PHP Arrays and Superglobals

Chapter 9

Randy Connolly and Ricardo Hoar

Objectives

1. Arrays
2. $_GET and $_POST Superglobal arrays
3. $_SERVER Array
4. $_FILES Array
5. Reading/Writing Files

Section 1 of 5

ARRAYS

Arrays

Background

An array is a data structure that

- Collects a number of related elements together
- Allows the set to be Iterated
- Allows access of any element

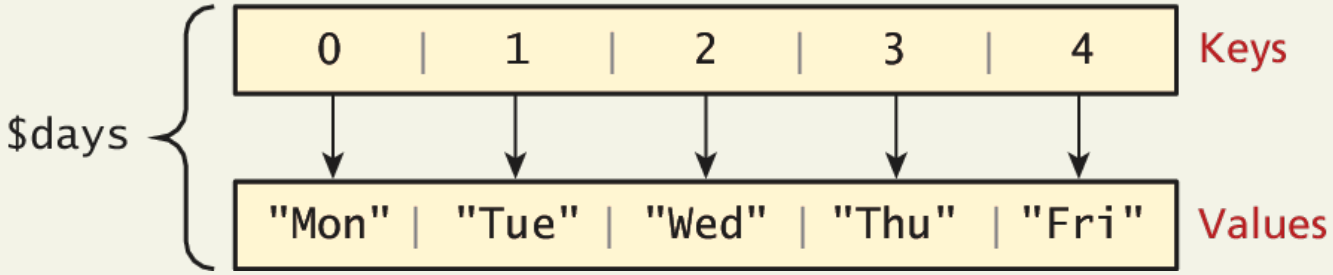
Since PHP implements an array as a dynamic structure:

- Add to the array
- Remove from the array

Arrays

Key Value

In PHP an array is actually an **ordered map**, which associates each value in the array with a key.



Arrays

Keys

Array keys are the means by which you refer to a single element in the array.

In most programming languages array keys are limited to integers, start at 0, and go up by 1.

In PHP, array keys *must* be either *integers* or *strings* and need not be sequential.

- Don't mix key types i.e. "1" vs 1
- If you don't explicitly define them they are 0, 1, …

Arrays

Values

Array values, unlike keys, are not restricted to integers and strings.

They can be any object, type, or primitive supported in PHP.

You can even have objects of your own types, so long as the keys in the array are integers and strings.

Arrays

Defining an array

The following declares an empty array named days:

$days = array();

You can also initialize it with a comma-delimited list of values inside the ( ) braces using either of two following syntaxes:

```
$days = array("Mon","Tue","Wed","Thu","Fri");

$days = ["Mon","Tue","Wed","Thu","Fri"]; // alternate
```

Arrays

Defining an array

You can also declare each subsequent element in the array individually:

```
$days = array();

$days[0] = "Mon"; //set 0th key's value to "Mon"

$days[1] = "Tue";

// also alternate approach

$daysB = array();

$daysB[] = "Mon"; //set the next sequential value to "Mon"

$daysB[] = "Tue";
```

Arrays

Access values

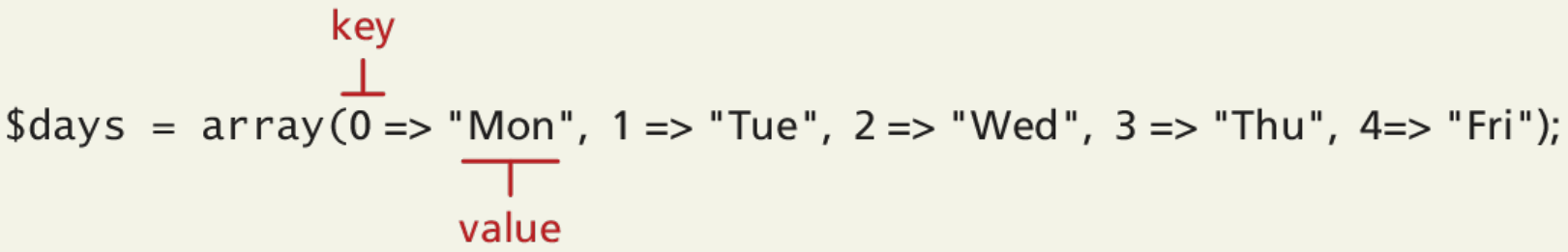
To access values in an array you refer to their key using the square bracket notation.

```
echo "Value at index 1 is ". $days[1];
```

Keys and Values

In PHP, you are also able to explicitly define the keys in addition to the values.

This allows you to use keys other than the classic 0, 1, 2, . . . , n to define the indexes of an array.



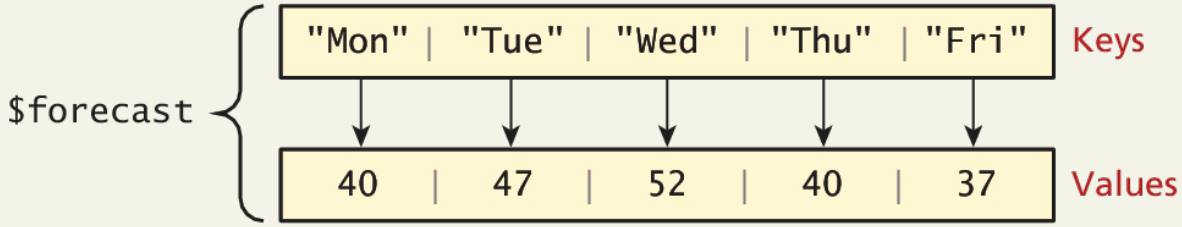
Keys and Values

Array declaration with string keys, integer values

```
$forecast = array("Mon" => 40, "Tue" => 47, "Wed" => 52, "Thu" => 40, "Fri" => 37);
```

key

value




```
echo $forecast["Tue"];  // outputs 47
echo $forecast["Thu"];  // outputs 40
```

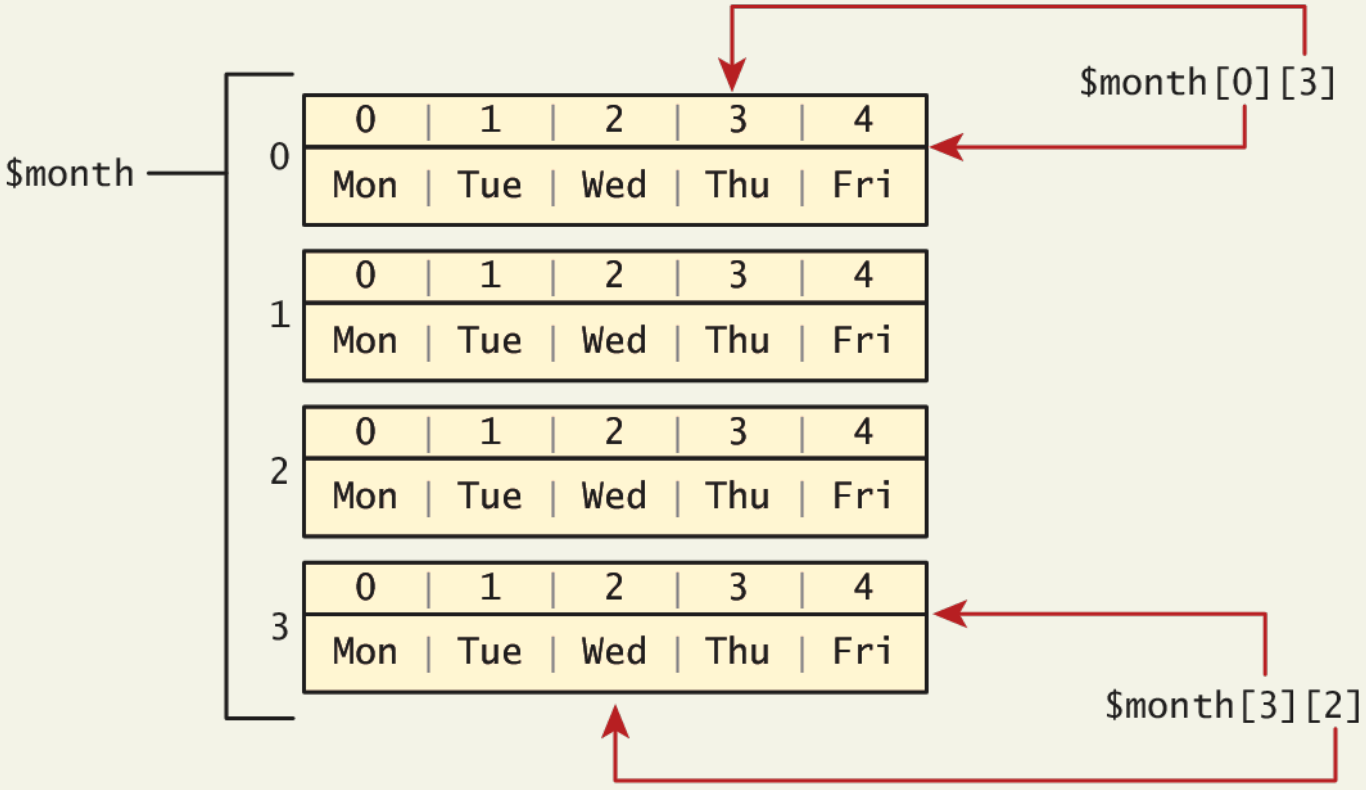
Multidimensional Arrays

Creation

```
$month = array(
        array("Mon","Tue","Wed","Thu","Fri"),
        array("Mon","Tue","Wed","Thu","Fri"),
        array("Mon","Tue","Wed","Thu","Fri"),
        array("Mon","Tue","Wed","Thu","Fri")
);
echo $month[0][3]; // outputs Thu
```

Multidimensional Arrays

Access



Multidimensional Arrays

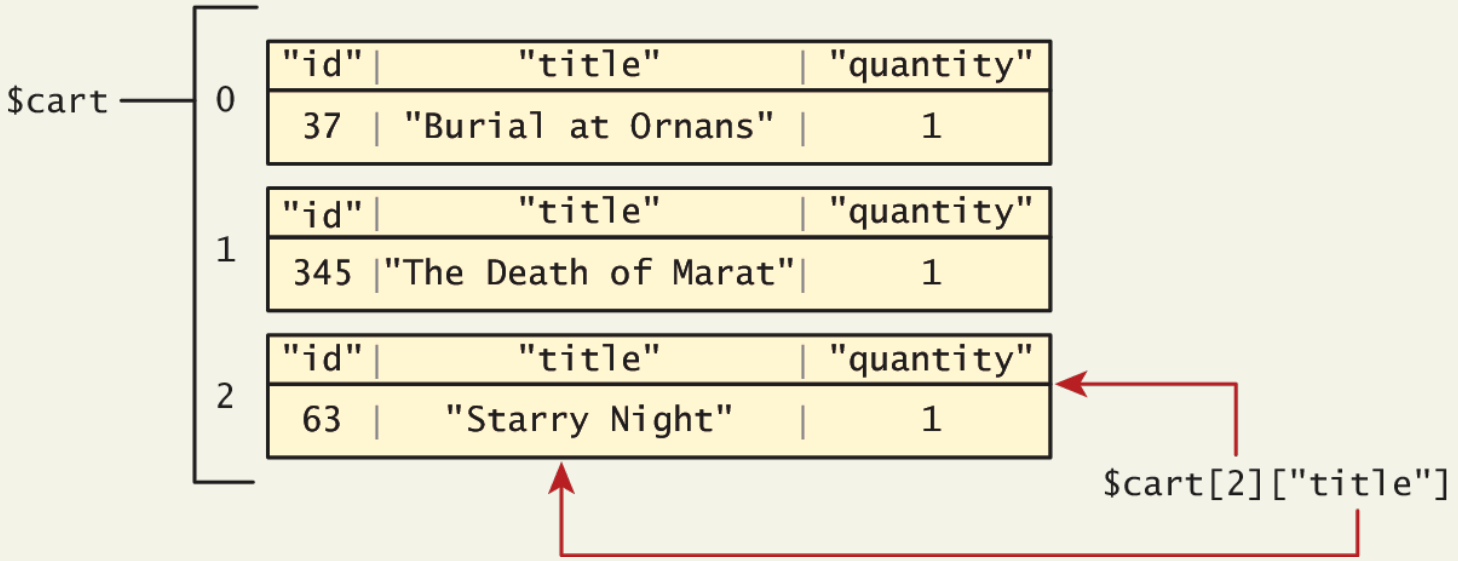
Another example

```
$cart = array();

$cart[] = array("id" => 37, "title" => "Burial at Ornans", "quantity" => 1);

$cart[] = array("id" => 345, "title" => "The Death of Marat", "quantity" => 1);

$cart[] = array("id" => 63, "title" => "Starry Night", "quantity" => 1);
```



Iterating through an array

```php
// while loop
$i=0;
while ($i < count($days)) {
    echo $days[$i] . "<br>";
    $i++;
}


// do While loop
$i=0;
do {
    echo $days[$i] . "<br>";
    $i++;
} while ($i < count($days));


// for loop
for ($i=0; $i<count($days); $i++) {
    echo $days[$i] . "<br>";
}
```

LISTING 9.2 Iterating through an array using while, do while, and for loops

Iterating through an array

Foreach loop is pretty nice

The challenge of using the classic loop structures is that when you have nonsequential integer keys (i.e., an associative array), you can't write a simple loop that uses the **$i++** construct. To address the dynamic nature of such arrays, you have to use iterators to move through such an array.

```php
// foreach: iterating through the values
foreach ($forecast as $value) {
   echo $value . "<br>";
}

// foreach: iterating through the values AND the keys
foreach ($forecast as $key => $value) {
   echo "day" . $key . "=" . $value;
}
```

LISTING 9.3 Iterating through an associative array using a foreach loop

Adding to an array

To an array

An element can be added to an array simply by using a key/index that hasn't been used

```
$days[5] = "Sat";
```

A new element can be added to the end of any array

```
$days[ ] = "Sun";
```

Adding to an array

And quickly printing

PHP is more than happy to let you “skip” an index

```
$days = array("Mon","Tue","Wed","Thu","Fri");

$days[7] = "Sat";

print_r($days);
```

Array ([0] => Mon [1] => Tue [2] => Wed [3] => Thu [**4**] => Fri [**7**] => Sat)

If we try referencing $days[6], it will return a **NULL** value

Deleting from an array

You can explicitly delete array elements using the **unset()** function

```
$days = array("Mon","Tue","Wed","Thu","Fri");

unset($days[2]);
unset($days[3]);

print_r($days); // outputs: Array ( [0] => Mon [1] => Tue [4] => Fri )

$days = array_values($days);
print_r($days); // outputs: Array ( [0] => Mon [1] => Tue [2] => Fri )
```

LISTING 9.4 Deleting elements

array_values() returns all the values from the argument array, indexed numerically (returns a *reindexed* array).

Checking for a value

Since array keys need not be sequential, and need not be integers, you may run into a scenario where you want to check if a value has been set for a particular key.

To check if a value exists for a key, you can therefore use the **isset()** function, which returns true if a value has been set, and false otherwise

```
$oddKeys = array (1 => "hello", 3 => "world", 5 => "!");
if (isset($oddKeys[0])) {
   // The code below will never be reached since $oddKeys[0] is not set!
   echo "there is something set for key 0";
}
if (isset($oddKeys[1])) {
   // This code will run since a key/value pair was defined for key 1
   echo "there is something set for key 1, namely ". $oddKeys[1];
}
```

LISTING 9.5 Illustrating nonsequential keys and usage of isset( )

Array Sorting

Sort it out

There are many built-in sort functions, which sort by key or by value. To sort the $days array by its *values* you would simply use:

sort($days);

As the values are all strings, the resulting array would be:

Array ([0] => Fri [1] => Mon [2] => Sat [3] => Sun [4] => Thu [5] => Tue [6] => Wed)

A different sort, one that would have kept keys and values associated together, is:

asort($days);

Array ([4] => Fri [0] => Mon [5] => Sat [6] => Sun [3] => Thu [1] => Tue [2] => Wed)

More array operations

- **array_keys($someArray)** returns all the key, or all the keys matching a given value (additional arguments)
- **array_values($someArray)** returns all the values
- **array_rand($someArray, $num=1)** returns a set of random keys
- **array_reverse($someArray)** returns a reversed array in the order of values (another argument can be used to keep keys anchored with values)
- **array_walk($someArray, $callback, $optionalParam)** executes the function callback on all the key-value pairs in the array, the last parameter is provided to the callback function
- **in_array($needle, $haystack)** searches haystack for needle value
- **shuffle($someArray)** shuffles the values, new keys are provided
- ...

Section 2 of 5

$_GET AND $_POST SUPERGLOBAL ARRAYS

Superglobal Arrays

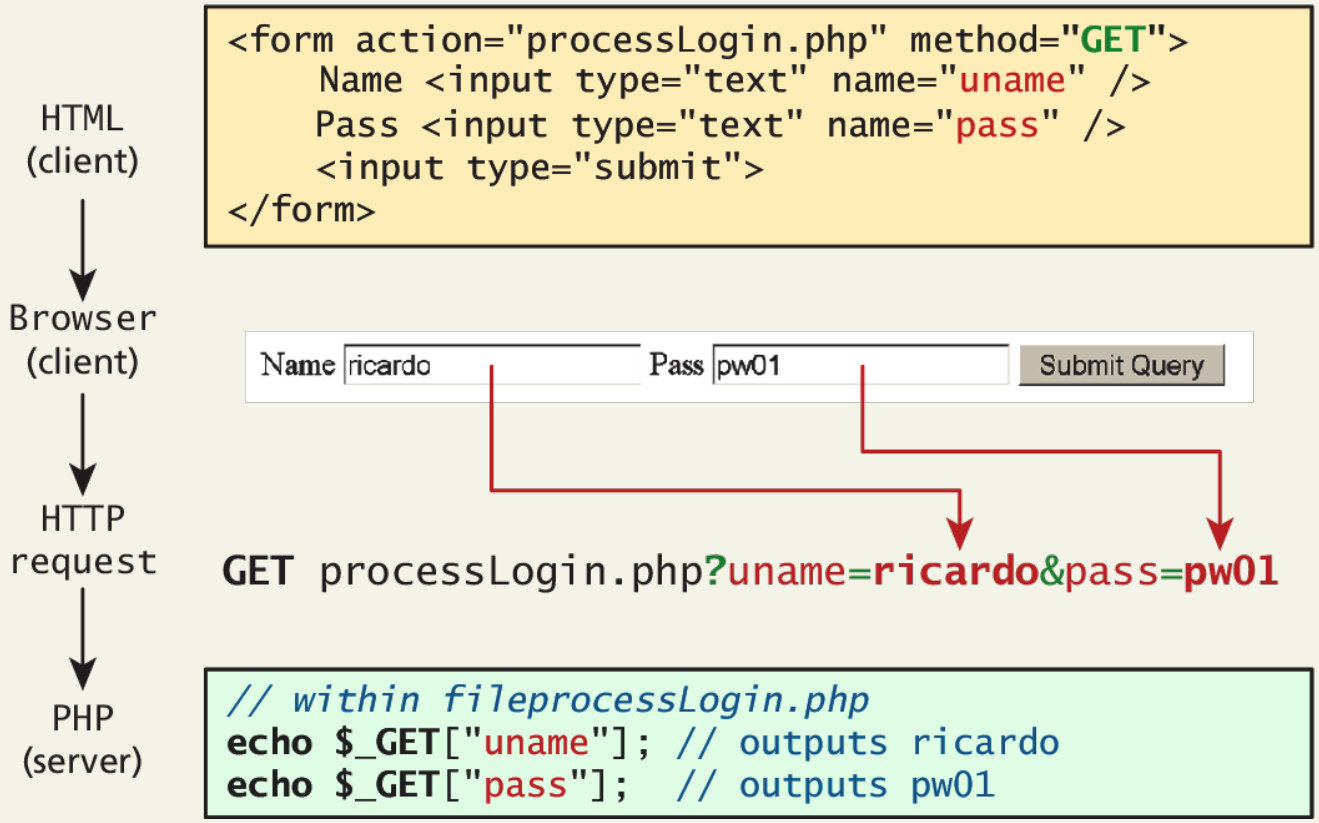
PHP uses special predefined associative arrays called **superglobal variables** that allow the programmer to easily access HTTP headers, query string parameters, and other commonly needed information.

They are called superglobal because they are always in scope, and always defined.

$_GET and $_POST

Sound familiar?

The **$_GET** and **$_POST** arrays are the most important superglobal variables in PHP since they allow the programmer to access data sent by the client in a query string.



$_GET and $_POST

Sound familiar?

- GET requests parse query strings into the $_GET array
- POST requests are parsed into the $_POST array

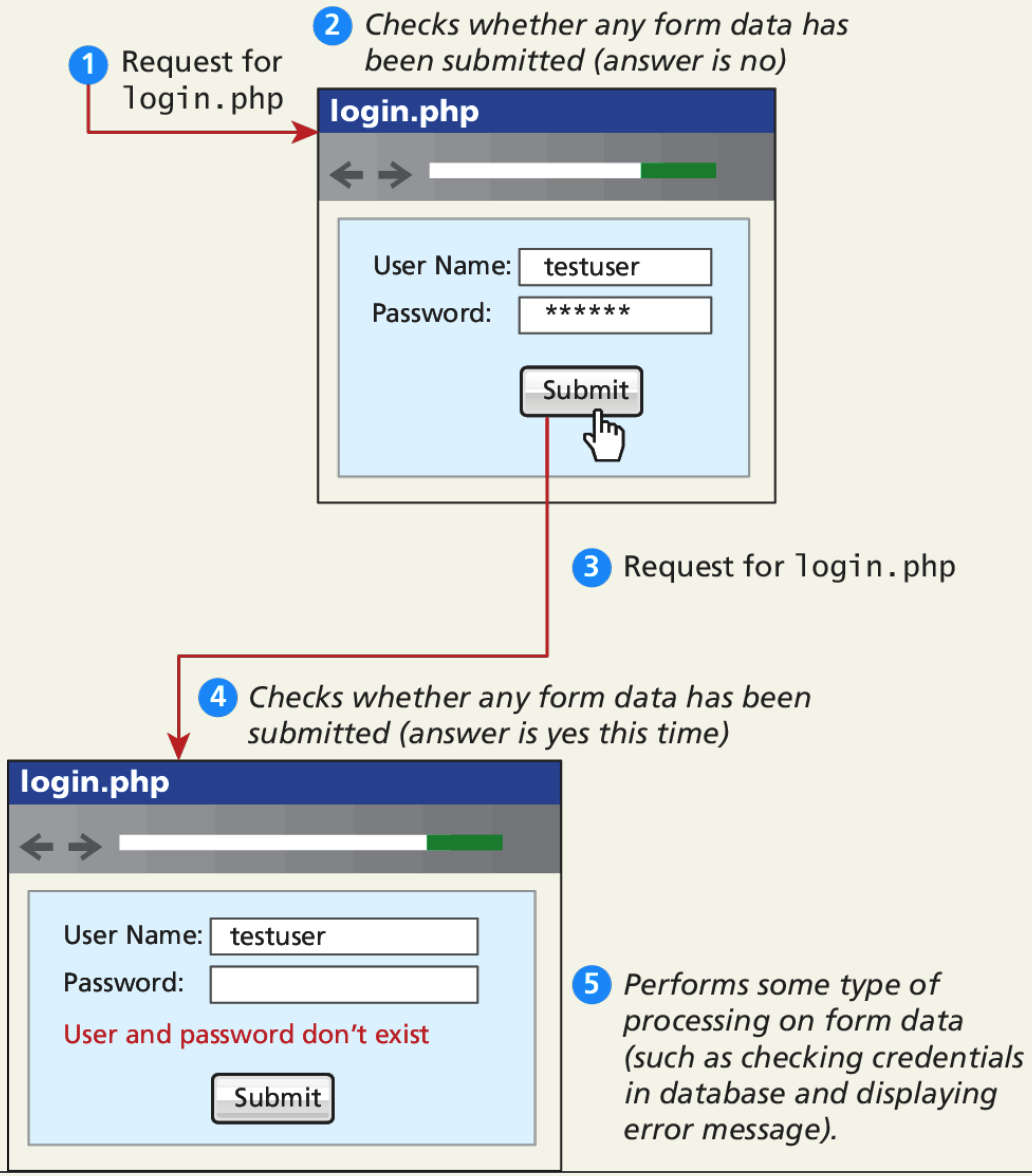
This mechanism greatly simplifies accessing the data posted by the user, since you need not parse the query string or the POST request headers!

Determine if any data sent

```
<!DOCTYPE html>
<html>
<body>
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   if ( isset($_POST["uname"]) && isset($_POST["pass"]) ) {
      // handle the posted data.
      echo "handling user login now ...";
      echo "... here we could redirect or authenticate ";
      echo " and hide login form or something else";
   }
?>
<h1>Some page that has a login form</h1>
<form action="samplePage.php" method="POST">
   Name <input type="text" name="uname"/><br/>
   Pass <input type="password" name="pass"/><br/>
   <input type="submit">
</form>
</body>
</html>
```

LISTING 9.6 Using isset() to check query string data

Determine if any data sent



Accessing Form Array Data

Sometimes in HTML forms you might have multiple values associated with a single name;

```
<form method="get">
   Please select days of the week you are free.<br />
   Monday <input type="checkbox" name="day" value="Monday" /> <br />
   Tuesday <input type="checkbox" name="day" value="Tuesday" /> <br />
   Wednesday <input type="checkbox" name="day" value="Wednesday" /> <br />
   Thursday <input type="checkbox" name="day" value="Thursday" /> <br />
   Friday <input type="checkbox" name="day" value="Friday" /> <br />
   <input type="submit" value="Submit">
</form>
```

LISTING 9.7 HTML that enables multiple values for one name

Accessing Form Array Data

HTML tweaks for arrays of data

Unfortunately, if the user selects more than one day and submits the form, the $_GET['day'] value in the superglobal array *will only contain the last value from the list* that was selected.

To overcome this limitation, you must change the name attribute for each checkbox from **day** to **day[]**.

```
Monday <input type="checkbox" name="day[]" value="Monday" />

Tuesday <input type="checkbox" name="day[]" value="Tuesday" />
```

Accessing Form Array Data

Meanwhile on the server

After making this change in the HTML, the corresponding variable $_GET['day'] will now have a value that is of type array.

```php
<?php

echo "You submitted " . count($_GET['day']) . "values";
foreach ($_GET['day'] as $d) {
   echo $d . ", ";
}

?>
```

LISTING 9.8 PHP code to display an array of checkbox variables

Using Query String in Links

Design idea

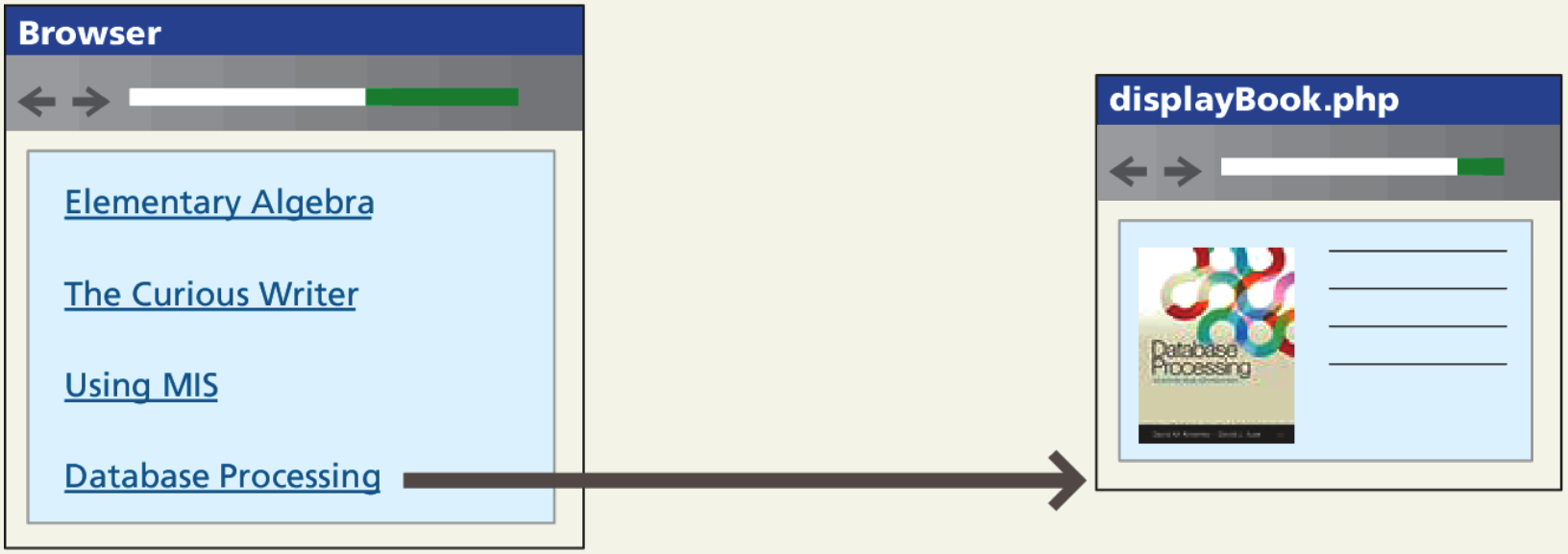
Imagine a web page in which we are displaying a list of book links. One approach would be to have a separate page for each book.

Using Query Strings in links

Not a great setup

Using Query Strings in links

Use the query string to reduce code duplication




```
<a href="displayBook.php?isbn=0132145375">Database Processing</a>
```

Query string

Sanitizing Query Strings

Just because you are expecting a proper query string, doesn't mean that you are going to get a properly constructed query string.

- **distrust all user input**

The process of checking user input for incorrect or missing information is sometimes referred to as the process of **sanitizing user inputs.**

Learn more about this later.

Section 3 of 5

$_SERVER ARRAY

$_SERVER

The **$_SERVER** associative array contains

- HTTP request headers (sent by client)
- configuration options for PHP

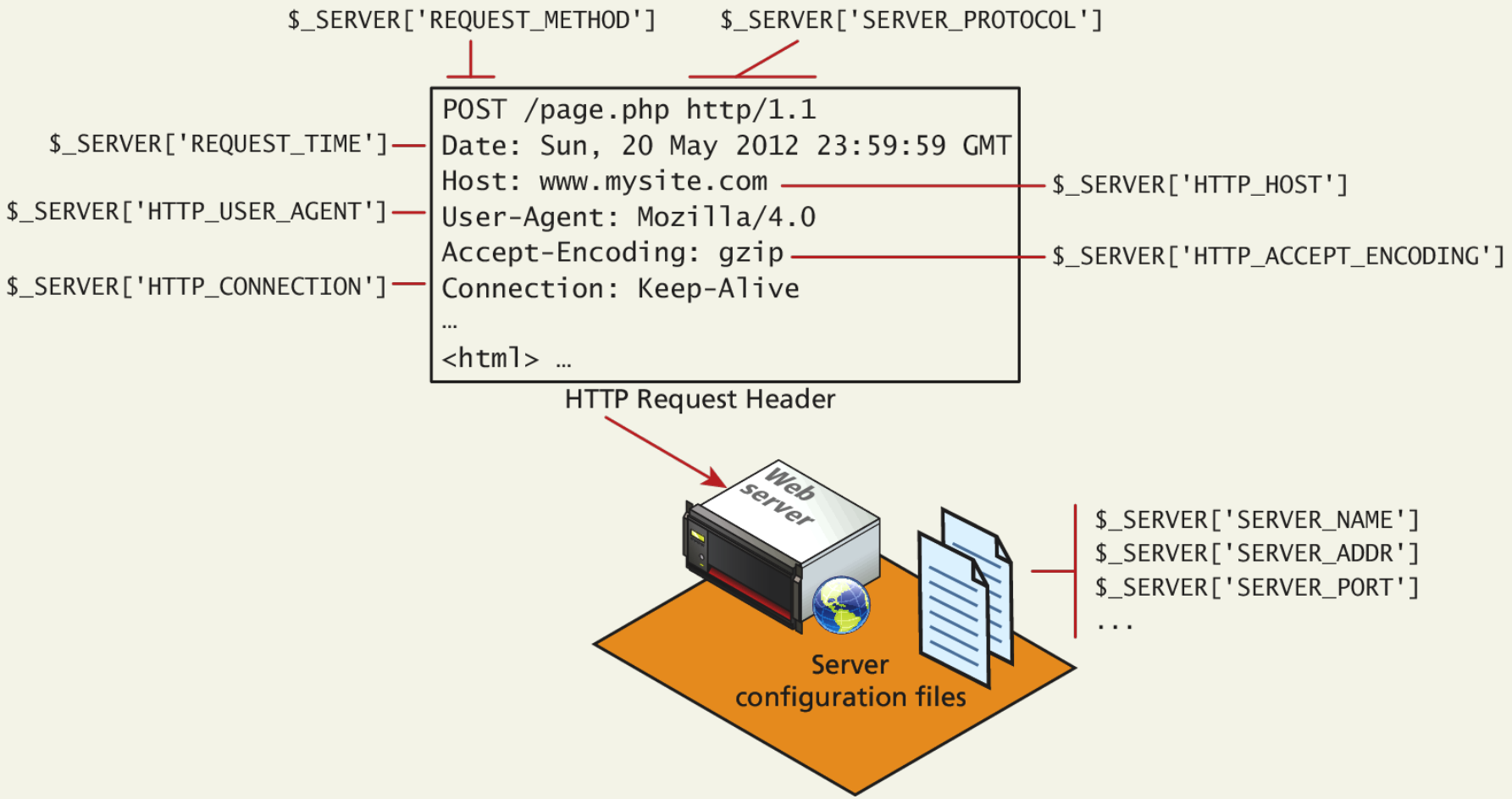
To use the $_SERVER array, you simply refer to the relevant case-sensitive keyname:

```
echo $_SERVER["SERVER_NAME"] . "<br/>";

echo $_SERVER["SERVER_SOFTWARE"] . "<br/>";

echo $_SERVER["REMOTE_ADDR"] . "<br/>";
```

$_SERVER


$_SERVER['REQUEST_METHOD']
$_SERVER['SERVER_PROTOCOL']
POST /page.php http/1.1
$_SERVER['REQUEST_TIME']
Date: Sun, 20 May 2012 23:59:59 GMT
Host: www.mysite.com
$_SERVER['HTTP_HOST']
$_SERVER['HTTP_USER_AGENT']
User-Agent: Mozilla/4.0
Accept-Encoding: gzip
$_SERVER['HTTP_ACCEPT_ENCODING']
$_SERVER['HTTP_CONNECTION']
Connection: Keep-Alive
…
<html> …
HTTP Request Header
Web server
$_SERVER['SERVER_NAME']
$_SERVER['SERVER_ADDR']
$_SERVER['SERVER_PORT']
...
Server configuration files

SERVER INFORMATION KEYS

- **SERVER_NAME** contains the name of the site that was requested
- **SERVER_ADDR** tells us the IP of the server
- **DOCUMENT_ROOT** tells us the location from which you are currently running your script
- **SCRIPT_NAME** key that identifies the actual script being executed

Request Header Keys

- **REQUEST_METHOD** returns the request method that was used to access the page: that is, GET, HEAD, POST, PUT
- **REMOTE_ADDR** key returns the IP address of the requestor
- **HTTP_USER_AGENT** contains the operating system and browser that the client is using
- **HTTP_REFERER** contains the address of the page that referred us to this one (if any) through a link

Header Access Examples

```php
<?php
echo $_SERVER['HTTP_USER_AGENT'];

$browser = get_browser($_SERVER['HTTP_USER_AGENT'], true);
print_r($browser);
?>
```

LISTING 9.10 Accessing the user-agent string in the HTTP headers

```php
$previousPage = $_SERVER['HTTP_REFERER'];
// Check to see if referer was our search page
if (strpos("search.php",$previousPage) != 0) {
   echo "<a href='search.php'>Back to search</a>";
}
// Rest of HTML output
```

LISTING 9.11 Using the HTTP_REFERER header to provide context-dependent output

Security

Headers can be forged

All headers can be forged!

- The HTTP_REFERER header can lie about where the referral came from
- The USER_AGENT can lie about the operating system and browser the client is using.

Section 4 of 5

$_FILES ARRAY

$_FILES Array

The **$_FILES** associative array contains items that have been uploaded in the current request.

A server script must process the upload file(s) in some way; the **$_FILES** array helps in this process.

the **$_FILES** array will contain a *key=value* pair for each file uploaded in the post

First a refresher on HTML forms for uploading files…

HTML Required for File Uploads

1. You must ensure that the HTML form uses the HTTP POST **method**, since transmitting a file through the URL is not possible.
2. You must add the **enctype="multipart/form-data"** attribute to the HTML form that is performing the upload so that the HTTP request can
3. You must include an input type of **file** in your form.

```
<form enctype='multipart/form-data' method='post'>
   <input type='file' name='file1' id='file1' />
   <input type='submit' />
</form>
```

LISTING 9.12 HTML for a form that allows an upload

Handling File upload in PHP

The $_FILES array will contain a key=value pair for each file uploaded in the post.

The key for each element will be the **name** attribute from the HTML form, while **the value will be an array** containing information about the file as well as the file itself.

The keys in that array are the **name**, **type**, **tmp_name**, **error**, and **size**.

Handling File upload in PHP

HTML (client)

```
<form enctype='multipart/form-data' method='post' action='upFile.php'>
   <input type='file'name='file1' />
   <input type='submit' />
</form>
```

Browser (client)

HTTP request

POST upFile.php

HTTP POST multipart/form-data

```
file1‰PNG ™‡»ªÎ ! ¾çÕkåFÊ÷Ï§29¾ªùÄ¡Šrá vÛ„ýìN üc/©(–Ä Á Z}/vë\Å(m–¼i± ¼6_Ë/Hí,÷ª“ …ÀÉA`) ,þ/
_êvŒoá˜™ ß½uÓaG¢ëN"ÕÓ?ÔÝ·ì°m ½€2h‹ ¶•˜|QÄóìçD“W)MíÛù_9ß9nÝO2'Â² km'Ñyþh $ž“›Ÿ˜ d®8í¬\
¡ûh»}>´‡¥[ ðB ªðèè[ ‡¥ué|ÝxR¾XL)à£›[ÕZ!pâ X¢ kÖ >Ö °cÙý}=‡ ÅÔt©Q¯íÙ±¥<3½üÔ3 ×l‘ó;¶l¬W+4èê
@å,Òò9'tâÈC ŒîÛ11Êëœ´B>{Úêå§® ®Öê[¶ïÞ¾soizªVydÙÊµ'ÂaÁ_ Ã ¯Ç jñzðceŒ%Ÿ~B ´=àã'~ÁÌÌ'¡íùú 5¾$1 “
Z³Ñ ð@ Ã¶ £…hÉGÄ ¢ª+¥i>ÊƒŽÏŒ‹ûT …À| –ð€[®Â±ç€_´nù»ßò®ùOmŸa ï ¿õ{?üö÷ï™ Û_ìé?\ËÃ /
æsÙ”8¾‰têy˜¨>qäíÛõÌÔøþ Ï=ó7ßù‹ ‡š8Ä, ßøô'?÷ÅíÏlX±æìl¾x8ô ?¿ ñÙLòoþî Ú7~î¦;¶ì>d§éâ >
®¶Â:K{ªG@ð±xÀ Üsx *wnÛ¼o×¶ñ±ýX]1Ii\ ^¹fÅêuéL6>yÈÁ¼ÇÐUÜ`ý#÷×ëÕøãà² ËW¯õ<wËSë'ÇGf&ÇãÌQ&›?eå
…
```

PHP (server)

```
echo $_FILES["file1"]["name"]          // "Sample1.png"
echo $_FILES["file1"]["type"]          // "image/png"
echo $_FILES["file1"]["tmp_file"]      // "/tmp/phpJ08pVh"
echo $_FILES["file1"]["error"]         //   0
echo $_FILES["file1"]["size"]          // 1219038
```

Handling File upload in PHP

Keys. We still have to do something with this data

- **name** is a string containing the full file name used on the client machine, including any file extension.
- **type** defines the MIME type of the file
- **tmp_name** is the full path to the location on your server where the file is being temporarily stored.
- **error** is an integer that encodes many possible errors and is set to UPLOAD_ERR_OK (integer value 0) if the file was uploaded successfully.
- **size** is an integer representing the size in bytes of the uploaded file.

Check for errors

For every uploaded file, there is an error value associated with it in the **$_FILES** array.

The value for a successful upload is **UPLOAD_ERR_OK**, and should be looked for before proceeding any further.

```
foreach ($_FILES as $fileKey => $fileArray) {
   if ($fileArray["error"] != UPLOAD_ERR_OK) { // error
      echo "Error: " . $fileKey . " has error" . $fileArray["error"]
           . "<br>";
   }
   else {   // no error
      echo $fileKey . "Uploaded successfully ";
   }
}
```

LISTING 9.13 Checking each file uploaded for errors

File Size Restrictions

There are three main mechanisms for maintaining uploaded file size restrictions:

- HTML in the input form
- via JavaScript in the input form
- via PHP coding.

HTML in the input form

Add a hidden input field before any other input fields in your HTML form with a name of **MAX_FILE_SIZE**

The file uploading must be complete before an error message can be received.

PHP only; mechanism cannot be trusted

```
<form enctype='multipart/form-data' method='post'>
   <input type="hidden" name="MAX_FILE_SIZE" value="1000000" />
   <input type='file' name='file1' />
   <input type='submit' />
</form>
```

LISTING 9.14 Limiting upload file size via HTML

Via JavaScript

Allows a client side check to happen before any data transmitted. (Easily overridden).

```
<script>
var file = document.getElementById('file1');
var max_size = document.getElementById("max_file_size").value;
if (file.files && file.files.length ==1){
   if (file.files[0].size > max_size) {
      alert("The file must be less than " + (max_size/1024) + "KB");

   }
}
</script>
```

LISTING 9.15 Limiting upload file size via JavaScript

via PHP

The only one you HAVE to do.

The third mechanism for limiting the uploaded file size is to add a simple check on the server side (just in case JavaScript was turned off or the user modified the MAX_FILE_SIZE hidden field).

```
$max_file_size = 10000000;
foreach($_FILES as $fileKey => $fileArray) {
   if ($fileArray["size"] > $max_file_size) {
      echo "Error: " . $fileKey . " is too big";
   }
   printf("%s is %.2f KB", $fileKey, $fileArray["size"]/1024);
}
```

LISTING 9.16 Limiting upload file size via PHP

Limiting the type of File Upload

I won't allow .abc, .def now let me be

What if you wanted the user to upload an image and they uploaded a Microsoft Word document?

You might also want to limit the uploaded image to certain image types, such as *jpg* and *png*, while disallowing bmp and others.

- examine the file extension
- and the type field

Limiting the type of File Upload

Example code

```
$validExt = array("jpg", "png");
$validMime = array("image/jpeg","image/png");
foreach($_FILES as $fileKey => $fileArray ){
   $extension = end(explode(".", $fileArray["name"]));
   if (in_array($fileArray["type"],$validMime) &&
          in_array($extension, $validExt)) {
      echo "all is well. Extension and mime types valid";
   }
   else {
      echo $fileKey." Has an invalid mime type or extension";
   }
}
```

LISTING 9.17 PHP code to look for valid mime types and file extensions

Also in this case the user could upload a file with a content that does not match the extension...

Moving the File

Finally!

You must move the temporary file to a permanent location on your server.

move_uploaded_file() takes in the temporary file location and the file's final destination.

```
$fileToMove = $_FILES['file1']['tmp_name'];
$destination = "./upload/" . $_FILES["file1"]["name"];
if (move_uploaded_file($fileToMove,$destination)) {
   echo "The file was uploaded and moved successfully!";
}
else {
   echo "there was a problem moving the file";
}
```

LISTING 9.18 Using move_uploaded_file() function

Section 5 of 5

READING/WRITING FILES

Reading/Writing

There are two basic techniques for read/writing files in PHP:

- **Stream access**. In this technique, our code will read just a small portion of the file at a time. While this does require more careful programming, it is the most memory-efficient approach when reading very large files.
- **All-In-Memory access**. In this technique, we can read the entire file into memory. While not appropriate for large files, it does make processing of the file extremely easy.

Stream Access

C style

C style file access. More difficult, but more memory efficient.

The function **fopen()** takes a file location or URL and access mode as parameters. The returned value is a **stream resource**, which you can then read sequentially.

Use **fread()** or **fgets()** to read ahead in the file.

fclose() is invoked when you are done.

Writing done much the same with **fwrite()**.

Stream Access

Just show me the code

```
$f = fopen("sample.txt", "r");
$ln = 0;
while ($line = fgets($f)) {
    $ln++;
    printf("%2d: ", $ln);
    echo $line . "<br>";
}
fclose($f);
```

LISTING 9.19 Opening, reading lines, and closing a file

In-Memory File Access

Easy as pie

- **file()** Reads the entire file into an array, with each array element corresponding to one line in the file
- **file_get_contents()** reads the entire file into a string variable
- **file_put_contents()** writes the contents of a string variable out to a file

In-Memory File Access

To read an entire file into a variable you simply use:

$fileAsString = file_get_contents(FILENAME);

To write the contents of a string $writeme to a file:

file_put_contents(FILENAME, $writeme);

In-Memory File Access

Consider a realistic example

Imagine we have a comma-delimited text file that contains information about paintings, where each line in the file corresponds to a different painting:

```
01070,Picasso,The Actor,1904

01080,Picasso,Family of Saltimbanques,1905

02070,Matisse,The Red Madras Headdress,1907

05010,David,The Oath of the Horatii,1784
```

In-Memory File Access

Parsing our file

```
// read the file into memory; if there is an error then stop processing
$paintings = file($filename) or die('ERROR: Cannot find file');

// our data is comma-delimited
$delimiter = ',';

// loop through each line of the file
foreach ($paintings as $painting) {

   // returns an array of strings where each element in the array
   // corresponds to each substring between the delimiters

   $paintingFields = explode($delimiter, $painting);

   $id= $paintingFields[0];
   $artist = $paintingFields[1];
   $title = $paintingFields[2];
   $year = $paintingFields[3];

   // do something with this data
   . . .
}
```

LISTING 9.20 Processing a comma-delimited file