elements

- It is often necessary to loop through each element of an array to either read or change the values contained therein.
  - One such mechanism is to use the foreach loop.
- The foreach loop works much like a **for** or **while loop**
  - allows you to iterate through each array element.
- There are two ways to use foreach.

Looping through PHP Array Elements

- The first assigns the value of the current element to a specified variable
  - which can then be accessed in the body of the loop.
- The syntax for this is: foreach ($arrayName as $variable)

```
$colorArray = array("Red", "Yellow", "Green", "Blue", "Indigo");

foreach ($colorArray as $color)
{
    echo "$color <br>";
}
```

Looping through PHP Array Elements

- For associative arrays the foreach keyword allows you to iterate through both the keys and the values
- Using the followint: foreach ($arrayName as $keyVariable=>$valueVariable)

```
$customerArray = array('customerName'=>'John Smith',
'customerAddress'=>'1 The Street', 'accountNumber'=>'123456789');

foreach ($customerArray as $key=>$value) {
   echo "Key = $key <br>";
   echo "Value = $value <br>";
}
```

Replacing Sections of an Array

- Entire blocks of array element can be modified using the array_splice() function.
- The array_splice() function takes two mandatory and two optional arguments.
  - The first argument is the name of the array on which the function is to work.
  - The second argument specifies the index into the array where the splice is to take effect.
  - The optional third argument specifies the end point of the splice
  - The final argument is an optional array containing elements to be used to replace the removed items.

```
$myArray = array ('One', 'Two', 'Three', 'Four', 'Five');

$myReplacements  = array ('Six', 'Seven', 'Eight');

$extract = array_splice ($myArray, 2, 4, $myReplacements);
```

Sorting a PHP Array

- An array can be sorted using the sort function.
- A number of different sorts are possible using the sort function.
  - The first argument is the name of the array.
  - The second indicates the sort algorithm to use.
  - The available algorithms are SORT_NUMERIC, SORT_STRING and SORT_REGULAR.
  - If no sort type is specified, SORT_REGULAR is used.
- Similarly array items can be sorted in descending order using the rsort function.

```
$colorArray = array("Red", "Yellow", "Green", "Blue", "Indigo");
sort($colorArray, SORT_STRING);
```

Sorting Associative Arrays

- Associative arrays can be sorted in two ways
  - by key
    - **ksort**
    - **krsort** reverse function
  - by value
    - **asort**
    - **arsort** reverse function
- The syntax and options for these functions are as outlined for the sort and rsort functions above.

Opening and Creating Files in PHP

- **Existing** files are opened, and **new** files created using the PHP **fopen()** function.
- The **fopen()** function accepts two arguments and returns a **file handle**
  - which is subsequently used for all future read and write interactions with that file.
  - The **first** argument is the name
    - This path is relative to the server filesystem root, not your web server root.
  - The **second** argument is an attribute indicating the mode in which to open the file

```
<?php
$fileHandle = fopen('/tmp/php_essentials.txt', 'w+')
?>
```

Closing Files in PHP

- Once a file has been opened it can be closed using the fclose() function.
- The fclose() function takes a single argument
  - the file handle returned by the fopen function when the file was first opened.

```
<?php
$fileHandle = fopen('/tmp/php_essentials.txt', 'w+')
 OR die ("Can't open file\n");

fclose ($fileHandle);
?>
```

Writing to a File using PHP

- We can do this using the PHP fwrite() and fputs() functions.
  - These are essentially the same function so either can be used.
- fwrite() takes two arguments
  - the file handle returned from the original fopen()
  - and the string to be written

```
<?
$fileHandle = fopen('/tmp/php_essentials.txt', 'w+') OR die ("Can't open");

$result = fwrite ($fileHandle, "This line of text was written by PHP\n");

if ($result) {
     echo "Data written successfully.<br>";
} else {
     echo "Data write failed.<br>";
}
fclose($fileHandle);
?>
```

Reading From a File using PHP

- Data can be read from a file using the PHP fread() function.
  - fread() accepts two arguments
    - the file handle
    - the number of bytes to be read from the file

```
<?
$fileHandle = fopen('/tmp/php_essentials.txt', 'w+') OR die ("Can't open");

fwrite ($fileHandle, "This line of text was written by PHP\n");
fclose($fileHandle);

$fileHandle = fopen('/tmp/php_essentials.txt', 'r') OR die ("Can't open");
$fileData = fread ($fileHandle, 1024);

echo "data = $fileData";
fclose($fileHandle);

?>
```

Checking Whether a File Exists

- The **file_exists** function can be used at any time to **find** if a file **already exists** in the filesystem.
  - The function takes a single argument
    - the path to the file in question
  - Returns a **boolean true** or **false**
    - depending on the **existence** of the file.

Moving, Copying and Deleting Files with PHP

- Files can be copied using the **copy**() function
- Renamed using the **rename**() function
- Deleted using the **unlink**() function

```
<?
   if (file_exists('/tmp/php_essentials.txt) {
   copy ('/tmp/php_essentials.txt, '/tmp/php_essentials.bak');
   rename ('/tmp/php_essentials.bak', '/tmp/php_essentials.old');
   unlink ('/tmp/php_essentials.old'); // Delete the file
}
?>
```

Accessing File Attributes

- The PHP stat() and fstat() functions provide a wealth of information about a file.
- The information is so copious that the results are returned as an associative array.
- The functions take a single argument.
  - stat() takes a string defining the path to the file
  - fstat() takes a file handle returned from an fopen() function call.

Key	Description
dev	Device number
ino	Inode number
mode	Inode protection mode
nlink	Number of links
uid	User ID of owner
gid	Group ID of owner
rdev	Inode device type
size	Size in bytes
atime	Last access (Unix timestamp)
mtime	Last modified (Unix timestamp)
ctime	Last inode change (Unix timestamp)
blksize	Blocksize of filesystem IO (platform dependent)
blocks	Number of blocks allocated

Accessing File Attributes

- With reference to the above table we can now extract some information about a file on the file system of our server:

```
<?php

$results = stat ("/tmp/php_essentials.txt");

echo "File size is $results[size]<br>";
echo "File last modifed on $results[mtime]<br>";
echo "File occupies $results[blocks] filesystem blocks<br>";
?>
```

Creating Directories in PHP

- A new directory can be created in PHP using the mkdir() function.
  - The second, optional argument allows the specification of permissions on the directory

```
<?php
$result = mkdir ("/path/to/directory", "0777");
?>
```

Deleting a Directory

- Directories are deleted in PHP using the rmdir() function.
- rmdir() takes a single argument, the name of the directory to be deleted.
- The deletion will only be successful if the directory is empty.
- If the directory contains files or other sub-directories the deletion cannot be performed
  - until those files and sub-directories are also deleted.

Finding and Changing the Current Working Directory

- The current working directory can be identified using the getCwd() function:

```
<?php
$current_dir = getCwd();

echo "Current directory is $current_dir";
?>
```

- The current working directory can be changed using the chdir() function.

```
<?php

$current_dir = getCwd();

echo "Current directory is $current_dir <br>";

chdir ("/tmp");

$current_dir = getCwd();

echo "Current directory is now $current_dir <br>";
?>
```

Listing Files in a Directory

- The files in a directory can be read using the PHP scandir() function.
  - The first argument is the path the directory to be scanned.
  - The second optional argument specifies how the directory listing is to be sorted.
    - If the argument is 1 the listing is sorted reverse-alphabetically.
    - If the argument is omitted or set to 0 the list is sorted alphabetically.

```
<?php
chdir ("/tmp");

$current_dir = getCwd();

echo "Current directory is now $current_dir";

$array = scandir(".", 1);

print_r($array);
?>
```

What is a PHP Session?

- PHP **Sessions** allow web pages to be **treated** as a **group**
  - allowing **variables** to be **shared** between different pages.
- One of the **weaknesses** of **cookies** is that the cookie is **stored** on the user's computer
  - This provides the user the ability to access, view and modify that cookie for potentially nefarious purposes.
- PHP sessions, on the other hand, store only an ID cookie on the user's system which is used to reference the session file on the server.
  - the user has **no access** to the content of the session file
  - thereby **providing** a **secure** alternative to cookies.
- PHP sessions also work when the user has disabled the browser's cookie support.

Creating a PHP Session

- PHP sessions are created using the **session_start()**
  - should the first function call of the PHP script on your web page
- The following example demonstrates the creation of a PHP session:

```
<?php
       session_start();
?>
<html>
<head>
<title>A PHP Session Example</title>
</head>
<body>
</body>
</html>
```

Creating and Reading PHP Session Variables

- Variables can be assigned to a session using the $_SESSION array
  - This is a global array that is accessible to all the pages on your web site.
  - This is also an associative array
  - it is possible to access array elements using the variable name as an index.
- Session variables can be any type of data
  - strings, numbers, arrays and objects.
- Session variables can be defined using a number of mechanisms.
  - Variables can be assigned directly to the $_SESSION array
    - using the assignment operator and variable name:

```
<?php
     $_SESSION['userName'] = 'Negroponte';
?>
```

Creating and Reading PHP Session Variables

- Another option is to use the PHP session_register() function.
  - session_register() takes two arguments
    - the string representing the variable name
    - the value to be assigned to the variable

```
<?php
session_register('username', 'Negroponte');
?>
```

Creating and Reading PHP Session Variables

- Session variables are accessed by using the variable name as an index key into the $_SESSION array.
- The session_is_registered() function can also be used to make sure the variable exists before attempting to read the value

```
<? session_start(); ?>
<html><head><title>Simple HTML Form</title></head>
<body>
<?php
   if (session_is_registered('userName') {
      $_SESSION['userName'] = 'Neil';
      echo 'userName = ' . $_SESSION['userName'];
   }
?>
</body></html>
```

Writing PHP Session Data to a File

- Session data only stays active on the web server until it expires or the session is deleted.
  - Once deleted, all the data associated with the session is lost.
- A snapshot of the session data can, however, be taken at any time and written out to a file.
  - Once saved, it can be reloaded when required.

Writing PHP Session Data to a File

- To save a session state the session_encode() function is used combined the PHP file I/O functions
- The session_encode() function returns an encoded string containing the session data.
  - Once this string has been obtained it can be written to a file:

```
<?
   $_SESSION['userName'] = 'Negroponte';
   $_SESSION['emailAddress'] = 'negroponte@gmail.com';
   $session_data = session_encode();

   // open a file write session data
   $filehandle = fopen ('/tmp/php_session.txt', 'w+');

   // write the session data to file
   fwrite ($filehandle, $session_data);

   fclose ($filehandle);
?>
```

Reading a Saved PHP Session

- Once session data has been written to a file it can be read back in
  - decoded and applied to the current session
- This is achieved using the session_decode() function:

```
<?php session_start(); ?>
<html><head></head><body>

<?php
      $filehandle = fopen ('/tmp/php_session.txt', 'r');
      $sessiondata = fread ($filehandle, 4096);
      fclose ($filehandle);
      session_decode($sessiondata); // Decode the session data
      print_r($sessiondata); // Display the session data
?>

</body></head>
```