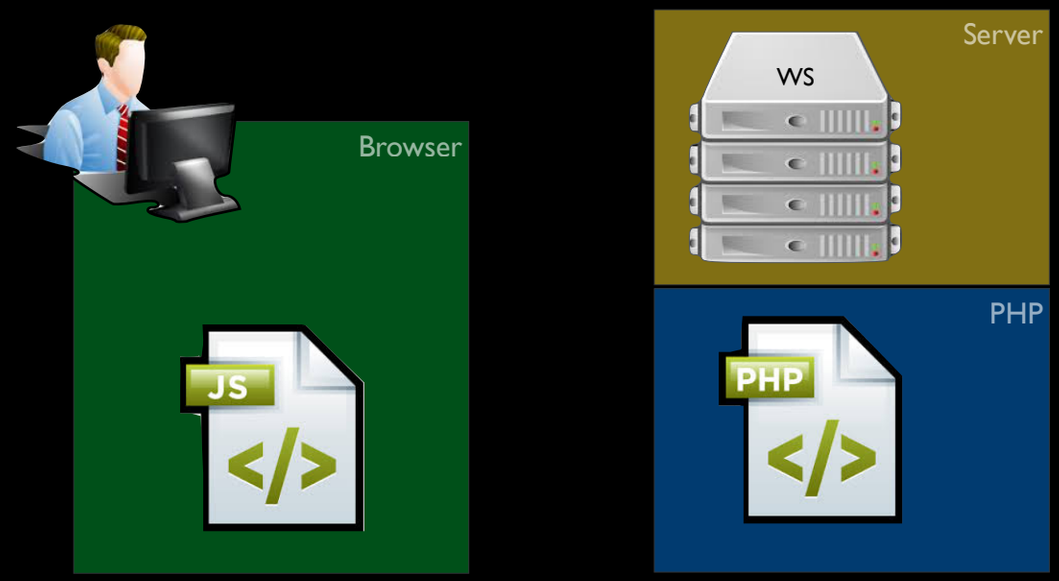
TU PHP

Objectives

- Client-side v server-side
- Php and Apache
- Php overview

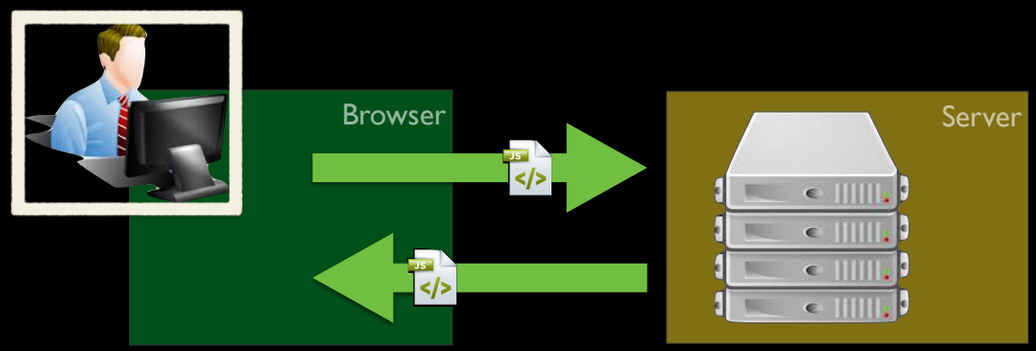
Client-side v Server-side



The primary scripting language on the client is JS, which runs inside the browser.

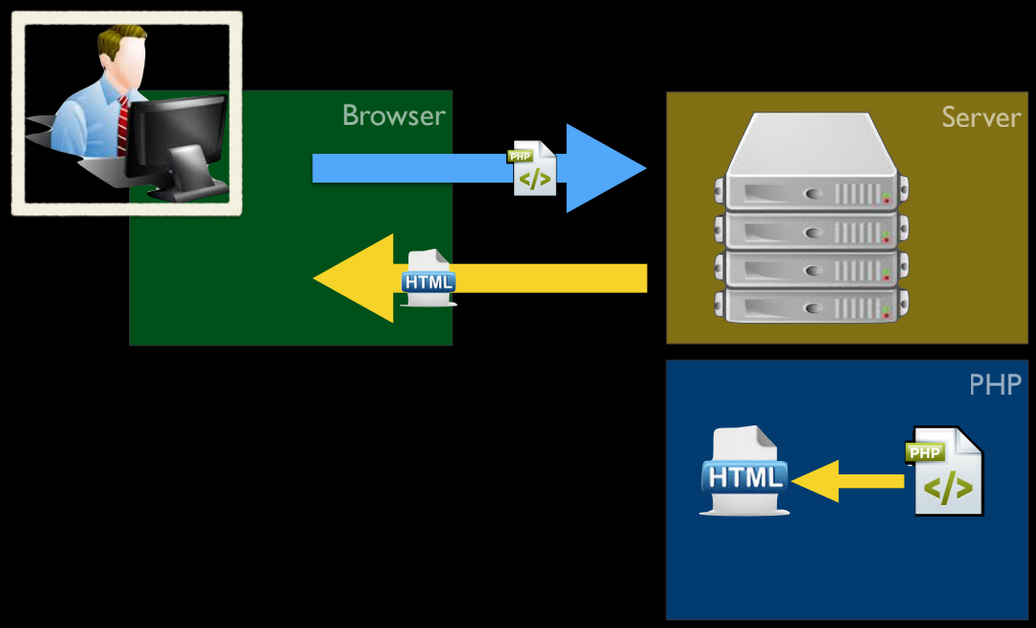
Their are several choices for server side scripting. We are using php, which fits nicely with MySQL.

JS script requested and delivered



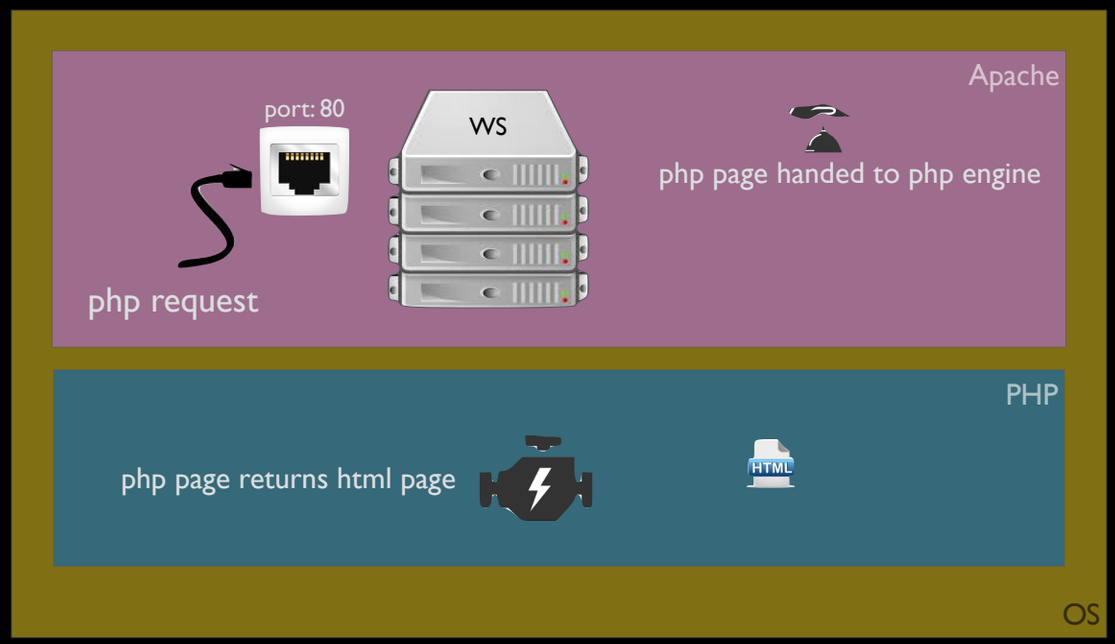
The client requests a js script, the ws obliges and the browser executes the script. Client-side.

php script requested
html delivered



The client requests a php script, the ws retrieves it passing it along to the php engine for processing. The php engine then creates the html file, gives to the ws which in turn returns to the client for rendering. Server-side.

Apache, Php and the OS work



Our development stack (XAMP or MAMP) includes Apache, a web server and php. The WS runs on your machine, but functions just like any ws on the internet would.

Requests to the ws come in through port:80, although the stack will most likely renumber it to avoid conflicts with other installed web servers, such as Microsofts web server.

Php files are processed by the php engine (a module loaded by Apache), and html files are the output of this processing. The user never sees this php, and thus it is considered a more secure way of development.

PHP
Hypertext Preprocessor

Originally: Personal Home Page

Now: Php: Hypertext Preprocessor

The good stuff

- dynamically / loosely typed
- supports OO
  - class
  - inheritance and polymorphism
  - interface
- control flow
- functions

Very must an OO language like C++, Java and others

Try not to embed PHP

```php
<?php $user = "Forest" ?>

<!DOCTYPE html>
<html>
  <body>
    <h1>Welcome
      <?php echo $user; ?></h1>
    <p> The Server time is
        <?php
        echo "<strong>";
        echo date("H:i:s");
        echo "</strong>"; ?></p>
  </body>
</html>
```

What the server processes

You will not be able to eliminate all php from the markup, but there are good ways and bad ways to do this.

Most of the php should be in separate files, with minimal php in the markup.

Notice here how the php is intermixed within the html which makes maintenance less than favorable.

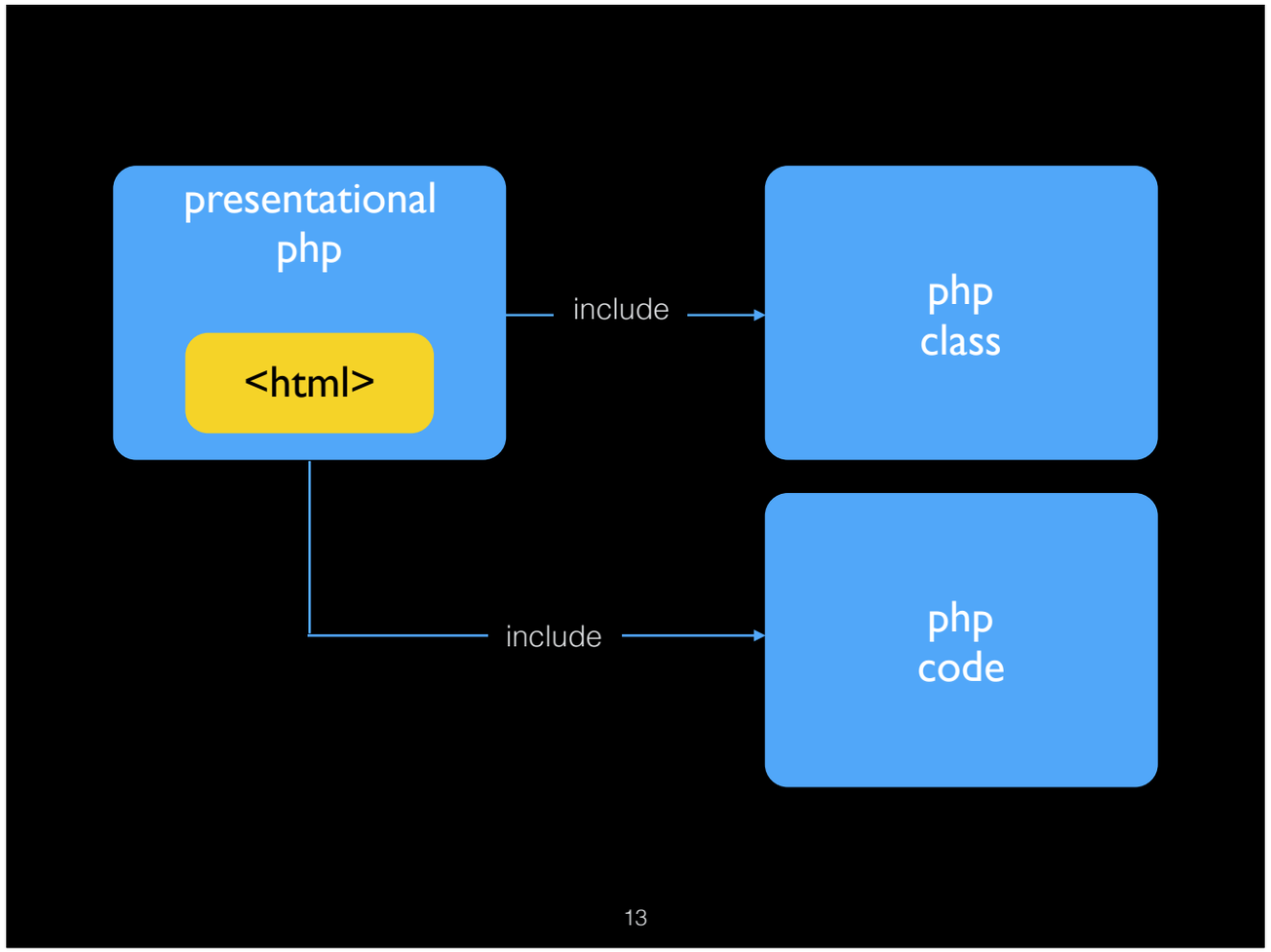
```
<!DOCTYPE html>
  <html>
  <body>
  <h1>Welcome
    Forest</h1>
  <p>The Server time is
     <strong>02:59:09</strong></p>
  </body>
  </html>
```

What the browser processes

The server processes all php and creates the equivalent html. This is why php is not visible to end users and thus more secure overall.

Notice also, that each <?php tag defines where the html is placed and thus keeping that in mind may lead to more readable final html files.

Use layers instead



The approach is preferred since we are layering the functionality and separating the various components.

The presentational file will contain the markup and the necessary php that uses the other external files, which are included.

What's up doc?

- # More readable? You decide
- // Deja vu? Look familiar?
- /* multiple lines
  * are easier to comment
  */

Know thy variables

```php
// Dynamically typed
$age = 21;                  // Integer
$name = 'Stark';            // String
$what;                      // Undefined

// Loosely typed
$var = 21;
$var = "Stark";
$var = [];                  // Array

// Case sensitive
$age = 21;
$Age = 22;

define('PI', 3.1415);    // constant
```


Dynamically type basically means that the type of the variable is inferred by its value.

Loosely type means it can take on any value, thus its type can change with each assignment.

Scope is two flavored

```
<?php Global
  function A() {  local A


  }
  function B() {  local B


  }
?>
```

```php
<?php
  $globallyScoped = "woo hoo";



  function iCantSeeYou() {
    echo $globallyScoped;    // local var

    global $globallyScoped;
    echo $globallyScoped;    // global var
  }
?>
```


In order for the function iCantSeeYou() to access the global variable $allCanSeeMe, it must declare it as global inside its body. That is,

global $allCanSeeMe;
...
Now, it can use it.

Data Types	
`Boolean`	true or false
`Integer`	whole numbers
`Float`	decimal numbers
`String`	letters
`Array`	collection of any type
`Object`	class instance

Variable variables?😳

```php
<?php

  $fab1 = "John";
  $fab2 = "Paul";
  $fab3 = "George";
  $fab4 = "Ringo";

  echo "Ladies and gentlemen the beatles!";
  for ($i = 1; $i <= 4; $i++) {
    echo ${"fab" . $i} . "<br>";
  }

?>
```


A variable can contain the name of a variable, which makes it extremely powerful and expressive.

Notice how we can use a loop to cycle through all the variables. We are building the variable name basically. Groovy!

echo caveats

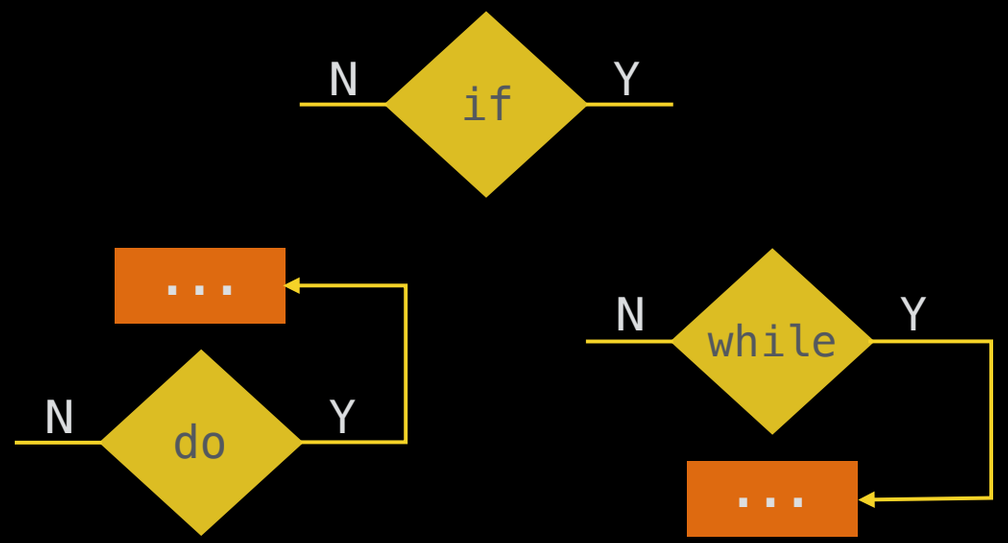
```php
// echo's output is the html file
echo ("hello");
echo "hello";




// string interpolation uses double quotes "
echo "$fab1";         // John
echo '$fab1';         // $fab1
```


You could also use the print statement instead of the echo statement.

```php
// string concatenation uses single dot .
echo "this" . "and that";     // thisand that
echo "this" . " and that";    // this and that




// printf() offers more formatting options
printf("Hello %s. Give me %d", $fab1, 5);

// Hello John. Give me 5
```


You could also use the print statement instead of the echo statement.

Controlling flow



```
if ($timeToQuit == true) {}


if ($timeToQuit == true) {}
else                     {}


if ($timeToQuit == true)        {}
else if ($timeToQuit == false) {}
else                            {}
```

```
switch($timeToQuit) {
  case true:  echo "Clock out."; break;
  case false: echo "Not yet.";   break;
  default:    echo "Huh?";       break;
}
```

```
$timeToQuit = ...
while ( !$timeToQuit) {
   ...
   $timeToQuit = ...
}
```

```
do {
   ...
   $timeToQuit = ...
} while ( !$timeToQuit);
```

```php
// Iterative
for ($i = 0; $i <= $upperBound; $i) {}


// foreach
$arr = [1,2,3];
foreach ($arr as $value) {}


foreach ($arr as $key => $value) {}
```

functions are it baby

```php
function iDontReturnAnything() {
  echo "I will not return. :-)";
}


function iDoReturnOnTheOtherHand() {
  return "I am returning this";
}
```


Functions begin with the keyword, function, followed by the name and the parameter list, which can be empty.

Functions can return a value, but that fact is not advertised in their header (unlike C++).

More good stuff

- Arrays
- Superglobals
- File upload
- File IO

Array surprises

- ordered maps (key => value)
- key - string or integer
  - don't have to be sequential
- value - any type

Arrays are multi-faceted to say the least.

You can think of them in the traditional array way, or as a map or dictionary, with a key: value pair.

Creating ...

```php
// Array constructor
$days = array();

$days = array("Mon", "Tue");
    // $days[0]: "Mon"
    // $days[1]: "Tue"


// Array literal
$days = [];

$days = ["Mon", "Tue");
    // $days[0]: "Mon"
    // $days[1]: "Tue"
```


Creating arrays is also very flexible, and you can use a ctor or literal notation, depending on your needs.

You may create in the spirit of an array and/or in the spirit of a map (key: value).

```php
// keys are implied
$days = ["Sun", "Mon"];
  // [0] => "Sun"
  // [1] => "Mon"


// keys are explicit
$days = [0 => "Sun", 1 => "Mon"];
$days = array(0 => "Sun", 1 => "Mon");


// keys are not consecutive
$days = [0 => "Sun", 6 => "Sat"];


var_dump($days);
```


The keys don't have to be continuous when numeric. Just like when they are strings.

Therefore, gaps in the key sequence is acceptable, but must be handled cautiously. No for statement here, but rather foreach.

Iterating ...

```php
// if keys are sequential, all is good
$days = ["Sun", "Mon"];

for ($day = 0; $day <= 1; $day++) {
  echo "$days[$day]<br>";
}
// Sun
// Mon

// but if not, oops!
$days = [0 => "Sun", 6 => "Sat"];

for ($day = 0; $day <= 1; $day++) {
  echo "$days[$day]<br>";
}
// Sun
```


The keys don't have to be continuous when numeric. Just like when they are strings.

Therefore, gaps in the key sequence is acceptable, but must be handled cautiously. No for statement here, but rather foreach.

```php
// use a foreach instead
$days = [0 => "Sun", 6 => "Sat"];

for ($days as $key => $value) {
  echo "$value<br>";
}
// Sun
// Mon
```


The keys don't have to be continuous when numeric. Just like when they are strings.

Therefore, gaps in the key sequence is acceptable, but must be handled cautiously. No for statement here, but rather foreach.

Checking ...

```php
$days = [0 => "Sun", 6 => "Sat"];

// If the key is not present, isset() will
// be false.
if ( isset($days[0]) ) { echo "[0] is set"; }
if ( isset($days[1]) ) { echo "[1] is set"; }

// [0] is set
```


Verifying the existence and validity of variables is huge, and so there is a lot to it. A lot of issues are of course common sense, and we shall see how to tackle them as we move through the course.

isset() is one way - check that the var is given a value other than NULL.

Sorting ...

```php
// Sorting with keys realigned
$days = [0 => "Mon", 1 => "Tue", 5 => "Fri"];

sort($days);
print_r($days);

// Array ([0] => Fri [1] => Mon [2] => Tue)


// Sort with keys retained
asort($days);
print_r($days);

// Array ([5] => Fri [0] => Mon [1] => Tue)
```

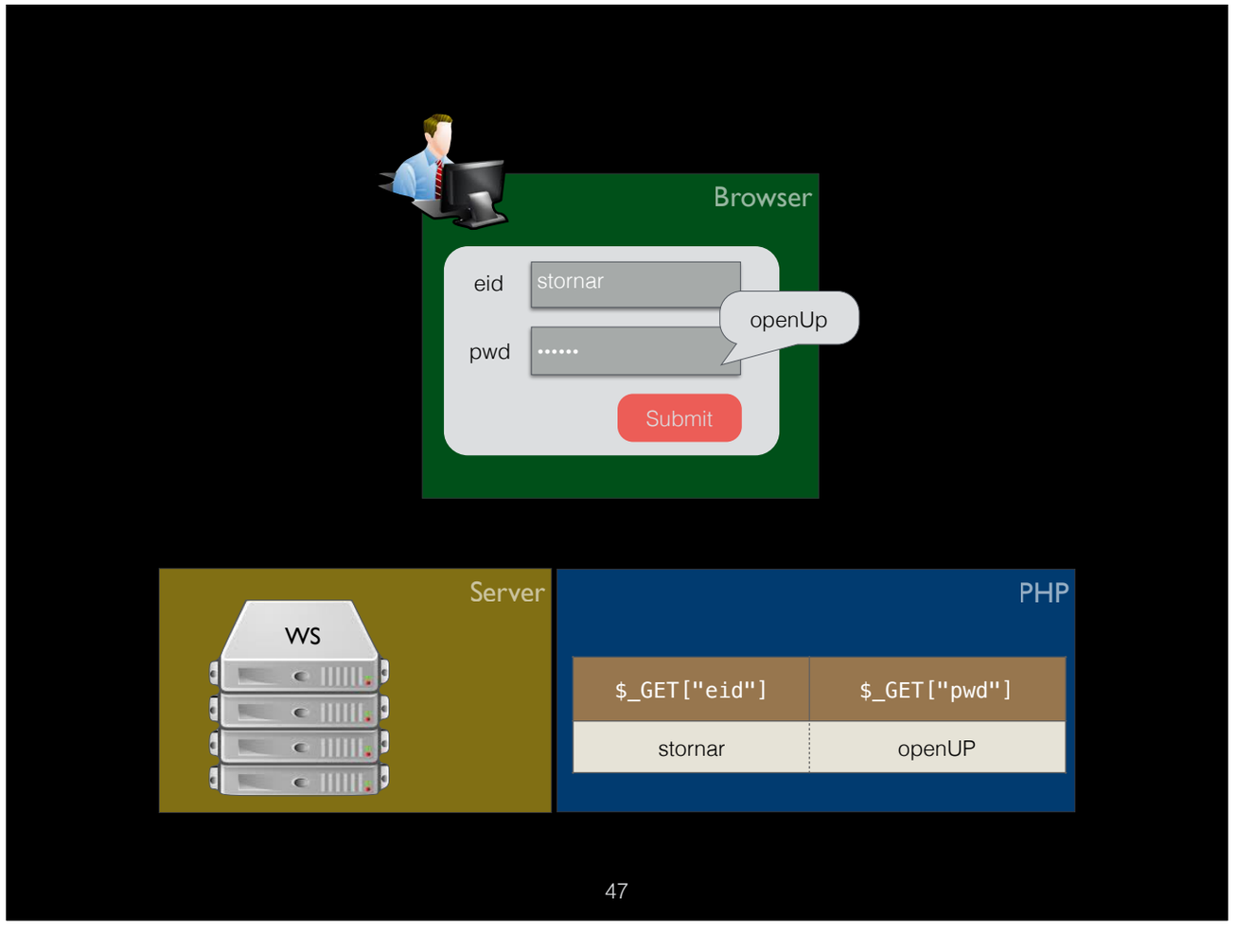

sort() - uses the values for its ordering and reindexes the values. Careful with this one if you want to maintain the original index to element relationship. You will loose it.

asort() - this one maintains the indexing, but still sorts by value.

Superglobals hold the power

`$GLOBALS`	anything goes here
`$_COOKIES`	cookies sent with request
`$_ENV`	server environment data
`$_FILES`	uploaded files
`$_GET`	query string data
`$_POST`	query string data
`$_REQUEST`	$_GET, $_POST, $_COOKIES
`$_SESSION`	session data
`$_SERVER`	request and server info

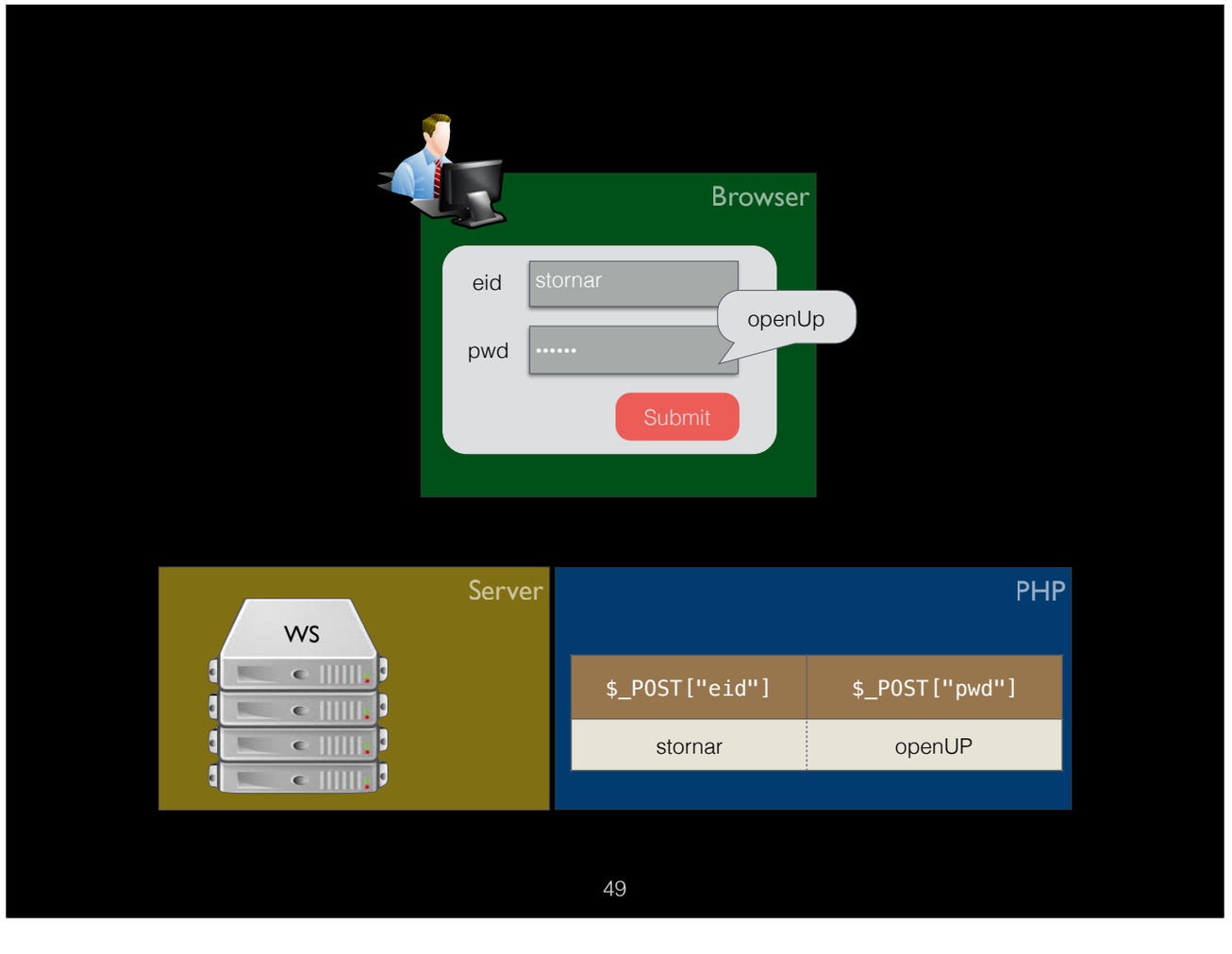
These arrays are all free for use, you do not have to declare any of them. Lots of valuable information inside these guys.


Browser
eid
stornar
pwd
••••••
openUp
Submit
Server
WS
PHP
$_GET["eid"]
$_GET["pwd"]
stornar
openUP

```
<form action="handle.php" method="GET">
  eid <input name="eid" type="text">
  pwd <input name="pwd" type="password">
  <input type="submit">
</form>
```

```
GET handle.php?eid=stornar&pwd=openUp
```


A 'GET' request uses the url to pass its query string, so it is visible for all to see. This is a security concern, so be diligent as to what you expose. No sensitive data in other words.


Browser
eid
stornar
pwd
••••••
openUp
Submit
Server
WS
PHP
$_POST["eid"]
$_POST["pwd"]
stornar
openUP

```
<form action="handle.php" method="POST">
  eid <input name="eid" type="text">
  pwd <input name="pwd" type="password">
  <input type="submit">
</form>
```

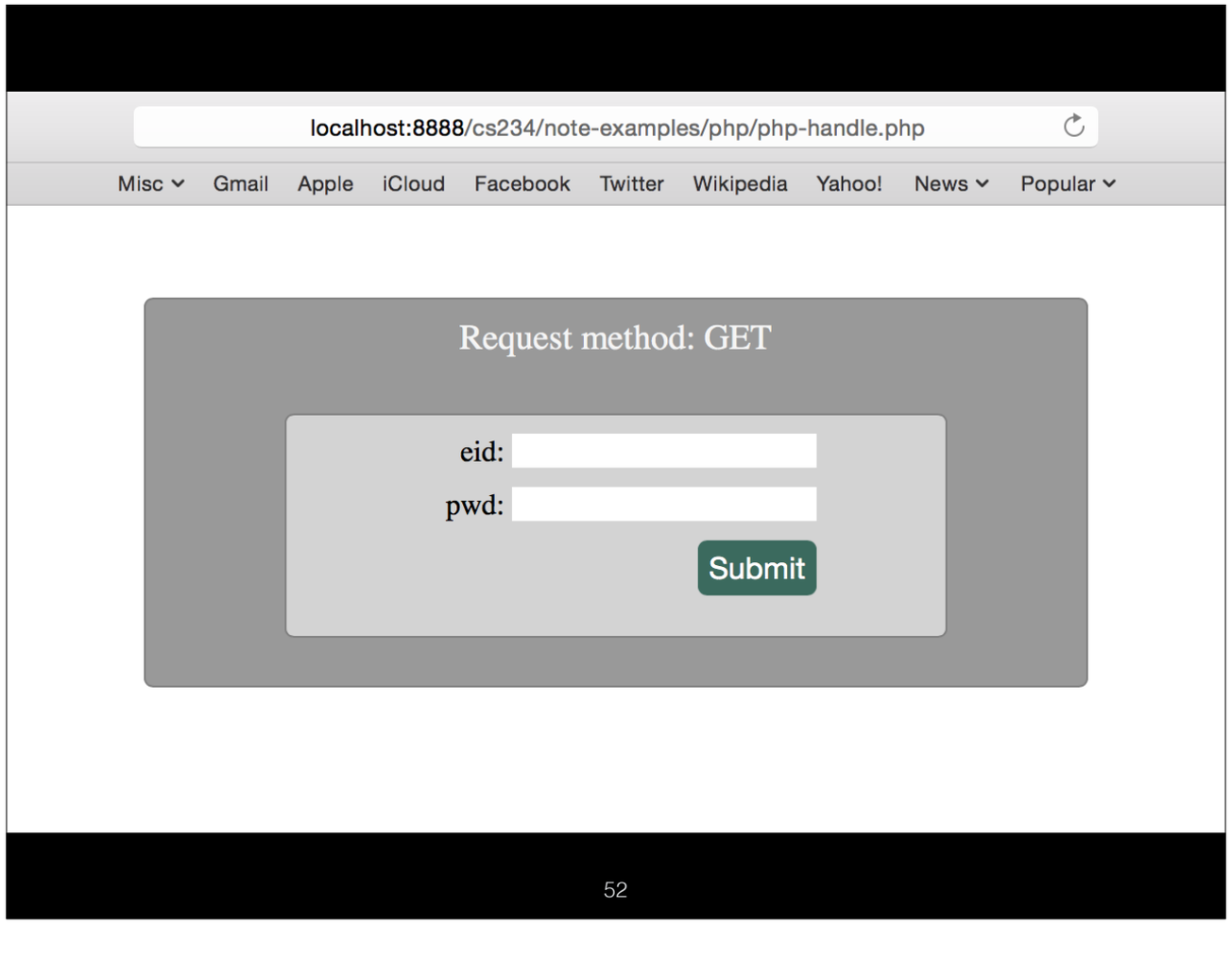
```
POST eid=stornar&pwd=openUp
```

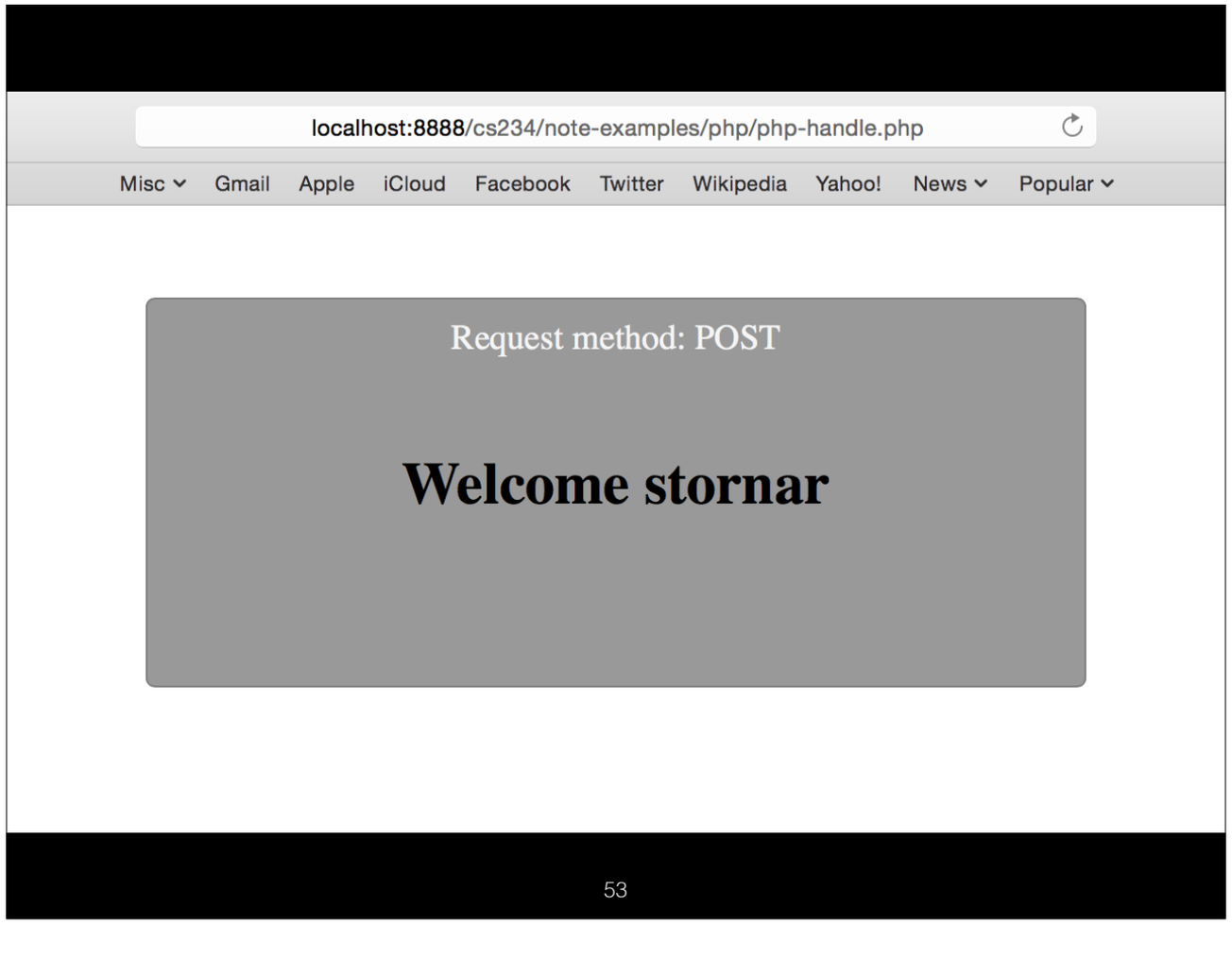

A 'POST' request uses HTTP headers for its communication, so to the untrained eye, data is more secure. More, not full proof.

Main request method for large set of data, since GET does have a character limit.

GET v POST

- GET - part of the url
  - visible, thus less secure ❌
  - limited data transfer ❌
- POST - part of HTTP headers
  - non-visible, thus more secure ✅
  - larger data transfer✅


localhost:8888/cs234/note-examples/php/php-handle.php
Misc
Gmail
Apple
iCloud
Facebook
Twitter
Wikipedia
Yahoo!
News
Popular
Request method: GET
eid:
pwd:
Submit


localhost:8888/cs234/note-examples/php/php-handle.php
Misc
Gmail
Apple
iCloud
Facebook
Twitter
Wikipedia
Yahoo!
News
Popular
Request method: POST
Welcome stornar

```
<?php /* File: php-handle.php */
      include 'php-inc.handle.php'; ?>
...
<main id="handle_page">
  <p id="request_method">
  <?php getRequestMethod(); ?></p>

  <div id="login_content">
  <!-- here we are going to use a function to
       show either a) the form or b) a welcome
       message -->
  <?php showContent(); ?>
  </div>
</main>
...
</html>
```


Notice here the two php drop-ins for the two function calls. Each one has a specific task to carry out.

Each function will generate the corresponding html that will then be inserted where those function calls are.

```php
<?php
/* File: php-inc.handle.php */

$request = $_SERVER['REQUEST_METHOD'];


function getRequestMethod() {
    global $request;
    echo "Request method: " . $request;

}// end getRequestMethod()
```


Here the php builds the html itself. Notice how a $form variable is build up with non-echo usage and then echoes it out when done.

Note the opening ' followed by the html and closed with another '.

```php
function showContent() {
  global $request;

  if ($request == "GET") { showForm(); }
  else {
    $eid = isset($_POST["eid"]);
    $pwd = isset($_POST["pwd"]);

    /* The credentials would normally be
     * validated against a db. */
    if ($eid && $pwd && $pwd == "openUp") {
      echo '<p id="welcome"><h1>Welcome ' .
           $eid . '</h1></p>';
    } else { showForm(); }
  }
}// end showContent()
```


This is primary php, with minimal html (one line basically).

Separation of the layers in other words.

```php
function showForm() {
    $form = '<div id="form_content">
              <form id="login_form"
                    action="php-handle.php"
                    method="POST">
              ...
              <p><input id="submit"
                        type="submit"></p>
              </form>
             </div>';
    echo $form;
}// end showForm()
```


Here the php builds the html itself. Notice how a $form variable is build up with non-echo usage and then echoes it out when done.

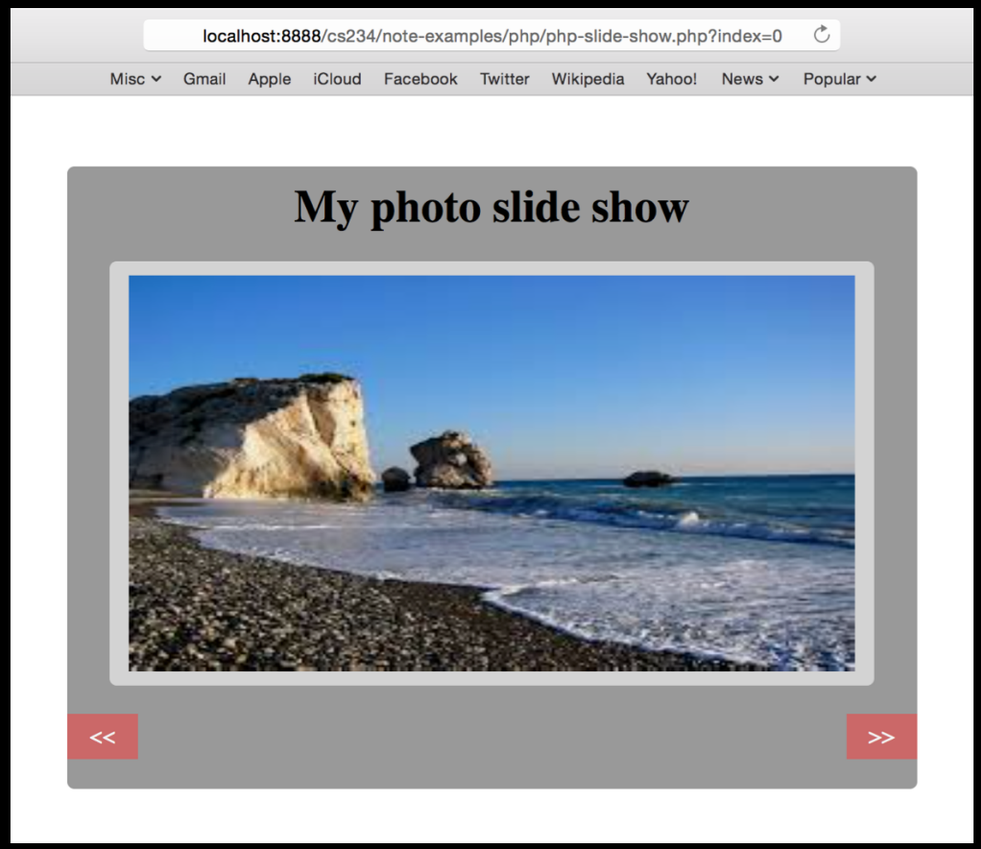
Note the opening ' followed by the html and closed with another '.

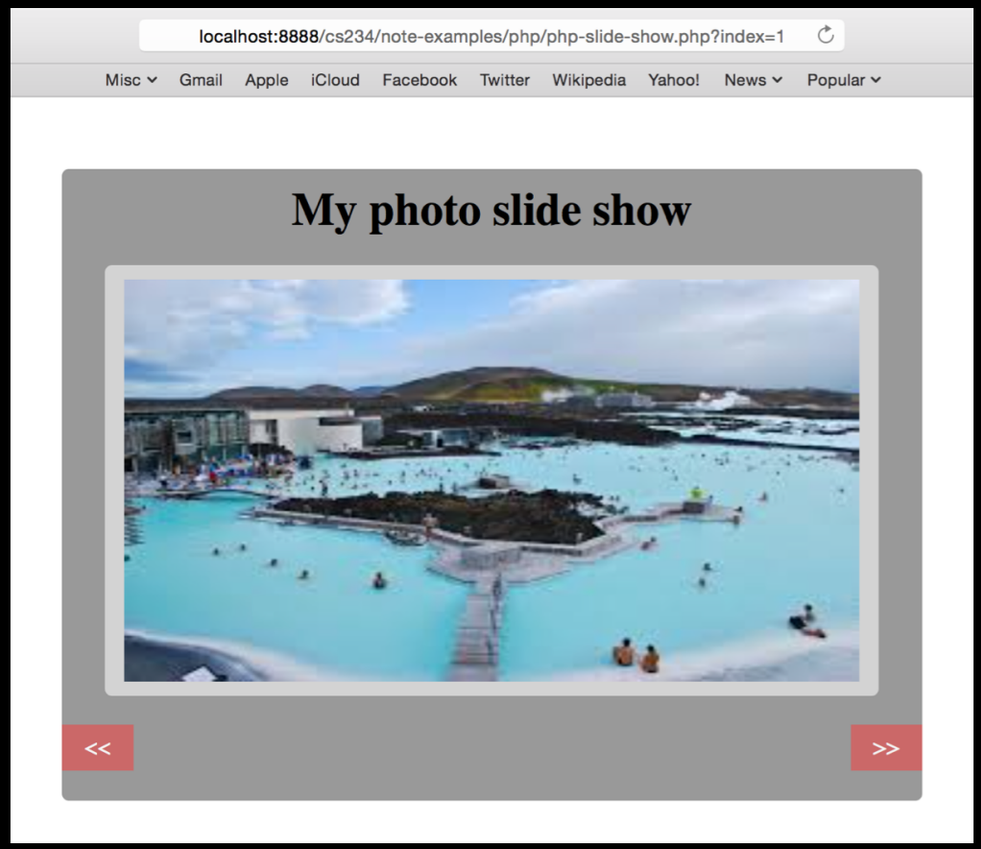
Linking is always an option

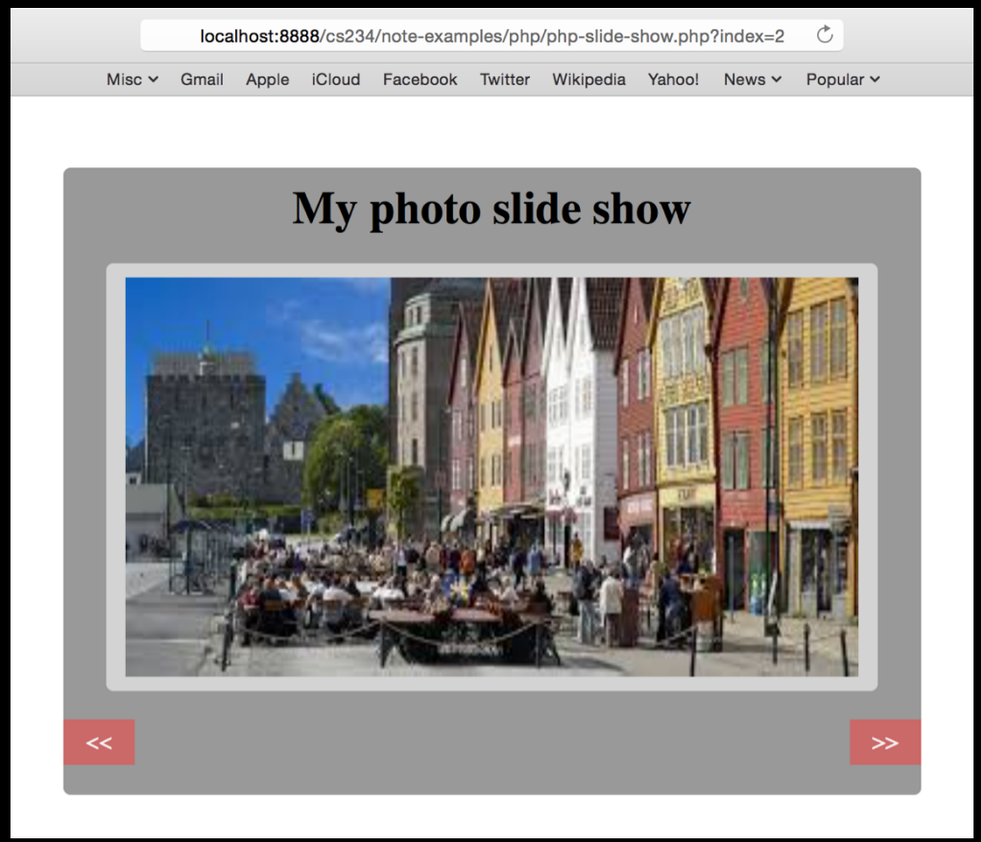
- >>

So far the browser made the initial request, the submit button the subsequent requests, but that is not all.

Links can now be used to navigate to other pages passing along their own query strings. Most often this query string will be added to the html via a php drop-in.


localhost:8888/cs234/note-examples/php/php-slide-show.php?index=0
Misc
Gmail
Apple
iCloud
Facebook
Twitter
Wikipedia
Yahoo!
News
Popular
My photo slide show
<<
>>


localhost:8888/cs234/note-examples/php/php-slide-show.php?index=1
Misc
Gmail
Apple
iCloud
Facebook
Twitter
Wikipedia
Yahoo!
News
Popular
My photo slide show
<<
>>


localhost:8888/cs234/note-examples/php/php-slide-show.php?index=2
Misc
Gmail
Apple
iCloud
Facebook
Twitter
Wikipedia
Yahoo!
News
Popular
My photo slide show
<<
>>

```
<?php /* File: php-slide-show.php */
      include 'php-inc.slide-show.php'; ?>
...
<p><h1>My photo slide show</h1></p>
<div id="content">
     <?php showImage($curIndex); ?></div>

<p id="next" class="goto_photo">
  <a href="php-slide-show.php?index=
  <?php echo nextIndex(); ?>">&gt;&gt;</a></p>

<p id="previous" class="goto_photo">
  <a href="php-slide-show.php?index=
  <?php echo previousIndex(); ?>">&lt;&lt;
  </a></p></main>
</body></html>
```


Notice the use of $curIndex in the showImage() function. Where is this variable defined?
The answer is inside the php-inc.slide-show.php file.

Recall, an include is a substitution or merging operation. The code inside that script, is available to everything in this file from that point on. It is a strange way of doing things, but it is done often, so join the club.

```php
<?php /* File: php-inc.slide-show.php */

$photos = ['0.jpg', '1.jpg', '2.jpg'];
$curIndex = $_GET['index'];

if (!isset($curIndex)) { $curIndex = 0; }

function showImage($curIndex) {
    global $photos;
    $image = '<img src="../images/' .
                $photos[$curIndex] . '">';
    echo $image;
}// end showImage()
```


Here we are setting up two global vars, $photos-an array, and $curIndex. These are used in the functions, so check the global declaration in each of them.

Careful attention to the paths. Relative to what exactly? This file (.inc) of the one including this file(the includer). Since an include is a merging operation you might think it is in relation to the includer file, but not so fast.

Let's break it down. php-slide-show.php includes php-inc.slide-show.php, but it must first be processed into html, and that then leads to the path being relative to the included file (php-inc.slide-show.php). The includer simply receives html at the end of the day.

```php
function nextIndex() {
    global $curIndex;
    return ($curIndex + 1) % 3;
}// end nextIndex()



function previousIndex() {
    global $curIndex;
    return ($curIndex + 2) % 3;
}// end previousIndex()
```


Here we are setting up two global vars, $photos-an array, and $curIndex. These are used in the functions, so check the global declaration in each of them.

Careful attention to the paths. Relative to what exactly? This file (.inc) of the one including this file(the includer). Since an include is a merging operation you might think it is in relation to the includer file, but not so fast.

Let's break it down. php-slide-show.php includes php-inc.slide-show.php, but it must first be processed into html, and that then leads to the path being relative to the included file (php-inc.slide-show.php). The includer simply receives html at the end of the day.

Sanitizing query strings

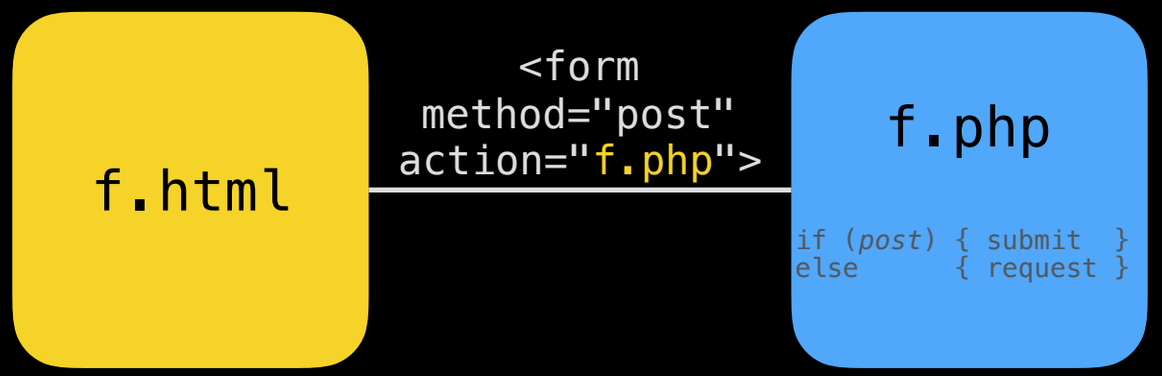
- Check your query string
  - if a parameter exists
  - if a parameter has a value
  - if a parameter is the correct type
  - if a db search value exists in a table

Sanitation is part of our defensive programming philosophy. It sounds cynical, but you should trust no one and thus develop with that in mind. Better be safe than sorry, any day.

This is a work in progress. Baby steps, but always be mindful of the need for it and its severity.

Validation strategies

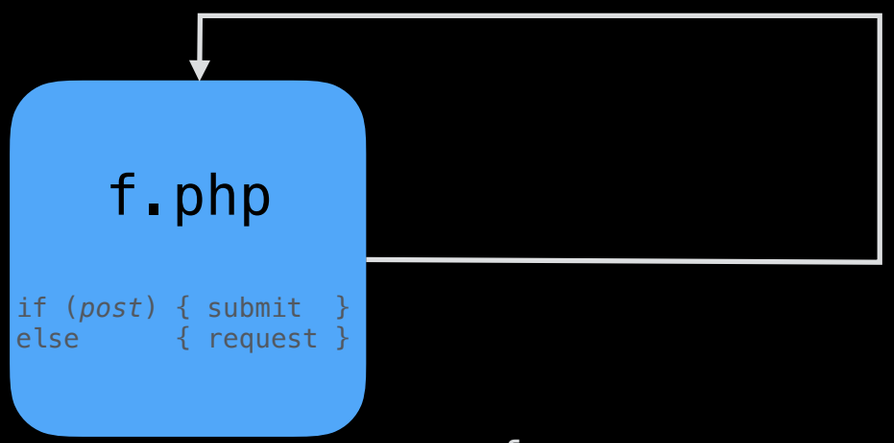
html-php



Here a .php will process the form in some way. An action will be taken, perhaps computing some values or querying a db.

If validating a form, you can redirect back to the .html, but fields will not be sticky, thus user must re-enter all values.

php-php




```
<form
method="post"
action="{$_SERVER['PHP_SELF']}">
```


A single .php file is used to show the form and validate it. This is the request/submit model, using a single file for both the initial request and the subsequent submissions.

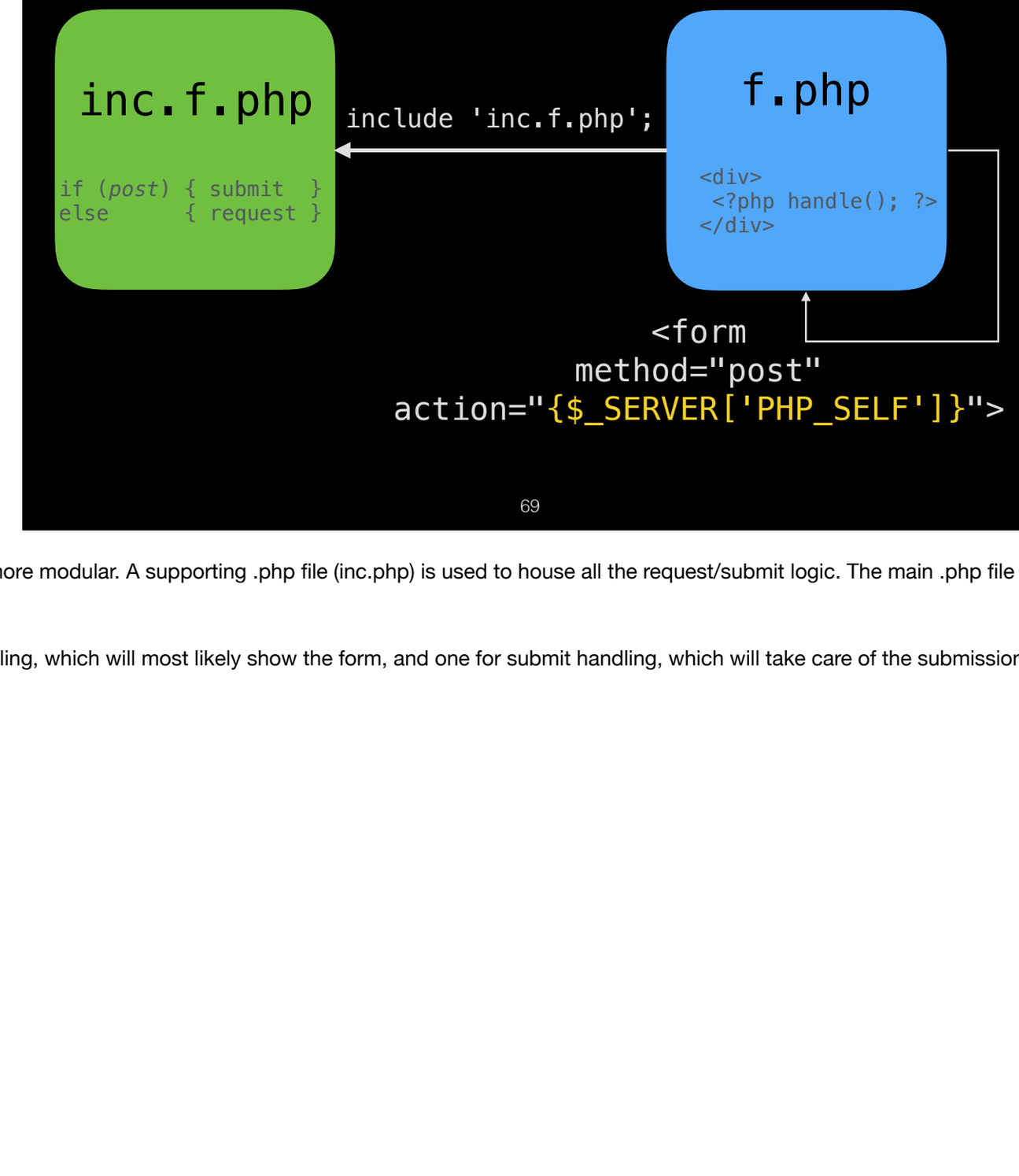
Logic can verify which task (request/submit) is handled and act accordingly. Here if you are validating a form, the fields could be made sticky (made to show the values entered by the user).

For example: <input type="text" name=username value="<?php echo $username; ?>" />

Based on the logic, you may redirect the user to another page using the header() function. Make sure however, that no echo statements are issued prior to the redirection. For example,

header("Location: http://somewhere.html");

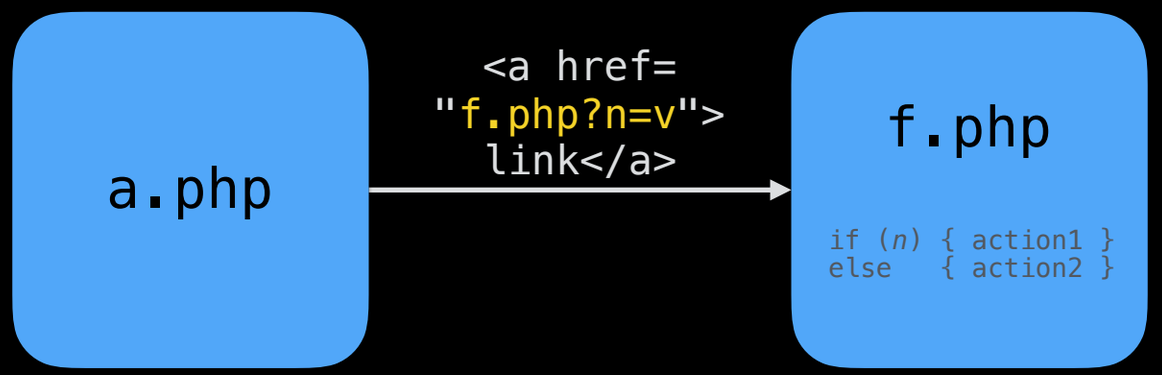
php-inc.php



This is similar to the previous set-up but a bit more modular. A supporting .php file (inc.php) is used to house all the request/submit logic. The main .php file simply makes calls to appropriate functions within the .inc file.

The .inc file will have a branch for request handling, which will most likely show the form, and one for submit handling, which will take care of the submission.

php-php



This technique was highlighted earlier, and allows for page to communicate with another by providing relevant data through the query string.

File IO

File IO is very similar in concept to other languages that use streams.

You have a choice of reading through a stream piece by piece or all at once (meaning the entire file in one felt swoop).

Stream

- Appropriate for large files
- Uses normal stream techniques

When writing files and even when reading them, one has to be aware of the line delimiters used by the different OSs.

Windows: \r\n
Linux, Unix: \n
Mac OS: \r

```php
// File in same dir as script
$file = fopen("test.txt", "w");
for ($i = 1; $i <= 10; $i++) {
  fwrite($file, $i . "\n");
}
fclose($file);


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

All-in-memory

- Appropriate for small files
- Entire file is real as one

```php
/* Read entire file into an array, one
 * record per element.
 */
$fileAsArray = file($file) or die('Error!');

/* reads entire file into a string
 * variable.
 */
$fileAsString = file_get_contents($file);


/* reads entire file into a string
 * variable.
 */
file_put_contents($file, $string);
```


These three beauties to the all-in-one reading that is so convenient.

file() puts contents into an array, while file_get_contents() puts them in a single string.

Even more good stuff

- 😶😶 PHP
- PHP classes
- PHP recap

OO PHP

- encapsulation
- inheritance (single)
- polymorphism
- interfaces

- Objects exist for the duration of the script
  - Build-up and tear-down constantly
- Serializing objects can offer persistence b/w requests

PHP Class

- Saved in its own file
- Named class.className.php
- Included when needed
- Instantiated with new

- Cannot overload methods
- May include static members, accessed with
  - `self::member` from within class
  - `classname::member` from outside class

- May include constants
  - `const MAX_COUNT = 31`
  - `self::`MAX_COUNT (w/in),
  - `classname::`MAX_COUNT (w/out)

Define the class

```php
class Artist {
  public $firstName;
  public $lastName;
  public $birthDate;
  public $birthCity;
  public $deathDate;
}
```

member access can be **public**, **protected** or **private**.

Yes, that is a $ sign in front of the property names. Don't forget it.

Instantiate the class

```php
$dali = new Artist();
$picasso = new Artist();
$leonardo = new Artist();
```

Access the class

```php
$picasso->firstName = "Pablo";
$picasso->lastName = "Picasso";
$picasso->birthCity = "Malaga";
$picasso->birthDate = "October 25, 1881";
$picasso->deathDate = "April 8, 1973";
```


When accessing the properties, notice that there is no $ sign this time around.

ctors offer flexibility

```php
class Artist {
  // ivars here - no change


function __construct($f, $l, $c,
                     $b, $d = null) {
    // this is a magic function
    $this->firstName = $f;
    $this->lastName = $l;
    $this->birthCity = $c;
    $this->birthDate = $b;
    $this->deathDate = $d;
  }
}
```


$this-> is the way to access members of a class. It is actually a requirement, not an option.

Also, notice the optional deathDate ($d) parameter, which is set to null.

during instantiation

```
$picasso = new Artist("Pablo",
                      "Picasso",
                      "Malaga",
                      "October 25, 1881",
                      "April 8, 1973");
```


$this-> is the way to access members of a class. It is actually a requirement, not an option.

Also, notice the optional deathDate ($d) parameter, which is set to null.

Additional magic functions

`__construct() / __destruct()`

`__call() / __callStatic()`

`__get() / __set()`

`__isset() / __unset()`

`__sleep() / __wakeup()`

```
__toString()
__invoke()
__set_state()
__clone()
__autoload()
```

Naming conventions

- Capitalize class names: `Author`
- Use underscores: `International_Order`
- Use camel case for members: `name`, `getName()`
- Use _ prefix for private properties: `$_private`
- Use all caps for constants: `DB_NAME`

Inheritance already?

```php
// Use the extends keyword
class Subclass extends Superclass {}



// Use parent:: to access superclass
parent::someNonPrivateMember



// Call super's ctor pronto!
function __construct() {
  parent::__construct();
  ...
}
```

override means Polymorphic

Interfaces

```php
// Use the interface keyword
interface IPayable {}


// Consists only of methods
interface IPayable {
  public function pay();
}


// Use the implements keyword with a class
class Employee implements Payable {}
```

white space

- PHP is white space insensitive (put code anywhere)
- HTML considers only one space
- PHP writes to html, so
  - each single-line echo writes to html at cursor
  - each multi-line echo writes to html on a new line
  - use of "\n" in code writes to new line in html

comments

- #, // for single-line
- /* */ for multi-line

variables

- Scalar: Boolean, Integer, Float, String
- Non-scalar: Array, Object
- NULL
- begin with $
- case sensitive
- interpolated within ""

constants

- `define ('PI', 3.14);`
- `echo PI;`

form validation

- empty()
  - true if value does not exist or is false
- isset()
  - true if value not NULL

arrays

- indexed
  - `echo $states[0];`
- associative
  - `echo $states['IL'];`
  - `echo "IL is { $states['IL'] }";`

When using an associative array and you want to echo within " ", use { } to surround the array access.

creating arrays

- []
  - `$days = [1,2,3];`
  - `$days = ['mon' => 1, 'tue' => 2];`
- array()
  - `$days = array(1,2,3);`
  - `$days = array('mon' => 1, 'tue' => 2);`

creating arrays

- range()
  - `$days = range(1, 3);`
    `// (1,2,3)`

iterating arrays

- for
  - `for (init; condition; step) {}`
- foreach
  - `foreach (array as value) {}`
  - `foreach (array as key => value) {}`