

Some PHP Issues

- System Architecture
 - how functions/operations/files, etc are organized.
- Objects and OOP in PHP
- Error Handling
- Using libraries
 - examples:
 - graphics generation
 - pdf generation

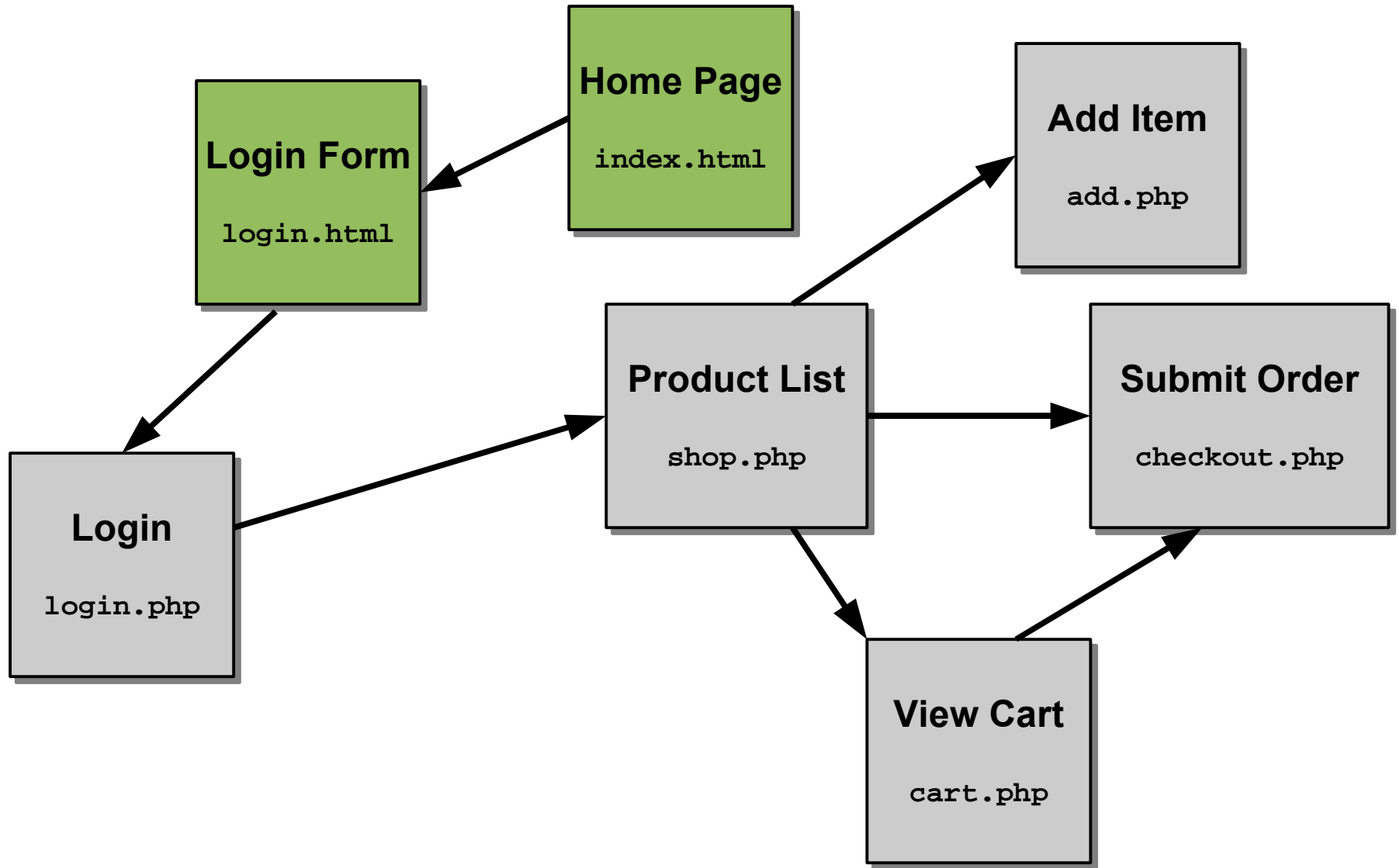
System Architecture

- Consider a complex system with many different kinds of operations.
- There are a number of ways to organize such a system:
 - different URLs (php programs) for different functions
 - Single URL that can handle any kind of request
 - uses query string (form fields) to distinguish operation.
 - different objects (more on OOP later).

Using lots of independent PHP programs.

- Advantages:
 - can develop and debug each program independently.
- Disadvantages:
 - probably lots of duplication of code/HTML
 - any change requires modifying many files.
 - hard to keep the appearance and operation consistent (which can determine whether the system is easy to use or not).
 - Each program requires it's own documentation and testing (more duplication of effort).

Example



Shared Code

- All the php programs in this example need to access the database
 - establishing the connection, etc.
- We probably want them all to look similar
 - lots of identical HTML
- Session control/identification code needed by all.
- Error handling (once again, consistency is an issue).

Single PHP program

- Probably a bad idea to try to put all the code for a complex system in a single file.
 - hard to find individual functions, etc.
 - multiple developers can mess each other up.
- Solution: include and require

```
include ("foo.php")
```

start reading from foo.php if possible

```
require ("blah.php")
```

start reading from blah.php
generate an error if blah.php is not found

include files

- Each file included is treated as a php file
 - can include HTML
 - can include PHP (inside `<?php ?>`)
- You can include a pure HTML file
 - file name and extension don't mean anything
- PHP will look for the file in the same directory as the *original* php program.
 - the one that received the current HTTP request.

includes in includes

- There is no limit on when/where you can use include (or require).
 - in some situations you need to worry about the location of the include file, for example if you create folders to hold different include files.
 - You can handle these problems with the following:

`dirname (__FILE__)`

This is the path to the directory
(folder) that holds the current file

One possible idea

```
<?php
include('head.html'); # page top and banner
include('body.php'); # the real work (code)
include('foot.html'); # footer
?>
```

This is a little extreme, but you get the idea...

Code Organization

- You can create *modules*
 - collections of code/data that deal with some particular operations or type of operations.
- For example:
 - db.php: put all code that deals with the database in a file. Some code (that you always want executed) can be at the global level, other code should be in function definitions (to be called later).

Sample db.php

```
<?php
$username="php"; $password="php"; $database="dbintro";
mysql_connect(localhost,$username,$password);
@mysql_select_db($database) or die( "Unable to select
    database" );

function lookupUser($userid) {
    ...
function getShoppingCart($userid) {
    ...
function getProductList() {
    ...
```

Alternative System Organization

- One main URL. All requests are handled by this main php program.
- A bunch of other php files that are included:
 - db.php: database connection and support functions.
 - login.php: functions for login
 - cart.php: functions to viewing, updating shopping cart.
 - products: functions for viewing product list, ...

index.php

```
<?php
require ( 'db.php' ) ;
require ( 'login.php' ) ;
require ( 'cart.php' ) ;
require ( 'products.php' ) ;

if ( $_REQUEST [ 'op' ] == "login" ) {
    login ( ) ;
elseif ( $_REQUEST [ 'op' ] == "view" ) {
    viewcart ( ) ;
...

```

Object Oriented PHP

- You can define object types (classes) and create objects.
- Basic idea is like C++, although not nearly as powerful.
 - Some differences between objects in PHP 4 vs. 5
- Single inheritance
- No fancy protection (everything is public).

Class definition

```
class student {  
    var $name;  
    var $grade="F";    # default grade  
  
    function student($stuname) {  
        $this->name = $stuname;  
    }  
    function assignGrade($g) {  
        $this->grade = $g;  
    }  
    function flunk() { $grade = "F"; }  
}
```

field (data member) declarations.

constructor

goofy required syntax

methods

PHP System Issues

Creating and using an object

```
$fred = new student("Fred");  
echo "<p>fred's grade is " . $fred->grade. "</p>";  
  
$fred->grade = "B";  
echo "<p>fred's grade is " . $fred->grade. "</p>";  
  
$fred->flunk();  
echo "<p>fred's grade is " . $fred->grade. "</p>";
```

Object Example - [calendar.php](#)

October 2006						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Error Handling

- **IMPORTANT!**
 - you don't want user's to see PHP errors
 - or MySQL errors.
 - This can actually be a security issue.
- There are a number of approaches:
 - always generate a simple error message and then call exit;
 - suppress automatically generated error messages.
 - capture errors with an error handler.

Simple Error Handling

- Doesn't work for PHP parse errors!
- Doesn't work for many other kinds of errors
 - for example, errors automatically generated by MySQL functions.
- General idea is to check for error conditions individually.
- *Wrapper functions* are one approach

Wrapper Example

```
function getUsername() {  
    if (isset($_REQUEST['username'])) {  
        return($_REQUEST['username']);  
    } else {  
        echo "<H3>Invalid Request</H3>\n";  
        exit;  
    }  
}
```

Error Suppression

- You can prefix any PHP expression with '@' to suppress automatic error messages:

```
$value = ($x / $y) ;
```

- If `$y == 0`, this will generate an error message.

```
$value = @ ($x / $y) ;
```

- Now no message is generated (but who knows what `$value` is...)
- This can only handle runtime errors.

Better Wrappers

- You can use error suppression in wrappers:

```
function dbConnect() {  
    @mysql_connect('localhost', 'php', 'php');  
    @mysql_select_db($database);  
}
```

- Unfortunately this simply prevents error messages from being sent to the user, it doesn't stop the errors from happening...

Global error suppression

- You can suppress all automatically generated error messages with:

```
error_reporting(0);
```

- You can also set this as the default in your system-wide `php.ini` (php configuration file).
- Only handles runtime errors (not parsing errors).

Error Handlers

- You can also tell PHP that you want to provide code that should handle all errors.

```
set_error_handler('function_name');
```

- Every error detected by PHP will result in a call to the function *function_name* (this is a function you create).
- You can write an error handler that generates a nice HTML error page.

Sample Error Handler

```
function errhandler($error,$error_string) {  
    echo "<h3>An error has occurred: $error_string</h3>\n";  
    echo "<h3>Please call customer support and report this  
error!</h3>\n";  
    exit;  
}  
  
set_error_handler('errhandler');
```

Other Error Handling Issues

- There are three levels of errors:
 - notice
 - warning
 - error
- You can control these somewhat independently
 - you could live with notices displaying a message, but suppress warnings and errors.

Using PHP Libraries

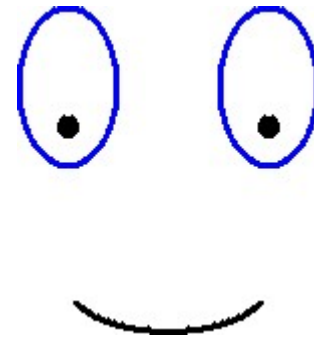
- There are many libraries available for PHP
 - content creation (images, sounds, pdfs, etc)
 - Database (MySQL, Postgress, Oracle, ...)
 - Networking
 - XML
 - email
- Some libraries are a collection of functions, some are provided as some classes and objects.

Example Library: GD

- GD is a graphics generation library
 - available as a PHP extension (try `phpinfo()` ; to see if you have this extension).
 - Allows you to generate images (GIF,png,jpeg) with a PHP program.
 - add text to any image file.
 - create a drawing.
 - create an image for a button.

Example GD code

```
<?php
$im = ImageCreate(400,400);
$white = ImageColorAllocate($im,0xff,0xff,0xff);
$black = ImageColorAllocate($im,0,0,0);
$blue = ImageColorAllocate($im,0,0,0xff);
ImageSetThickness($im,3);
ImageArc($im,100,100,50,80,0,360,$blue);
ImageArc($im,200,100,50,80,0,360,$blue);
ImageArc($im,100,120,10,10,0,360,$black);
ImageFill($im,100,120,$black);
ImageArc($im,200,120,10,10,0,360,$black);
ImageFill($im,200,120,$black);
ImageArc($im,150,200,100,45,20,160,$black);
header('Content-Type: image/png');
ImagePNG($im);
?>
```



GD Exercise

- Check out the documentation at:
us2.php.net/manual/en/ref.image.php
- Use PHP to create an image of a bullseye:



PDF generation

- There are a number of PDF generation libraries.
- fpdf is a simple and free library available at:
<http://www.fpdf.org/>
- The documentation is also available at this site.
- You can install the library in the folder with your PHP files and use require to include this code.

PDF with fpdf Example

```
<?php
require ("fpdf.php");
$pdf=new FPDF("P","in");
$pdf->AddPage();
$pdf->setFont('Times','B',14);
$pdf->Text(1.0,1.0,"Student Record for Joe Dohn");
$pdf->setFont('Helvetica','',12);
$y=1.5;
$pdf->Text(1.0,$y,"Name:");
$pdf->Text(2.0,$y,"Joe Dohn");
$y+=.25;
$pdf->Text(1.0,$y,"ID:");
$pdf->Text(2.0,$y,"66012334");
$y+=.25;
$pdf->Text(1.0,$y,"Grade:");
$pdf->Text(2.0,$y,"C++");
$pdf->Output();
?>
```

Student Record for Joe Dohn

Name:	Joe Dohn
ID:	66012334
Grade:	C++

PDF Exercise

- Create a PDF that shows a shopping cart.
 - PHP program can look for userid in the query.
 -
- fpdf is pretty easy to use, the PHP pdf support described on php.net is a little more complex and requires a license for commercial use.