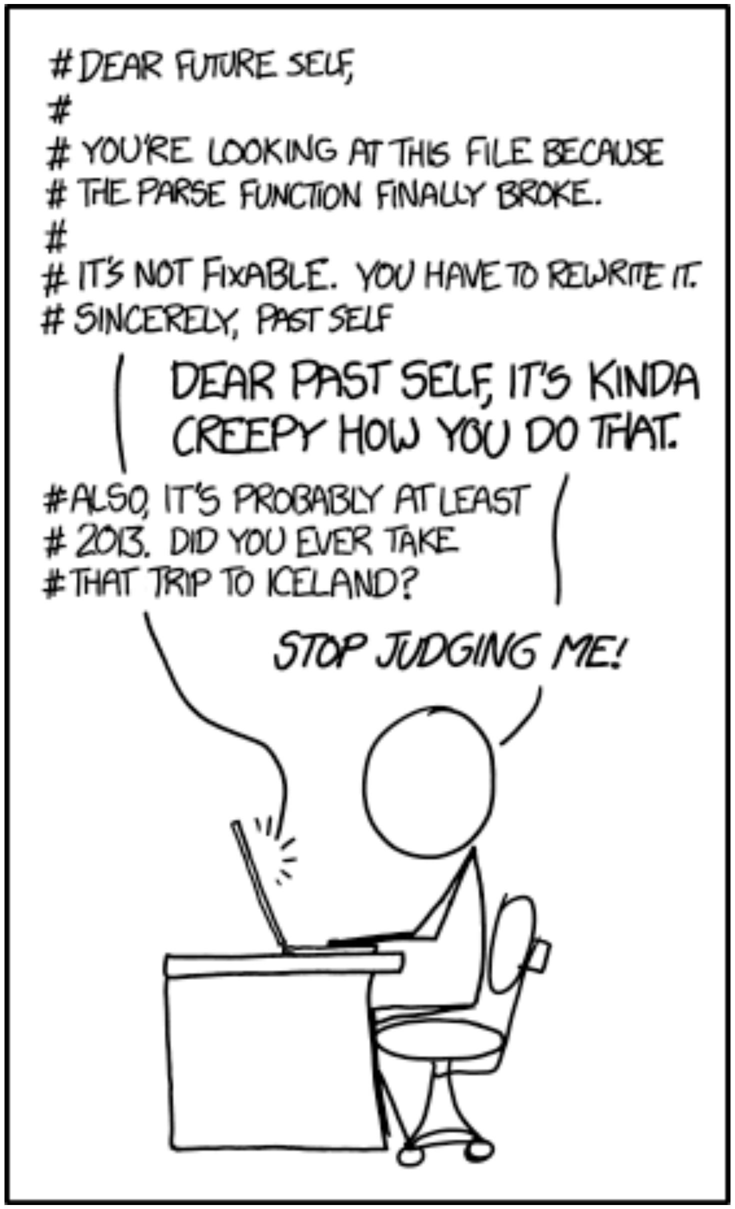
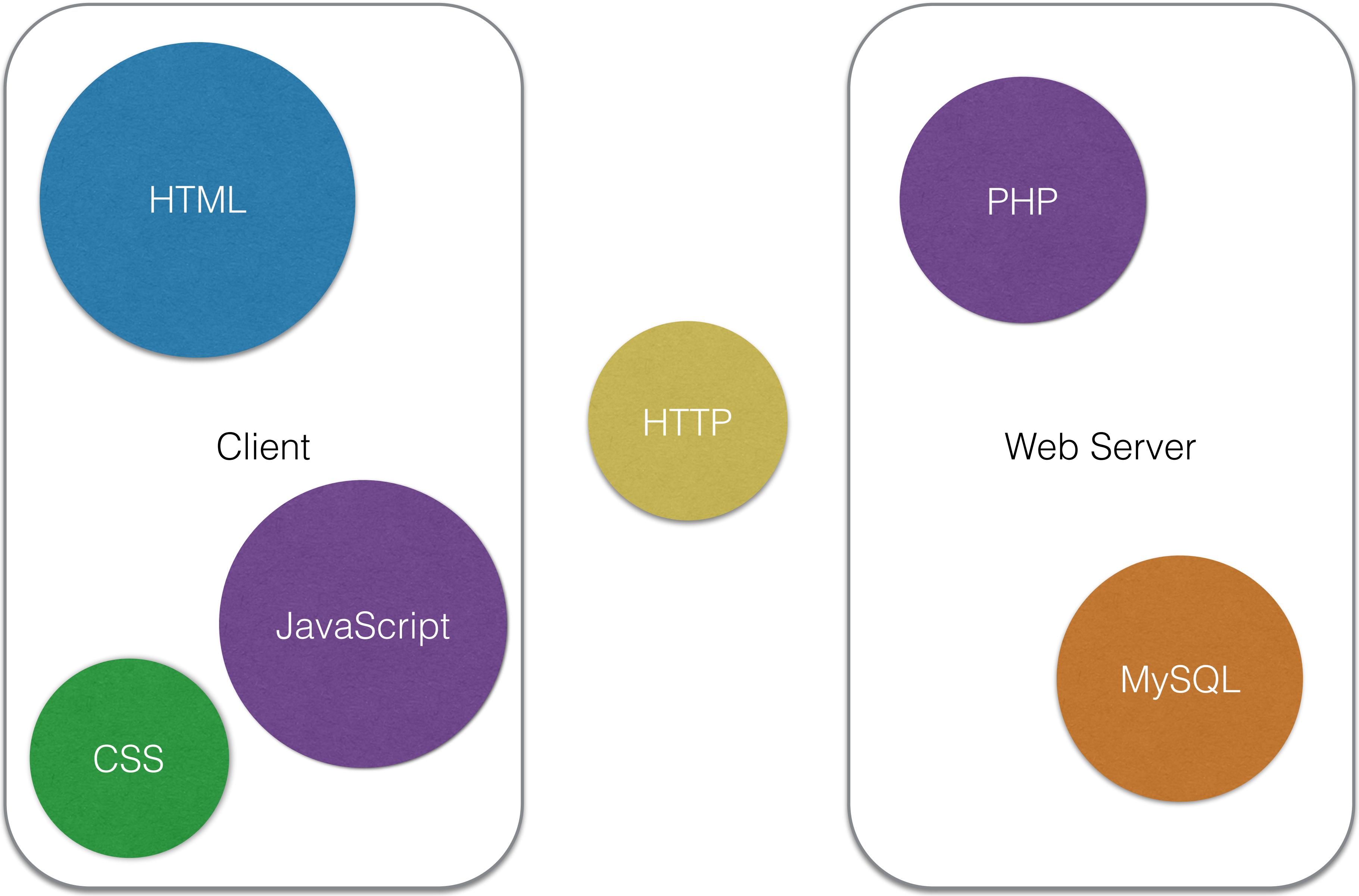
PHP


#DEAR FUTURE SELF,
#
# YOU'RE LOOKING AT THIS FILE BECAUSE
# THE PARSE FUNCTION FINALLY BROKE.
#
# IT'S NOT FIXABLE. YOU HAVE TO REWRITE IT.
# SINCERELY, PAST SELF
DEAR PAST SELF, IT'S KINDA CREEPY HOW YOU DO THAT.
#ALSO, IT'S PROBABLY AT LEAST
# 2013. DID YOU EVER TAKE
#THAT TRIP TO ICELAND?
STOP JUDGING ME!

http://xkcd.com/1421

PHP

- Personal Home Page
- PHP Hypertext Preprocessor
- Pretty Horrible Programming Language


HTML
Client
JavaScript
CSS
HTTP
PHP
Web Server
MySQL

PHP

"There are only two kinds of languages: the ones people complain about and the ones nobody uses"

—Bjarne Stroustrup (the creator of C++)

http://www.stroustrup.com/bs_faq.html#really-say-that

- PHP gets a lot of hate, but it is an easy to approach language that is the basis for a lot of very successful projects.

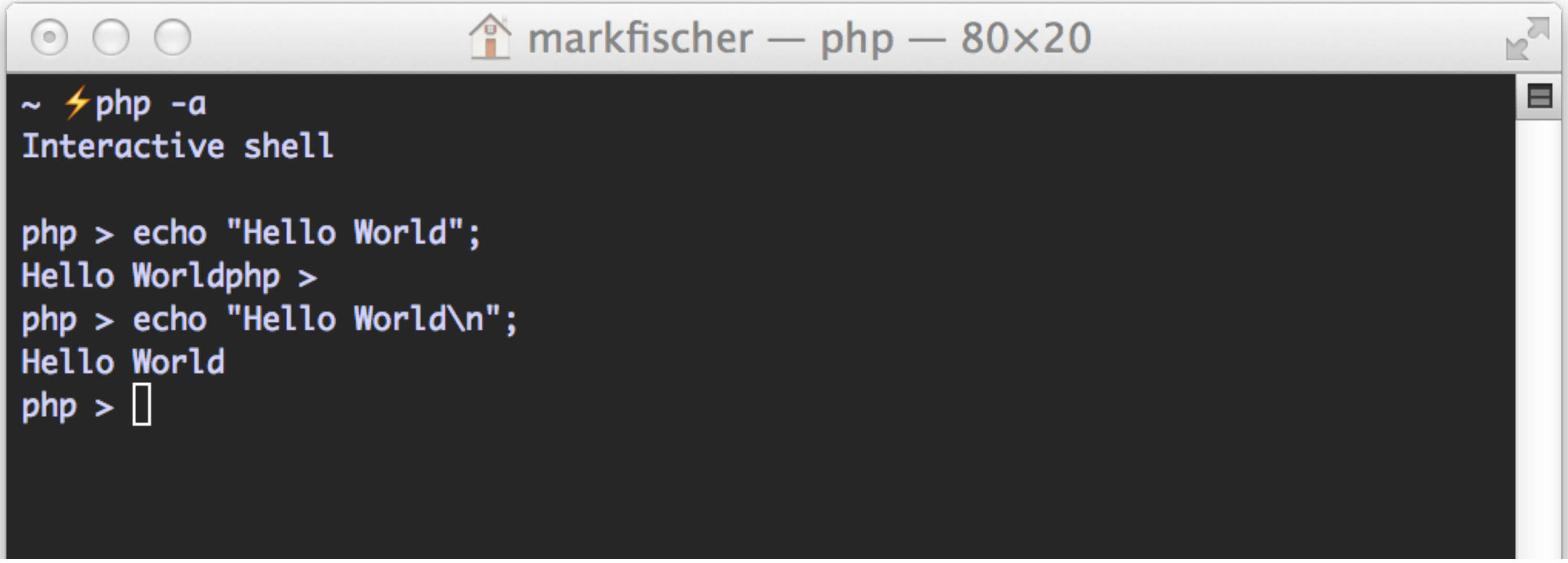
PHP: History

- 1994 - Rasmus Lerdorf wrote a series of Common Gateway Interface (CGI) binaries in C to maintain his homepage.
- 1995 - Lerdorf released “PHP Tools 1.0”
- 1997 - Zeev Suraski and Andi Gutmans rewrote the parser which formed the basis for PHP 3.
- 2000 - PHP 4 released
- 2004 - PHP 5 released, adding true objects, and an improved PHP Standard Library
- PHP 5.6 - 2014 We'll be working on this version
- PHP 7 - Just released December 2015

http://en.wikipedia.org/wiki/PHP

PHP Basics

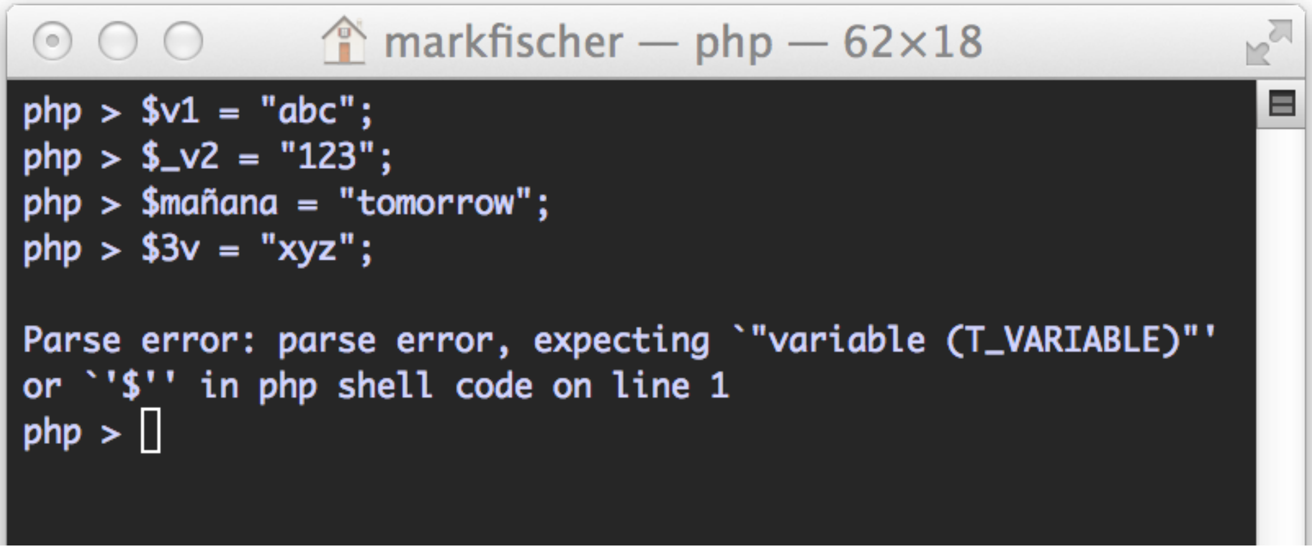
- PHP has a REPL too
  - php -a
  - Except it doesn’t work on windows…



```
~ ⚡php -a
Interactive shell

php > echo "Hello World";
Hello Worldphp >
php > echo "Hello World\n";
Hello World
php >
```

Variables

- All PHP variables are prefixed with a dollar sign: `$`
- Variable names must start with a letter or an underscore.
- Variable names can consist of letters, numbers, underscores, and the bytes 127 through 255.



```
php > $v1 = "abc";
php > $_v2 = "123";
php > $mañana = "tomorrow";
php > $3v = "xyz";

Parse error: parse error, expecting `"variable (T_VARIABLE)"'
or `'$'' in php shell code on line 1
php > 
```

Variables

- Like Javascript, variables in PHP are *not typed*.
- This doesn’t mean there are no types in PHP, it just means that a particular named variable is not tied to any one data type.

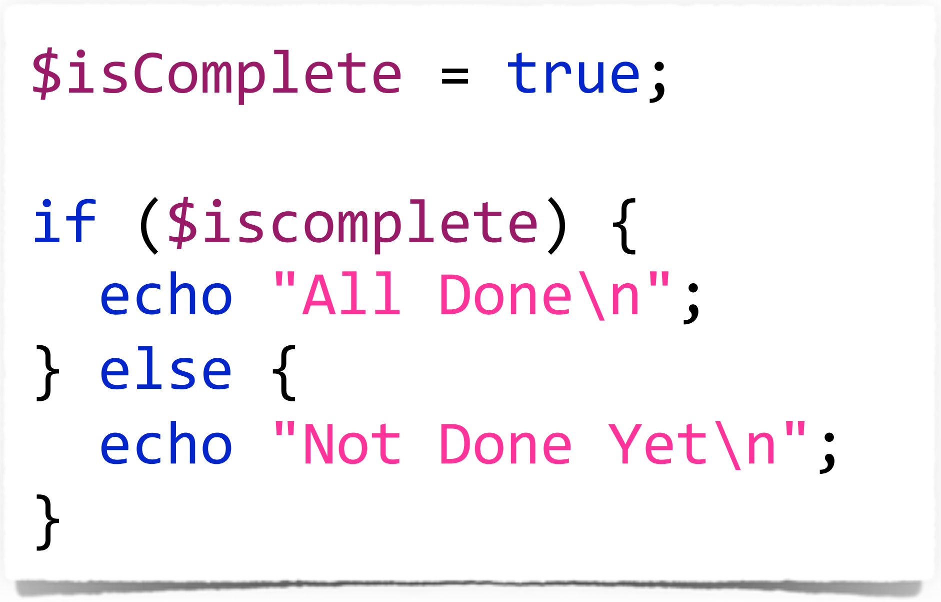
Type Checking

- Slight aside… Type Checking
- Instead of thinking about "Strongly Typed" or "Untyped" languages, think about *when* type checking is performed.

	Compile Time	**Run Time**
`C`	**Only**	None
`Java`	**Yes**	**Yes**
`PHP`	None	**Only**
`Python`	None	**Only**

Variables

- Variable names are case-sensitive. `$foo` and `$FOO` are different variables.
- Variables do not need to be declared. They spring magically into existence wherever they're needed.
- This can be a good thing, and a bad thing.

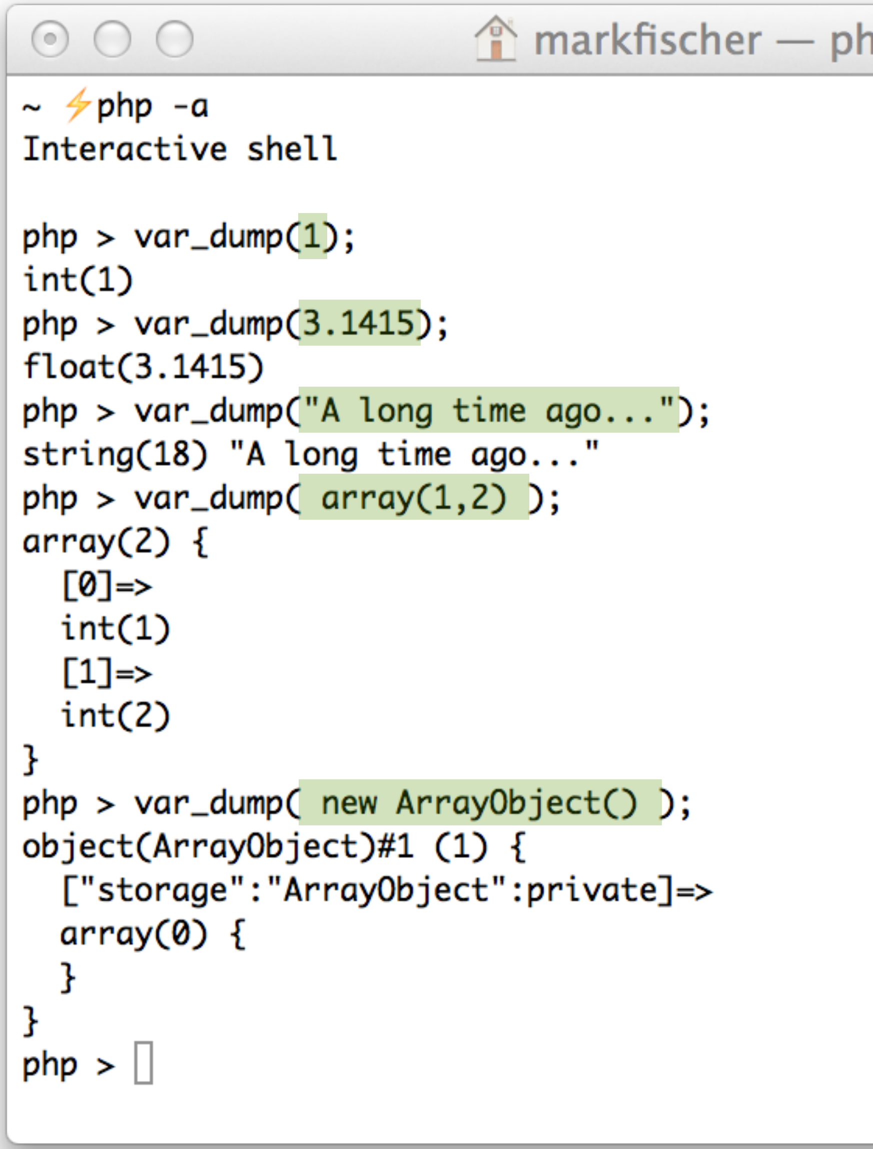


```
$isComplete = true;

if ($iscomplete) {
  echo "All Done\n";
} else {
  echo "Not Done Yet\n";
}
```

var_dump()

- What's in a variable?
- var_dump will show you the type and contents of any variable.
- Prints its output directly to STDOUT

```
php > var_dump(3.1415);
float(3.1415)
```



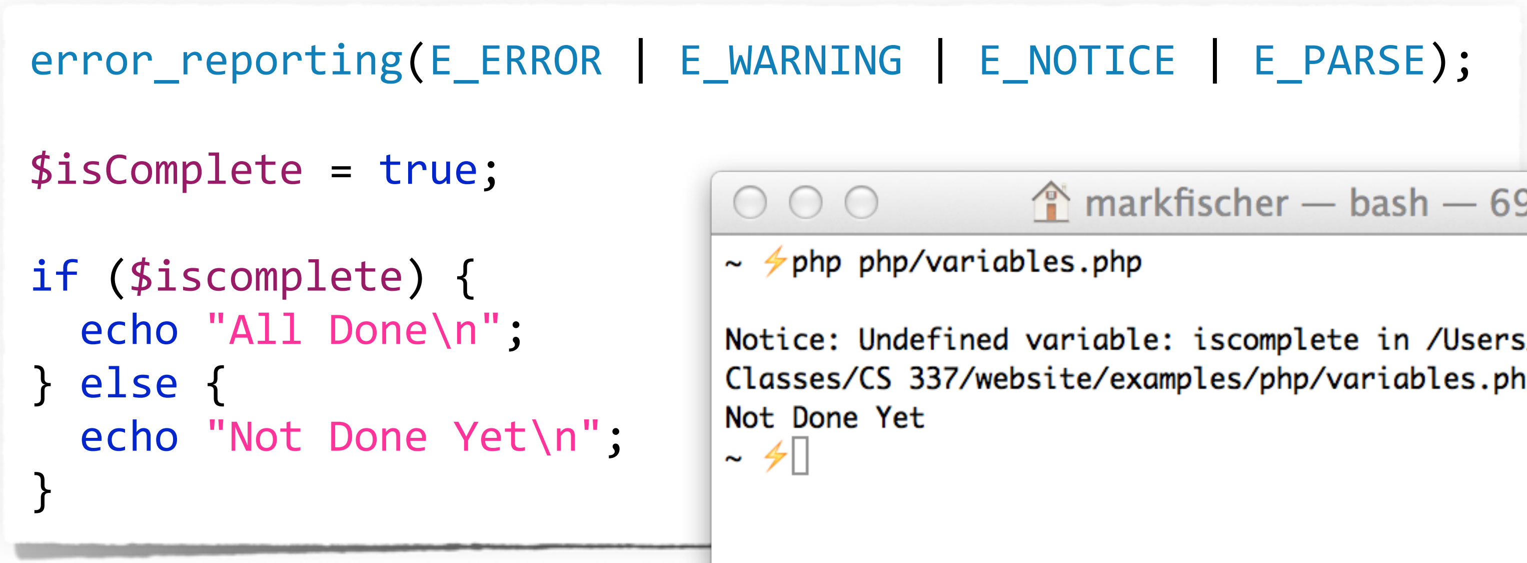
```
~ ⚡php -a
Interactive shell

php > var_dump(1);
int(1)
php > var_dump(3.1415);
float(3.1415)
php > var_dump("A long time ago...");
string(18) "A long time ago..."
php > var_dump( array(1,2) );
array(2) {
  [0]=>
  int(1)
  [1]=>
  int(2)
}
php > var_dump( new ArrayObject() );
object(ArrayObject)#1 (1) {
  ["storage":"ArrayObject":private]=>
  array(0) {
  }
}
php > 
```

Error Reporting

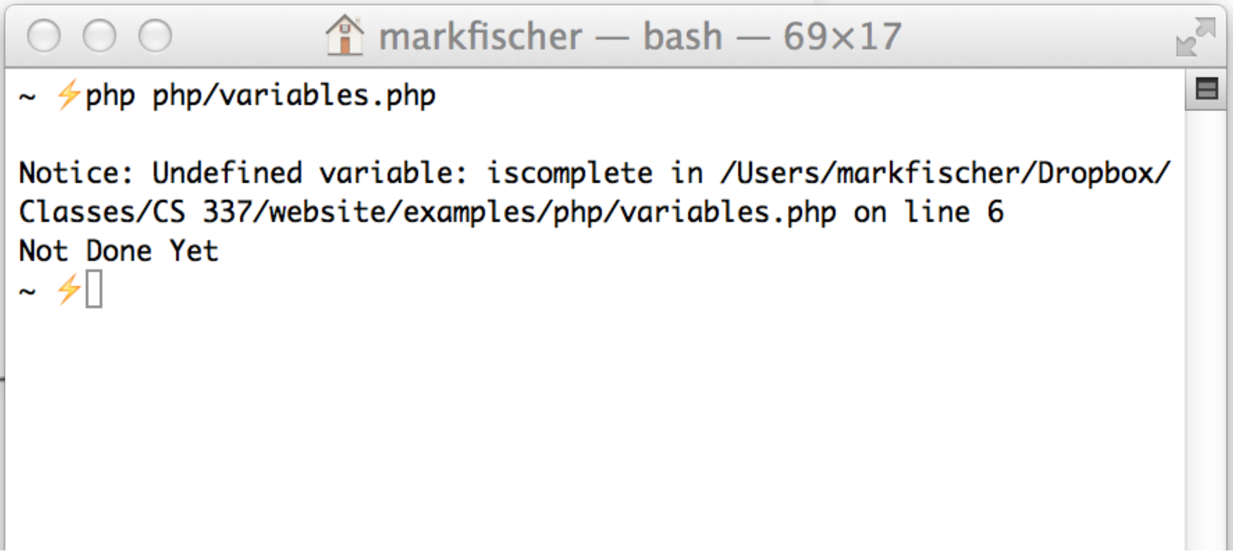
- You can change the level of error reporting.
- Config file, or at run time.
- Using `error_reporting( … )` at runtime

Error Reporting

- Setting the error reporting level down to E_NOTICE can be very useful during development.
- Incredibly spammy in production!



```php
error_reporting(E_ERROR | E_WARNING | E_NOTICE | E_PARSE);

$isComplete = true;

if ($iscomplete) {
  echo "All Done\n";
} else {
  echo "Not Done Yet\n";
}
```



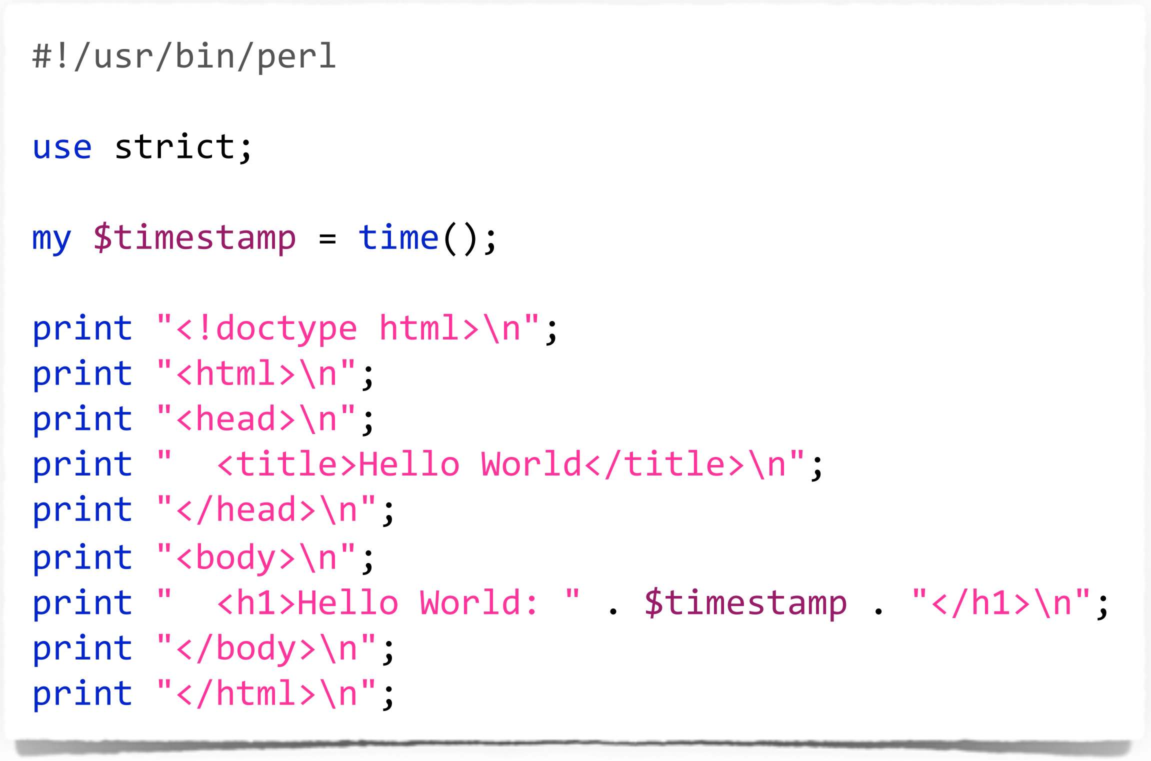
```
~ ⚡php php/variables.php

Notice: Undefined variable: iscomplete in /Users/markfischer/Dropbox/
Classes/CS 337/website/examples/php/variables.php on line 6
Not Done Yet
~ ⚡
```

PHP Structure

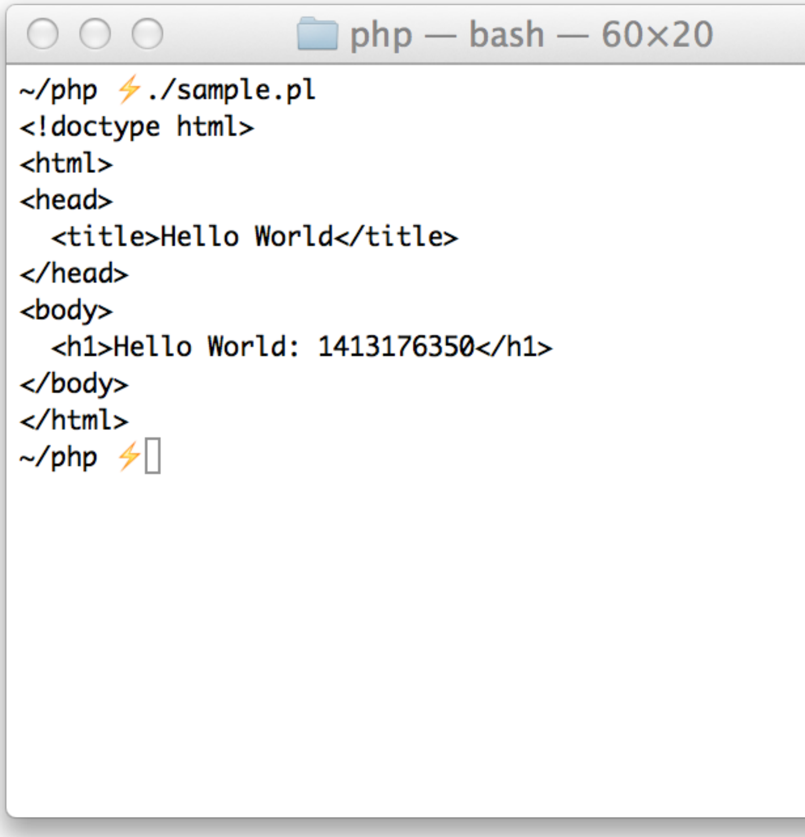
- PHP is sort of like the inverse of most languages when it comes to what gets output.
- Most languages have special features for printing things to the screen (or browser), and everything else is code.
- PHP has special features for defining where the code is, and everything else is output!

PHP Structure

- A Perl program and its output



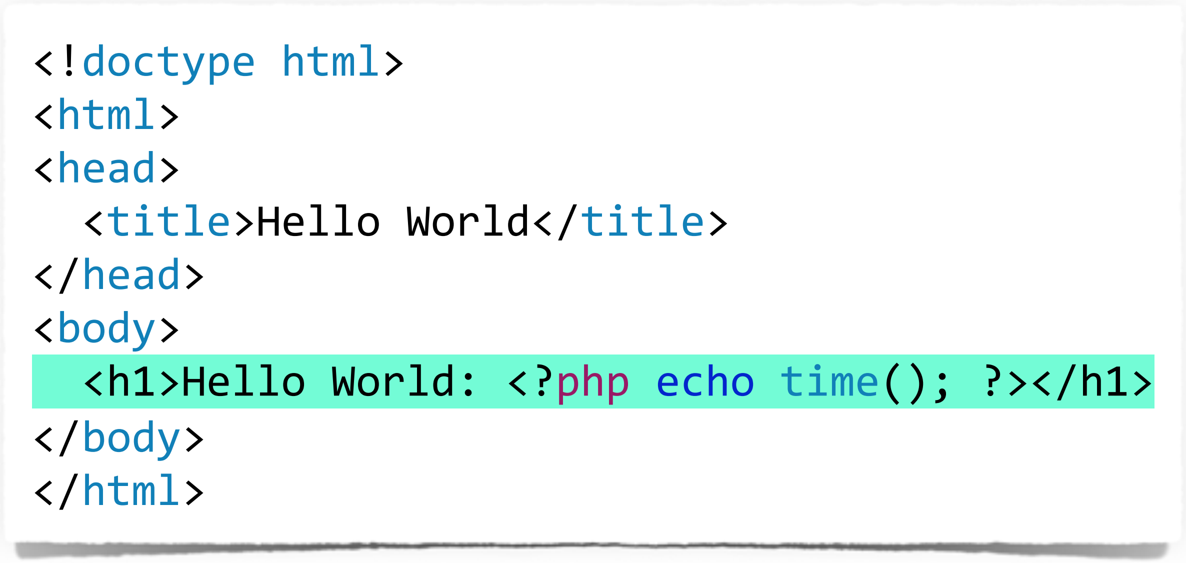
```perl
#!/usr/bin/perl

use strict;

my $timestamp = time();

print "<!doctype html>\n";
print "<html>\n";
print "<head>\n";
print "  <title>Hello World</title>\n";
print "</head>\n";
print "<body>\n";
print "  <h1>Hello World: " . $timestamp . "</h1>\n";
print "</body>\n";
print "</html>\n";
```



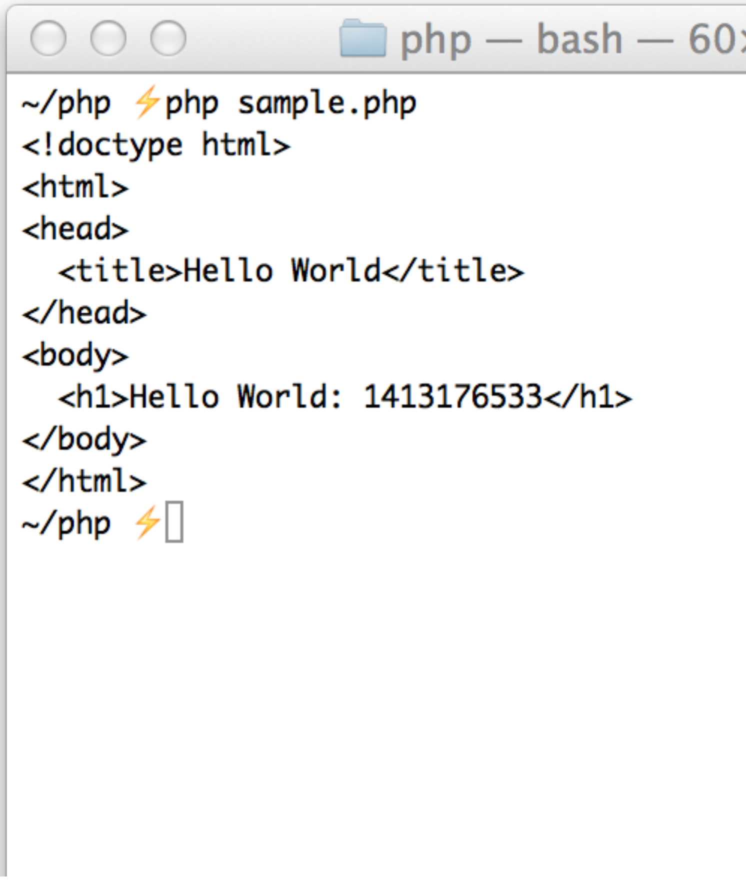
```
~/php ⚡./sample.pl
<!doctype html>
<html>
<head>
  <title>Hello World</title>
</head>
<body>
  <h1>Hello World: 1413176350</h1>
</body>
</html>
~/php ⚡
```

PHP Structure

- A PHP program and its output



```
<!doctype html>
<html>
<head>
  <title>Hello World</title>
</head>
<body>
  <h1>Hello World: <?php echo time(); ?></h1>
</body>
</html>
```



```
~/php ⚡php sample.php
<!doctype html>
<html>
<head>
  <title>Hello World</title>
</head>
<body>
  <h1>Hello World: 1413176533</h1>
</body>
</html>
~/php ⚡
```

PHP Structure

- The PHP parsing engine only executes code the follows a `<?php` sequence.
- The closing portion `?>` is required to stop parsing of PHP code
- The End of File (EOF) is treated the same as a closing `?>`

PHP Structure

- PHP Web Pages typically begin with HTML and have blocks of PHP code interspersed within it.
- PHP Code Files typically begin with an opening `<?php` tag right on the first line of the file, and then have no closing `?>` tag, leaving the EOF to close the PHP code.
    - This prevents stray characters outside of the `<?php // code ?>` blocks from being sent as output

PHP Structure

```
<!doctype html>
<html>
<head>
  <title>Hello World</title>
</head>
<body>
  <h1>Hello World: <?php echo time(); ?></h1>
</body>
</html>
```

Web Page

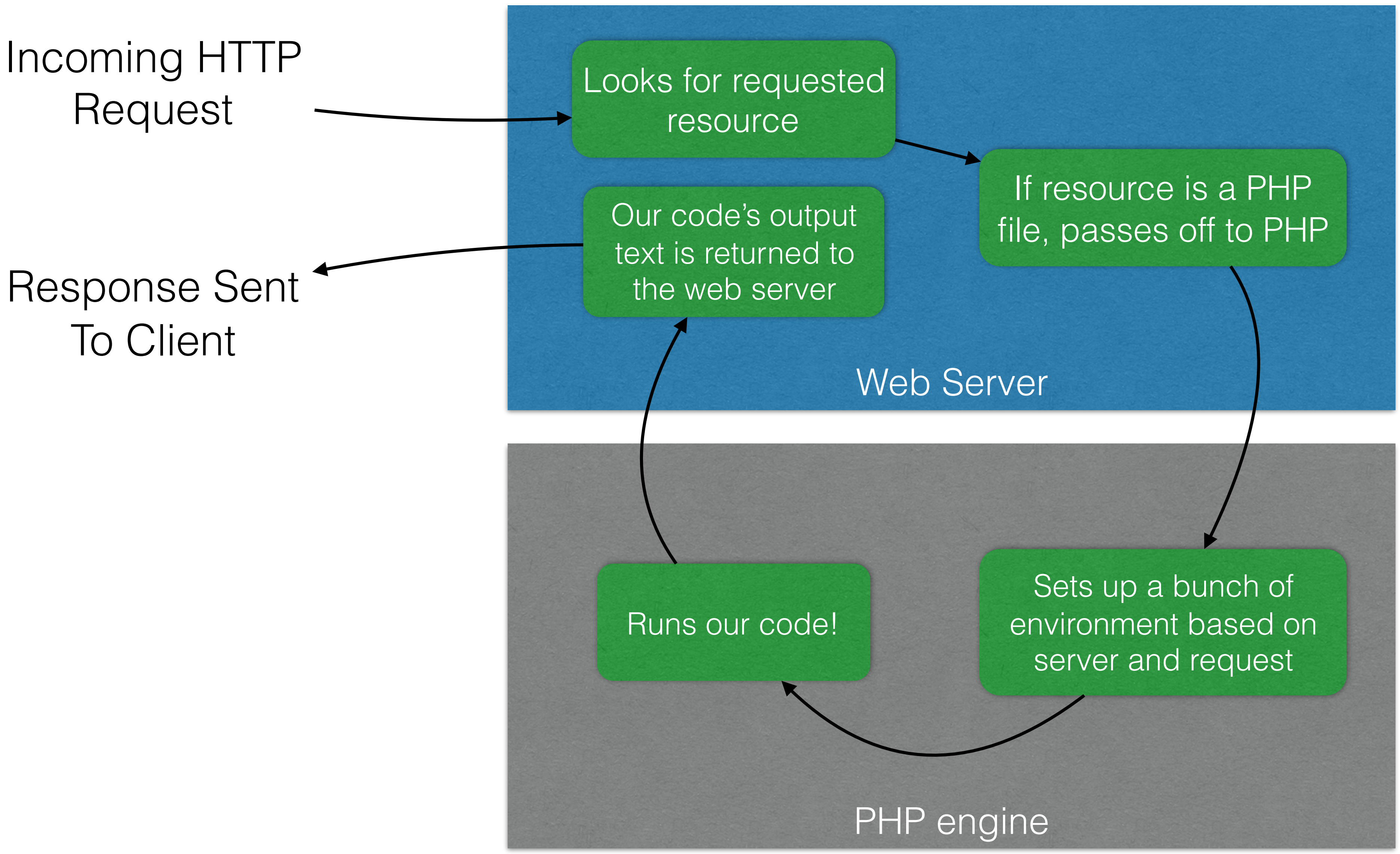
```
<?php
/**
 * Sample PHP Class
 * filename: php/structure1.php
 */
class first {
  private $foo;
  private $bar;

  public __construct() {
    $this->foo = "4";
    $this->bar = "2";
  }

  public answer() {
    return $this->foo . $this->bar;
  }
}
```

Pure Code File

Web Servers and PHP

- The Web Server does a lot before PHP ever gets invoked.
- PHP does a lot of setup work before our code gets invoked

Web Servers and PHP


Incoming HTTP Request
Looks for requested resource
If resource is a PHP file, passes off to PHP
Sets up a bunch of environment based on server and request
Runs our code!
Our code's output text is returned to the web server
Response Sent To Client
Web Server
PHP engine

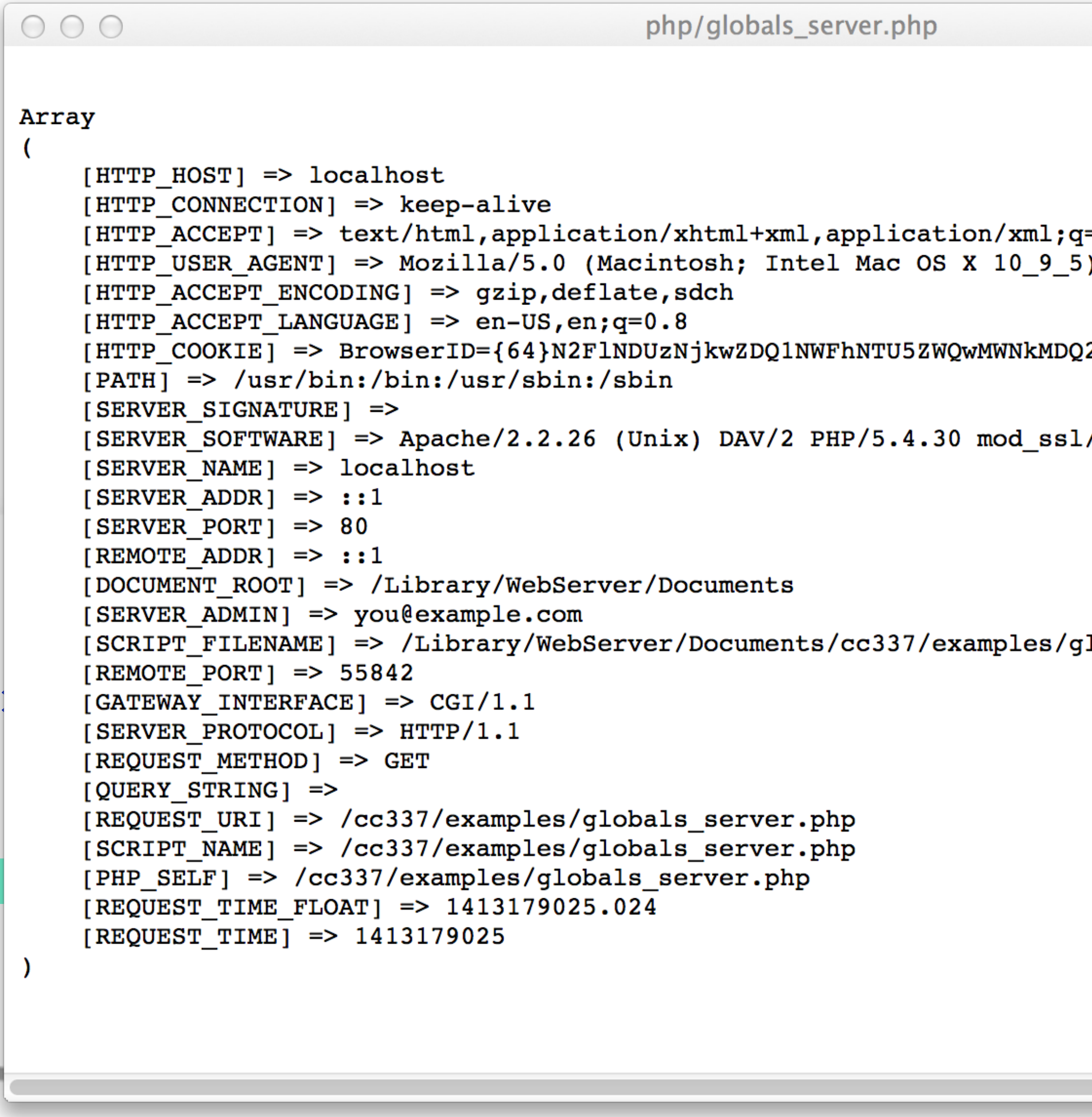
Web Servers and PHP

- What’s in all that setup that the web server and PHP does before we ever get to our code?
- The Web Server may re-write the request path, add additional information, etc.
- PHP creates a set of “Super Global” variables which we have access to.

$_SERVER

- The $_SERVER superglobal contains a bunch of information about the request, the server, and our environment.

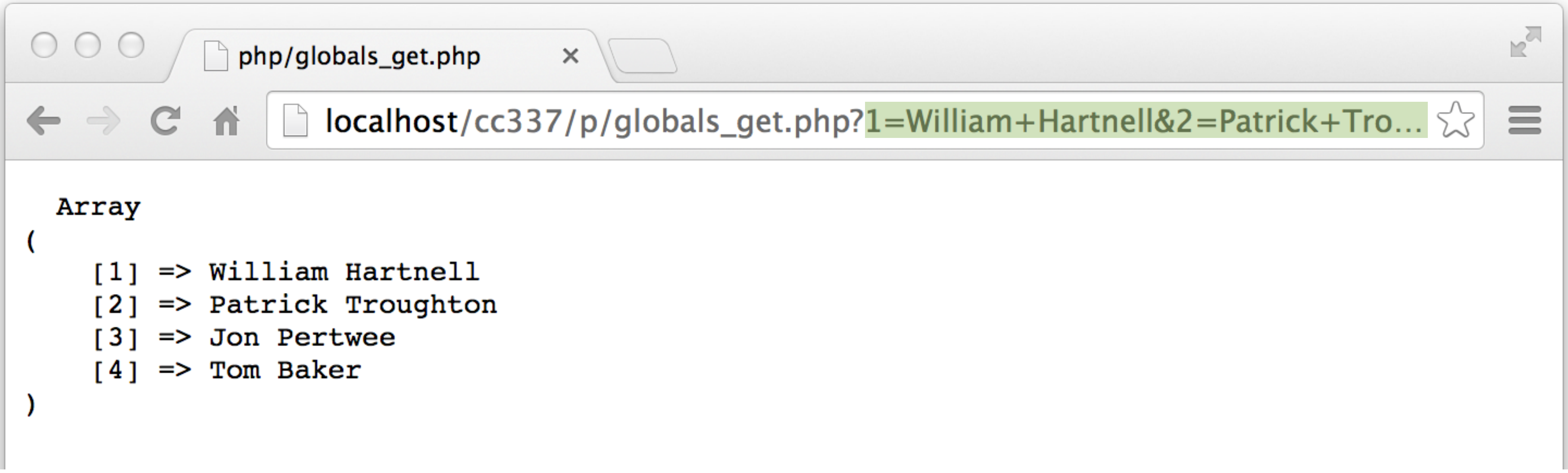
```
<!doctype html>
<html>
<head>
  <title>php/globals_server.php</title
</head>
<body>
  <pre>
  <?php print_r($_SERVER); ?>
  </pre>
</body>
</html>
```


php/globals_server.php

```
Array
(
    [HTTP_HOST] => localhost
    [HTTP_CONNECTION] => keep-alive
    [HTTP_ACCEPT] => text/html,application/xhtml+xml,application/xml;q=
    [HTTP_USER_AGENT] => Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5)
    [HTTP_ACCEPT_ENCODING] => gzip,deflate,sdch
    [HTTP_ACCEPT_LANGUAGE] => en-US,en;q=0.8
    [HTTP_COOKIE] => BrowserID={64}N2FlNDUzNjkwZDQ1NWFhNTU5ZWQwMWNkMDQ
    [PATH] => /usr/bin:/bin:/usr/sbin:/sbin
    [SERVER_SIGNATURE] =>
    [SERVER_SOFTWARE] => Apache/2.2.26 (Unix) DAV/2 PHP/5.4.30 mod_ssl/
    [SERVER_NAME] => localhost
    [SERVER_ADDR] => ::1
    [SERVER_PORT] => 80
    [REMOTE_ADDR] => ::1
    [DOCUMENT_ROOT] => /Library/WebServer/Documents
    [SERVER_ADMIN] => you@example.com
    [SCRIPT_FILENAME] => /Library/WebServer/Documents/cc337/examples/gl
    [REMOTE_PORT] => 55842
    [GATEWAY_INTERFACE] => CGI/1.1
    [SERVER_PROTOCOL] => HTTP/1.1
    [REQUEST_METHOD] => GET
    [QUERY_STRING] =>
    [REQUEST_URI] => /cc337/examples/globals_server.php
    [SCRIPT_NAME] => /cc337/examples/globals_server.php
    [PHP_SELF] => /cc337/examples/globals_server.php
    [REQUEST_TIME_FLOAT] => 1413179025.024
    [REQUEST_TIME] => 1413179025
)
```

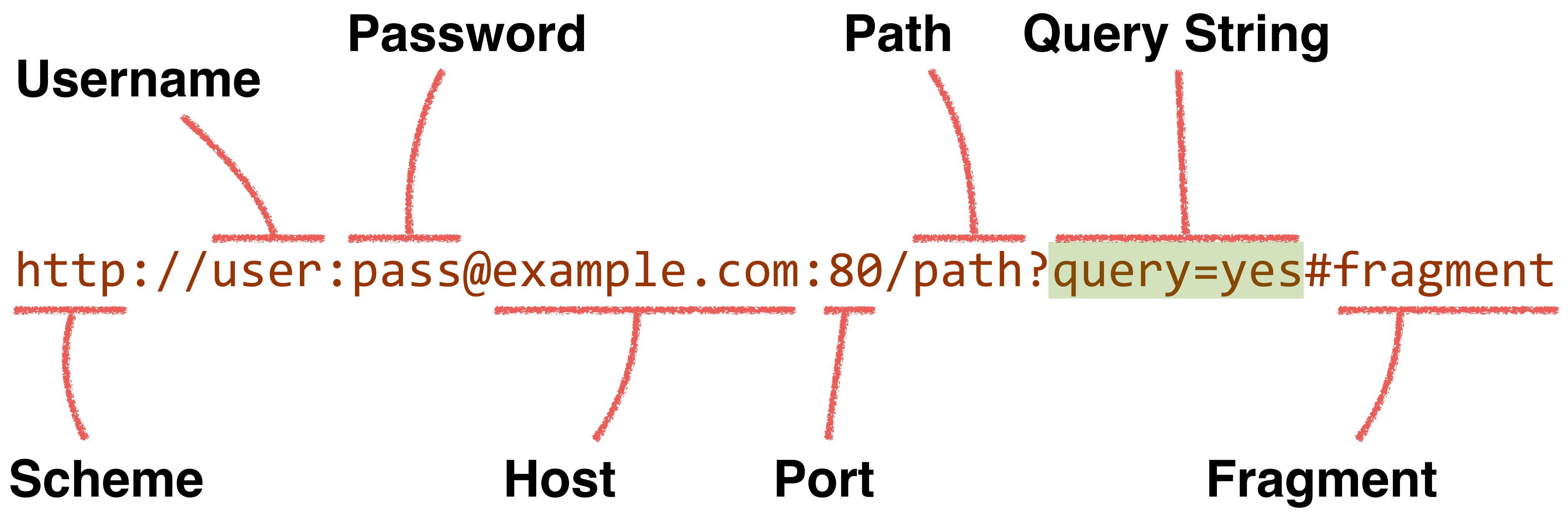
$_GET

- The **`$_GET`** superglobal contains all variables passed in via the *Query String* portion of the *URL*


php/globals_get.php
localhost/cc337/p/globals_get.php?1=William+Hartnell&2=Patrick+Tro...

```
  Array
(
    [1] => William Hartnell
    [2] => Patrick Troughton
    [3] => Jon Pertwee
    [4] => Tom Baker
)
```

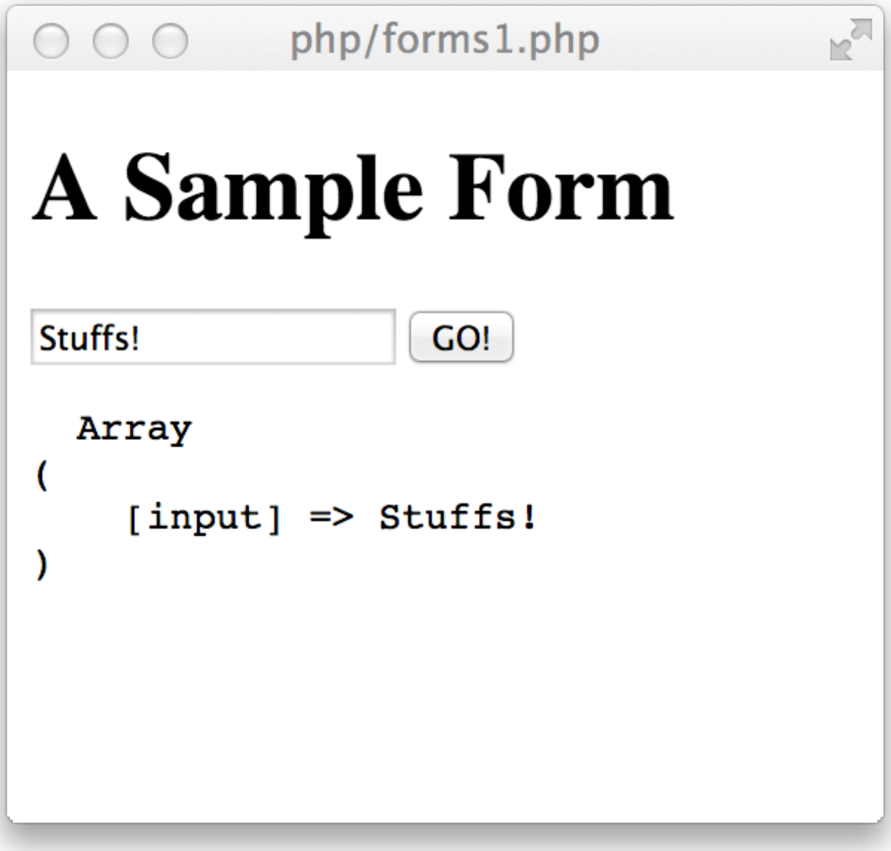
Query String



- Key / Value Pairs
- URL Encoded Values

Forms

- Forms processing is one of the major uses for server side code.
- More HTML elements!!
- Example

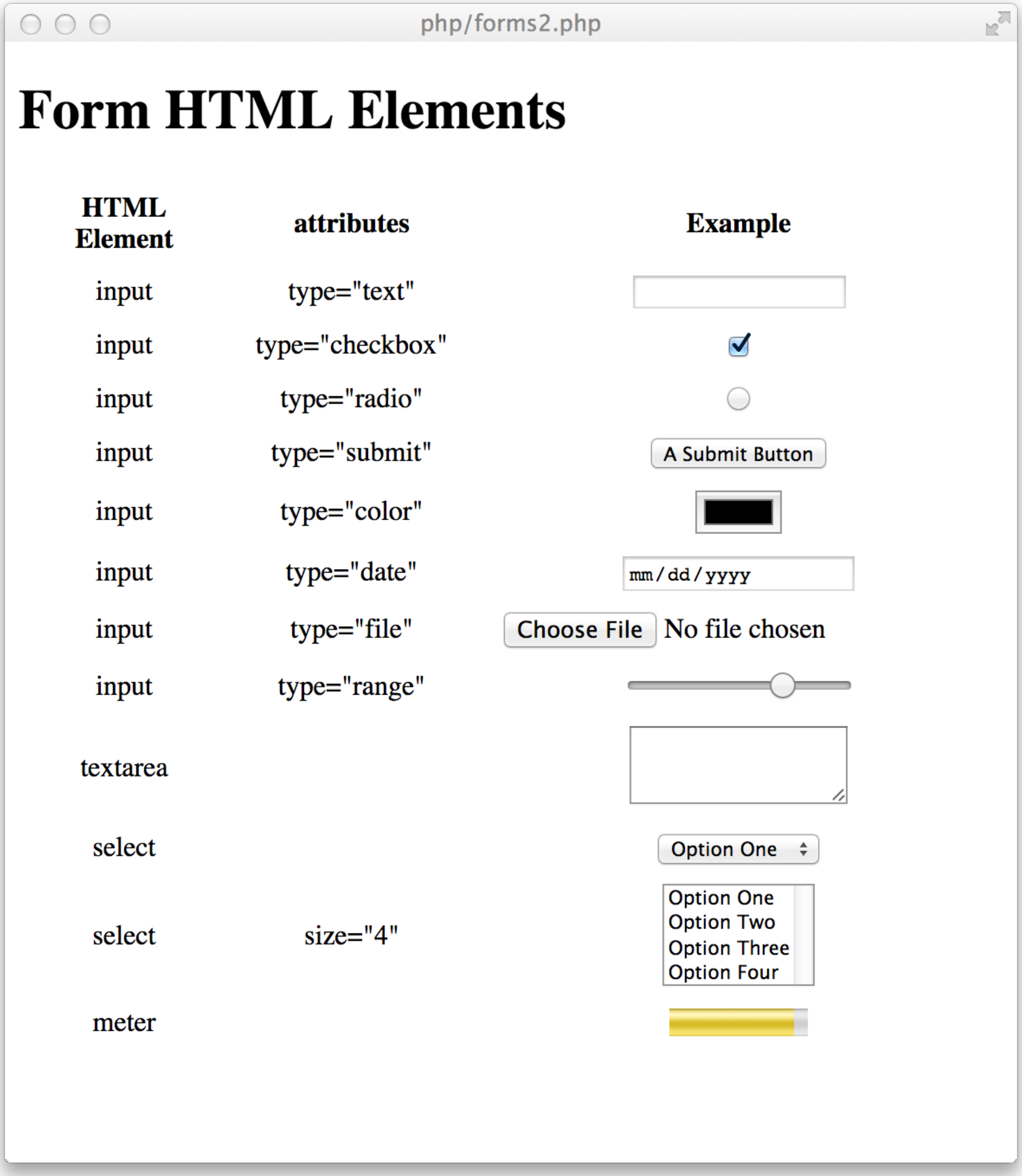


A Sample Form

```
  Array
(
    [input] => Stuffs!
)
```

Forms

- A bunch of different HTML form elements.



Form HTML Elements

HTML Element	attributes	Example
input	type="text"	
input	type="checkbox"	✓
input	type="radio"	○
input	type="submit"	A Submit Button
input	type="color"	■
input	type="date"	mm / dd / yyyy
input	type="file"	Choose File No file chosen
input	type="range"	
textarea		
select		Option One
select	size="4"	Option One Option Two Option Three Option Four
meter		

`<form>`

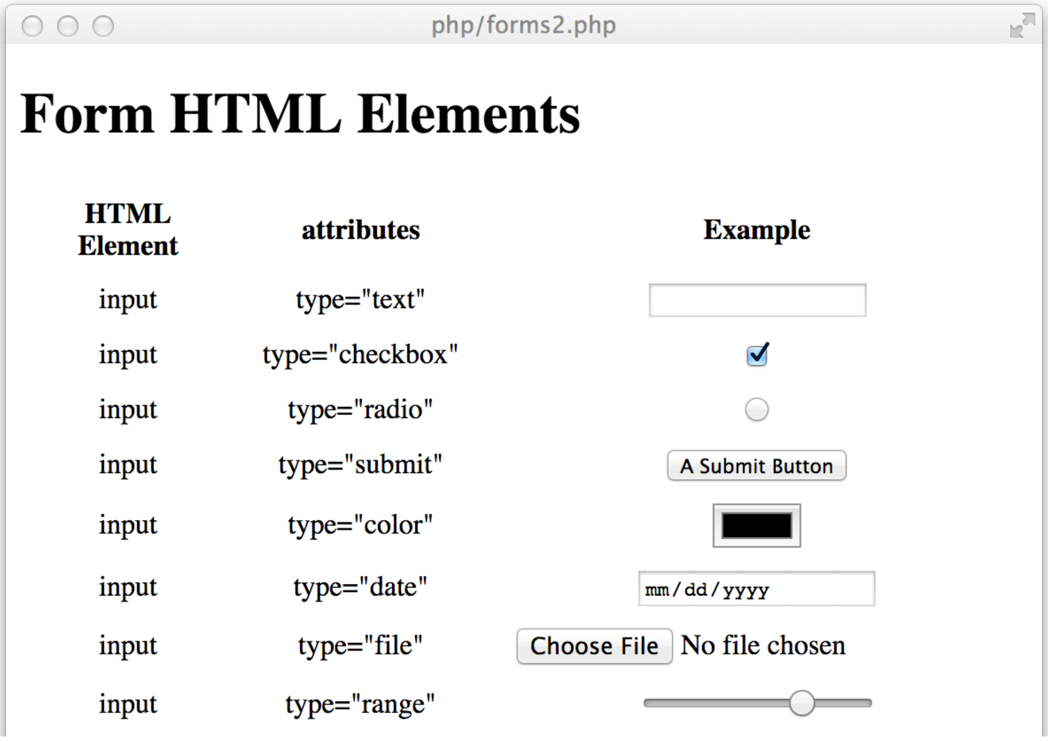
- The `<form>` element defines an HTML form, and dictates where the form data is sent, and how.
  - the **`action`** attribute says where to send this form's data when the form is submitted.
  - the **`method`** attribute says how to send the data, either with an **`HTTP GET`** command or **`POST`**.

```
<form action="forms3.php" method="POST">
  <input type="text" name="input" size="20">
  <input type="submit" value="GO!">
</form>
```

`<input>`

- The `<input>` element is the basic, and most flexible of the form elements.
- Basic text input fields.
- Submit buttons.
- Password fields.
- Checkboxes and radio buttons

```
<input type="text" name="input" size="20">

<input type="checkbox" name="check" checked>

<input type="radio" name="radioset">

<input type="submit" value="A Submit Button">
```


php/forms2.php

Form HTML Elements

HTML Element	attributes	Example
input	type="text"	
input	type="checkbox"	✓
input	type="radio"	○
input	type="submit"	A Submit Button
input	type="color"	■
input	type="date"	mm / dd / yyyy
input	type="file"	Choose File No file chosen
input	type="range"	

`$_GET` and `$_POST`

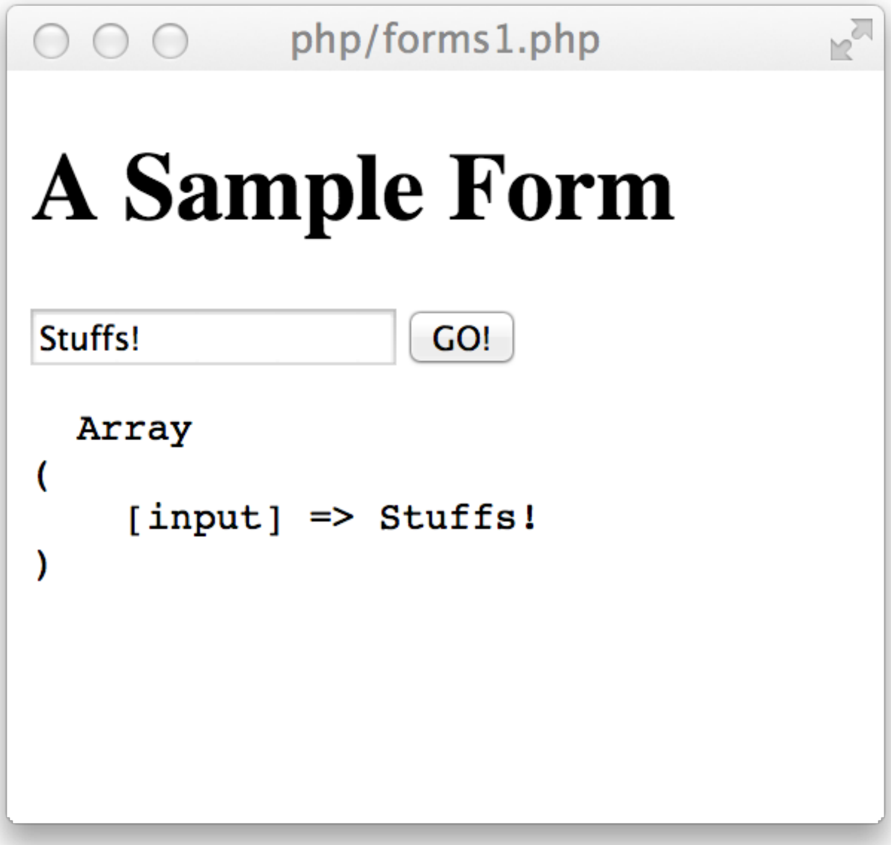
- PHP provides us with these superglobal arrays
- User input
- Don’t Trust it!

$_POST

- The `$_POST` superglobal array contains all key/value pairs passed in via a `POST HTTP` request.
- Usually as the result of a **Form** submission

$_POST

```
<!doctype html>
<?php
  $input = "";
  if (!empty($_POST)) {
    $input = print_r($_POST, true);
  }
?>
<html>
<head>
  <title>php/forms1.php</title>
</head>
<body>
  <h1>A Sample Form</h1>
  <form action="forms1.php" method="POST">
    <input type="text" name="input" size="20">
    <input type="submit" value="GO!">
  </form>
  <pre>
  <?php echo $input; ?>
  </pre>
</body>
</html>
```



A Sample Form

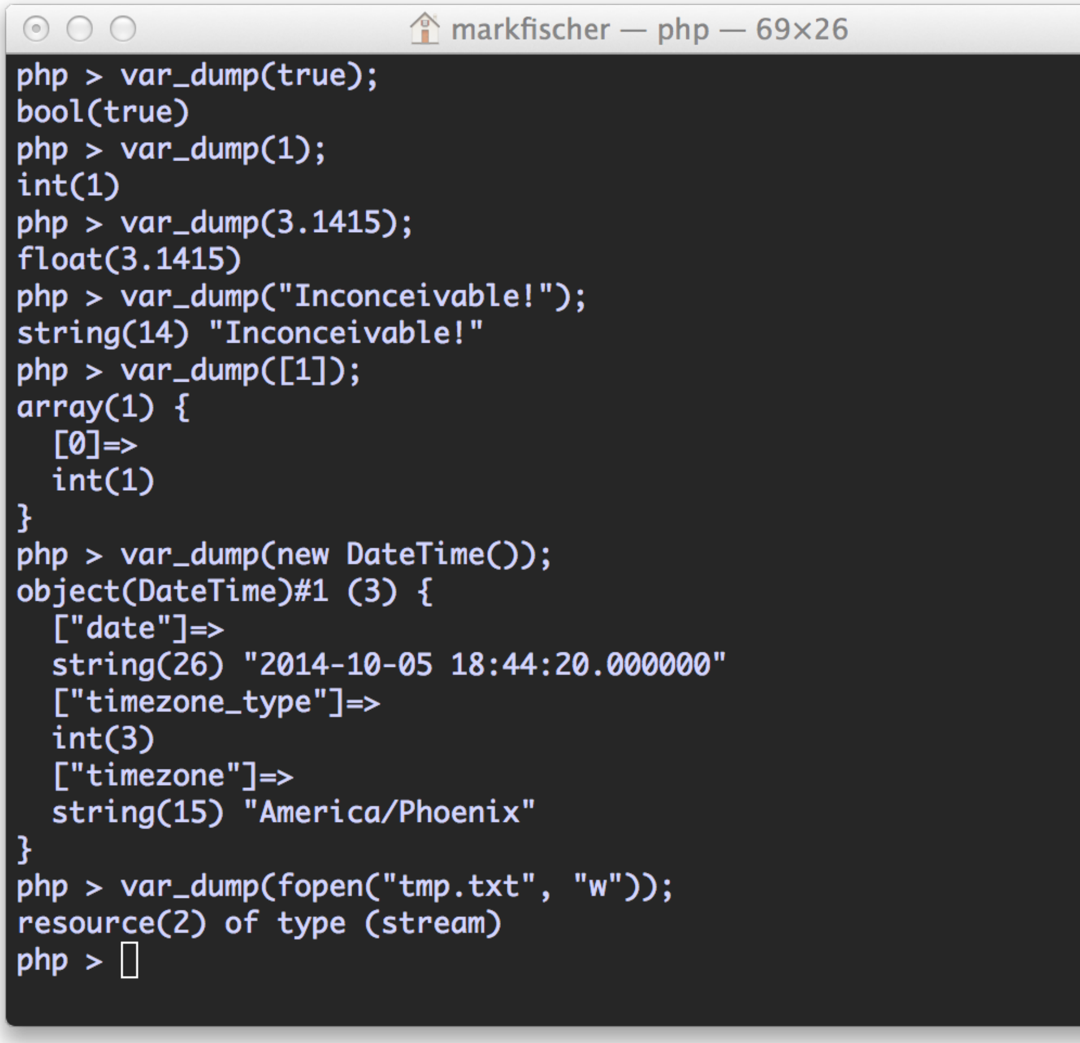
```
  Array
(
    [input] => Stuffs!
)
```

Datatypes

- PHP only does type checking at *run time*.
- Variables have an internal type, but are aggressively type converted based on situation.

Datatypes

- Boolean
- Integer
- Float (Double)
- String
- Array
- Object
- Resource
- NULL



```
php > var_dump(true);
bool(true)
php > var_dump(1);
int(1)
php > var_dump(3.1415);
float(3.1415)
php > var_dump("Inconceivable!");
string(14) "Inconceivable!"
php > var_dump([1]);
array(1) {
  [0]=>
  int(1)
}
php > var_dump(new DateTime());
object(DateTime)#1 (3) {
  ["date"]=>
  string(26) "2014-10-05 18:44:20.000000"
  ["timezone_type"]=>
  int(3)
  ["timezone"]=>
  string(15) "America/Phoenix"
}
php > var_dump(fopen("tmp.txt", "w"));
resource(2) of type (stream)
php > 
```

http://php.net/manual/en/language.types.php

Built In Functions

- PHP has ‘em. Seriously, lots of them.
- Different from Java, or C, where the language defines very little in the way of functionality.
  - Functions are included manually via import statements.
- PHP defines hundreds of built-in functions, available in the global scope.

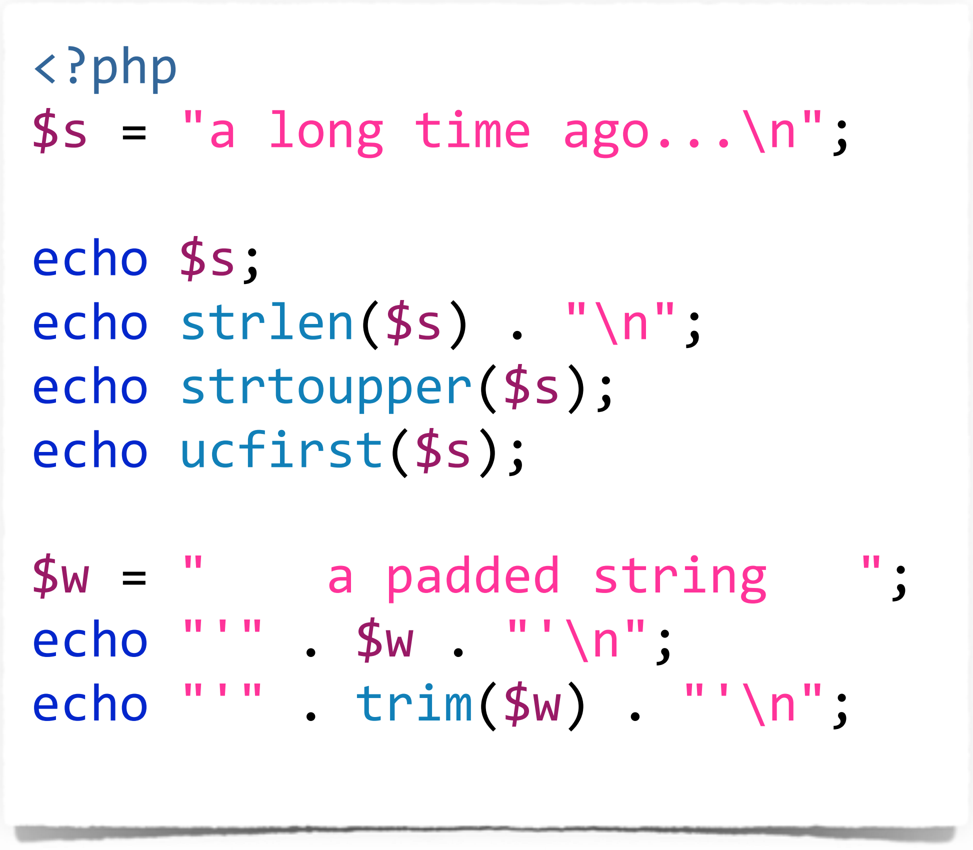
String Functions

- Its probably more useful to talk about datatypes as they relate to built in functions.

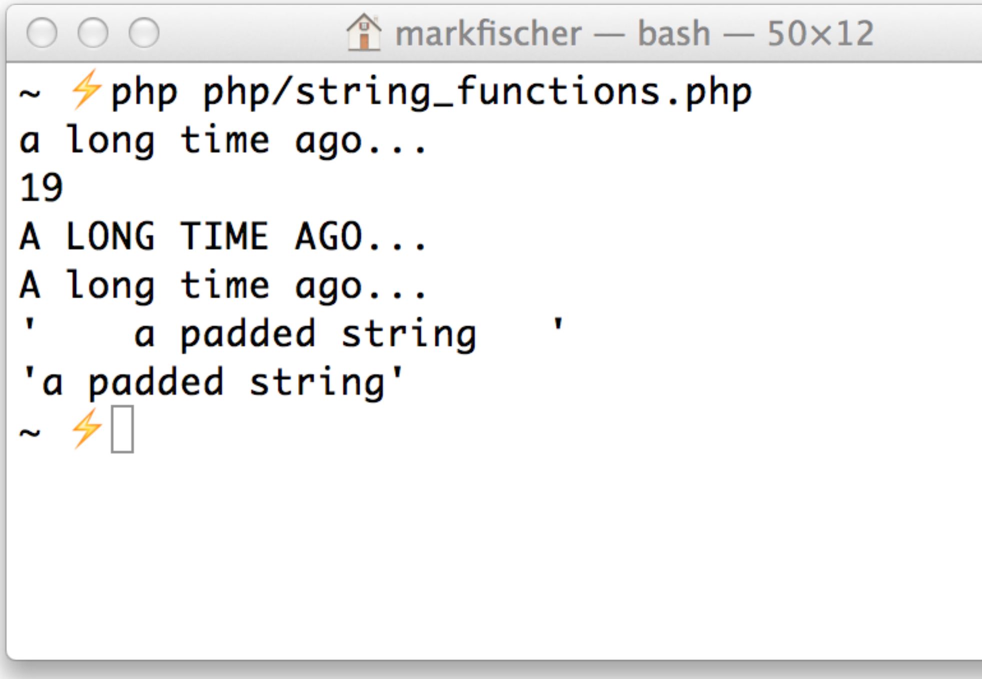
`echo`	Outputs a string
`printf`	Print a C style formatted string
`strlen`	Gets the length of a string
`strtoupper`	Returns an uppercase string
`trim`	Remove whitespace from the beginning and end of a string
`ucfirst`	Uppercase the first character of a string
	nearly 100 more…

[`http://php.net/manual/en/book.strings.php`](http://php.net/manual/en/book.strings.php)

String Functions



```php
<?php
$s = "a long time ago...\n";

echo $s;
echo strlen($s) . "\n";
echo strtoupper($s);
echo ucfirst($s);

$w = "     a padded string    ";
echo "'" . $w . "'\n";
echo "'" . trim($w) . "'\n";
```

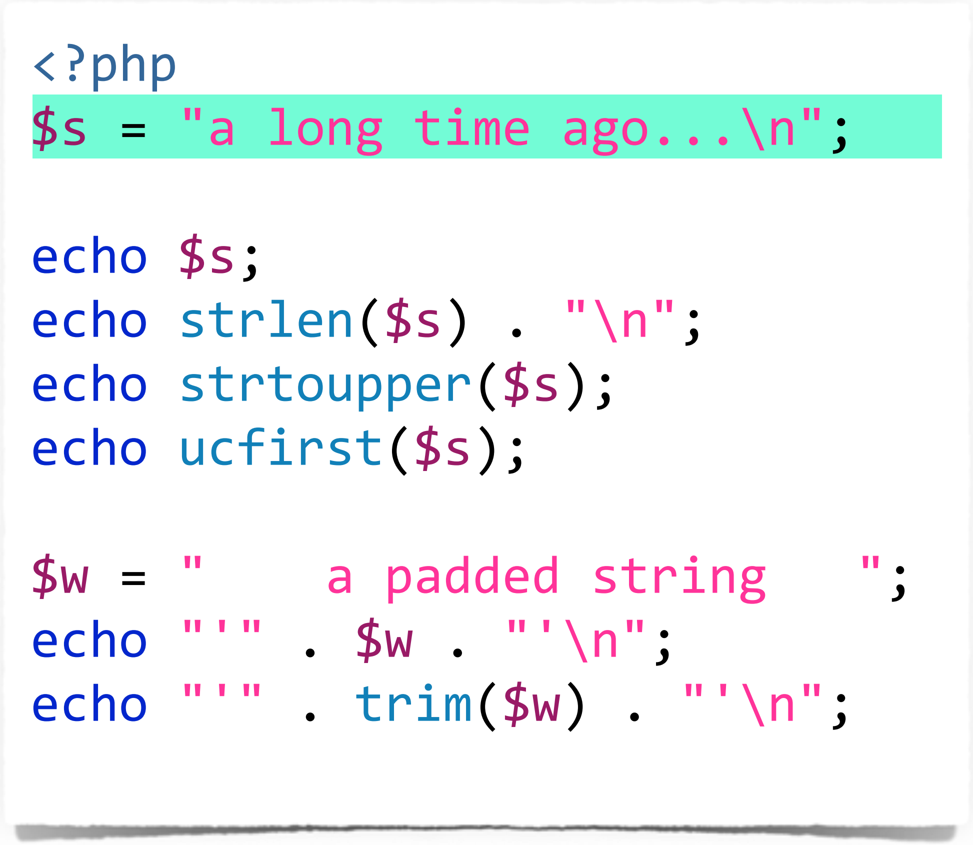


```
~ ⚡php php/string_functions.php
a long time ago...
19
A LONG TIME AGO...
A long time ago...
'     a padded string    '
'a padded string'
~ ⚡
```

http://php.net/manual/en/book.strings.php

String Escaped Characters

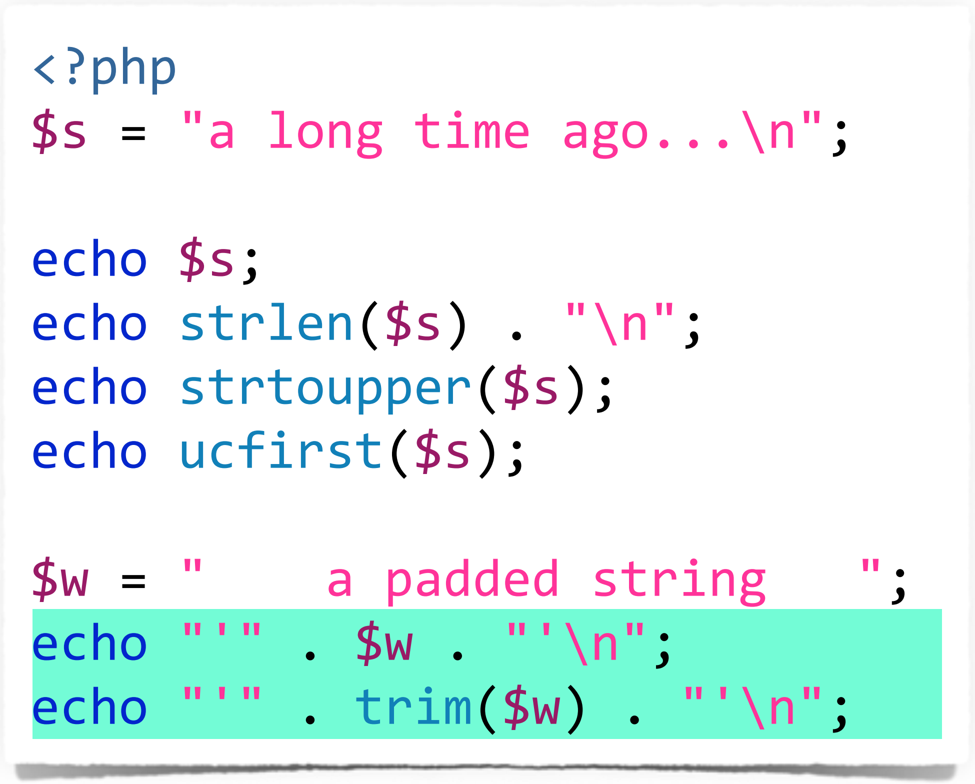
- Standard sort of escape mechanism for things like newlines and tabs.
- `\n` for a newline
- `\t` for a tab



```php
<?php
$s = "a long time ago...\n";

echo $s;
echo strlen($s) . "\n";
echo strtoupper($s);
echo ucfirst($s);

$w = "    a padded string   ";
echo "'" . $w . "'\n";
echo "'" . trim($w) . "'\n";
```

String Concatenation

- The period . is our concatenation operator in PHP



```php
<?php
$s = "a long time ago...\n";

echo $s;
echo strlen($s) . "\n";
echo strtoupper($s);
echo ucfirst($s);

$w = "    a padded string   ";
echo "'" . $w . "'\n";
echo "'" . trim($w) . "'\n";
```

Integers

- Formally, an integer in PHP is a member of the set:

$$\mathbb{Z} = \{\ldots, \text{-}2, \text{-}1, 0, 1, 2, \ldots\}$$

- `$a = 0;             // A decimal integer`
- `$a = -123;          // A negative decimal integer`
- `$a = 0123;          // An octal integer: 83`
- `$a = 0x2A;          // A hexadecimal integer: 42`
- `$a = 0b11111111;  // A binary integer: 255`

Floats

- Floats, Doubles, Reals. PHP calls them all Floats
- `$a = 3.1415;`
- `$a = 1.2e4;`
- `$a = 7E-10;`
- All ways to define a float value

Arithmetic Operators

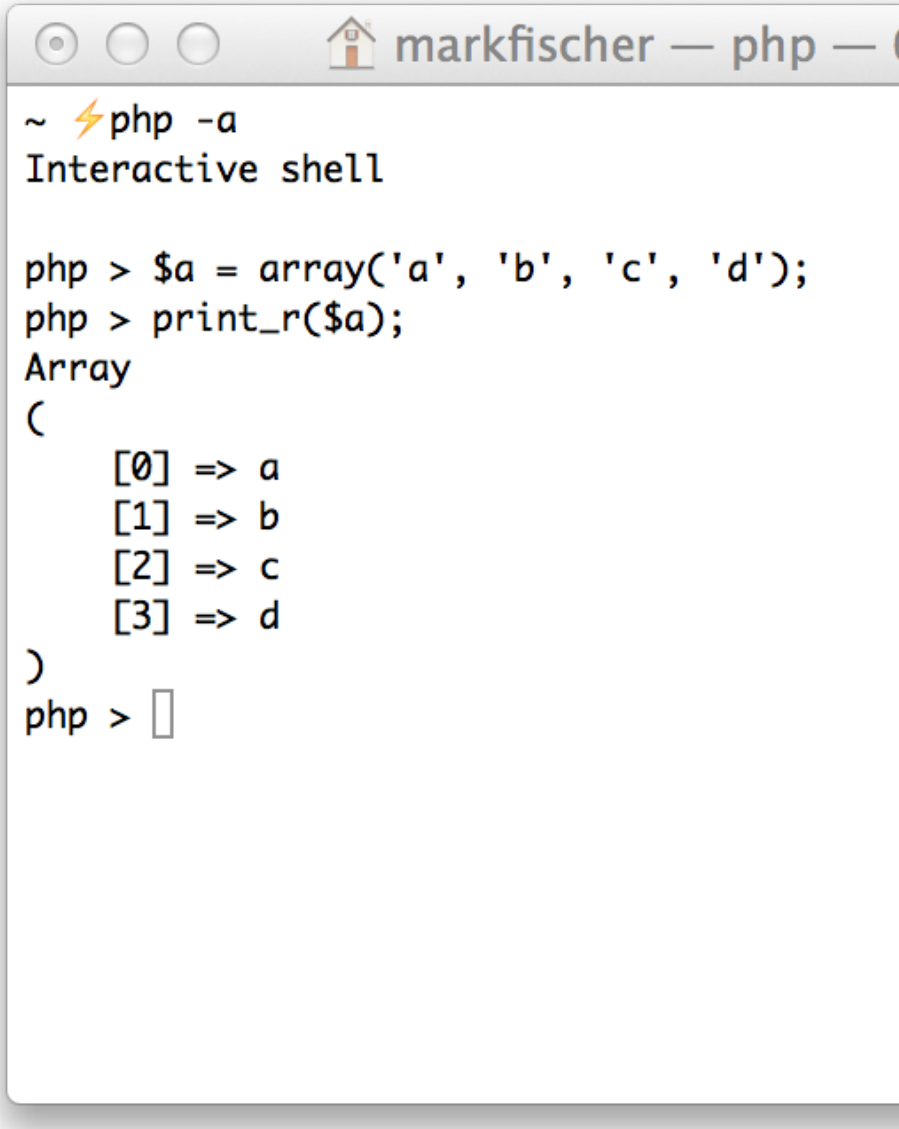
- You can, you know… do math, and stuff.
- PHP will convert an integer to a float before arithmetic
- `$a = 1 + 2;` // int = int + int
- `$a = 5 - 2.45;` // float = (int cast to float) + float
- `$a = 5.5 / 0.5;` // float = float ÷ float

Arithmetic Operators

`$a + $b`	Addition
`$a - $b`	Subtraction
`-$a`	Negation
`$a / $b`	Division
`$a * $b`	Multiplication
`$a % $b`	Modulo
`$a ** $b`	Exponent (`$a` raised to the `$b` power) New in PHP 5.6

Arrays

- Arrays in PHP are all *ordered Maps* under the hood.
- A map associates Keys and Values
- The basic array structure associates numerical keys (0, 1, 2, 3, 4) with their values.
- `$a = array('a', 'b', 'c', 'd');`

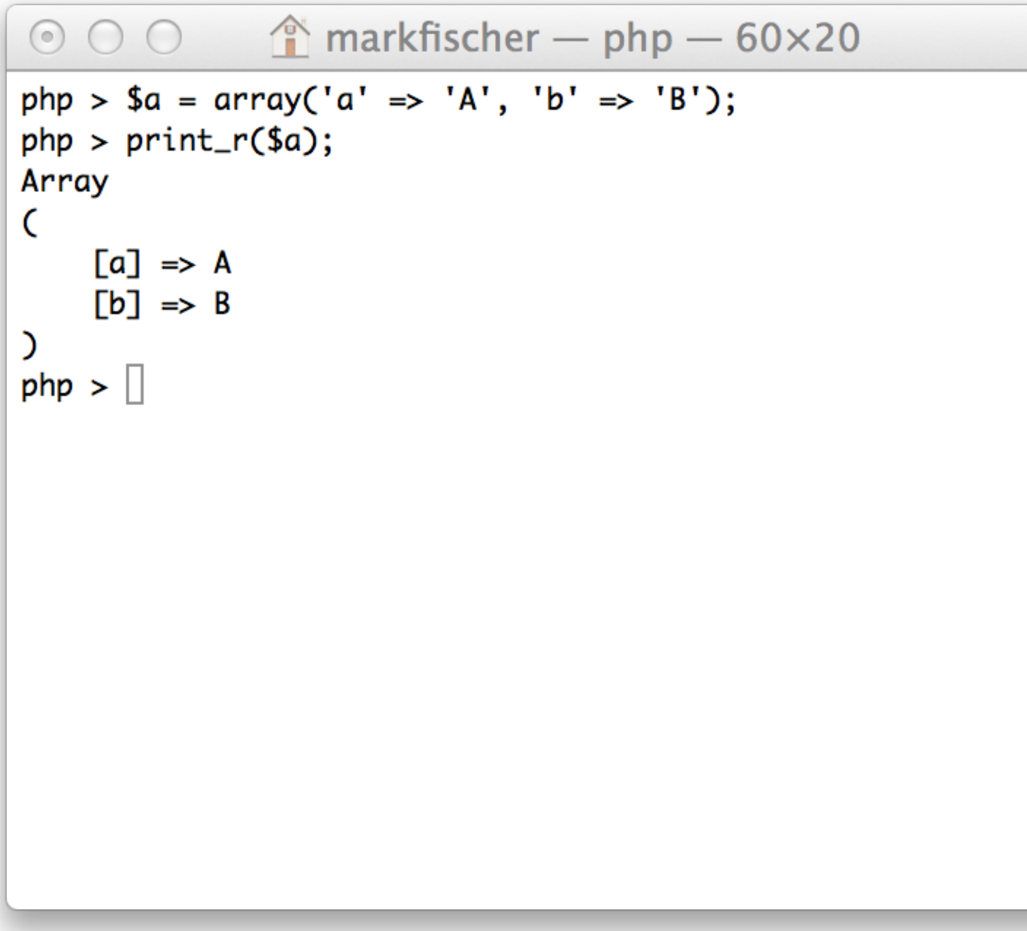


```
~ ⚡php -a
Interactive shell

php > $a = array('a', 'b', 'c', 'd');
php > print_r($a);
Array
(
    [0] => a
    [1] => b
    [2] => c
    [3] => d
)
php > 
```

Arrays

- You can specify the keys for arrays using the `key => value` syntax.

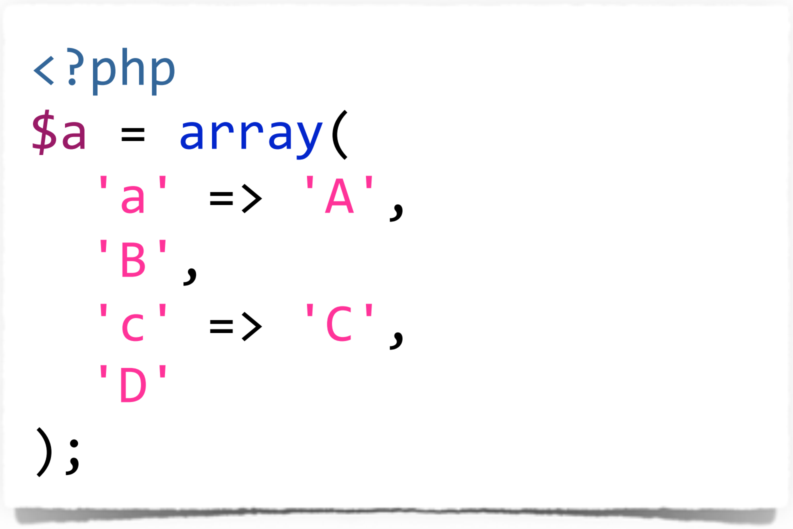
- `$a = array('a' => 'A');`



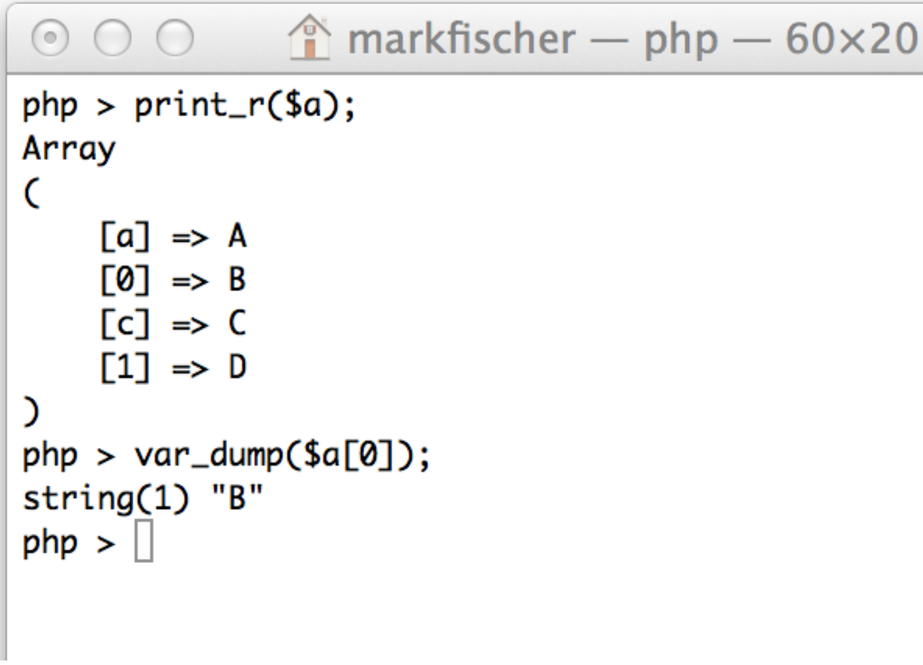
```
php > $a = array('a' => 'A', 'b' => 'B');
php > print_r($a);
Array
(
    [a] => A
    [b] => B
)
php > 
```

Arrays

- Any element who's key is not explicitly set receives and auto-increment key.
- They start incrementing as they're used, so `$a[0]` does not *always* indicate the first element of an array!



```php
<?php
$a = array(
  'a' => 'A',
  'B',
  'c' => 'C',
  'D'
);
```

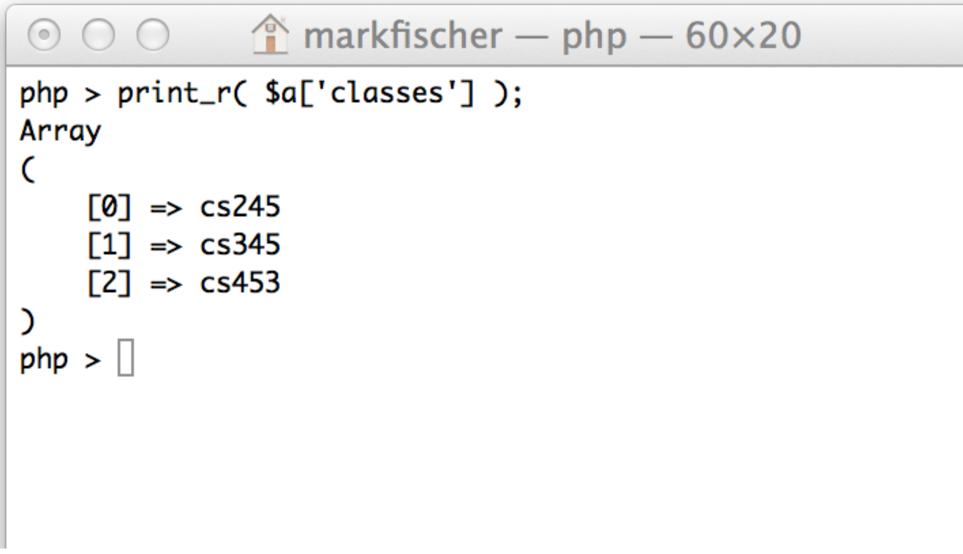


```
php > print_r($a);
Array
(
    [a] => A
    [0] => B
    [c] => C
    [1] => D
)
php > var_dump($a[0]);
string(1) "B"
php > 
```

Arrays

- Array values can be any valid type.
- A given array can have values of many different types.
- You can have arrays as values in an array element, leading to complex nested structures.

```php
$a = array(
  'name' => 'Mark',
  'classes' => array(
    'cs245',
    'cs345',
    'cs453'
  )
);
```



markfischer — php — 60×20

```
php > print_r( $a['classes'] );
Array
(
    [0] => cs245
    [1] => cs345
    [2] => cs453
)
php > 
```

Array Functions

- There are quite a lot of array functions!

`$a["foo"] = 1`	Assigns the value 1 to the element with a key of “key”
`array_push($a, 2)`	Appends a new element to the end of the array with a value 2
`$a[] = 2`	Same as above. Shortcut for `array_push()`
`array_pop($a)`	Pops an element off the end of the array and returns its value.
`array_keys($a)`	Returns an array of all the keys for the array `$a`
`sort($a)`	Sort the elements in array `$a` by their keys.
	There are 75 more!

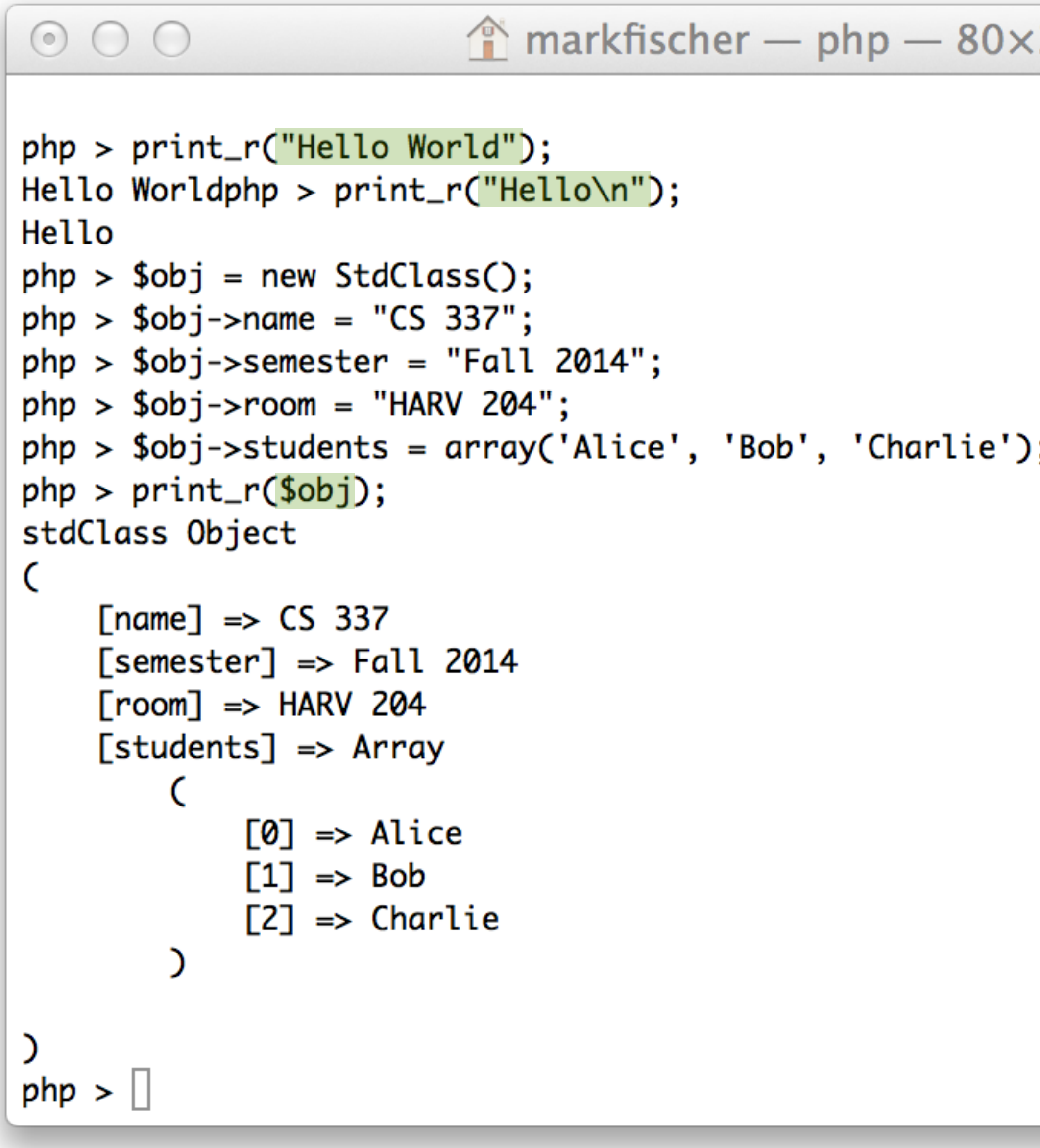
`http://php.net/manual/en/ref.array.php`

`print_r()`

- Similar to var_dump(), print_r() will print the contents of an object to STDOUT
  - Can be made to return a string instead of printing to STDOUT
- It doesn’t report anything about data types
- Looks a little bit nicer
- Doesn’t append line breaks

print_r()

- Similar to var_dump(), print_r() will print the contents of an object to STDOUT
  - Can be made to return a string instead of printing to STDOUT
- It doesn't report as much about data types (still some though)
- Looks a little bit nicer
- Doesn't append line breaks (except with arrays and objects)



```
php > print_r("Hello World");
Hello Worldphp > print_r("Hello\n");
Hello
php > $obj = new StdClass();
php > $obj->name = "CS 337";
php > $obj->semester = "Fall 2014";
php > $obj->room = "HARV 204";
php > $obj->students = array('Alice', 'Bob', 'Charlie')
php > print_r($obj);
stdClass Object
(
    [name] => CS 337
    [semester] => Fall 2014
    [room] => HARV 204
    [students] => Array
        (
            [0] => Alice
            [1] => Bob
            [2] => Charlie
        )

)
php > 
```

Booleans

- Truth or dare! Well.. `true` or `false`
- Case *insensitive*
    - `true TRUE True trUE` // All of these are true!
    - `FALSE false fALsE` // Yup, all false

Booleans

- Most values in PHP are `true`, there are also many which are `false`.
- Some of the things that are `false` (there are others):
  - `false` (well… duh)
  - the integer value `0` // this one causes us problems later…
  - the float value `0.0`
  - an empty string, i.e. `""`
  - an array with zero elements
  - the special type `NULL`
  - any unset variable (think `undefined` from javascript)

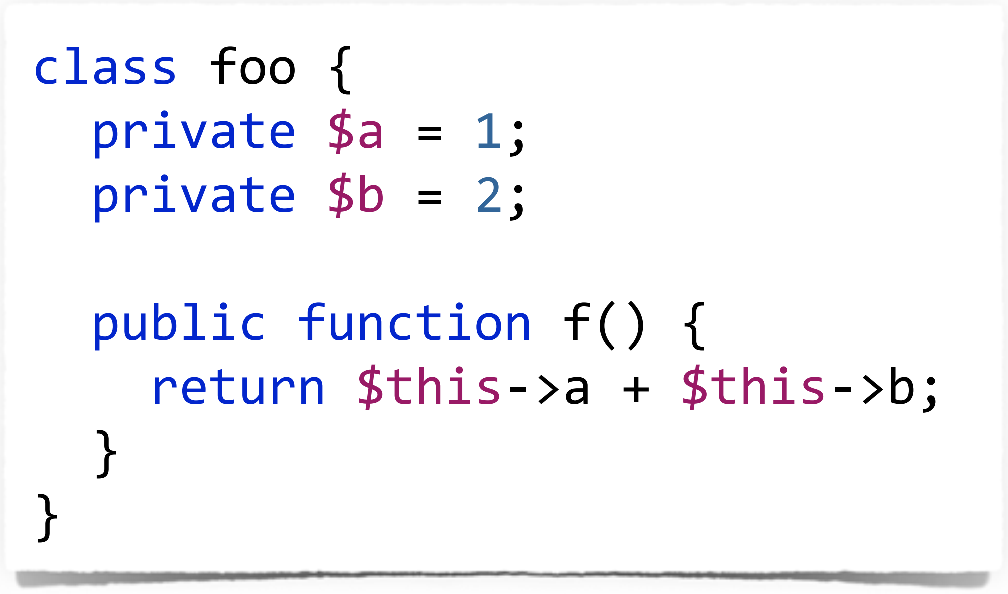
`http://php.net/manual/en/language.types.boolean.php`

Objects

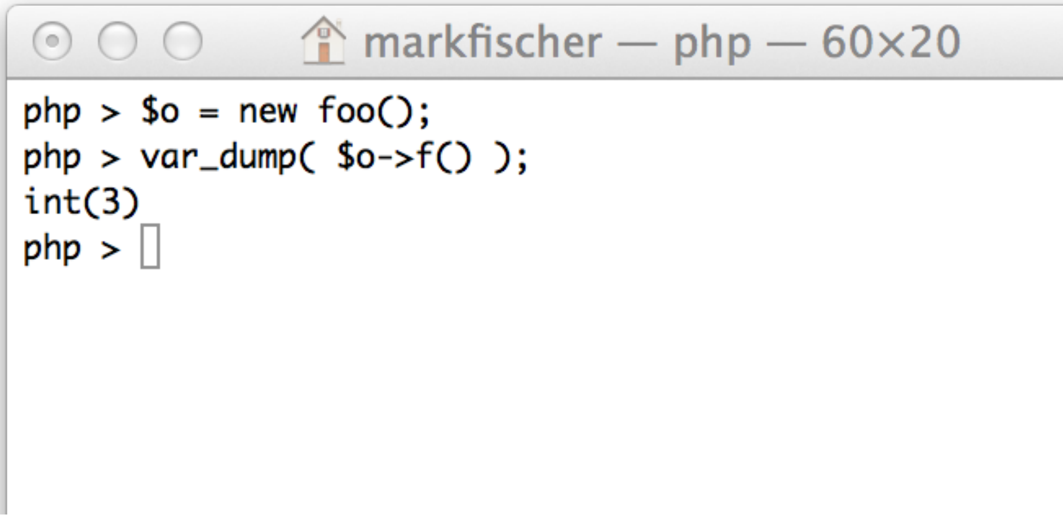
- PHP gained true object oriented support in PHP 5.0
- Classes are declared and inherited
- Instances are created of classes via the `new` keyword.

Objects

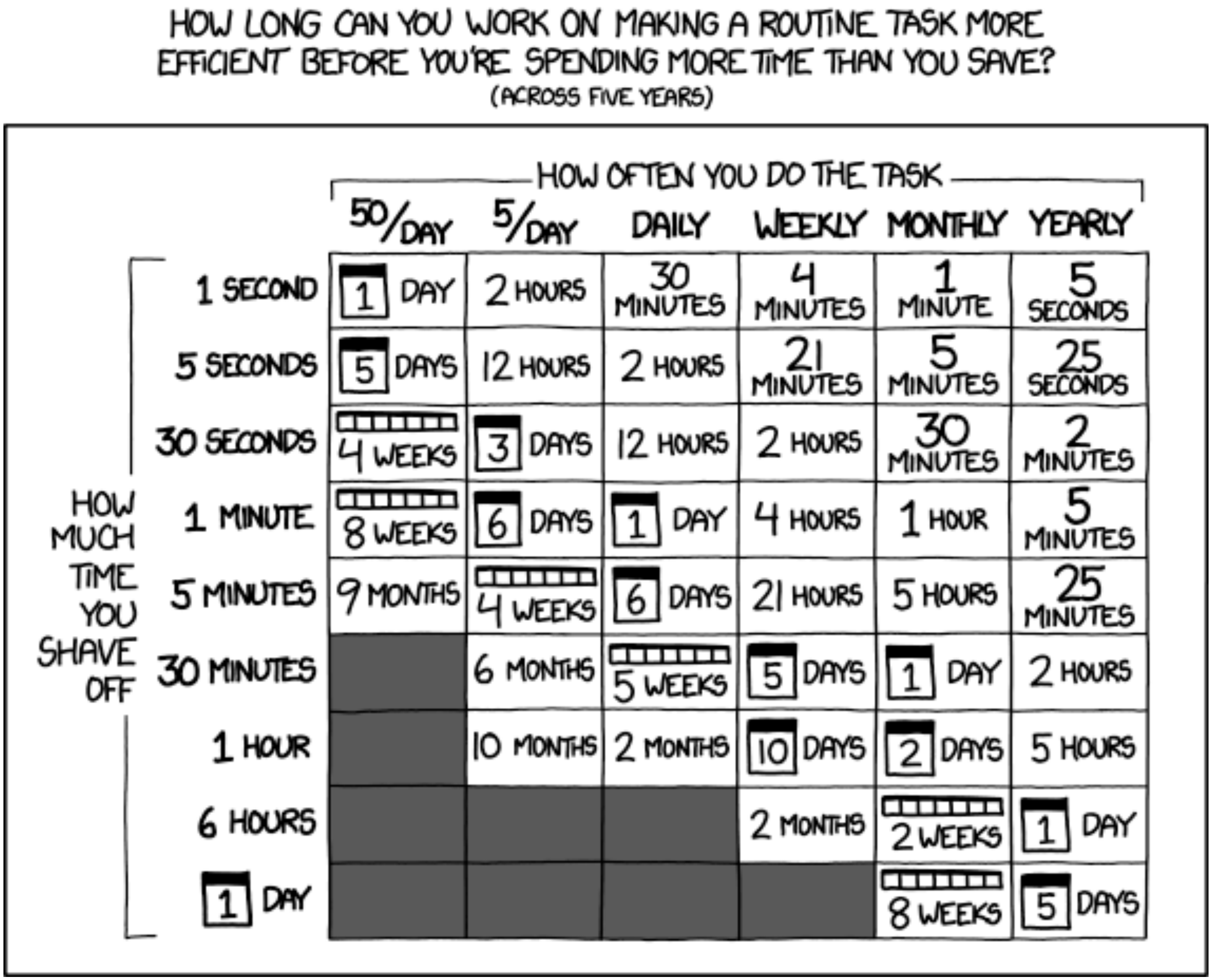
- Objects can have properties, methods, constructors
- Supports single inheritance
- Supports public, private, protected visibility
- Lots more on objects as we go



```php
class foo {
  private $a = 1;
  private $b = 2;

  public function f() {
    return $this->a + $this->b;
  }
}
```



```
php > $o = new foo();
php > var_dump( $o->f() );
int(3)
php >
```



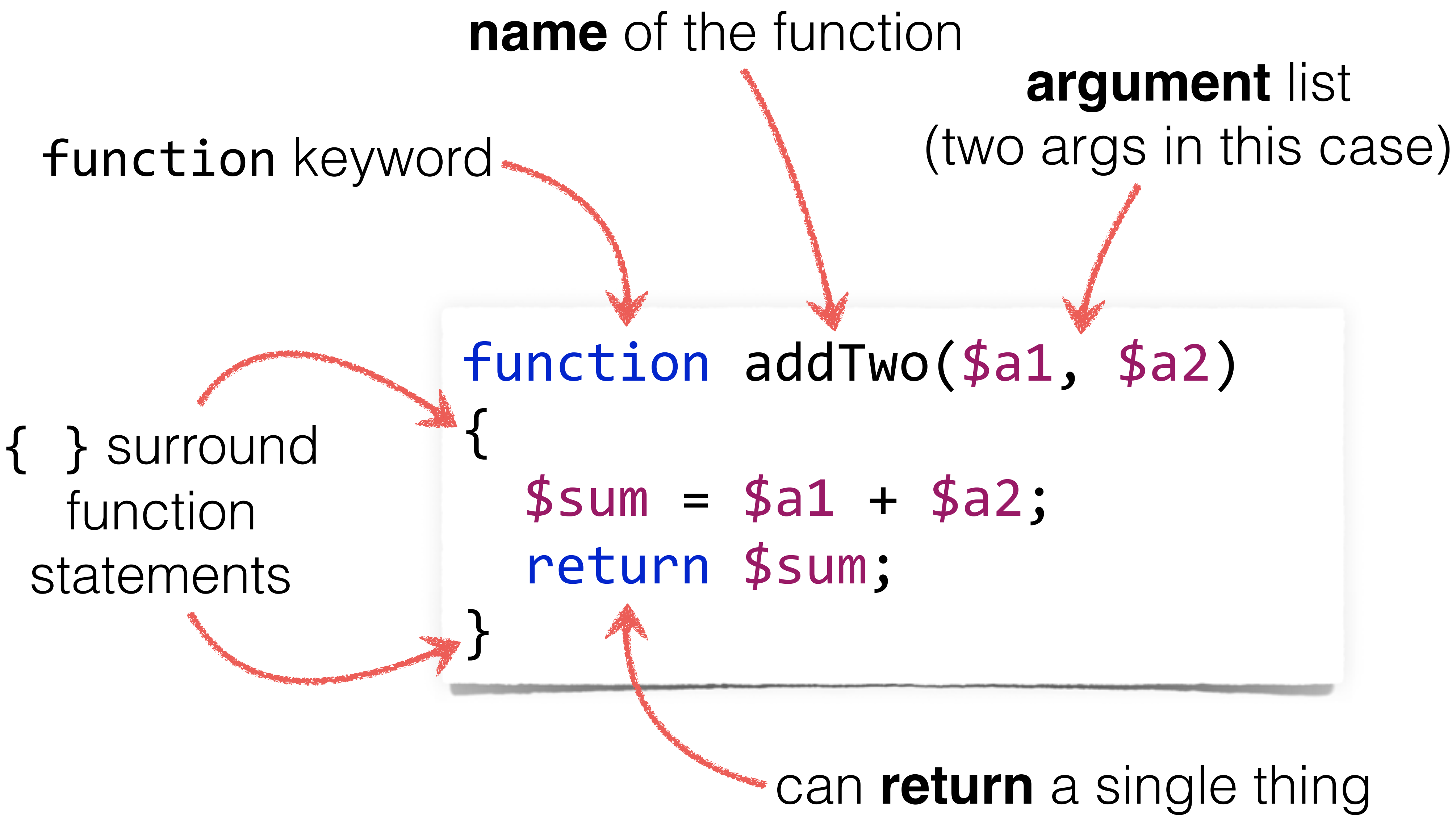
HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?
(ACROSS FIVE YEARS)

HOW MUCH TIME YOU SHAVE OFF	HOW OFTEN YOU DO THE TASK					
	50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
1 SECOND	1 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
5 SECONDS	5 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
30 SECONDS	4 WEEKS	3 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
1 MINUTE	8 WEEKS	6 DAYS	1 DAY	4 HOURS	1 HOUR	5 MINUTES
5 MINUTES	9 MONTHS	4 WEEKS	6 DAYS	21 HOURS	5 HOURS	25 MINUTES
30 MINUTES	▨	6 MONTHS	5 WEEKS	5 DAYS	1 DAY	2 HOURS
1 HOUR	▨	10 MONTHS	2 MONTHS	10 DAYS	2 DAYS	5 HOURS
6 HOURS	▨	▨	▨	2 MONTHS	2 WEEKS	1 DAY
1 DAY	▨	▨	▨	▨	8 WEEKS	5 DAYS

Functions

- PHP began life as a procedural & function based language.
- Only added Objects late in life.
- PHP loves functions.

Anatomy of a Function



```
function addTwo($a1, $a2)
{
   $sum = $a1 + $a2;
   return $sum;
}
```

Functions

- Functions can be declared at the top level, or inside other functions
- Functions have global scope, no matter where they are declared
- Scope is different than namespaces, we won’t go into namespaces

```php
<?php
function foo() {
  function foo2() {
    return "bar!";
  }

  return foo2();
}

// Cannot call foo2() here,
// it doesn't exist yet!

var_dump(foo());

// Now we can call foo2, its been
// defined by calling foo()
var_dump(foo2());
```

```
string(4) "bar!"
string(4) "bar!"
```

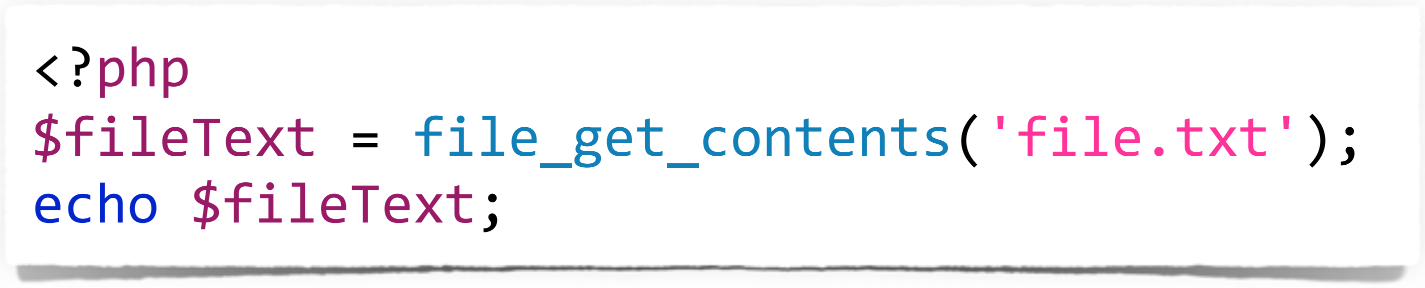
File IO

- Reading from a local or remote file is pretty straight forward
- Writing to files is a bit more complicated

Reading from a File

- `file_get_contents(“path/to/file”)`

- Reads the entire contents of a file into memory and returns it as a string.



```php
<?php
$fileText = file_get_contents('file.txt');
echo $fileText;
```

- This example reads the entire contents of ‘file.txt’ into a variable called **`$fileText;`**

Reading from a File

- `fopen('path/to/file', 'r')`
- Creates a file handle that can be referenced by further function calls.
- Can open files in read mode, or write mode.
- Doesn’t read the entire file into memory, so useful for working with large files, or for files where you don’t want everything, just specific pieces.

```
file.txt
This is a text file.
It has a few lines of text in it.
Nothing much to see here.
```

```php
<?php
// Open a file handle to 'file.txt'
$fileHandle = fopen('file.txt', 'r');

// Read one line from the $fileHandle
$aLine = fgets($fileHandle);

// Read another line from the $fileHandle
$anotherLine = fgets($fileHandle);

echo $anotherLine;
```

```
~/php ⚡ php fopen.php
It has a few lines of text in it.
~/php ⚡
```

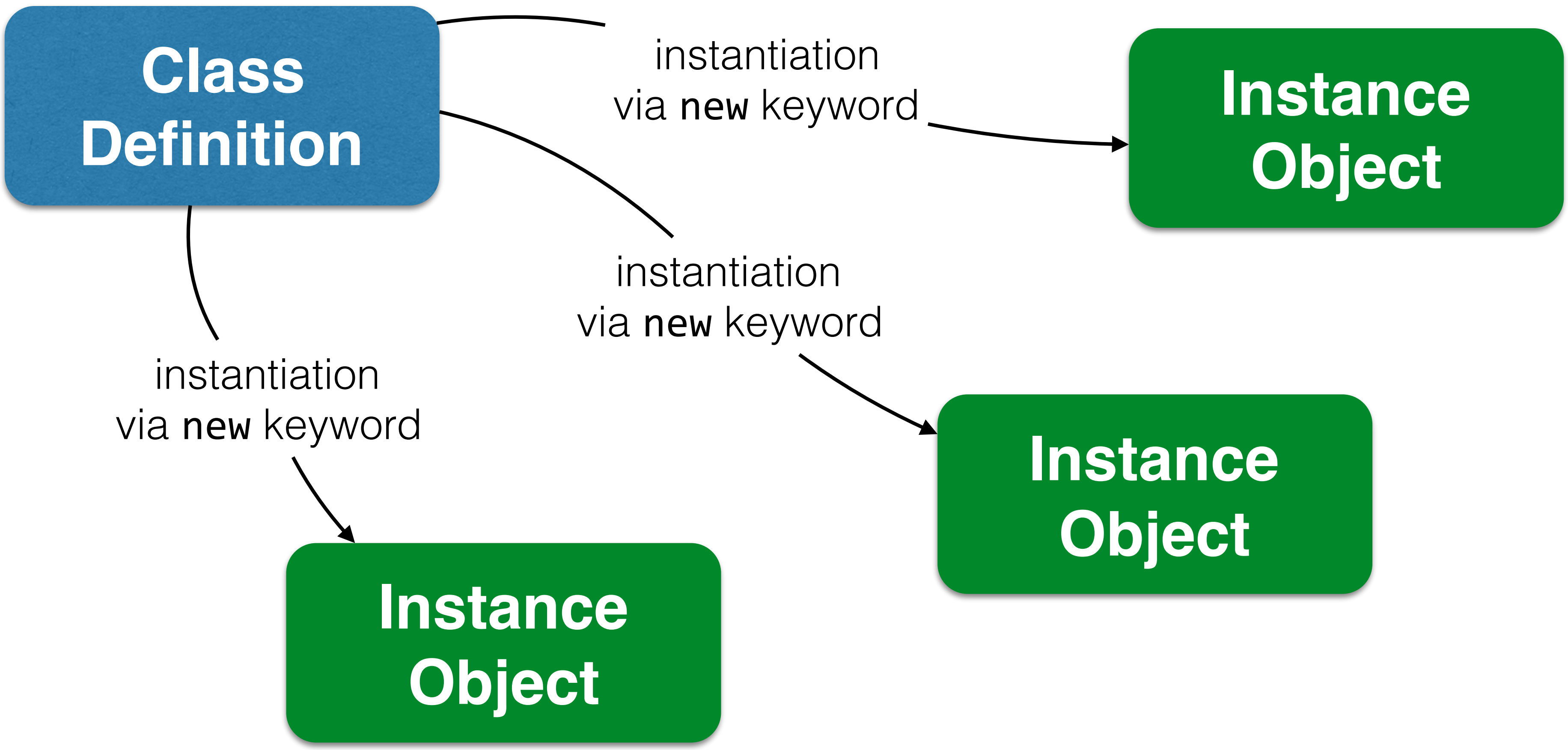
Remote Files

- Most PHP file operations that take a path can accept any type of stream.
- Get the remote contents of a URL

```php
<?php

$webpage = file_get_contents("http://www.example.com");

echo $webpage;
```

Objects

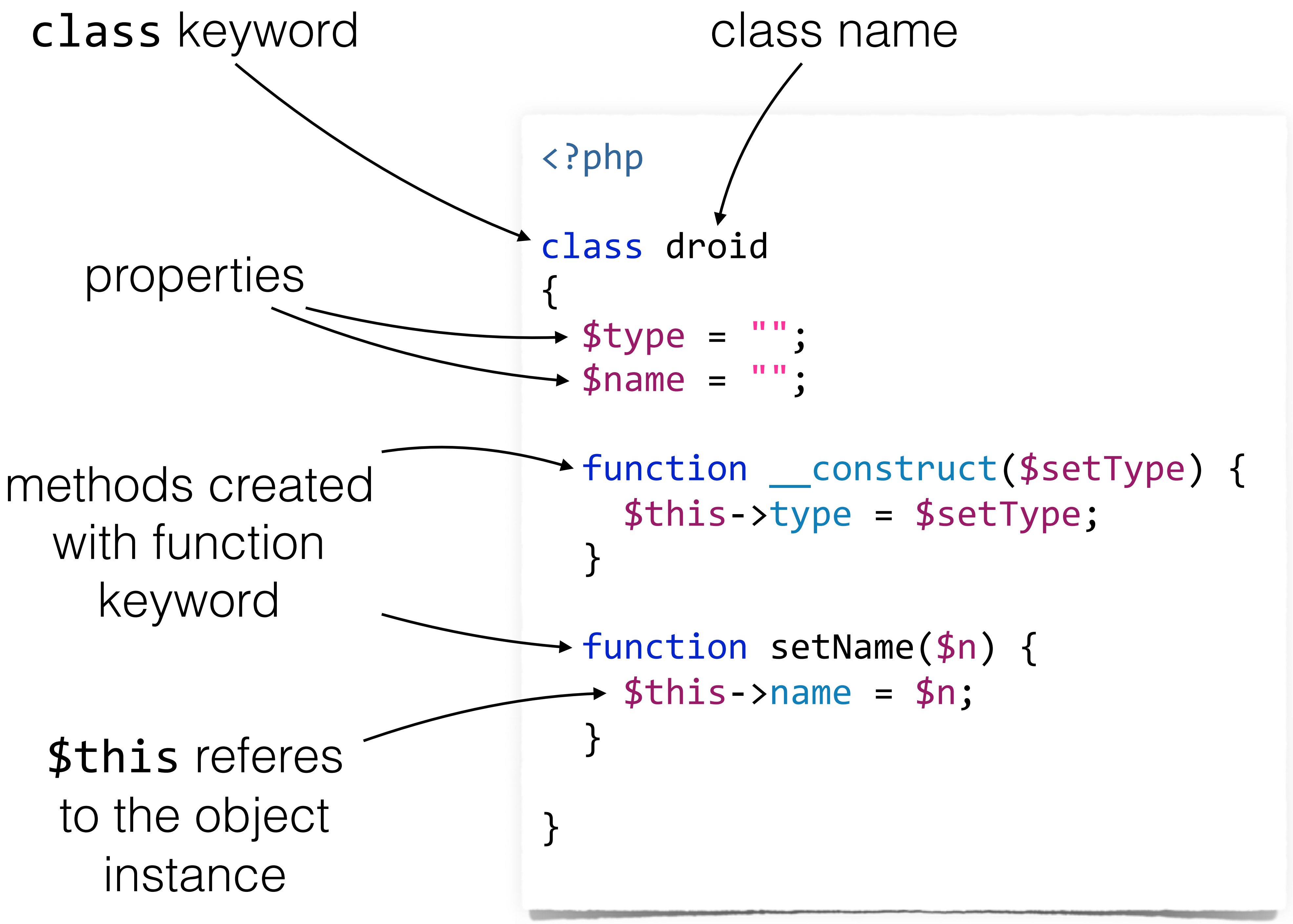
- PHP 5 introduced full well thought out objects.


Class Definition
instantiation via new keyword
Instance Object
instantiation via new keyword
Instance Object
instantiation via new keyword
Instance Object

Objects

- Classes are defined with the `class` keyword.
- New objects are created with the `new` keyword.

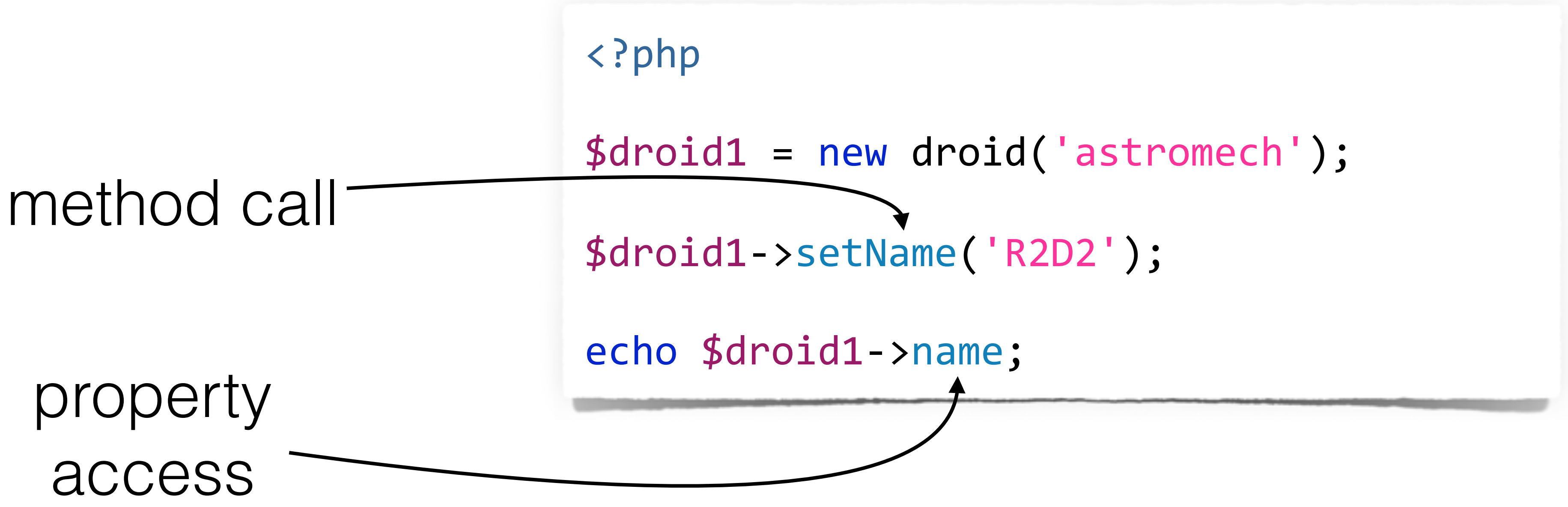
```
<?php

class droid
{
  $type = "";

  function __construct($setType) {
    $this->type = $setType;
  }
}

$droid1 = new droid('protocol');
$droid2 = new droid('astromech');
```


class keyword
class name
properties
methods created with function keyword
$this referes to the object instance

```php
<?php

class droid
{
  $type = "";
  $name = "";

  function __construct($setType) {
    $this->type = $setType;
  }

  function setName($n) {
    $this->name = $n;
  }

}
```

Objects

- PHP uses the `->` characters to do object access. Works pretty much the same way that a period `.` does in Java and Javascript.



```php
<?php

$droid1 = new droid('astromech');

$droid1->setName('R2D2');

echo $droid1->name;
```

Objects

- Special `__construct()` method

- This method is called and passed any parameters when being instantiated via the `new` keyword.

```php
<?php

class droid
{
  $type = "";

  function __construct($setType) {
    $this->type = $setType;
  }
}

$droid1 = new droid('protocol');
$droid2 = new droid('astromech');
```

Control Structures

- if .. else
- for
- foreach
- while
- continue
- break

if … elseif … else

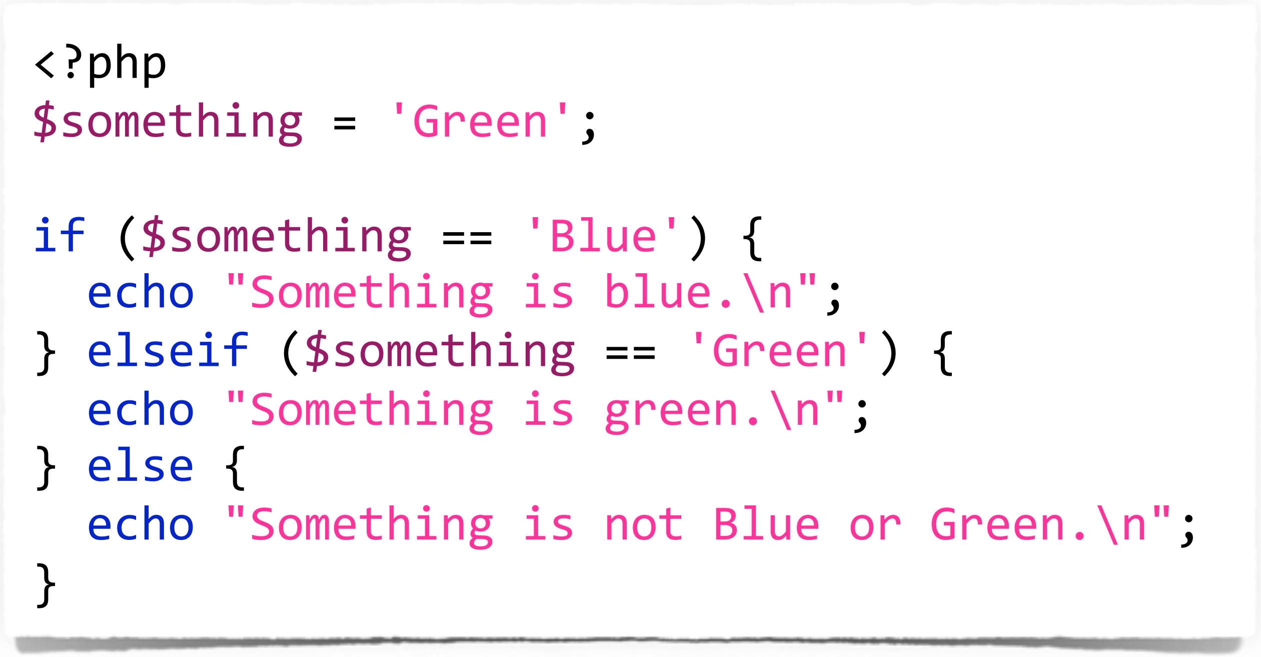
- Basic branching logic.
- If an expression is TRUE, do one thing, otherwise do something **else**

```php
<?php
$expression = false;

if ($expression == true) {
  echo "Something is true.\n";
} else {
  echo "Something is false.\n";
}
```

http://php.net/manual/en/control-structures.elseif.php

if … elseif … else

- Can test multiple conditions with the **`elseif`** keyword
- It's all one word – **`elseif`** not two words
- ~~`else if`~~



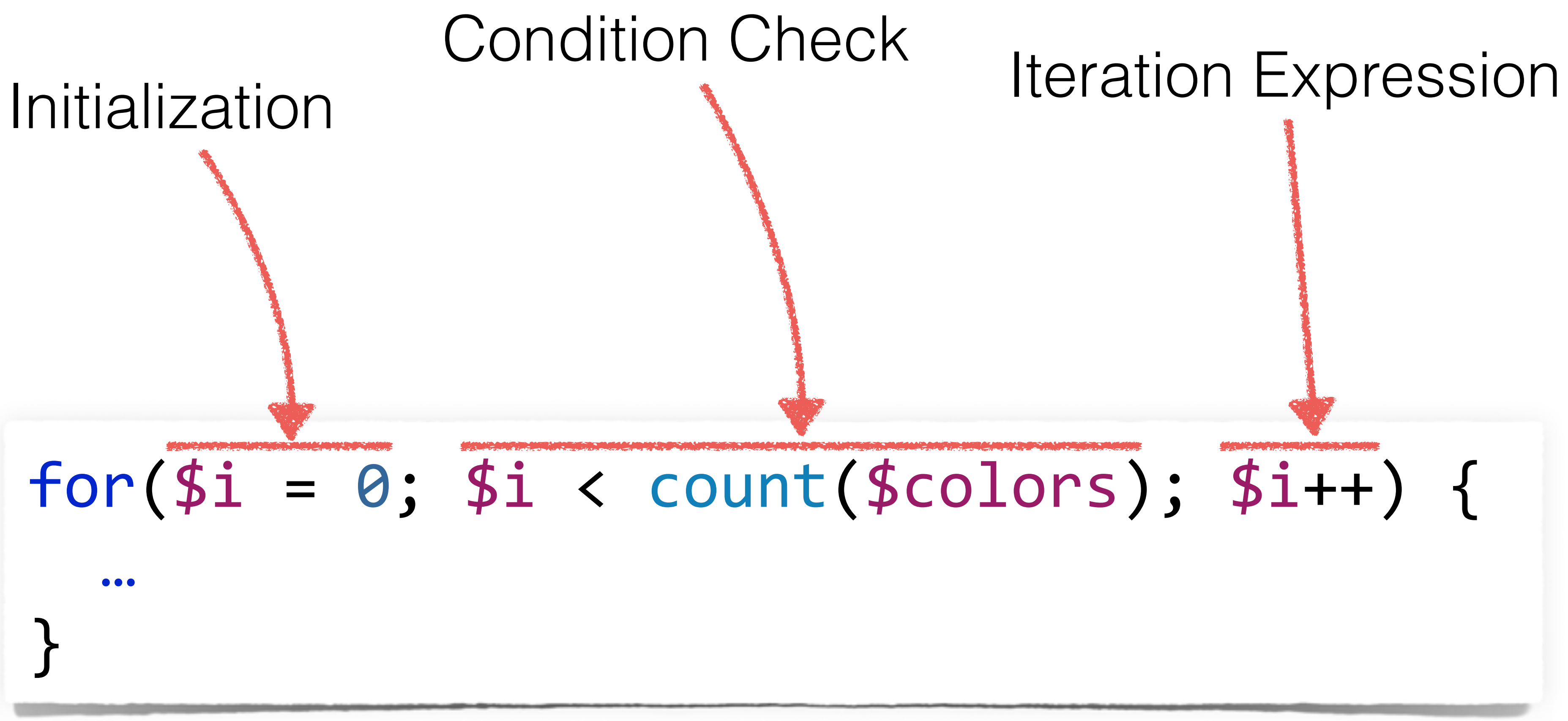
```php
<?php
$something = 'Green';

if ($something == 'Blue') {
  echo "Something is blue.\n";
} elseif ($something == 'Green') {
  echo "Something is green.\n";
} else {
  echo "Something is not Blue or Green.\n";
}
```

for (;;) { }

- Basic C style for loop

```php
<?php
$colors = array("red", "orange", "yellow");

for($i = 0; $i < count($colors); $i++) {
  echo "Color: " . $colors[$i];
}
```

`for (;;) { }`



```
for($i = 0; $i < count($colors); $i++) {
   …
}
```

http://php.net/manual/en/control-structures.for.php

foreach()

- Do something ***for each*** element in a collection

```php
<?php
$colors = array("red", "orange", "yellow");

foreach($colors as $c) {
  echo "Color: $c\n";
}
```

foreach()

- Works on all types of keys, not just numerical

```php
<?php
$person = array(
   "name"     => "Mark Fischer",
   "role"     => "Instructor"
);

foreach($colors as $key => $val) {
   echo "$key: $val\n";
}
```

while()

- Keep doing something until a condition is false

```php
<?php
$fh = fopen('somefile.txt', 'r');

while ($line = fgets($fh)) {
  doWorkOn($line);
}

fclose($fh);
```

continue

- Stop this iteration of a loop, and go on to the next iteration

```php
<?php
$people = array(
  array("name" => "Mark Fischer","role" => "Instructor"),
  array("name" => "Margrit McIntosh","role" => "Student"),
  array("name" => "Michale Hirst","role" => "Student"),
);

// Echo only students
foreach($people as $p) {
  if ($p['role'] == "Instructor") {
    continue;
  }

  echo $p['name'] . "\n";
}
```

break

- Stops all iterations of a loop

```php
<?php
$numbers = range(0, 100);

$numEvens = 0;
foreach($numbers as $n) {
  echo $n . "\n";

  if ( ($n % 2) == 0 ) {
    $numEvens++;
  }

  if ($numEvens >= 5) {
    break;
  }
}
```

Troubleshooting

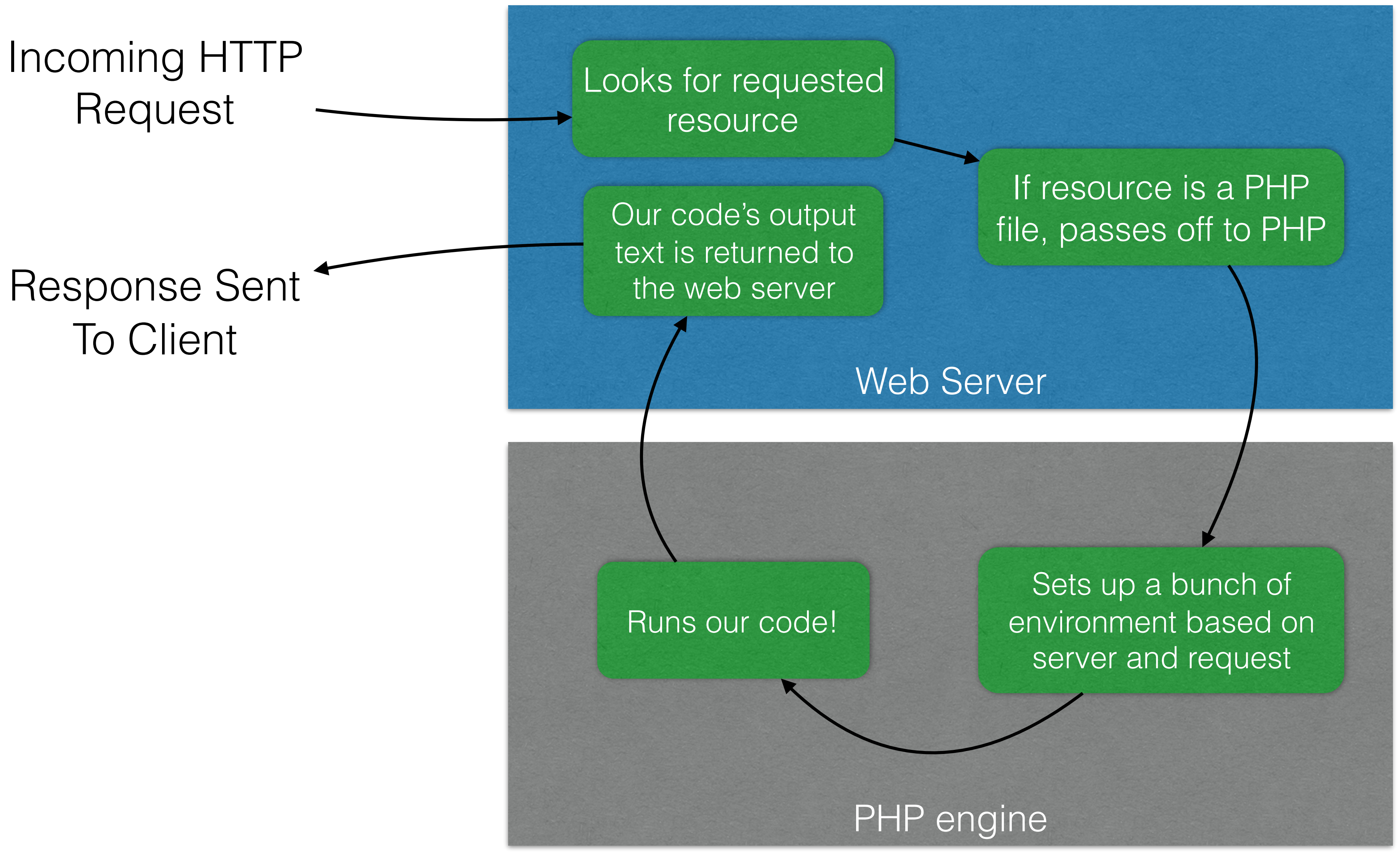
- White Screen of Death
- Error Reporting
- Display Errors

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

PHP Sessions

- One way to solve the stateless nature of the Web
- Each Request is an isolated event
- How do we keep track of people between page views?

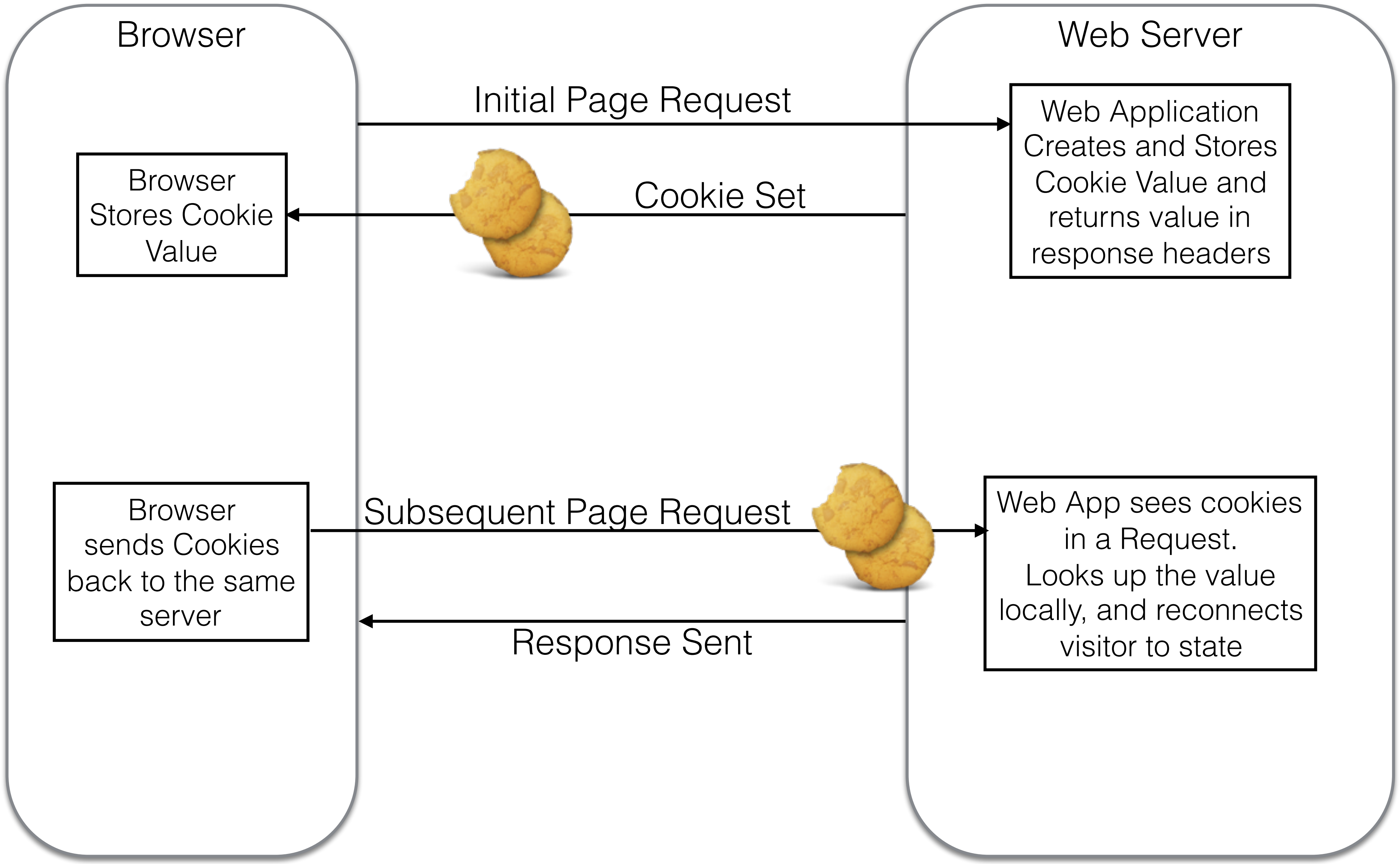
Web Servers and PHP


Incoming HTTP Request
Looks for requested resource
If resource is a PHP file, passes off to PHP
Our code's output text is returned to the web server
Response Sent To Client
Web Server
Runs our code!
Sets up a bunch of environment based on server and request
PHP engine

Cookies

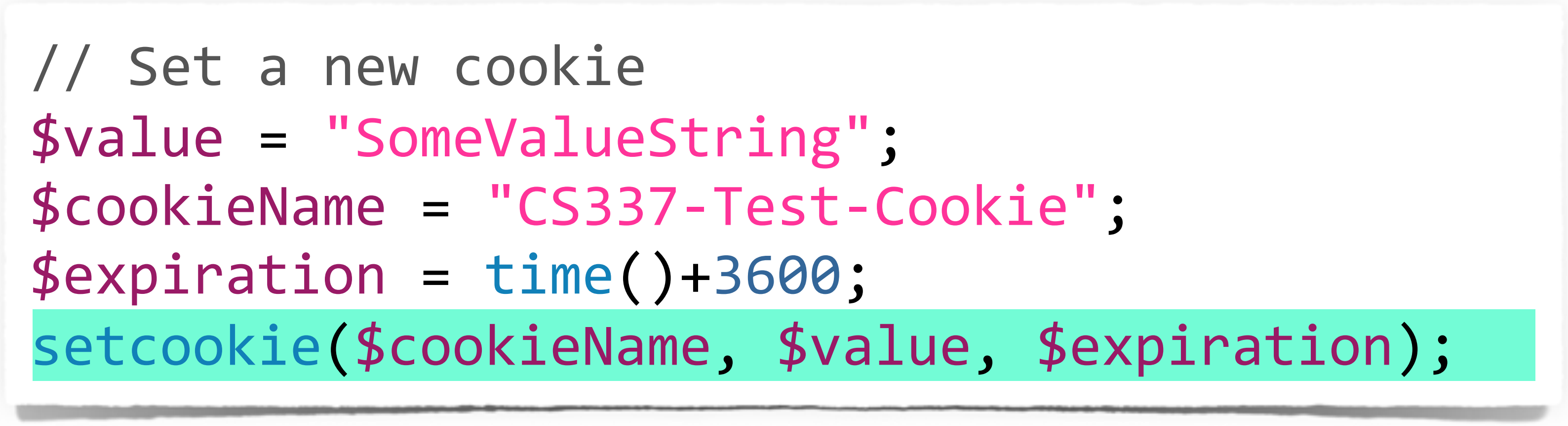
- Web browsers allow sites to store small bits of information – cookies – locally on our computers
- Cookies are sent to the browser as part of the HTTP response headers
- Sent back to the server on subsequent requests
- The server keeps track of who has which cookie ID, and can keep track of visitors.

Cookies


Browser
Web Server
Initial Page Request
Web Application Creates and Stores Cookie Value and returns value in response headers
Cookie Set
Browser Stores Cookie Value
Browser sends Cookies back to the same server
Subsequent Page Request
Web App sees cookies in a Request. Looks up the value locally, and reconnects visitor to state
Response Sent

PHP Cookies

- PHP has a **`setcookie()`** function that handles the details of constructing a properly formatted **`Set-Cookie`** response header.



```php
// Set a new cookie
$value = "SomeValueString";
$cookieName = "CS337-Test-Cookie";
$expiration = time()+3600;
setcookie($cookieName, $value, $expiration);
```

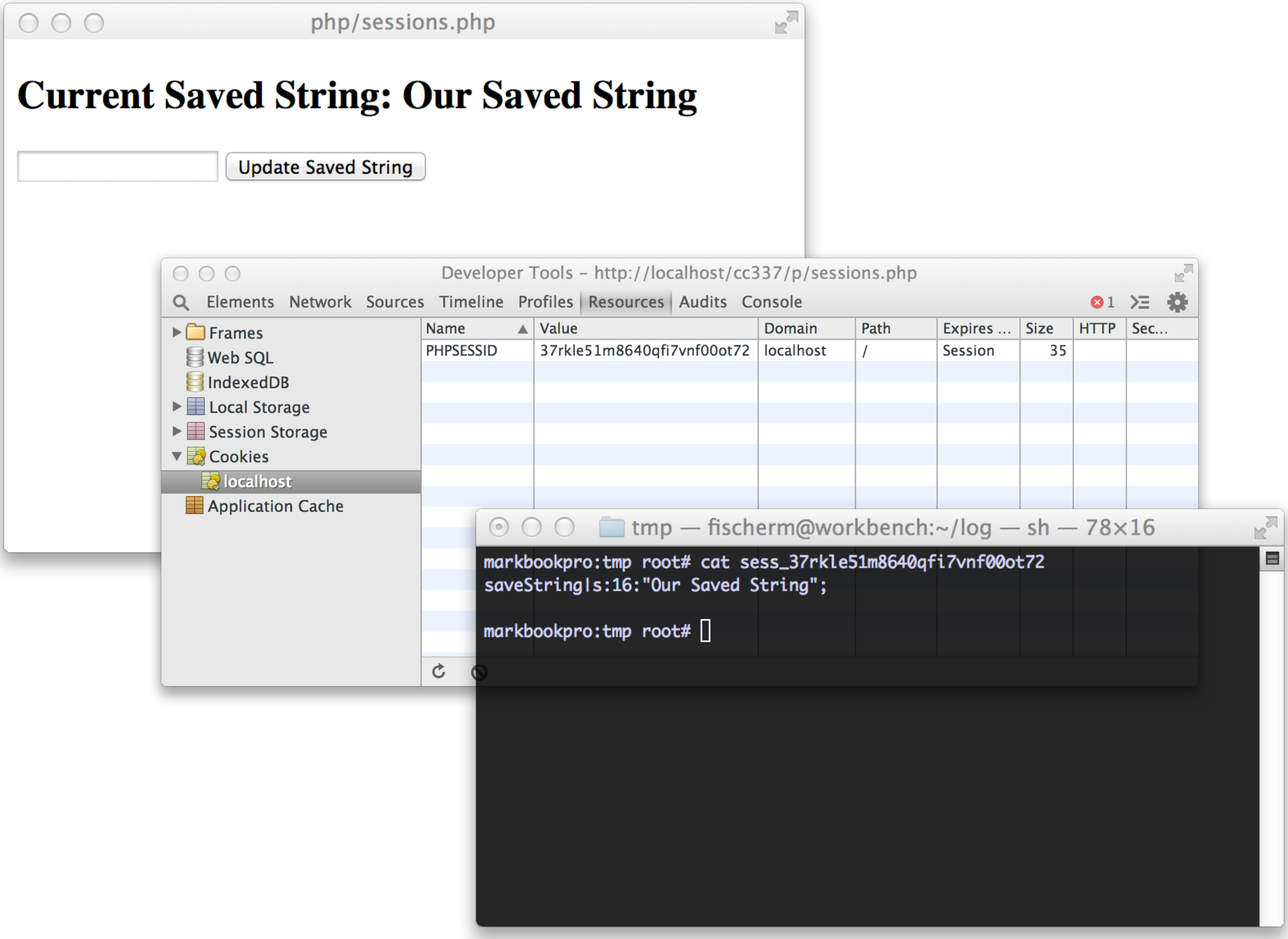
`http://localhost/cc337/p/cookies.php`

PHP Sessions

- PHP has a session handling system built in.
- Based on cookies, and server-side file storage by default.
- Beginning a PHP session sets a cookie on the client.
- That cookie is then used to retrieve locally stored data from the server, and present it in the `$_SESSION` superglobal.

PHP Sessions Example

```php
<?php
session_start();

// If we're POSTing to this page, its probably a form update
if (!empty($_POST)) {
  $newSavedString = $_POST['saveString'];
  $_SESSION['saveString'] = $newSavedString;
  // Redirect to the page via GET to fix the back button issue
  header('Location: sessions.php');
}
?>
<!doctype html>
<html>
<head>
  <title>php/sessions.php</title>
</head>
<body>
  <section>
    <h2>
      Current Saved String: <?php echo $_SESSION['saveString']; ?>
    </h2>
  </section>

  <section>
    <form action="sessions.php" method="POST">
      <input name="saveString" type="text">
      <input type="submit" value="Update Saved String">
    </form>
  </section>

</body>
</html>
```



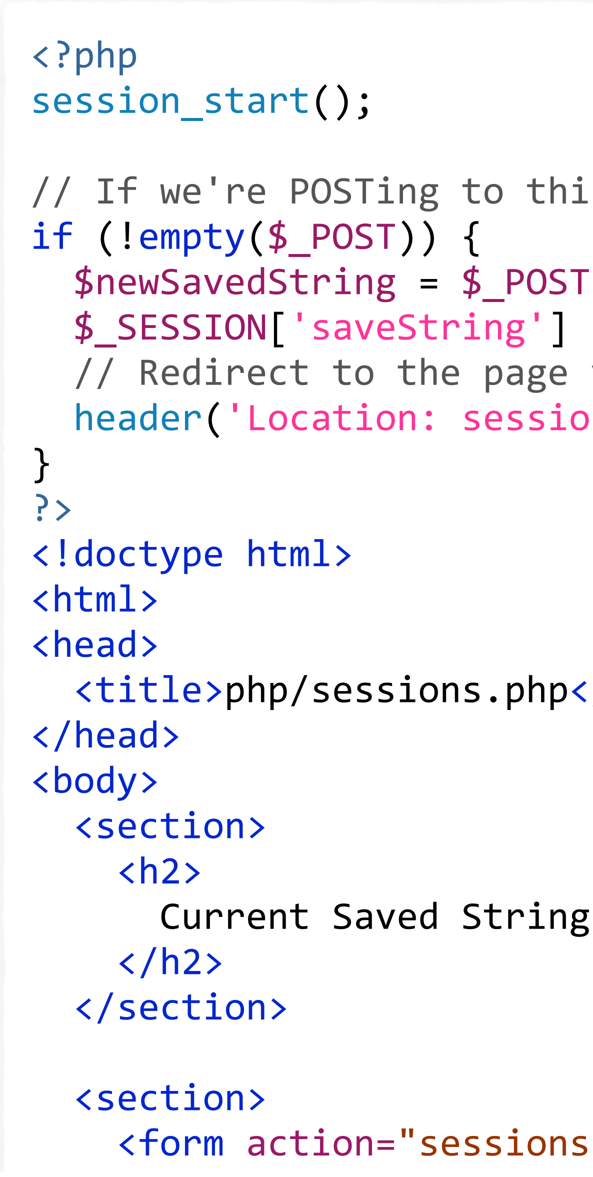
Current Saved String: Our Saved String

Name	Value	Domain	Path	Expires ...	Size	HTTP	Sec...
PHPSESSID	37rkle51m8640qfi7vnf00ot72	localhost	/	Session	35		

```
markbookpro:tmp root# cat sess_37rkle51m8640qfi7vnf00ot72
saveString|s:16:"Our Saved String";

markbookpro:tmp root# 
```

PHP Sessions

- *MUST* call **`session_start()`** before sending *ANY* response to the browser.
- Once the server begins sending text back to the browser, all headers must be sent first.
- Since sessions depend on cookies, the cookie must be sent along with the response headers, before any content.



```
<?php
session_start();

// If we're POSTing to thi
if (!empty($_POST)) {
  $newSavedString = $_POST
  $_SESSION['saveString']
  // Redirect to the page
  header('Location: sessio
}
?>
<!doctype html>
<html>
<head>
  <title>php/sessions.php<
</head>
<body>
  <section>
    <h2>
      Current Saved String
    </h2>
  </section>

  <section>
    <form action="sessions
```

PHP Sessions

- What Can I keep in `$_SESSION` ?
- Any *serializable* value
  - Scalars (int, float, string, bool, etc)
  - Arrays – As long as all array elements are also serializable
  - Objects – Again, as long as all properties are serializable

PHP Sessions

- What isn't allowed in `$_SESSION` ?
- Mostly resources:
  - Open file handles
  - Network sockets
  - Streams
- Closures
- Some objects – Any objects with references to non-serializable things

Go Talk About MySQL