# PHP

CSC 337, Fall 2013
The University of Arizona
William H. Mitchell
whm@cs

## Big picture

HTML – Hypertext Markup Language

Specifies structure and meaning

CSS – Cascading Style Sheets

Specifies presentation

PHP – PHP: Hypertext Preprocessor

One of many back-end languages popular for generating HTML and CSS.

JavaScript

Lets us write code that runs on the front-end (i.e, in the browser.)

## Background

## What is PHP?

"PHP is a popular general-purpose scripting language that is especially suited to web development."—php.net

Recursive acronym: "PHP: PHP Hypertext Preprocessor"

w3techs.com reports that PHP is used on 81.2% of all websites

php.net claims PHP "is installed on" 244 million websites

PHP is used by Facebook, Wikipedia, Twitter and many other large sites.

PHP underlies many Content Management Systems including Drupal and WordPress.

## PHP: Good news and bad news

phpsadness.com

Comedy Central for language designers

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

—Bjarne Stroustrup (the creator of C++)

Confession: PHP is the newest language in my toolbox.

My current opinion: "Lots of dopey stuff, too many must-know details, but sorta fun to use. Lots of influence from C but has automatic memory management."

## Quick history

1994: Rasmus Lerdorf writes some Perl CGI (Common Gateway Interface) scripts for his personal homepage.

1995: Lerdorf announces *Personal Home Page Tools (PHP Tools) version 1.0*, an evolved rewrite in C of those tools.

May 1998: 60,000 sites using PHP (about 1% of all sites.)

June 1998: PHP 3; first release that resembles today's PHP.

2004: PHP 5; improved support for OO programming

Later releases: PHP 5.3—2009, 5.4—2012, 5.5—2013

PHP 6 is on indefinite hold (but there are PHP 6 books!)

More at php.net/manual/en/history.php.php

## PHP.net

PHP.net is the official PHP website and is the primary resource for PHP. It is maintained by the PHP Group.

My "search engines" (browser keywords) for PHP:

| | |
|---|---|
| pm | php.net/manual/en/ |
| ps | php.net/results.php?q=%s |
| pf | php.net/manual/en/function.%s.php |
| pt | php.net/manual/en/language.types.%s.php |
| pops | php.net/manual/en/language.operators.php |
| psf | php.net/manual/en/ref.strings.php |
| pn | php.net |

## PHP books

I still haven't found a PHP book I like even a tenth as much as HFHC. But having said that...

*Programming PHP*, 3rd edition by Tatroe, MacIntyre and Lerdorf

*PHP and MySQL Web Development*, 4th edition, by Welling and Thomson

*Learning PHP, MySQL, JavaScript, and CSS*, 2nd edition, by Nixon

*Head First PHP & MySQL* by Beighley and Morrison

## Interacting with PHP

## PHP's interactive mode

PHP's interactive mode, invoked with `php -a`, provides an environment to interactively evaluate PHP expressions.  It's great for learning and experimenting.

Sadly, PHP's interactive mode doesn't work on Windows with standard PHP distributions, which lack "readline" support.

If you've got a Windows machine, use PuTTY to login on lectura and use `php -a` there.

On Mac OS X, just run `php -a` in a Terminal or iTerm window.

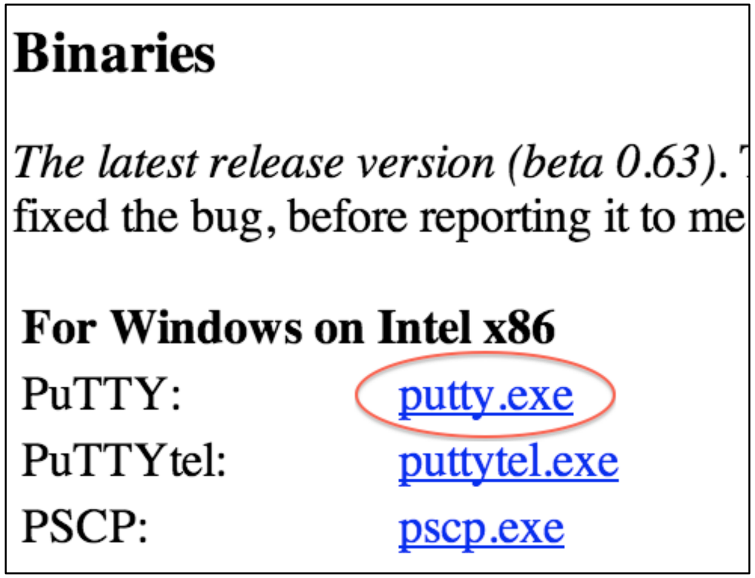
## Sidebar: Getting and running PuTTY

If you Google for "putty", the first hit should be this:

PuTTY Download Page

[www.chiark.greenend.org.uk/~sgtatham/putty/download.html](http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html)

Download putty.exe. It's just an executable—no installer!



**Binaries**

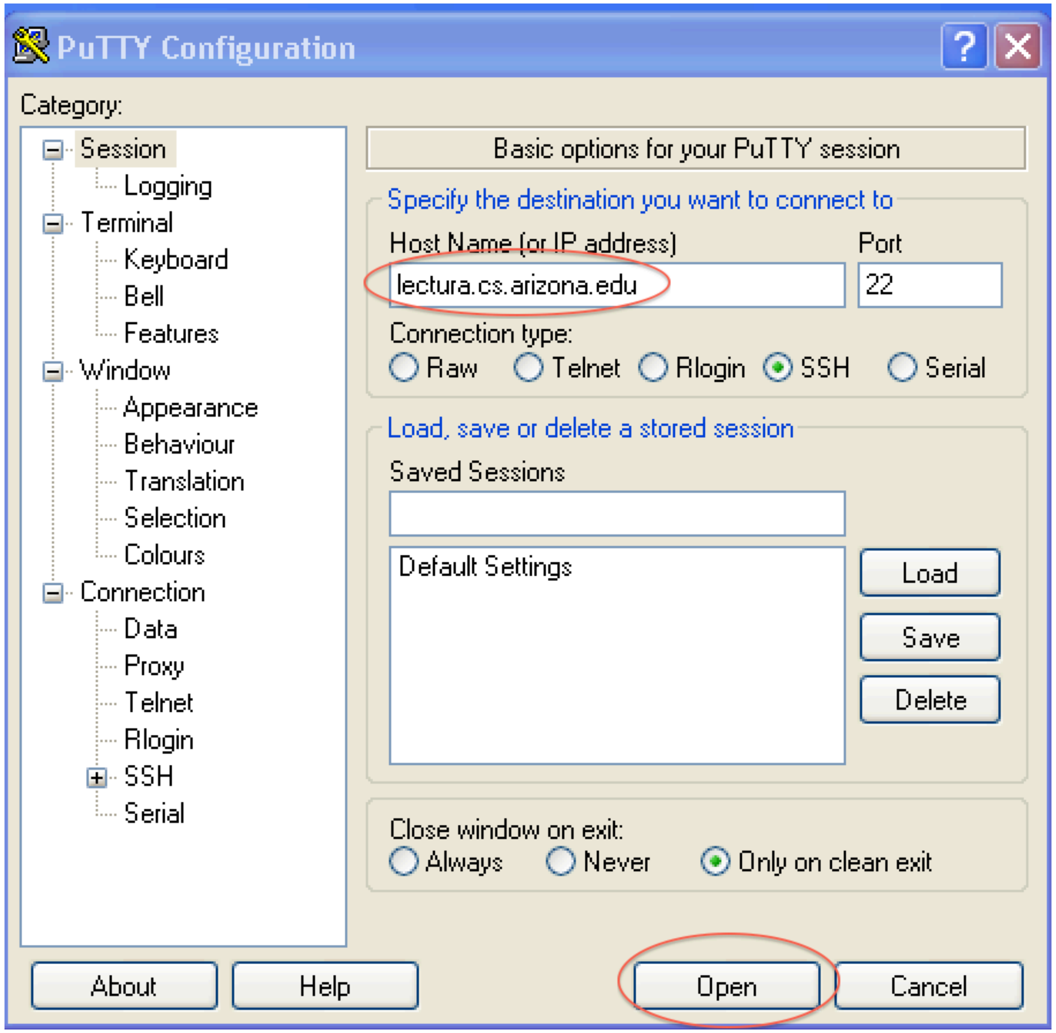
*The latest release version (beta 0.63).*
fixed the bug, before reporting it to me

**For Windows on Intel x86**

| | |
|---|---|
| PuTTY: | [✓ putty.exe] |
| PuTTYtel: | puttytel.exe |
| PSCP: | pscp.exe |

# PuTTY, continued

Click on putty.exe to run it. In the dialog that opens, fill in lec.cs.arizona.edu for Host Name and click Open.


PuTTY Configuration

Category:

- Session
  - Logging
- Terminal
  - Keyboard
  - Bell
  - Features
- Window
  - Appearance
  - Behaviour
  - Translation
  - Selection
  - Colours
- Connection
  - Data
  - Proxy
  - Telnet
  - Rlogin
  - SSH
  - Serial

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): lectura.cs.arizona.edu

Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Load, save or delete a stored session

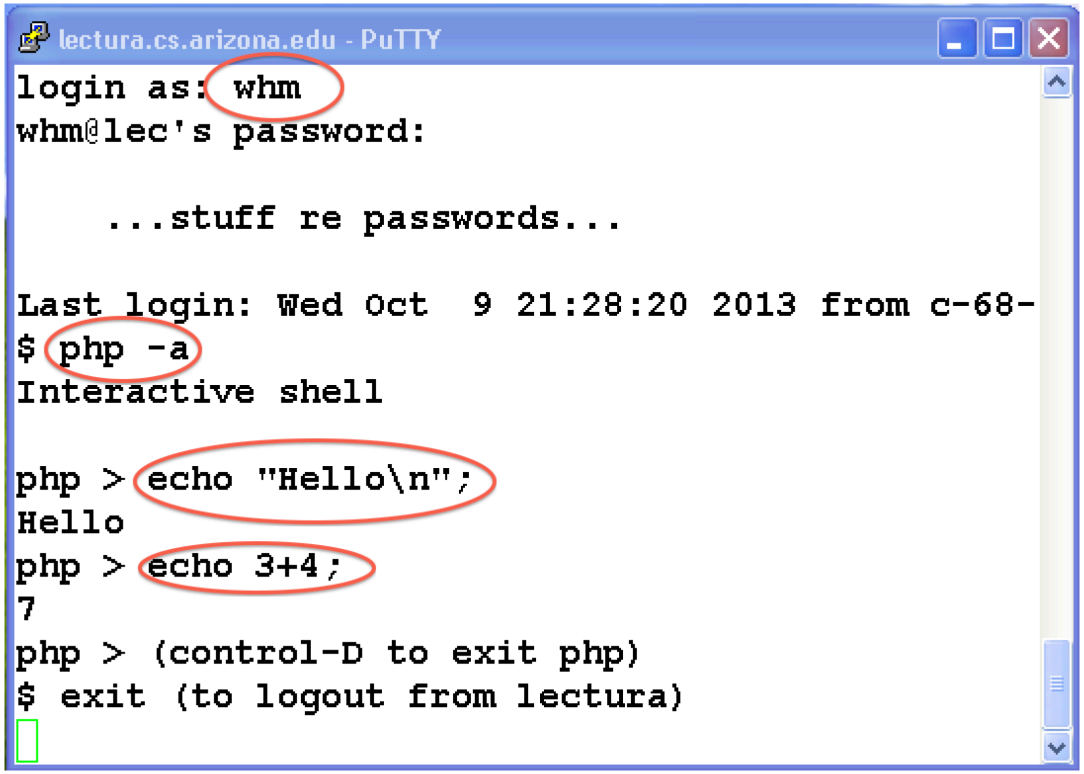
Saved Sessions

Default Settings

Close window on exit: Always Never [✓ Only on clean exit]

## `php -a` on lectura

Login to lectura using your netid. Run `php -a`, and try a couple of `echo` statements:



```
login as: whm
whm@lec's password:

    ...stuff re passwords...

Last login: Wed Oct  9 21:28:20 2013 from c-68-
$ php -a
Interactive shell

php > echo "Hello\n";
Hello
php > echo 3+4;
7
php > (control-D to exit php)
$ exit (to logout from lectura)
```

Go to http://cs.arizona.edu/computing/services and use "Reset my forgotten Unix password" if needed.

## Extra Credit Assignment 1

Due: Monday, October 14 at 2:45pm
Worth: 3 points
Why: To get you up and running with `php -a`

What:

Use `php -a` to do ten `echo` commands with some amount of variety among them. (Hint: `echo 1; echo 2;` ... will be worth about zero points.)

Capture the interaction as text, save it as a file named php.txt and turn in that file (no zip!) via the eca1 D2L Dropbox. <u>Note: Text only! No screenshots!</u>

## Interactive mode

`php -a` starts an interactive mode. Statements are executed as they are entered.

One way to see the value an expression produces is to use the `echo` statement:

```
% php -a
Interactive shell
php > echo 1 + 2;
3

php > echo 1, 2, 3;
123
```

Don't forget the semicolon!!

## Some basics

## Arithmetic operators

The arithmetic operators are about what you'd expect.

```
php > echo 1+2, " ", 3*4;
3 12

php > echo 5-6, " ", 7 % 2;
-1 1

php > echo 4/3, "   ", 120/3;
1.3333333333333   40
```

Any surprises?

php.net/manual/en/language.operators.arithmetic.php

## Variables

Variable names are always preceded with a dollar sign.

```
php > $width = 10;
php > $height = 20;
php > $area=$width*$height; # spaces not needed
php > echo $area;
200
php > echo $area + $arae;
200
```

Undefined variables default to context-based values.

Variable names can start with letters and underscores, and can contain digits after the first character.

There are no variable declarations (like `int x` in Java.)

## Sidebar: error_reporting() – a PHP "knob"

Note this difference in behavior between lectura...

```
% php –a
php > $x=$y;
Notice:  Undefined variable: y in php shell code on line 1
php > echo error_reporting();
22527
```

and my Mac...

```
% php –a
php > $x=$y;
php > echo error_reporting();
22519
php > error_reporting(22527);  # adjust it! (E_NOTICE on)
php > $x=$y;
Notice: Undefined variable: y in php shell code on line 1
```

## Variables have no type

In languages like Java and C, variables are declared to have a type, like char, int [], String, List, and Object.

When a program is compiled, the compiler ensures that all operations are valid with respect to the types involved.

<u>Variables in PHP do not have a type. Instead, type is associated with values.</u>

In Java you say, "What's the type of `x`?"

In PHP you say, "What's the type of the value held by `x`?"

## Variables have no type, continued

The `gettype()` function returns the type of a value.

```
php > echo gettype(2/3);
double

php > $x = 2/3;
php > echo gettype($x);
double

php > $x = 7;
php > echo gettype($x);
integer

php > echo gettype($x/3);
double
```

## Characterizing type-checking

Some people use the term "strongly typed" to characterize languages like Java and C, and "weakly typed" to characterize languages like PHP and Python but the merit of those terms is debatable.

A better way to characterize a language is to say <u>when</u> type checking is done.

- Java does compile-time type checking and some run-time type checking, too.
- C does compile-time type checking but absolutely no run-time type checking.
- PHP and Python do only run-time type checking.

## The `string` type

PHP has a `string` type. Literals can be enclosed in single- or double-quotes

The concatenation operator is dot.

```
php > echo 1, "...", 2, "...", 3;
1...2...3


php > echo "abc" . 'xyz';
abcxyz
```

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

## String literals

Literals can be enclosed in single- or double-quotes but the only backslash escapes recognized in single-quotes are \' and \\.

```
php > echo "Test\nthis\x21";
Test
this!

php > echo 'Test\nthis\x21';
Test\nthis\x21

php > echo 'How\'s\nthis?';
How's\nthis?
```

## `string`, continued

Individual characters in a string can be accessed with zero-based *offsets* enclosed in square brackets.

If an offset is out of bounds, an empty string is returned.

```
php > $s = "abcdef";
php > echo $s[2];
c
php > echo gettype($s[2]);
string
php > echo $s[10];
php > echo strlen($s[10]);
0
```

Anything notable?

## `string`, continued

PHP strings are mutable!

```
php > $s = "abcd";
php > $s[0] = "X";
php > echo $s;
Xbcd

php > $s[1] = "YYY";   # Note: only $s[1] is changed
php > echo $s;
XYcd

php > $s[2] = "";
php > echo $s;          # ~~Assigning "" deletes!~~ WRONG!
XYd
```

## `string` functions

Almost 100 string functions are described in
[http://php.net/manual/en/ref.strings.php](http://php.net/manual/en/ref.strings.php)

```
php > echo substr("abcdef", 2, 4);
cdef

php > echo strtoupper("hey!");
HEY!

php > echo rtrim("huh?!?", "!?");
huh

php > echo htmlentities("<&>");
&lt;&amp;&gt;

php > echo str_replace("to", "2", "tomato");
2ma2

php > echo json_encode("\x61\x62\x63\x09\xa\x21");
"abc\t\n!"
```

[added post-handouts]

## Variable expansion in string literals

If a double-quoted literal contains a $, the following characters are treated as a variable name. The value of the variable is inserted. Braces can be used to delimit the name.

```
php > $x = 10; $y = 20;

php > echo "x: $x; $ym in length";
x: 10;  in length

php > echo "x: $x; {$y}m in length";
x: 10; 20m in length
```

Was ${y}m

Idiom: Use variable expansion, not a bunch of concatenations to form a string from many values.

## The `boolean` type

The literals for PHP's `boolean` type are `tRUe` and `FalSE` and are <u>case insensitive</u>.

Comparison and logical operators produce boolean values

Note how booleans are printed and converted to strings:

```
php > $f = false; $t=true;
php > echo $t, $f;
1
php > echo false;
php > echo "false: $f, true: $t";
false: , true: 1
php > var_dump($f);~~echo var_export($f)~~
bool(false)    # why not boolean(false)?
```

## Several ways to be `false`

Along with the keyword `false`, PHP considers these values to be false, too:

- `integer` and `double` zeros (`0` and `0.0`)
- The empty string (`""`)
- The string "0"
- An array with no elements
- NULL
- SimpleXML objects created from empty tags

<u>Every other value is TRUE.</u> (Lots and lots of ways to be true!)

## Comparison operators

These comparison operators are somewhat similar to their counterparts in Java, C, and Python:

```
<     >     <=     >=     ==    !=
```

Some conversions ("type juggling") are surprising:

```
php > ~~echo~~ var_export("00" == "000");
true
php > ~~echo~~ var_export(20 < 100);
true
php > ~~echo~~ var_export("20" < "100");
true
php > var_export(strcmp("20","100"));
1
```

## Equality

Some PHP programmers never use the == or != operators and instead use === and !==, "identical" and "not identical".

```
php > ~~echo~~ var_export("00" === "0");
false
php > echo var_export(1 === true);
false
php > echo var_export(1 === 1.0);
false
php > echo var_export(false === "");
false
php > echo var_export("10" !== 10);
true
```

## Full programs

Unlike Java but like Python, you don't need ~~any~~ much boilerplate to have a runnable PHP program.

```
% cat hello.php          (> type hello.php on Windows)
<?php
echo "Hello, world!\n";

% php hello.php
Hello, world!
```

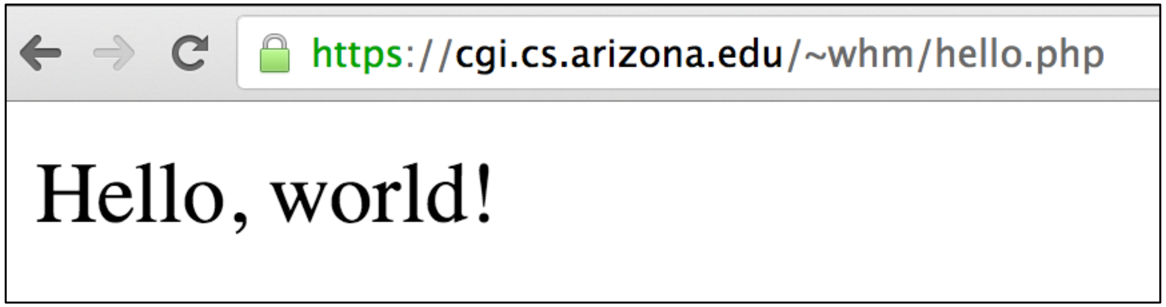
`echo` writes to standard output: [added post-handouts]

```
% php hello.php > x
% cat x
Hello, world!
```

## PHP hits the web

The web server on lectura supports PHP. If we ask for a file with a .php extension, we get not the contents of that file but the data written to standard output when that file is run as a PHP program. [reworded post-handouts]

```
% cat hello.php
<?php
echo "Hello, world!\n";
% scp hello.php lec.cs.arizona.edu:/cs/cgi/people/
whm/public_html/.
```


https://cgi.cs.arizona.edu/~whm/hello.php

Hello, world!

How does the URL differ from the file name? What's the mapping?

## Extra Credit Assignment 2

Due: ~~??~~ Wednesday, October ~~??~~ 16 at 2:45pm
Worth: 3 points
Why: To get you to put some PHP on the web

What:

Put a file named eca2.php in place on lectura such that hitting
https://cgi.cs.arizona.edu/~YOUR-NETID/eca2.php
runs it. It must produce at least one byte of output due to execution of a PHP statement but feel free to have it do more.

There's nothing to "turn in"—we'll have a script hit the URL for every student. Note: Everybody on the net will be able to hit it, too, so don't have it print your SSN or anything like that!

Some students may run into problems with permissions and other thing. If so, don't panic but let us know ASAP.

## Sidebar: Copying a file to lectura

Mac OS X: Just use scp like I did.

Windows:

1. Get pscp.exe from the same place you got PuTTY.

2. Copy pscp.exe into the directory with your PHP files.

3. c:\...> pscp hello.php NETID@lec.cs.arizona.edu:/cs/cgi/people/NETID/public_html/.

Or...

Get WinSCP and read the instructions. Along with simple copying there's Commands>Keep Remote Directory Up To Date, which is very handy!

## `while` loops

## The `while` loop

The ***general form*** of a while loop is just like Java and C:

```
   while (expression)
      statement
```

Like Python but unlike Java, the value  produced by *`expression`* is permitted to be of any type—`int`, `boolean`, `string`, and more!

Like Java and C, statement can be a single statement terminated by a semicolon, or a compound statement grouping zero or more statements in curly braces.

Pythoners: Indentation does not matter in PHP!  Note also that *`expression`* must be enclosed in parentheses.

## `while`, continued

For reference:

```
   while (expression)
      statement
```

Example:

```
  <?php
  $i = 1;
  while ($i <= 10) {
       echo "$i\n";
       $i += 1; // $i=$i+1;
       }
```

```
% php while0.php
1
2
3
4
5
6
7
8
9
10
```

What's *`expression`*?  What's its type?

What's *`statement`*?

## `while`, continued

Here is while1.php. Is it valid? If so, what does it do?

```
<?php

$i = 10;

while ($i) {
    echo "$i\n";
    $i -= 1;
    }
```

What would the behavior be if we left off the braces?

## `while`, continued

What does this program do?

```
<?php
echo "
<!doctype html>
<title>Countdown</title>
<ul>";

$i = 10;
while ($i) {
    $em = 1 + $i/10;
    echo "<li style=font-size:{$em}em>$i";
    $i -= 1;
    }
echo "</ul>"; // while2.php
```

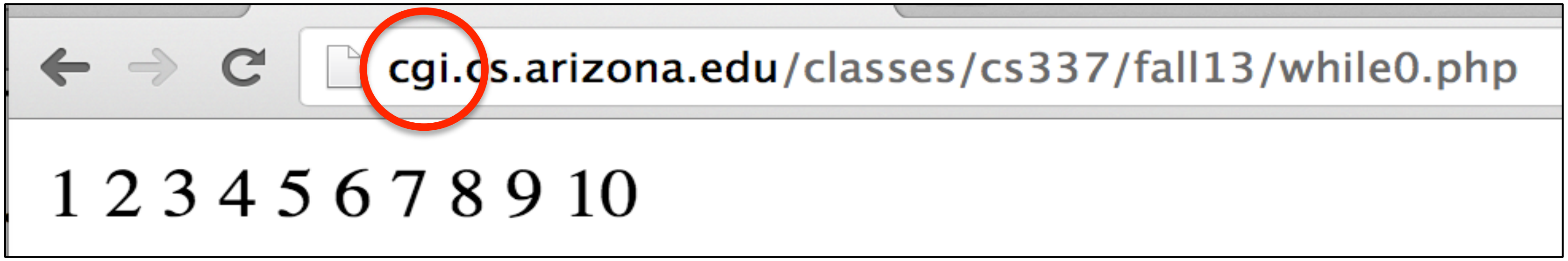
☜ String literal split across lines. Later we'll see a more idiomatic construct.

## Sidebar: Where are the examples?

Hitting

http://**cgi**.cs.arizona.edu/classes/cs337/fall13/while0.php

runs while0.php and displays its standard output:



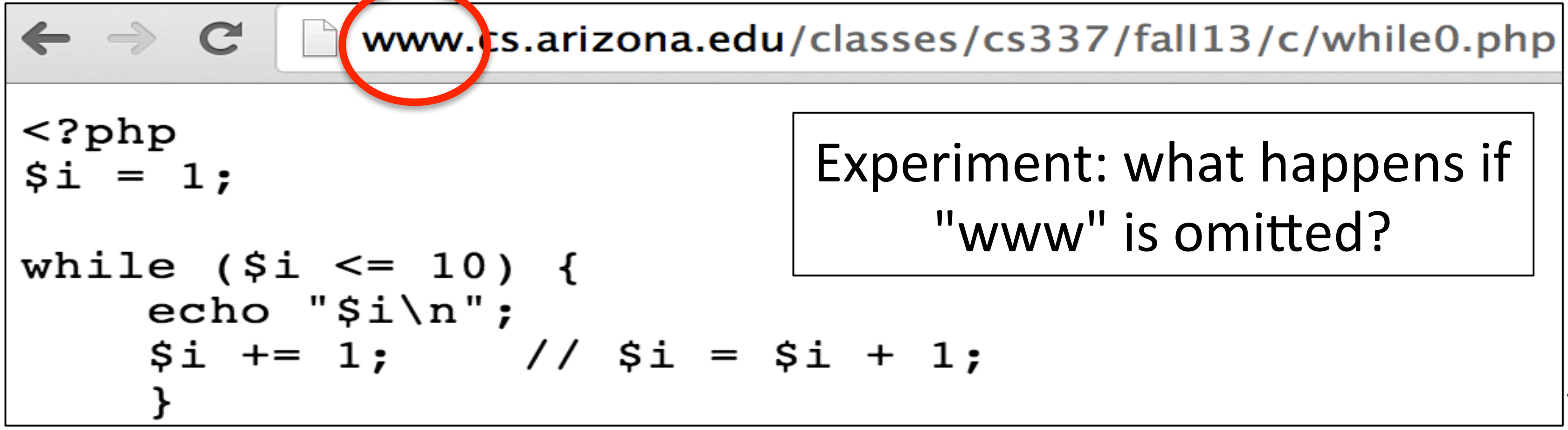
cgi.cs.arizona.edu/classes/cs337/fall13/while0.php

```
1 2 3 4 5 6 7 8 9 10
```

That URL corresponds to this path on lectura:

/cs/cgi/classes/cs337/fall13/while0.php

On lectura, /cs/www/classes/cs337/fall13/c is a *symlink* to /cs/cgi/classes/cs337/fall13. Note what this URL shows:



www.cs.arizona.edu/classes/cs337/fall13/c/while0.php

```
<?php
$i = 1;

while ($i <= 10) {
    echo "$i\n";
    $i += 1;      // $i = $i + 1;
    }
```

Experiment: what happens if "www" is omitted?

## Reading from files and more

## `fopen` and `fgets`: open a file and read it

The `fopen` function opens a file for reading (or writing).
`fgets` reads lines from a file one line at a time.

```
php > $f = fopen("five.txt", "r");
php > echo gettype($f);
resource
php > var_dump($f);
resource(2) of type (stream)
php > echo fgets($f);
one
php > echo json_encode(fgets($f));
"two\n"          (string has trailing newline)
php > echo json_encode(rtrim(fgets($f)));
"three"          (use rtrim to strip trailing newline)
```

At end of file, `fgets` returns `false`.

## Problem: reverse order of lines

Problem: Write a program that reads lines from the file `five.txt` and prints them in reverse order: last line first, first line last.  (No arrays yet!)

Expected behavior:

```
 % php tacfive.php
five
four
three
two
one
%
```

## Problem: reverse order of lines

tacfive.php:

```
<?php

$f = fopen("five.txt", "r");

$result = "";

while ($line = fgets($f))
    $result = $line . $result;

echo $result;
```

Let's be sure we understand the operation of the `while`'s *`expression`*.

Because no path, like `/w/337/five.txt`, is specified, the program will look for `five.txt` in the current directory.

## `fopen`: Not just for files!

taccities.php: Same as tacfive.php but with this instead:

```
    $f = fopen("http://cs.arizona.edu/classes/cs337/fall13/a1/
cities.txt", "r");
```

Execution:

```
    % php taccities.php
    .
    Blacklight Dodgeball Tournament/...
    Old Train Station:trainstation.jpg/...
    toknc.com
    Gateway to Walkertown
    Kernersville
    ...
    The City of Love
    Paris
    %
```

## Reading from standard input

tac.php reads from standard input using a *constant* named `STDIN`:

```
<?php
$result = "";

while ($line = fgets(STDIN))
    $result = $line . $result;

echo $result;
```

Execution:

```
% php tac.php < five.txt
five
four
three
two
one
%
```

Speculate: What would the following do?

% php tac.php < five.txt | php tac.php

## More control structures

## The `if-else` statement

PHP's if-else statement looks just like Java and C:

```
if (expression)
   statement1
else
   statement2
```

Like PHP's `while` statement, *`expression`* can have any type, not just `boolean`.

The `else` clause is optional.

*`statement1`* and *`statement2`* can each be a single statement or a compound statement, with zero or more statements enclosed in curly braces.

## Example: Computing a mean

```
$sum = $n = $blanks = 0;

while ($line = fgets(STDIN)) {
   $line = trim($line);
   if (strlen($line) == 0)
      $blanks += 1;
   else {
      $sum += $line;
      $n += 1;
      }
   }

if ($n > 0) {
   $mean = $sum / $n;
   echo "mean = $mean; $blanks blank lines\n";
   }
```

```
% php mean.php
5


  2
^Z (control-Z RET on Win)
mean = 3.5; 1 blank lines
% seq 100 | php mean.php
mean = 50.5; 0 blank lines
```

## `else-if` and `elseif`

Nested "else-if"s can be done like in Java or, using `elseif`, which is like Python's `elif`. These two versions are equivalent:

```
while ($line = fgets(STDIN)) {
    $avg = trim($line);
        if ($avg >= 90) {
                $grade = "A";
                }
        else if ($avg >= 80) {
                $grade = "B";
                }
        else if ($avg >= 70) {
                $grade = "C";
                }
        else {
                $grade = "F";
                }
        echo "$avg -> $grade\n";
        }
```

```
while ($line = fgets(STDIN)) {
    $avg = trim($line);
    if ($avg >= 90) {
        $grade = "A";
        }
    elseif ($avg >= 80) {
        $grade = "B";
        }
    elseif ($avg >= 70) {
        $grade = "C";
        }
    else {
        $grade = "F";
        }
    echo "$avg -> $grade\n";
    }
```

elseif3.php and elseif.php, respectively

# `break` and `continue`

`break` and `continue` are just like their counterparts in Java, C, and Python:

> `break` exits the enclosing loop.
>
> `continue` skips the balance of the current iteration of the enclosing loop and begins the next iteration.

## break and continue, continued

This is break1.php. What does it do?

```
<?php

while (true) {
    $line = trim(fgets(STDIN));

    if ($line == ".")
        break;

    if (!is_numeric($line))
        continue;

    while ($line) {
        echo "x";
        $line -= 1;
        }
    echo "\n";
    }
```

## Functions

## Function basics

Like Python but unlike Java, PHP allows functions to be "freestanding"—not associated with a class.

Here's a call to a trivial function, followed by the function's definition:

```
<?php
say_hello();

function say_hello()
{
    echo "Hello, world!\n";
}
```

The keyword `function` must precede the function's name.

A function's definition need not precede a call to it.

## Function basics, continued

Here's a function that takes two parameters and returns a value:

```
<?php
// return $count copies of $string
function repl($string, $count)
{
    $result = "";

    while ($count--)
        $result .= $string;

    return $result;
} // repl.php
```

Usage:

```
php > include("repl.php");
php > echo repl("abc", 2);
abcabc
php > echo strlen(repl("abc", 1000));
3000
php >
```

Observations?

## Function basics, continued

At hand:

```
function repl($string, $count)
{
    $result = "";

    while ($count--)
        $result .= $string;

    return $result;
}
```

Things to note:

- Parameters prefixed with $, just like variables.
- No declaration of parameter or return types.
- If no `return` or just "return;" NULL is returned.
- The function `include($file)` loads code at run-time, but functions can't be redefined.

## Function basics, continued

The type of the value returned by a function can vary!

```
function twice($x)    // twice.php
 {
    if (gettype($x) === "string")
        return $x . $x;
    elseif (gettype($x) == "integer")
        return $x * 2;
    else
        return "huh?";
 }
```

```
php > include("twice.php");
php > echo twice("abc");
abcabc
php > echo twice(7);
14
php > echo twice(7.0);
huh?
```

## PHP uses call-by-value (but...)

Like Java and Python, PHP uses "call-by-value"—changing a scalar parameter in a function doesn't affect the value in the caller.

```
php > function zero($x) { $x = 0; }
php > $a = 10;
php > zero($a);
php > echo $a;
10
```

However, we'll later see a way to make the above work.

## The `global` keyword

The first assignment to, or access of a variable `$x` in a scope causes it to be created ~~in~~ with that scope.

Code outside of a function is considered to in the global scope.

Code inside a function is considered to be in a local scope created that invocation of that function.

The call to `f()` prints `x =` because the `$x` in `f()`, with local scope has not been initialized!

```
% cat global1.php
<?php
$x = "g"; // This $x has global scope
f();

function f()
{
   // This $x has local scope
   echo "x = $x\n";
}

% php global1.php
x =
```

## `global`, continued

The `global` keyword allows a function to declare that it wants to use the instance of a variable that's at global scope.

```
$x = "g";

f();
g();

function f() {
    global $x;
    echo "x = $x\n";
    $x = "from f";
}

function g() {
    global $x;
    echo "x = $x\n";
}
```

Execution:

```
% php global2.php
x = g
x = from f
```

The `global $x` declarations in `f()` and `g()` cause references to `$x` in those functions to reference the instance of `$x` with global scope rather than individual, local scope instances of `$x`.

## Default values for parameters

PHP allows default values to be specified for parameters.

```
function wrap($s, $wrap = "<>")        # wrap.php
{
    return $wrap[0] . $s . $wrap[1];
}
```

Execution:

```
php > echo wrap("abc");
<abc>
php > echo wrap("abc", "()");
(abc)
```

There are several related rules. One is that all parameters without defaults must precede all parameters with defaults.

## Running PHP with XAMPP

## It started with "LAMP"

Over time this "solution stack" for web applications emerged as a popular choice:

**L**inux as the operating system
**A**pache HTTP Server (a.k.a. "httpd")
**M**ySQL as the database
**P**HP/**P**erl/**P**ython as programming languages

All components are Open Source, and there are no licensing fees.

The stack was later ported to Windows and Mac OS X. WampServer and MAMP provide one-step installs of the Apache Server, MySQL, PHP and more.

## XAMPP

XAMPP is an AMP stack that runs on Windows, Mac OS X, Linux, and Solaris. The "X" is for cross-platform.

XAMPP is my current recommendation if you'd like to have an AMP stack on your machine.

Alternatively, you can use cgi.cs.arizona.edu, which is in fact a LAMP stack.

Both need to be secured against access by other students in the class.

XAMPP can be easily secured via firewall settings. Working on cgi.cs.arizona.edu requires .htaccess and .htpasswd files.

## PHP version headaches

php.net shows these stable versions:

5.3.27
5.4.20
5.5.4

cgi.cs.arizona.edu runs 5.3.10 and that's not likely to change soon.

Compromise: I suggest XAMPP 1.8.2, which has PHP 5.4, putting us not too far ahead of cgi.cs.arizona.edu and not too far behind the leading edge.

## Installing XAMPP

XAMPP 1.8.2 installer for Windows:

http://www.apachefriends.org/download.php?xampp-win32-1.8.2-2-VC9-installer.exe

XAMP 1.8.2 installer for Mac OS X:

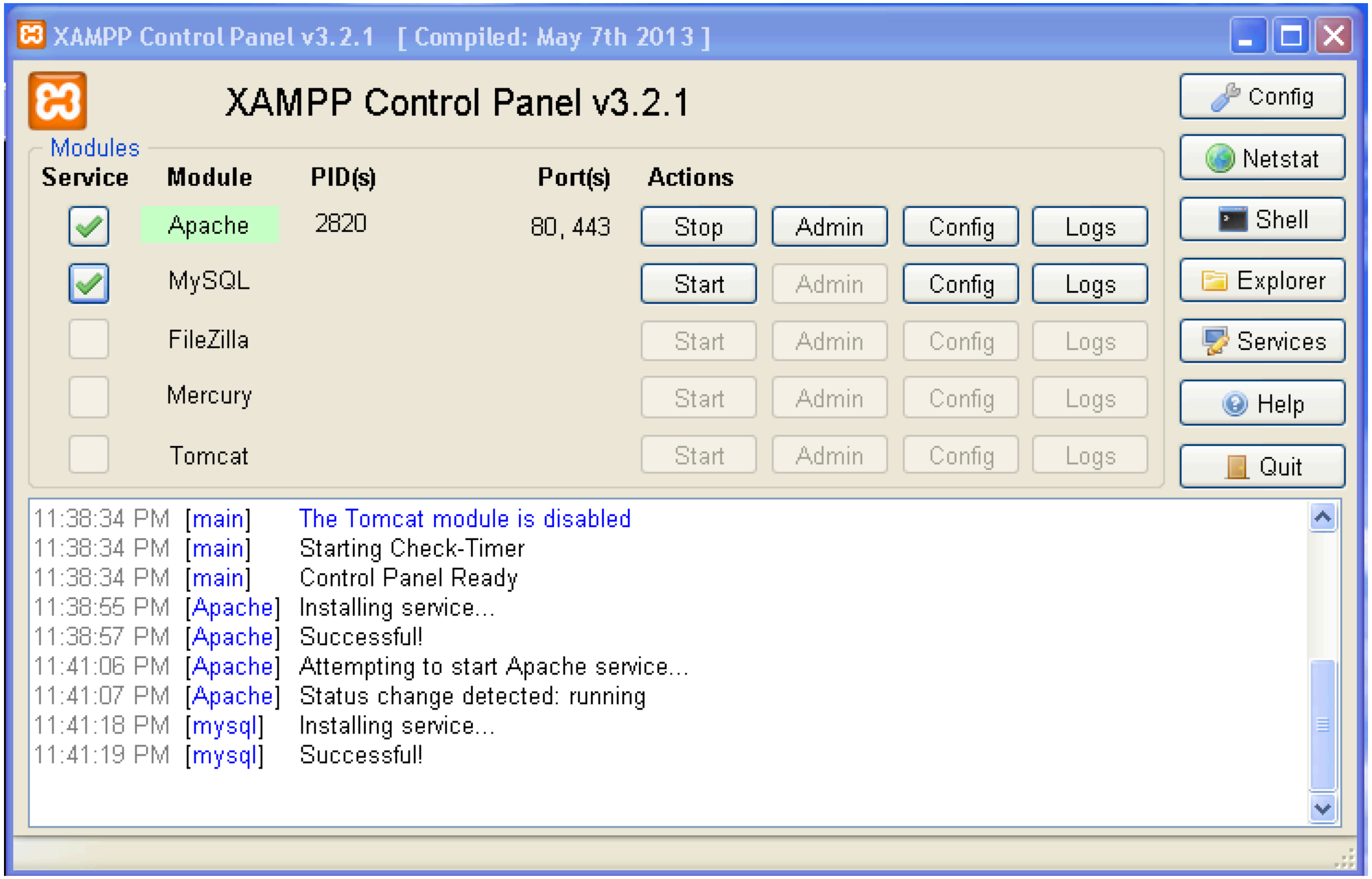
http://www.apachefriends.org/download.php?xampp-osx-1.8.2-2-installer.dmg

You can save space by "unchecking" Tomcat, Perl, FileZilla FTP Server, and Mercury Mail Server.

By default, XAMPP installs to C:\XAMPP on Windows and /Applications/XAMPP on OS X.

# The XAMPP control panel on Windows

The XAMPP control panel below shows that Apache is running and MySQL is stopped. (We don't need it yet.)



XAMPP Control Panel v3.2.1 [ Compiled: May 7th 2013 ]

XAMPP Control Panel v3.2.1

Modules

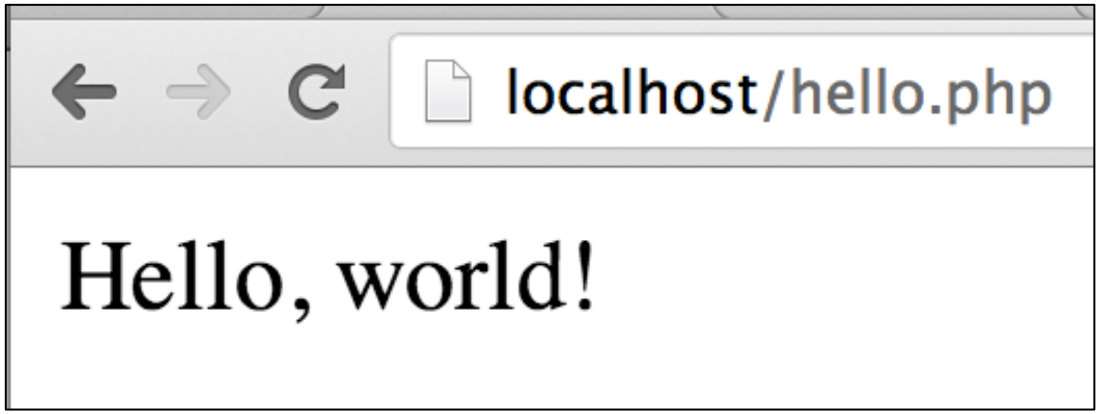
| Service | Module | PID(s) | Port(s) | Actions | | | |
|---|---|---|---|---|---|---|---|
| ✓ | Apache | 2820 | 80, 443 | Stop | Admin | Config | Logs |
| ✓ | MySQL | | | Start | Admin | Config | Logs |
| | FileZilla | | | Start | Admin | Config | Logs |
| | Mercury | | | Start | Admin | Config | Logs |
| | Tomcat | | | Start | Admin | Config | Logs |

```
11:38:34 PM  [main]     The Tomcat module is disabled
11:38:34 PM  [main]     Starting Check-Timer
11:38:34 PM  [main]     Control Panel Ready
11:38:55 PM  [Apache]   Installing service...
11:38:57 PM  [Apache]   Successful!
11:41:06 PM  [Apache]   Attempting to start Apache service...
11:41:07 PM  [Apache]   Status change detected: running
11:41:18 PM  [mysql]    Installing service...
11:41:19 PM  [mysql]    Successful!
```

## The `htdocs` directory

XAMPP's `htdocs` directory is the `DocumentRoot` for Apache. On Windows it is `C:\XAMPP\htdocs`. On OS X it is `/Applications/XAMPP/htdocs`.

If you copy hello.php to htdocs, you can hit it like this:


localhost/hello.php

Hello, world!

If hello.php was in htdocs/337, you'd instead hit localhost/337/hello.php

## Using XAMPP's PHP from the command line

You can use XAMPP's PHP directly on both Windows:

```
W:\337> c:\xampp\php\php --version
PHP 5.4.19 (cli) (built: Aug 21 2013 01:12:03)
...
W:\337> hello.php     (If error, try c:\xampp\php\php hello.php)
Hello, world!
```

And OS X:

```
% /Applications/XAMPP/bin/php --version
PHP 5.4.19 (cli) (built: Aug 26 2013 14:04:00)

% /Applications/XAMPP/bin/php hello.php
Hello, world!
```

It's not hard to set your "path" so you can type just "php" but a number of factors can come into play. Google, or see us during office hours. My old 352 slides on Piazza talk about it.

## Securing /cs/cgi/people/NETID

## Securing /cs/cgi/people/NETID

If no special steps are taken, anybody on the Internet can see what's in /cs/cgi/people/NETID/public_html by hitting http://cgi.cs.arizona.edu/~NETID

Access can be controlled with appropriate .htaccess and .htpasswd files in /cs/cgi/people/NETID/public_html.

## Securing /cs/cgi/people/NETID, continued

Here is my .htaccess file. It is minimal but sufficient.

```
$ cat /cs/cgi/people/whm/public_html/.htaccess
AuthUserFile /cs/cgi/people/whm/public_html/.htpasswd
AuthName "Who Dat?"
AuthType Basic
Require valid-user
```

**AuthUserFile** specifies a password file. We'll see it soon.

**AuthName** is used in a username/password prompt.

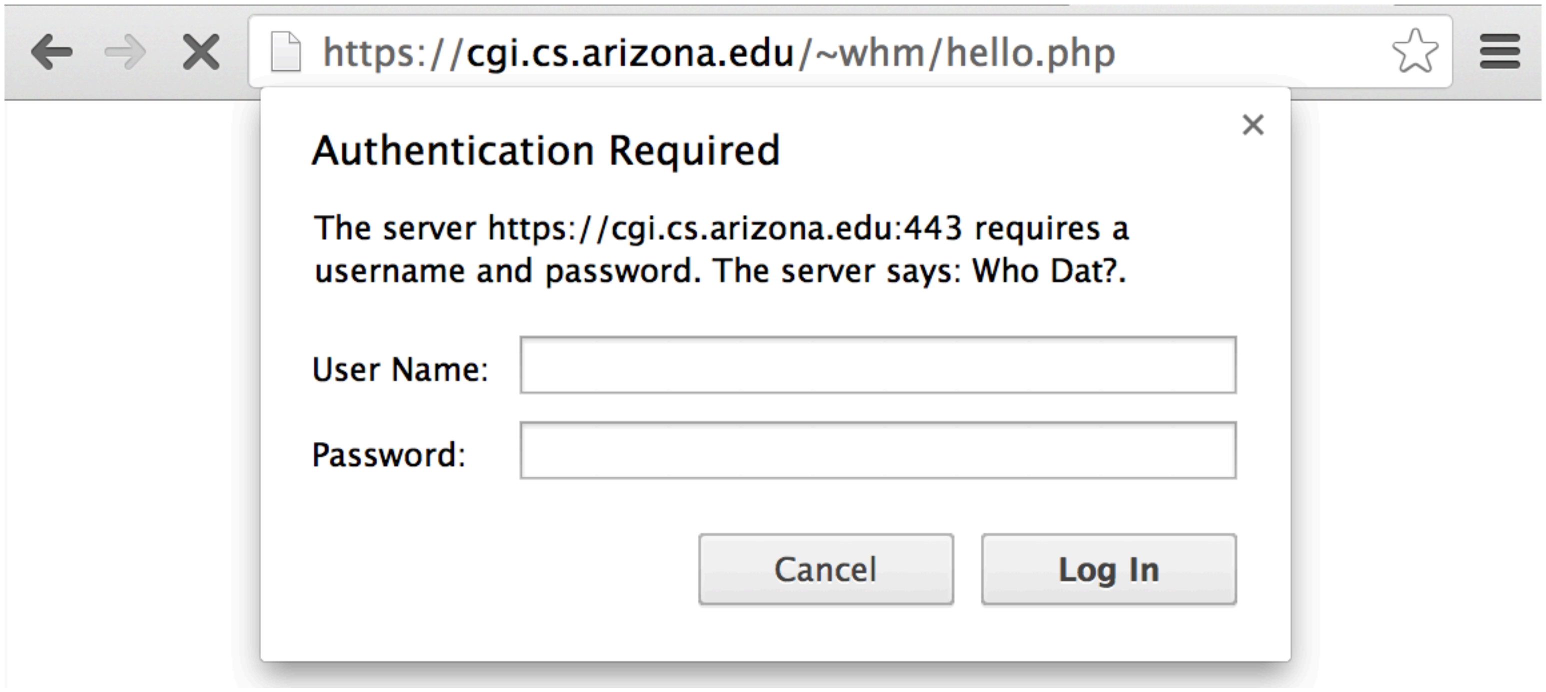
**AuthType Basic** specifies that "Basic authentication" is to be used. It is the simplest to configure but passwords are sent unencrypted so **it should only be used with HTTPS!**

**Require valid-user** activates access control. If it is commented out with a #, everybody still has access.

Docs: http://httpd.apache.org/docs/current/mod/directives.html

## Securing /cs/cgi/people/NETID, continued

With that .htaccess file in place, hitting a URL in .../~whm/... produces an authentication dialog:


https://cgi.cs.arizona.edu/~whm/hello.php

Authentication Required

The server https://cgi.cs.arizona.edu:443 requires a username and password. The server says: Who Dat?.

User Name:

Password:

**IMPORTANT: ALWAYS USE** http**S**://... with sites that use Basic Authentication!

When you set up your .htaccess file, confirm that the dialog appears iff **Require valid-user** is present. (Try commenting with #.)

## Securing /cs/cgi/people/NETID, continued

Recall the **AuthUserFile** line from .htaccess:

```
AuthUserFile /cs/cgi/people/whm/public_html/.htpasswd
```

Use *your* netid!

Here is .htpasswd. It has entries for two users.

```
% cat .htpasswd
whm:$apr1$uRHVz9Rz$CMUHL1KQN72CUMwx7cF6i1
test:$apr1$niU5LU6b$ZT68slBua25B4UlGEngzb0
```

Entries for .htpasswd can be generated with the `htpasswd` command on lectura. See options for htpasswd with **man htpasswd.** The following invocation simply prints the user:password entry. You can paste it into .htpasswd with an editor.

```
% htpasswd -n whm
New password:  (I typed secret)
Re-type new password: (ditto)
whm:$apr1$uRHVz9Rz$CMUHL1KQN72CUMwx7cF6i1
```

# Securing /cs/cgi/people/NETID, continued

At this point people/NETID is secure from Internet access but it must also be secured from your untrustworthy classmates!

Use this command to change the directory permissions:

```
% chmod 750 /cs/cgi/people/NETID
```

When done, check it and confirm the **drwxr-x---** sequence

```
% ls -ld /cs/cgi/people/NETID
drwxr-x--- 4 NETID 33 4 Oct 17 15:28 /cs/cgi/people/NETID
```

chmod is the "change (file) mode" command. It's used to change the value of nine bits that specify who can can access a file. Mode 750, which is shown as rwxr-x---, means that (1) the owner of the directory (you) can read it, write it, and "search" it, (2) the "group" (which ends up being the Apache server) can read it and search it, and (3) all other users can't access it in any way.

## Extra Credit Assignment 3

Due: Monday, October 21 at 2:45pm
Worth: 3 points

What:

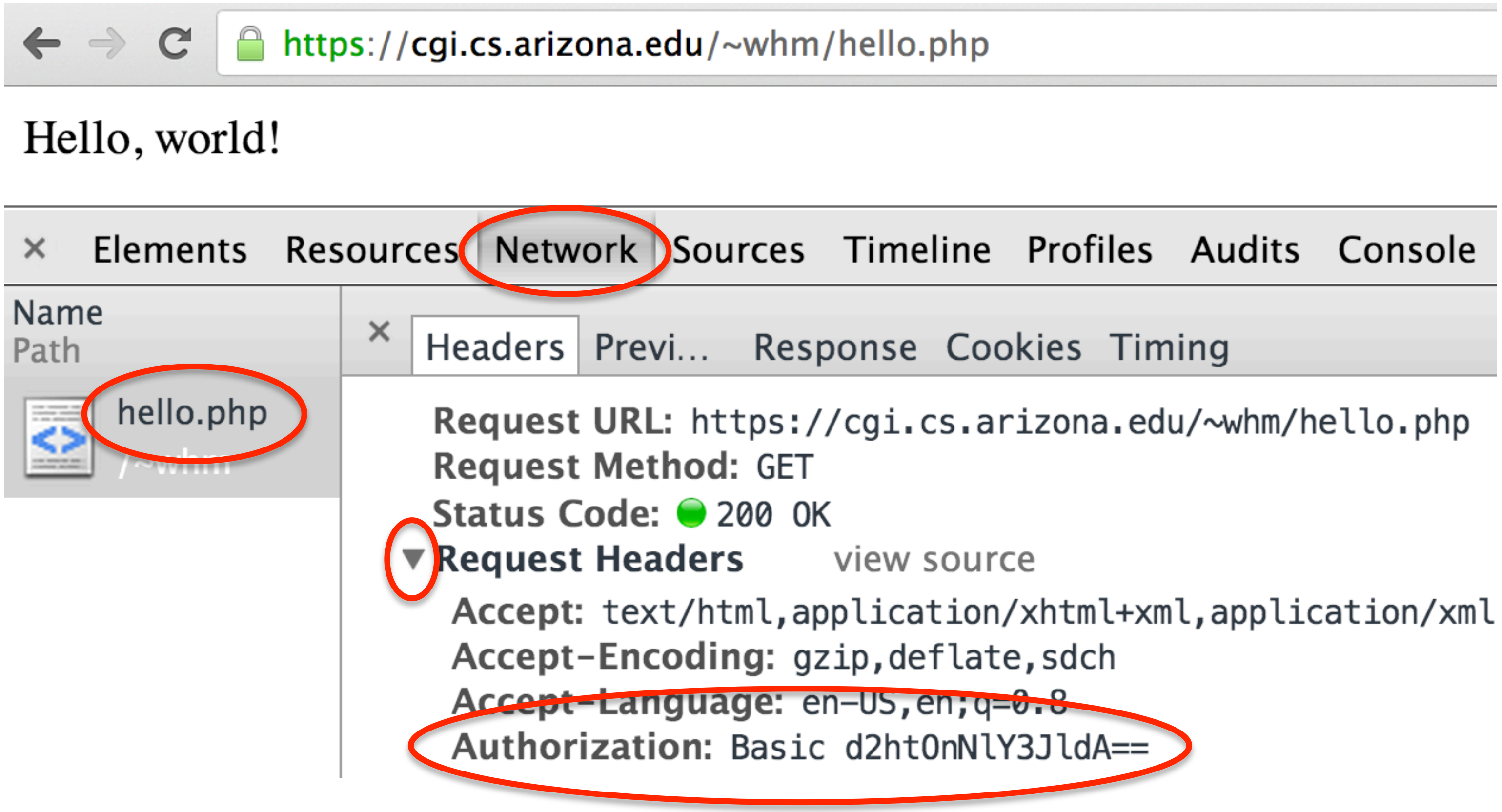
Secure https://cgi.cs.arizona.edu/~YOUR-NETID as shown on the preceding slides.

There's nothing to turn in. We'll test by hitting **https://cgi.cs.arizona.edu/~YOUR-NETID** and also checking directory permissions. If we get an authentication dialog and the mode is 750, you get three points!

Some students may run into problems with permissions and other things. If so, don't worry but let us know ASAP.

## Securing /cs/cgi/people/NETID, continued

With the Network tab in Chrome DevTools we can see the authentication information that the browser sends:



```
Request URL: https://cgi.cs.arizona.edu/~whm/hello.php
Request Method: GET
Status Code: ● 200 OK
▼Request Headers      view source
  Accept: text/html,application/xhtml+xml,application/xml
  Accept-Encoding: gzip,deflate,sdch
  Accept-Language: en-US,en;q=0.8
  Authorization: Basic d2htOnNlY3JldA==
```

```
php > echo base64_decode("d2htOnNlY3JldA==");
whm:secret
```

## Securing /cs/cgi/people/NETID, continued

Once authenticated, browsers send the Basic Authentication information along with every request sent to that site.

Exiting the browser causes the authentication information to be discarded, unless you saved the password for the site. (Passwords saved by browsers are trivially accessible, BTW!)

An .htaccess file protects both the directory it is in and all subdirectories of that directory, too.

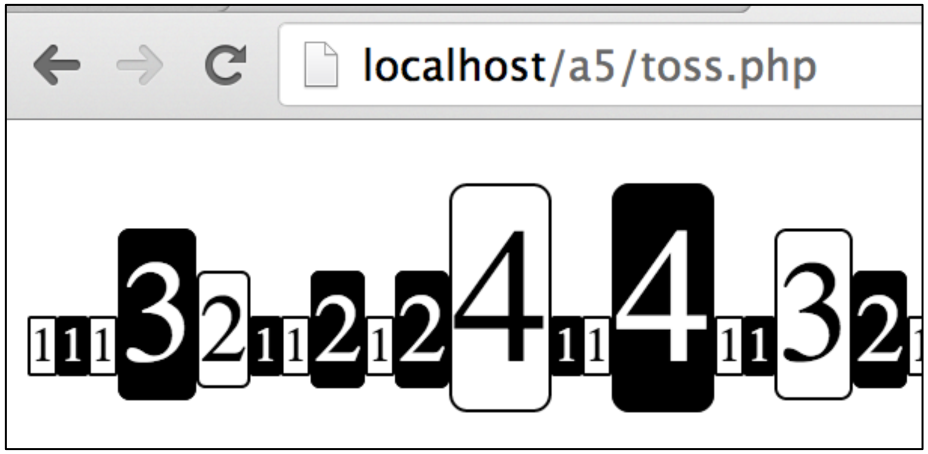
XAMPP's Apache HTTP server can be secured with .htaccess and .htpasswd but there's no need to do that if the firewall on your machine is configured to block external access to the server.

We'll talk more about the HTTP protocol later.

## Testing your a5 solutions

## Testing with XAMPP

If XAMPP is installed, you might test an a5 solution by putting it in htdocs/a5 and hitting it on localhost:



There's nothing magic about "a5"; the example just points out that the URL can specify a file in a subdirectory.

If trouble, use View>Developer>View source to see the generated HTML and CSS.

## Testing with the command-line on Windows

XAMPP's default install on Windows associates the extension .php with c:\xampp\php\php.exe. Typing the name of a php file at the command prompt causes the file to be run:

```
C:\xampp\htdocs\a5> pattern.php
<!doctype html><title>Pattern</title>
<blockquote><img src=blackpixel.png...LOTS MORE...
^C
```

Redirect the output into a file and then open the file:

```
C:\xampp\htdocs\a5> pattern.php > x.html
C:\xampp\htdocs\a5> x.html
C:\xampp\htdocs\a5> blog.php > blog.html & blog.html
```

## Testing with the command line on OS X

You can use the Apple-supplied php like this:

```
% php pattern.php
<!doctype html><title>Pattern</title>
<blockquote><img src=blackpixel.png...LOTS MORE...
^C
```

Redirect the output into a file and open the file with the "open" command, which uses OS X file associations:

```
% php pattern.php > x.html
% open x.html
% php pattern.php > x.html ; open x.html
```

Or maybe add an alias for XAMPP's php and use it:

```
% alias xp=/Applications/XAMPP/bin/php
% xp pattern.php >x2.html
```

To avoid typing the alias each time you start Terminal, put it in one of the bash initialization files like ~/.profile or ~/.bashrc.  (See my 352 slides.)

## Testing on lectura

If running PHP on your own machine is not an option, you could edit files in /cs/cgi/people/NETID/public_html on lectura and then hit http://cs.cgi.arizona.edu/NETID/...

A variant would be to edit files on your own machine and copy them to lectura to test them.

I use this "search engine"/keyword for testing:

```
    wc      https://cgi.cs.arizona.edu/~whm/%s
```

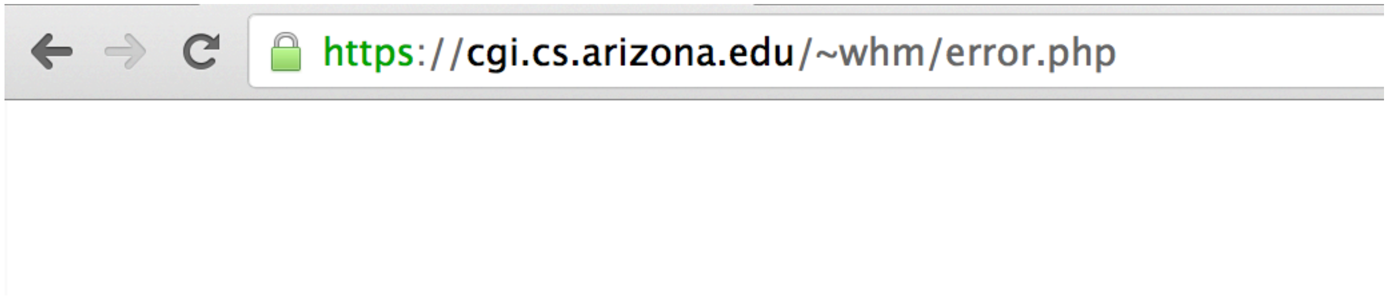
Let us know if you find yourself having to frequently enter your password when hitting cs.cgi.arizona.edu.

## The White Screen of Death

Here's a problem with testing on lectura:

```
% cat /cs/cgi/people/whm/public_html/error.php
<?php
echo 1  # missing semicolon!
```

When it's hit, we get The White Screen of Death:


https://cgi.cs.arizona.edu/~whm/error.php

I am unaware of anything that can be put in error.php to cause that syntax error to be shown. (Let me know if you've got a solution!)

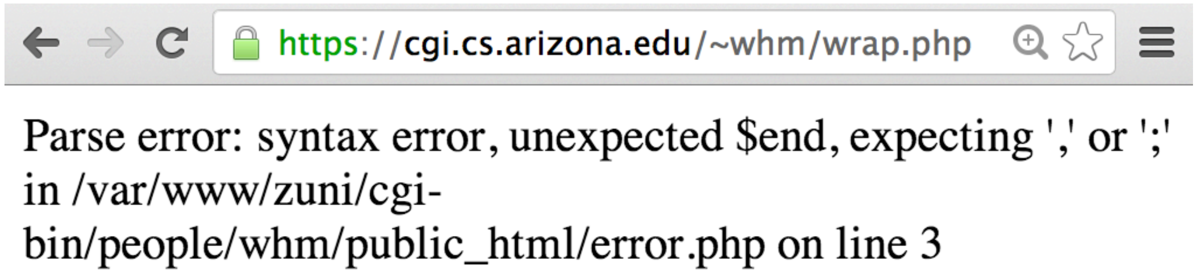
Note: XAMPP's configuration doesn't have this problem.

## A workaround for The White Screen of Death

We can use a "wrapper" that loads the file of interest <u>after</u> turning on the display_errors configuration option:

```
% cat /cs/cgi/people/whm/public_html/wrap.php
<?php
ini_set('display_errors', 'On');
include("error.php");
```

When we hit the wrapper, we see the error:


https://cgi.cs.arizona.edu/~whm/wrap.php

```
Parse error: syntax error, unexpected $end, expecting ',' or ';'
in /var/www/zuni/cgi-
bin/people/whm/public_html/error.php on line 3
```

Putting the ini_set(...) in error.php doesn't work because compilation fails <u>before</u> ini_set(...) is called. ini_get(...) can be used to see the initial value of display_errors.

## A round of odds and ends

## NULL

There is a null type with one value, represented by the literal NULL, which is case-insensitive.

A variable's value is NULL if it has never been assigned a non-null value or it has been `unset()`.

```
php > var_dump(null);
NULL
php > var_dump($x);
NULL
php > $x = 1;
php > unset($x);
php > var_dump($x);
NULL
```

# Automatic type conversions

One way to compare languages is to consider the type of the result of a binary operator for various operand types.

Java

| + | i | d | s |
|---|---|---|---|
| i | i | d | s |
| d | d | d | s |
| s | s | s | s |

Python

| + | i | f | s |
|---|---|---|---|
| i | i | f | E |
| f | f | f | E |
| s | E | E | s |

C

| + | i | d | c |
|---|---|---|---|
| i | i | d | i |
| d | d | d | d |
| c | i | d | i |

PHP ⇔

| + | i | f | s |
|---|---|---|---|
| i | i | f | i or f |
| f | f | f | i or f |
| s | i or f | i or f | i or f |

| . | i | f | s |
|---|---|---|---|
| i | s | s | s |
| f | s | s | s |
| s | s | s | s |

## Automatic type conversions, continued

As shown in the previous tables, the result of binary + is <u>always</u> a number, assuming no error:

```
php > var_dump(1 + "2" + "7/11");
int(10)
php > var_dump(1 + "2" + "7.0/11");
float(10)
php > var_dump(null + false + true);
int(1)
```

The string concatenation operator (.) <u>always</u> yields a string:

```
php > var_dump(1 . "2" . "7/11" . true);
string(7) "127/111"
php > var_dump(null . false . true);
string(1) "1"
```

## "Casting"

(I believe) C introduced the term "casting" for explicit type conversions. PHP uses the same syntax:

```
php > $x = (int)2.3;          # like int(2.3) in Python
php > var_dump($x);
int(2)

php > $y = (double)2;
php > var_dump($y);
float(2)

php > $z = (string)("10" + "20");
php > var_dump($z);
string(2) "30"
```

Automatic conversions are called "implicit casting" in PHP literature. That seems like an oxymoron: implicit explicit conversions!

## is_*SOMETHING* functions

PHP has a number of functions to test whether a value meets some criteria. A few examples:

```
php > var_dump(is_int(10));
bool(true)

php > var_dump(is_int(10.0));          # is_float would be true
bool(false)

php > var_dump(is_numeric("10"));  # "10.1" would be true, too
bool(true)

php > var_dump(is_string(1 . 1));
bool(true)

php > var_dump(is_file(".."));          # is_dir("..") is true
bool(false)

php > var_dump(is_file("a5/blog.txt"));
bool(true)
```

## Logical operators

Like C and Java, PHP has logical operators !, &&, ||.

PHP also has `and`, `or`, and `xor`, which have lower precedence that the assignment operators. PHP does not have `not`.

```
php > var_dump(1 < 2 && 3 > 4);
bool(false)

php > var_dump(1 < 2 or 3 > 4);
bool(true)

php > var_dump(true and true and true and false);
bool(false)

php > var_dump(true and true and true and !false);
bool(true)

php > var_dump(true xor true);
bool(false)
```

## Constants

PHP has the notion of a *constant*. It is an identifier that is given a scalar value (int, float, boolean, or string) by calling the `define` function:

```
php > define("DOTS", "...");
php > define("NL", "\n");
php > echo 10, DOTS, NL;
10...
php > var_dump(DOTS);
string(3) "..."
```

On an earlier slide we saw the constant STDIN:

```
$line = fgets(STDIN);
```

Constants have global scope, even if defined in a function.

Using a constant implies that the value won't change, but a constant can be changed!

## Bare strings

A "bare string" is an unquoted sequence of alphanumeric characters starting with an underscore or a letter.

If it hasn't been defined as a constant, it's treated as a string literal!

```
php > echo hello, world;
helloworld

php > include(wrap  .".".  php);
php > var_dump(wrap(TESTING, E3));
string(9) "ETESTING3"

php > define(TESTING, xxxxxxxx);
php > var_dump(wrap(TESTING, E3));
string(10) "Exxxxxxxx3"
```

## Arrays

## Array basics

A PHP array is in fact an *ordered map*.

PHP arrays are used in cases where a Java programmer might use an array, a Map, or  List; or where a Python programmer might use a tuple, list, or dictionary.  (One type does it all!)

php.net/manual/en/language.types.array.php describes the basics of the type.

There are dozens of library functions specifically for manipulating arrays, and dozens of others that use and/or produce arrays.

php.net/manual/en/ref.array.php describes the array-manipulating functions.

## Array basics, continued

An array can be created with the `array` *construct*:

```
php > $a = array("one"=>1,"ten"=>10.0,"five"=>"V");
php > var_dump($a);
array(3) {
  ["one"]=> int(1)
  ["ten"]=> float(10)
  ["five"]=> string(1) "V"
}
```

The array $a ~~is~~ can be thought of as a map/dictionary with these key/value associations:

| Key | Value |
|---|---|
| "one" | integer 1 |
| "ten" | float 10 |
| "five" | string "V" |

## Array basics, continued

The value associated with a key can be accessed with a subscripting notation.

```
php > $a = array("one"=>1,"ten"=>10.0,"five"=>"V");
php > var_dump($a["one"]);
int(1)
php > var_dump($a["ten"]);
float(10)
php > var_dump($a["twenty"]);
NULL
```

Note that NULL is produced if a key is not found. With error_reporting(22527), "Notice: Undefined offset: ..." is printed.

## Example: Tallying words

```
% cat tallywords.php
<?php
$counts = array();

while ($word = fgets(STDIN)) {
   $word = trim($word);
   $counts[$word] += 1;   # Very expressive—that's good!
                          # How is first occurrence handled?
   }
var_dump($counts);

% php tallywords.php  < tallywords.1
array(4) {
  ["to"]=> int(2)
  ["be"]=> int(2)
  ["or"]=> int(1)
  ["not"]=>  int(1)
}
```

```
% cat tallywords.1
to
be
or
not
to
be
```

## Implicit values for keys

If array(...) is invoked with only values (no =>), the values are associated with integer keys 0, 1, 2, ...

```
php > $a = array(1, 10.0, "V", false);
php > var_dump($a);
array(4) {
  [0]=> int(1)
  [1]=> float(10)
  [2]=> string(1) "V"
  [3]=> bool(false)
}
php > var_dump($a[0], $a[2], $a[4]);
int(1)
string(1) "V"
NULL
```

## Exploding a string

Many PHP functions return arrays. Here's one:

```
explode (string $delim, string $s [, int $limit ] )
```

Usage:

```
php > $parts = explode("/", "a:3/b:4/c:2");
php > var_dump($parts);
array(3) {
  [0]=> string(3) "a:3"
  [1]=> string(3) "b:4"
  [2]=> string(3) "c:2"
}
```

What does explode() do?

Problem: Write f($s) such that f("a:3/b:10/xy:2") returns "aaabbbbbbbbbbxyxy". Here's function that will help: count(array(10,2,4)) returns 3.

## Exploding a string, continued

Solution: (explode1.php)

```
    function f($s)
    {
       $result = "";
       $segs = explode("/", $s);

         $i = 0;
         while ($i < count($segs)) {
              $parts = explode(":", $segs[$i]);
              $result .= str_repeat($parts[0], $parts[1]);
              $i += 1;
              }

         return $result;
    }

    var_dump(f("a:3/b:10/xy:2"));
    var_dump(f("10:0/1:10"));
    var_dump(f(":1000000000000/xxxx:0")); # a trillion(?)
```

## parse_url(...)

PHP's parse_url() function returns an array where the the keys name the parts of a URL:

```
php > var_dump(parse_url("http://safaribooks.com.
ezproxy1.library.arizona.edu/978144068/ch03_html?
readerfullscreen=1&readerleftmenu=0=#X2ludGVybm"));
array(5) {
  ["scheme"]=>  string(4) "http"
  ["host"]=>  string(44)
"safaribooks.com.ezproxy1.library.arizona.edu"
  ["path"]=>  string(20) "/978144068/ch03_html"
  ["query"]=>  string(36) "readerfullscreen=1&leftmenu=0="
  ["fragment"]=>  string(10) "X2ludGVybm"
}
```

The various elements can be referenced with names rather than integer offsets.

## The `foreach` statement

PHP's `foreach` statement provides a way to easily iterate over elements in an array:

```
php > foreach ($a as $key => $value)
php >     echo "key=$key, value=$value\n";
key=one, value=1
key=ten, value=10
key=five, value=V
```

Any names can be used for the key and value; I typically use `$k` and `$v`.

Note that `foreach` is one word.

## `foreach`, continued

Instead of specifying **... $key => $value ...**, we can specify a single variable:

```
php > $a = array("one"=>1,"ten"=>10.0,"five"=>"V");
php > foreach ($a as $x)
php >     echo "$x\n";
1
10
V
```

When only a single variable is named, which is iterated over, the keys or the values? What if we wanted to iterate over the others?

## Arrays and functions

m_to_n creates an array holding a sequence of integers:

```
function m_to_n($start, $end {
    $r = array();
    $n = $start;
    while ($n <= $end) {
        $r[] = $n; $n += 1;
        }
    return $r;
} // m_to_n.php
```

a_to_s returns a string with the values from an array:

```
function a_to_s($a) {
    $r = $sep = "";
    foreach ($a as $value) {
        $r .= $sep . $value;
        $sep = ",";
        }
    return "array($r)";
} // m_to_n.php
```

Usage:

```
php > echo a_to_s(m_to_n(-3,3));
array(-3,-2,-1,0,1,2,3)
php > echo a_to_s(array_reverse(m_to_n(1,7)));
array(7,6,5,4,3,2,1)
```

## Lots of array functions...

Some fun with a few of PHP's many array-related functions:

```
php > $x = array_slice(str_split("pickle"),1,3);
php > echo a_to_s($x);
array(i,c,k)

php > array_unshift($x, "t"); array_push($x, "s");
php > echo a_to_s($x);
array(t,i,c,k,s)

php > sort($x); echo a_to_s($x);
array(c,i,k,s,t)

php > $nums = m_to_n(1,12);
php > shuffle($nums); echo a_to_s($nums);
array(4,12,7,11,5,2,3,6,9,8,10,1)
```

Some array functions are *applicative*; some are *imperative*.

## Passing parameters by reference

In PHP, call-by-value is used with arrays, too! Because of that, this function has no effect: (It works in Java and Python.)

```
function twice($nums) {      # callbyref1.php
    $i = 0;
    while ($i < count($nums)) {
        $nums[$i] *= 2;
        $i += 1;
        }
}
```

Usage:

```
php > $a = m_to_n(1,5);
php > twice($a);
php > echo a_to_s($a);
array(1,2,3,4,5)
```

## Passing parameters by reference, continued

We could make twice() *applicative* by returning a new array but another option is to pass the array by reference, indicated by preceding the parameter with an ampersand:

```
function twice(&$nums) {   # callbyref2.php
    $i = 0;
    while ($i < count($nums)) {
        $nums[$i] *= 2;
        $i += 1;
        }
}
```

Now this *imperative* version works as if it were Java or Python:

```
php > $a = m_to_n(1,5);
php > twice($a);
php > echo a_to_s($a);
array(2,4,6,8,10)
```

## PHP converts various values for keys!

Keys can be only strings or integers.  Others are converted.

```
php > $a = array();             # Actual key
php > $a["10"] = "10";          # int 10
php > $a["010"] = "010";        # string
php > $a[7.2] = 7.2;            # int 7
php > $a[false] = false;        # int 0
php > $a[true] = true;          # int 1
php > $a[null] = null;          # empty string

php > var_dump($a);
array(6) {
  [10]=> string(2) "10"
  ["010"]=> string(3) "010"
  [7]=> float(7.2)
  [0]=> bool(false)
  [1]=> bool(true)
  [""]=> NULL
}
```

## array_merge

The array_merge function can be used to concatenate arrays with integer keys:

```
php > $a=m_to_n(1,5);
php > $b=m_to_n(20,25);
php > $c = array_merge($a, $b, $a);
php > a_to_s($c);
php > echo a_to_s($c);
array(1,2,3,4,5,20,21,22,23,24,25,1,2,3,4,5)
```

Things are more "interesting" if the arrays have any non-integer keys or if there's a mix. (Try it!)

## PHP's $_GET array

## The HTTP GET request

When we hit a URL in a browser, the browser sends a "GET" request to a server using the HTTP protocol.

The server responds with data that the browser renders as HTML.

We can use the `curl` command to see the complete interaction.

curl is available on lectura and OS X. There's a Cygwin curl package, too.

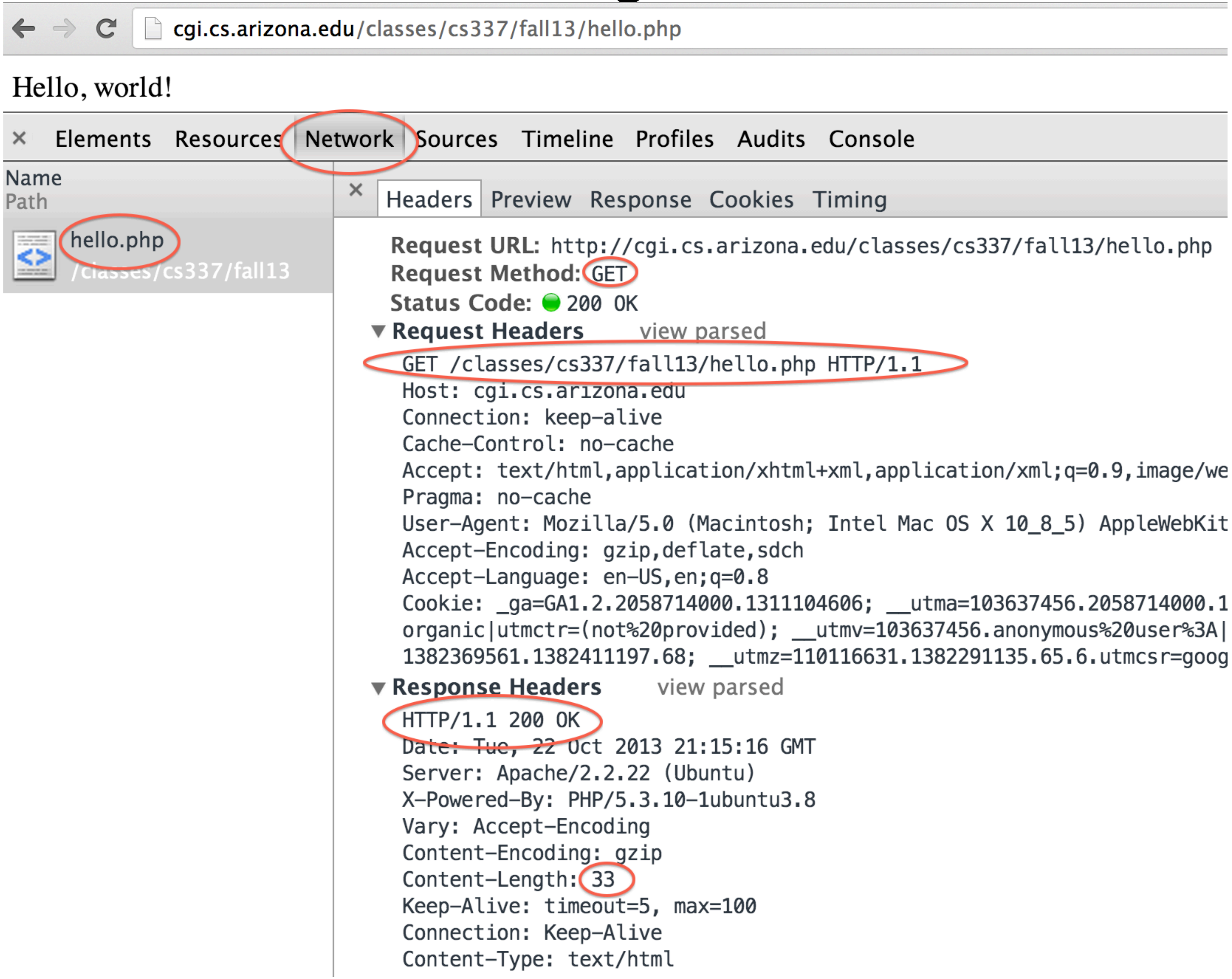
# Watching a GET with CURL

```
% curl -v http://cgi.cs.arizona.edu/classes/cs337/fall13/hello.php
* Connected to cgi.cs.arizona.edu (192.12.69.39) port 80 (#0)
> GET /classes/cs337/fall13/hello.php HTTP/1.1             (1)
> User-Agent: curl/7.25.0 (x86_64-apple-darwin10.8.0) l/... (2)
> Host: cgi.cs.arizona.edu
> Accept: */*
>                       (3)
< HTTP/1.1 200 OK       (4)
< Date: Tue, 22 Oct 2013 20:44:33 GMT  (5)
< Server: Apache/2.2.22 (Ubuntu)
< X-Powered-By: PHP/5.3.10-1ubuntu3.8
< Content-Type: text/html
<                       (6)
Hello, world!           (7)
* Closing connection #0
```

Note the...
(1) GET
(2) Headers
(3) Blank line
(4) 200 OK
(5) Headers
(6) Blank line
(7) Data

">" lines were sent by curl; "<" lines were received from the server; "*" lines are info printed by curl.

# Viewing GET results with Chrome



cgi.cs.arizona.edu/classes/cs337/fall13/hello.php

Hello, world!

Elements Resources Network Sources Timeline Profiles Audits Console
Name
Path
hello.php
/classes/cs337/fall13
Headers Preview Response Cookies Timing

```
Request URL: http://cgi.cs.arizona.edu/classes/cs337/fall13/hello.php
Request Method: GET
Status Code: ● 200 OK
▼ Request Headers      view parsed
  GET /classes/cs337/fall13/hello.php HTTP/1.1
  Host: cgi.cs.arizona.edu
  Connection: keep-alive
  Cache-Control: no-cache
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/we
  Pragma: no-cache
  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_5) AppleWebKit
  Accept-Encoding: gzip,deflate,sdch
  Accept-Language: en-US,en;q=0.8
  Cookie: _ga=GA1.2.2058714000.1311104606; __utma=103637456.2058714000.1
  organic|utmctr=(not%20provided); __utmv=103637456.anonymous%20user%3A|
  1382369561.1382411197.68; __utmz=110116631.1382291135.65.6.utmcsr=goog
▼ Response Headers      view parsed
  HTTP/1.1 200 OK
  Date: Tue, 22 Oct 2013 21:15:16 GMT
  Server: Apache/2.2.22 (Ubuntu)
  X-Powered-By: PHP/5.3.10-1ubuntu3.8
  Vary: Accept-Encoding
  Content-Encoding: gzip
  Content-Length: 33
  Keep-Alive: timeout=5, max=100
  Connection: Keep-Alive
  Content-Type: text/html
```

## URL parameters

Note the result of this parse_url call:

```
php> var_dump(parse_url("http://localhost/c/get1.php?
start=10&end=20"));
array(4) {
  ["scheme"]=> string(4) "http"
  ["host"]=> string(9) "localhost"
  ["path"]=> string(11) "/c/get1.php"
  ["query"]=> string(15) "start=10&end=20"
}
```

The *query string* has two name/value pairs. These are known as *URL parameters*.

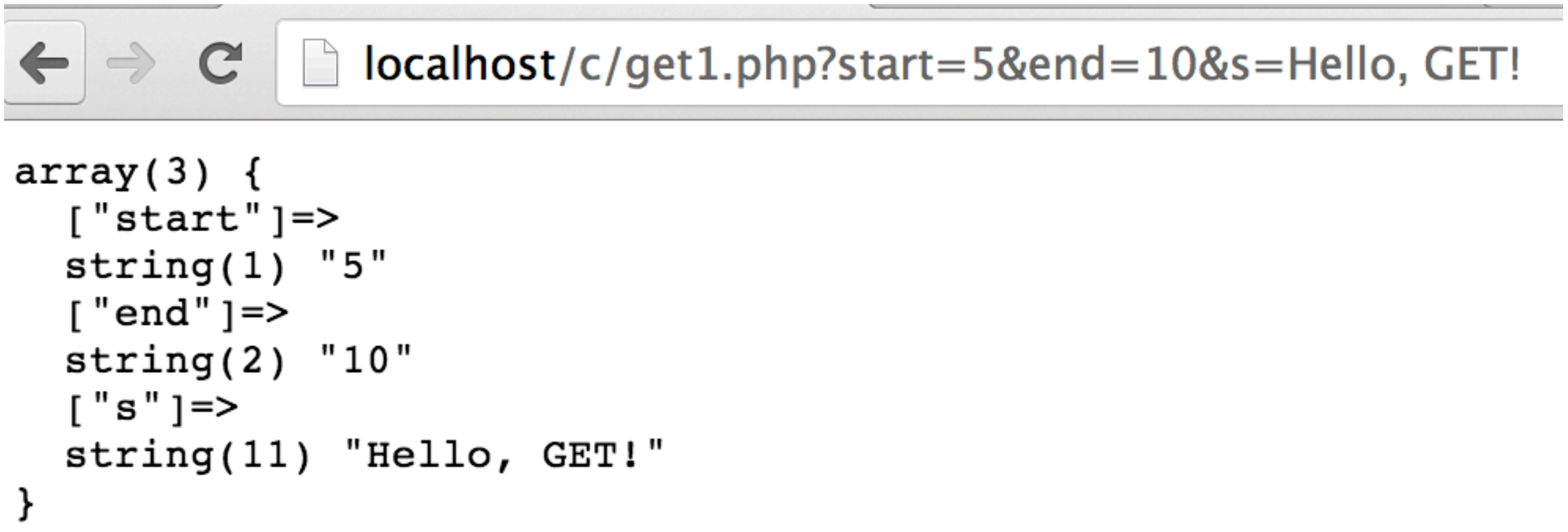
URL parameters are one way to provide data to a web application.

## Accessing URL parameters with $_GET

PHP makes URL parameters available in the $_GET array.

```
<?php        # get1.php
echo "<pre>";
var_dump($_GET);
echo "</pre>";
```

Note the result when we hit get1.php

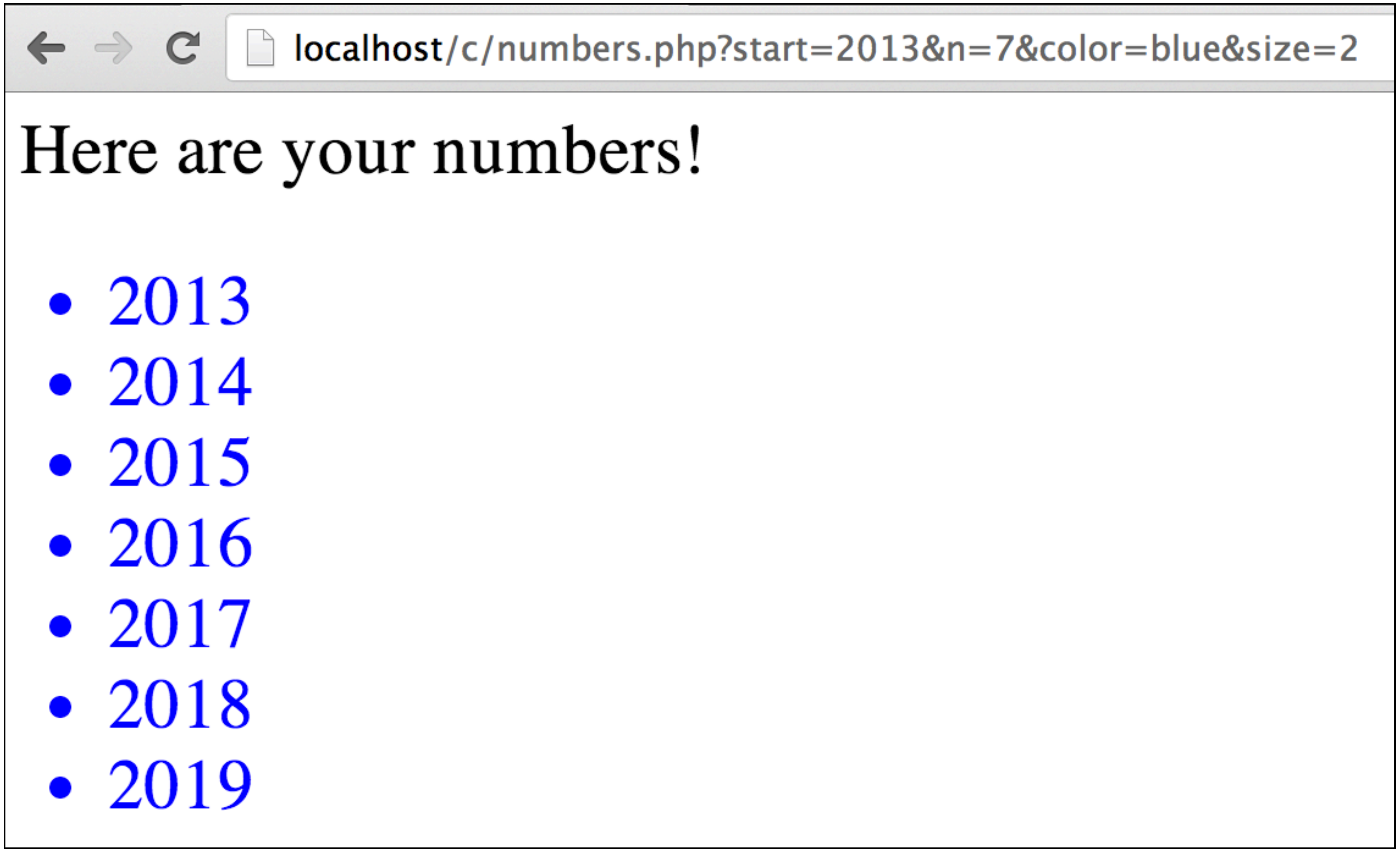


```
localhost/c/get1.php?start=5&end=10&s=Hello, GET!
```

```
array(3) {
  ["start"]=>
  string(1) "5"
  ["end"]=>
  string(2) "10"
  ["s"]=>
  string(11) "Hello, GET!"
}
```

## Problem: Sequence of numbers

Make it so!



localhost/c/numbers.php?start=2013&n=7&color=blue&size=2

Here are your numbers!

- 2013
- 2014
- 2015
- 2016
- 2017
- 2018
- 2019

## Solution: Sequence of numbers

```
<?php                                    # numbers.php
echo "<!doctype html>
<title>Numbers</title>
<body style=font-size:{$_GET['size']}em>
Here are your numbers!
<ul style=color:{$_GET['color']}>";

$i = $_GET["start"];
$end = $i + $_GET["n"] - 1;
while ($i <= $end) {
   echo "<li>$i";
   $i += 1;
   }
```

What's the behavior if a parameter is omitted?

## HTML Forms

## The HTML `input` element

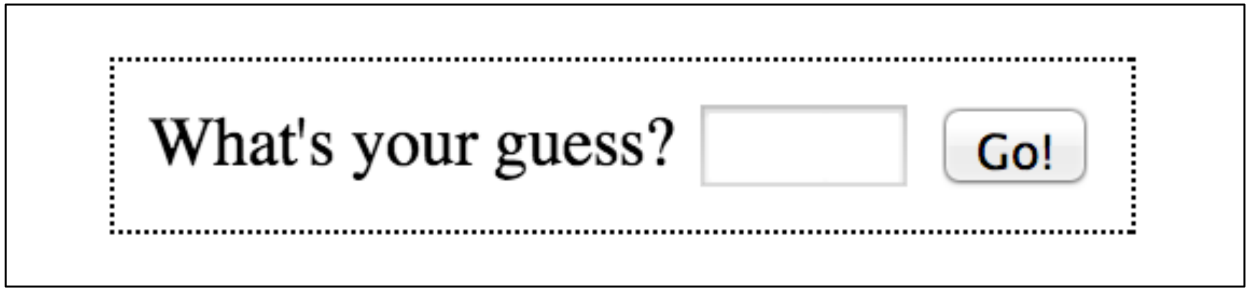
HTML forms typically contain one or more `input` elements.
HTML5 has 20+ types of input elements.

Two common `input` types are `text` and `submit`:

```
<div style='border:dotted 1px; padding: .5em; display:
    inline-block'>
What's your guess?
<input type=text name=guess size=5>
<input type=submit value='Go!'>
</div>
```

input1.html

Rendering:



## The HTML `form` element

Input elements are usually children of an HTML `form` element:

```
<form method=get action='guess.php'>
    <div style=...>
    What's your guess?
    <input type=text name=guess size=5 required>
    <input type=submit value='Go!'>
</form>                              form1.html
```

boolean attribute!

If 777 is entered in the text field and the Go! button is clicked, the browser sends this HTTP request:

```
GET /.../guess.php?guess=777 HTTP/1.1
```

Let's try it!

## `<form>`, continued

For reference:

```
<form method=get action='guess.php'>
    ...
    <input type=text name=guess size=5>
    <input type=submit value='Go!'>
</form>
```

Key points:

Clicking the Go! button generates an HTTP GET request that references the URL specified by the form's `action`.

The GET request will include a URL parameter that specifies a value for `guess`.

The form's `method=get` attribute indicates that the request is to be a GET. (We'll see POSTs soon.)

Like any other HTML element, a <form> might be on a manually typed HTML page or be generated by PHP execution.

What do we need now?

## guess.php

A simple back-end:

```
<?php
$guess = $_GET['guess'];
$referer = $_SERVER['HTTP_REFERER'];

if (!isset($guess)) die("Oops! Expected a URL parameter!");

if ($guess === "")
    die("No guess?! (<a href='$referer'>Try again</a>)");

echo "Wow! $guess is a great guess, but it is incorrect! <br><a
href='$referer'>Try again!</a>";
```

How does it work? How could we improve it?

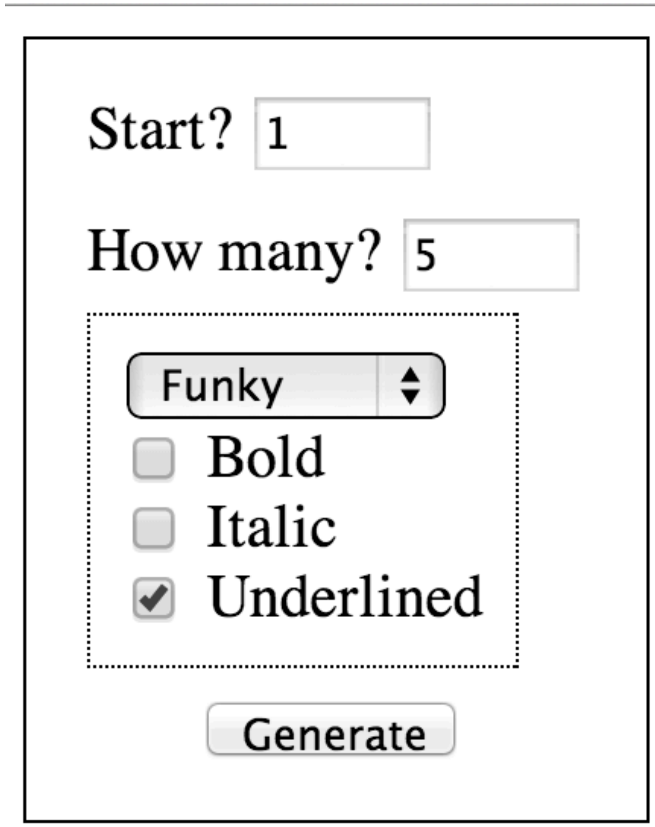
Try it with the DevTools Network tab. Do Rt.clk>Copy as cURL, too!

Notes: $_SERVER['HTTP_REFERER'] is the page we came from. The die(...) function outputs its arguments and exits PHP.

[REPLACEMENT SET!]
Discard sheet with slides 127-133
from Friday, 10/25/13

# Longer example: sequence.php

A form that specifies a sequence of numbers to generate:



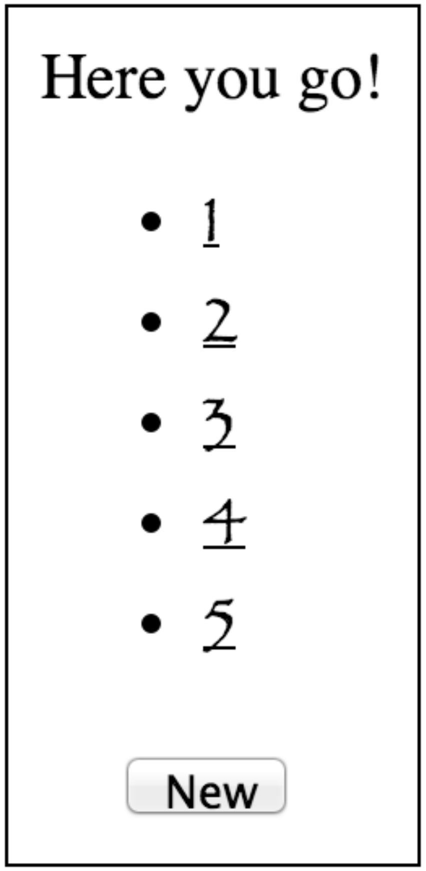
Start? 1

How many? 5

Funky

- [ ] Bold
- [ ] Italic
- [x] Underlined

Result:



Here you go!

- 1
- 2
- 3
- 4
- 5

What <input>s are new?

Run it: http://cgi.cs.arizona.edu/classes/cs337/fall13/sequence.php

## sequence.php, continued

Top-level control and some styling:

```
write_header();

if (count($_GET) == 0)
    write_form();
else
    write_numbers();

function write_header()
{
    echo "<!doctype html><title>Sequence</title>
    <style>
    #controls { border: 1px solid; padding: 0.5em 1em;
                    line-height: 2em; float: left; }
    #attrs { border:dotted 1px; width:6em; line-height: 1em;
                    padding: 0.5em }
    #results { border: 1px solid; float:left; margin-left:-1px;
                    padding: 0.5em }
    </style>";
}
```

## sequence.php, continued

```
function write_form()
{
  echo "<div id=controls>
    <form method=get action='sequence.php'>
    Start? <input name=start type=text size=5 value=1 required><br>
    How many? <input name=n type=text size=5 value=5 required><br>
    <div id=attrs>
        <select name=font-family>
            <option value=sans-serif>Sans Serif</option>
            <option value=cursive>Cursive</option>
            <option value=fantasy selected>Funky</option>
        </select>
        <br>
        <input type=checkbox name=attrs[] value=bold> Bold<br>
        <input type=checkbox name=attrs[] value=italic> Italic<br>
        <input type=checkbox name=attrs[] value=underlined> Underlined<br>
    </div>
    <div style=text-align:center><input type=submit value='Generate'></div>
    </form>
    </div>";
}
```

See HFHC Chapter 14 for lots of detail on form elements. (Skim it!)

## sequence.php, continued

```
function write_numbers()
{
    $attrmap = array("bold"=>"font-weight:bold;", "italic"=>"font-style:italic;",
        "underlined"=>"text-decoration:underline;");

    $style = "font-family:{$_GET['font-family']};";
    if (isset($_GET["attrs"])) {
        foreach ($_GET["attrs"] as $attr) {
            $style .= $attrmap[$attr];
            }
        }

    echo "<div id=results>Here you go!<ul style='$style' >";

    $i = $_GET["start"];
    $end = $i + $_GET["n"] - 1;
    while ($i <= $end) {
        echo "<li>$i"; $i += 1;
        }

    echo "</ul><form method=get action='sequence.php' style=text-align:center>
    <input type=submit value='New'></form></div>";
}
```

## Sidebar: GETs are one-line test cases

If the code that handles form submission has a bug, we have two choices for debugging:

(1) The slow way, by repeatedly filling out the form and clicking a button to submit it.

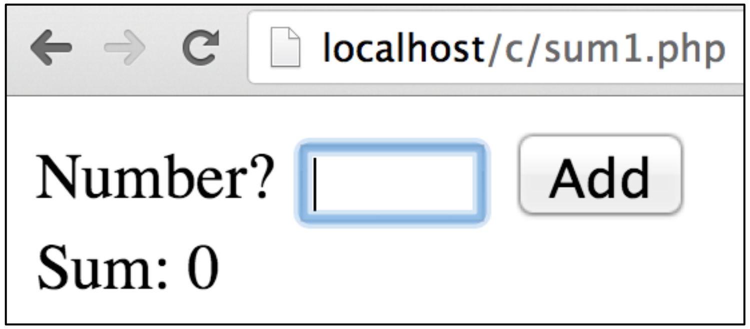
(2) The fast way, by saving the URL and hitting it again.

Let's try it with seqbug.php, a buggy sequence.php.

The URL for a GET can be mailed, put in a text file, added to a bug report, etc. It's a one-line test case!

## Accumulating a sum

Problem: Write a PHP app that sums a series of numbers entered on a form one at a time.


localhost/c/sum1.php

Number? [ ] Add

Sum: 0

sum1.php, **which doesn't work**:

```
error_reporting(22519);
$sum += $_GET["value"];
echo "
<form action='sum1.php'
        method=get>
    Number? <input type=text
                    name=value size=5>
    <input type=submit value=Add>
    <br>
    Sum: $sum
</form>";
```

What's the problem?

## Accumulating a sum, continued

If we hit the URL `sum1.php?value=7`, we'll execute this line:

```
   $sum += $_GET["value"];
```

What's `$sum` now?

If the user next enters 9, we'll hit `sum1.php?value=9` and do `$sum += $_GET["value"]` again.

Then what will `$sum` be?

It might look like the app is running continuously but <u>each form submission causes sum1.php to be run from scratch</u>. There's no retention of any information from the last run.

## One solution: a hidden field

This is sum2.php. It uses a *hidden field*:

```
error_reporting(22519);
$sum = $_GET["sum"] + $_GET["value"];
echo "<!doctype html><title>Hidden</title>
<form action='sum2.php' method=get>
    <input type=hidden name=sum value=$sum>
    Number? <input type=text name=value size=5>
    <input type=submit value=Add> <br>
    Sum: $sum
</form>";
```

If 7 is entered, we hit sum2.php?sum=0&value=7
If 9 is then entered, we hit sum2.php?sum=7&value=9

## Sidebar: The `autofocus` attribute

The `autofocus` binary attribute lets us specify that a control is to be given *input focus* when a form loads. sum2a.php uses it:

```
error_reporting(22519);
$sum = $_GET["sum"] + $_GET["value"];
echo "
<!doctype html>
<title>Hidden</title>
<form action='sum2a.php' method=get>
   <input type=hidden name=sum value='$sum'>
   Number? <input type=text name=value
         size=5 autofocus>
   <input type=submit value=Add> <br>
   Sum: $sum
</form>";
```

Try it!

## "Sticky" values

With sum2.php the number field clears after the form is submitted.

sum3.php makes the number field have a "sticky" value—instead of clearing, the field is prepopulated with the last-entered value.

```
error_reporting(22519);
$value = $_GET["value"];
$sum = $_GET["sum"] + $value;
echo "<!doctype html><title>Sticky</title>
<form action='sum3.php' method=get>
  <input type=hidden name=sum value='$sum'>
  Number? <input value='$value' type=text name=value
        size=5 autofocus>
  <input type=submit value=Add> <br>
  Sum: $sum
</form>";
```

Let's try it!

## Multiple submit buttons

If a form has multiple submit buttons we can add a `name` attribute to distinguish between them. Here is sum4.php:

```
if ($_GET["submit"] == "Add") {
   $value = $_GET["value"]; $sum = $_GET["sum"] + $value;
   }
else {
   $value = null; $sum = 0;
   }
echo "<!doctype html><title>Reset</title>
<form action='sum4.php' method=get>
   <input type=hidden name=sum value='$sum'>
   Number? <input value='$value' type=text name=value size=5
autofocus>
   <input type=submit name=submit value=Add>
   <input type=submit name=submit value=Reset> <br>
   Sum: $sum
</form>";
```

## Assignment 6 stuff

## Arrays of arrays

A PHP array can have arrays as values.

```
php > $odds = array(1,3); $evens = array(2,4);
php > $both = array($odds, $evens);
php > var_dump($both);
array(2) {
  [0]=>  array(2) {
                [0]=> int(1)
                [1]=> int(3)
                }
  [1]=>  array(2) {
                [0]=> int(2)
                [1]=> int(4)
                }
}
```

$both = array(array(1,3),array(2,4)); works, too.
Python equivalent: both = [[1,3],[2,4]].  How about Java?

## arrays of arrays, continued

mexplode() explodes a string first by slashes and then by colons:

```
function mexplode($s)
{
    $r = array();
    foreach (explode("/", $s) as $part)
        $r[]= explode(":", $part);

    return $r;
} // mexplode.php
```

```
php > $a =
   mexplode("a:10/bbb:5");
php > var_dump($a);
array(2) {
  [0]=>
  array(2) {
    [0]=> string(1) "a"
    [1]=> string(2) "10"
  }
  [1]=>
  array(2) {
    [0]=> string(3) "bbb"
    [1]=> string(1) "5"
  }
}
```

## arrays of arrays, continued

Here's a function that takes an mexplode(...) result as an argument and produces a string with replications:

```
function mrepl($a)
{
    $r = "";
    foreach ($a as $spec)
        $r .= str_repeat($spec[0], $spec[1]);

    return $r;
}
```

Usage:

```
php > $a = mexplode("a:3/xy:4/ccc:10");
php > echo mrepl($a);
aaaxyxyxyxyccccccccccccccccccccccccccccc
php > echo mrepl(mexplode("[]:3/():5/@-@:2"));
[][][]()()()()()@-@@-@
```

# load_entries.php on a6

```
% cat entries1.txt
I've started a blog!
tags: x,y,z
2013-08-28
Line 1
Line 2
.end
Phone trouble...
2013-09-19
First (and last) line.
.end
```

```
% php test_load_entries.php
array(2) {
  [0]=>  array(4) {
    ["title"]=> string(20) "I've started a blog!"
    ["date"]=> string(10) "2013-08-28"
    ["text"]=> string(14) "Line 1...
    ["tags"]=>    array(3) {
      [0]=> string(1) "x"
      [1]=> string(1) "y"
      [2]=> string(1) "z"
      }
    }
  [1]=>array(4) {
    ["title"]=> string(16) "Phone trouble..."
    ["date"]=> string(10) "2013-09-19"
    ["text"]=> string(23) "First (and last)
line...."
    ["tags"]=> array(0) { }
  }
}
```

## Demos of shapes.php and blog2.php

## The HTTP POST request

## form1.html revisited

Here's a revision of form1.html from slide 124, trivially recast as PHP, but also with `get` changed to `post` and a slightly revised back-end (guesspost.php).

```
<?php
echo "<!doctype html><title>form</title>
<link rel=stylesheet href=form1.css type=text/css>
<form method=post action=guesspost.php>
   <div>
   What's your guess?
   <input type=text name=guess size=5 required>
   <input type=submit value='Go!'>
   </div>
</form> ";
```

Try form1get.php and form1post.php. What differences do you see?

## guesspost.php

There's only one difference between guess.php and guesspost.php:

```
$guess = $_POST['guess'];
$referer = $_SERVER['HTTP_REFERER'];

if (!isset($guess))
    die("Oops! Expected a URL parameter!");

if ($guess === "")
    die("No guess?! (<a href='$referer'>Try again</a>)");

echo "Wow! $guess is a great guess, but it is incorrect!<br>
<a href='$referer'>Try again!</a>";
echo "</div>";
```

Confirm with diff:

```
% diff guess.php guesspost.php
8c8
< $guess = $_GET['guess'];
---
> $guess = $_POST['guess'];
```

## GET vs. POST

w3.org/Protocols/rfc2616/rfc2616-sec9.html defines HTTP "methods" (a.k.a. verbs). Excerpts:

"The GET method means retrieve whatever information is identified by the [URL]."

"The PUT **(note: PUT, NOT POST!)** method requests that the enclosed entity be stored under the supplied [URL].

"The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the [URL]."

Examples: (also from the RFC)

- Adding a message to a group of articles
- Providing a block of a data to a data-handling process
- Extending a database through an append operation

## GET vs. POST continued

| GETs... | POSTs... |
|---|---|
| Can be bookmarked | Can't be bookmarked |
| Are kept in browser history | Not kept in browser history |
| Have a maximum length | No maximum length |
| URL parameters are logged; also are visible OtS. | Data not logged |

Broad rules of thumb:
- Use GET when requesting data
- Use POST when changing data
- Feel free to bend rules when developing/debugging

MUST use POST if data might be "long", like an uploaded image or document.

There are caching issues, too...

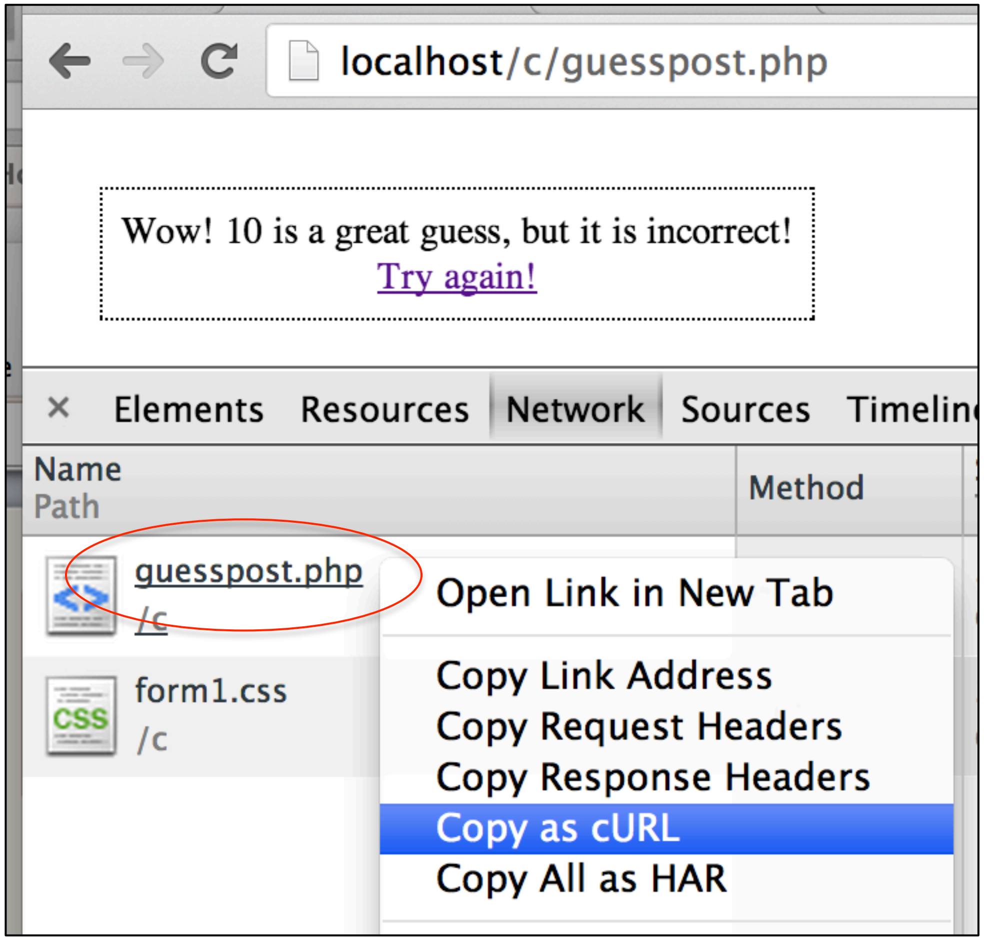
## Sidebar: Apache's access_log

Here are some lines from /Applications/XAMPP/logs/access_log on my machine. What have the users been doing?

```
127.0.0.1 [31/Oct/2013:16:08:58] "GET /a6/blog2.php HTTP/1.1" 200 15643
127.0.0.1 [31/Oct/2013:16:08:58] "GET /a6/tile1.png HTTP/1.1" 304 -
127.0.0.1 [31/Oct/2013:16:08:58] "GET /a6/folly.jpg HTTP/1.1" 304 -
127.0.0.1 [31/Oct/2013:16:08:58] "GET /a6/raffle.jpg HTTP/1.1" 304 -
...
127.0.0.1 [31/Oct/2013:16:09:06] "GET /a6/blog2.php?filter=World%20Series
HTTP/1.1" 200 3701
127.0.0.1 [31/Oct/2013:16:09:18] "GET /a6/form1.html HTTP/1.1" 404 1030
127.0.0.1 [31/Oct/2013:16:09:23] "GET /c/form1.html HTTP/1.1" 304 -
127.0.0.1 [31/Oct/2013:16:09:23] "GET /c/form1.css HTTP/1.1" 304 -
127.0.0.1 [31/Oct/2013:16:09:29] "GET /c/guess.php?guess=1234 HTTP/1.1"
200 208
127.0.0.1 [31/Oct/2013:16:11:24] "GET /c/sequence.php?
start=-99&n=200&font-family=fantasy&attrs%5B%5D=bold&attrs%5B
%5D=underlined HTTP/1.1" 200 1813
```

Run `tail -f /Applications/XAMPP/logs/access_log` and hit some URLs.

# Chromes' *Copy as cURL*



Wow! 10 is a great guess, but it is incorrect!
Try again!

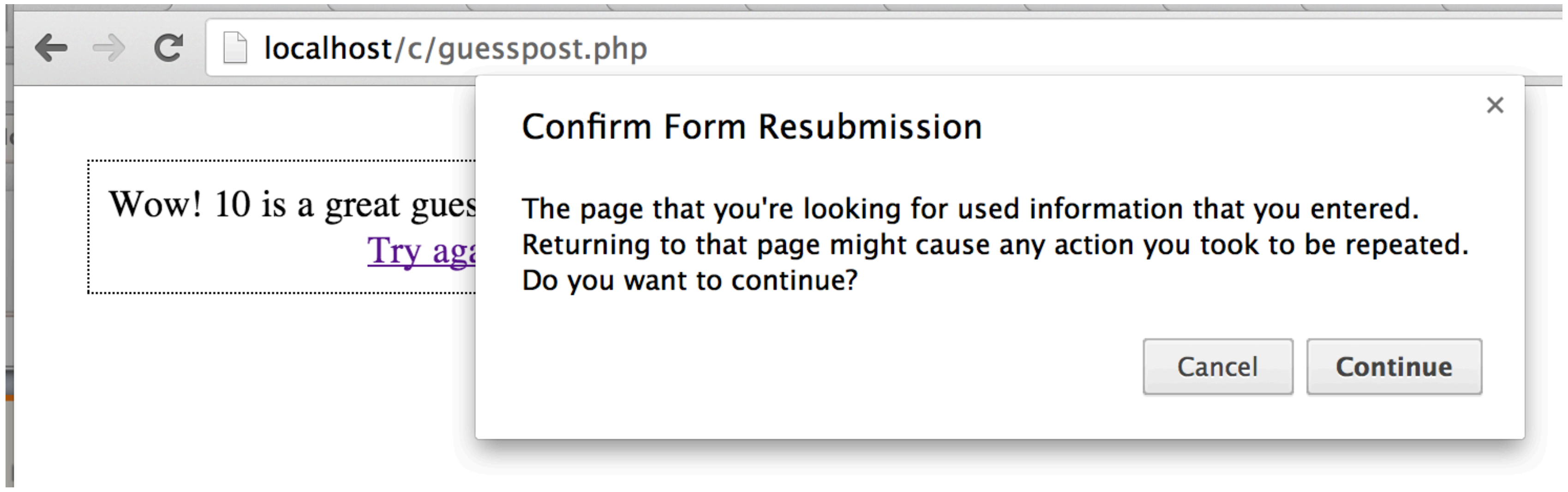
A right-click on a request shows a menu with "Copy as cURL".

Clicking it puts a curl command on the clipboard that reproduces the request.

% **pbpaste** (cat /dev/clipboard on Cygwin)
curl 'http://localhost/c/guesspost.php' -H 'Origin: http://localhost' (...more -H header args...) -H 'Referer: http://localhost/c/form1post.php' **--data 'guess=10'**

## Refreshing a page generated by a POST

If a page is produced by a POST and we do "View>Reload this Page", we get a prompt:


localhost/c/guesspost.php

Wow! 10 is a great gues

Try aga

Confirm Form Resubmission

The page that you're looking for used information that you entered. Returning to that page might cause any action you took to be repeated. Do you want to continue?

Why?

## File uploads

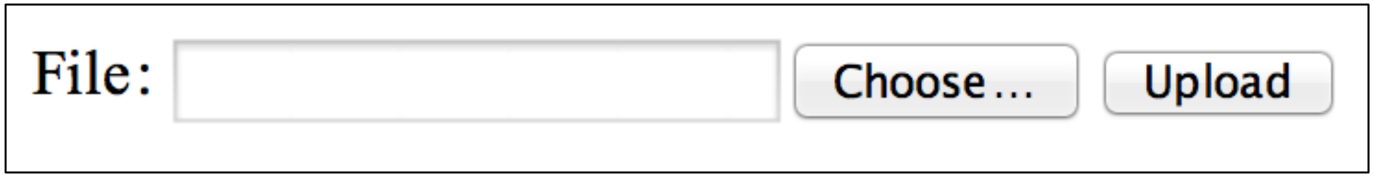
## File uploads

File uploads are almost always done with POST. (Why?)

Here's a form with an input element for uploading a file:

```
<form  enctype=multipart/form-data method=post
          action='http://localhost:4000/xyz.php' >
  File: <input name=f1 type=file>
  <input type=submit value=Upload>
</form>                                        upload1.php
```

It renders like this:



"Choose..." brings up an OS-specific file-choosing dialog.

Note the form's enctype attribute. It is required for uploads.

## File uploads, continued

Recall the form attributes:

```
<form  enctype=multipart/form-data method=post
        action='http://localhost:4000/xyz.php' >
```

XAMPP's Apache HTTP server listens on port 80 by default.

In order to see what exactly what gets sent by this POST, we'll use "netcat" (nc) to listen on port 4000 and print out whatever gets sent to it:

```
% nc -l 4000
```

netcat will wait silently, producing no output until something connects to port 4000 on this machine and sends data.

## File uploads, continued

If we hit upload1.php and upload a.txt, we see this from netcat:

```
% nc -l 4000
POST /xyz.php HTTP/1.1
User-Agent: Opera/9.80 (Macintosh; Intel Mac OS X 10.8.5) ...
Host: localhost:4000
Referer: http://localhost/c/upload1.php
Content-Length: 208
Content-Type: multipart/form-data; boundary=----------Jr6Wp5V
WjjjoMvPwdfrNBJ

------------Jr6Wp5VWjjjoMvPwdfrNBJ
Content-Disposition: form-data; name="f1"; filename="a.txt"
Content-Type: text/plain

Testing at...
Thu Oct 31 20:05:34 MST 2013

------------Jr6Wp5VWjjjoMvPwdfrNBJ—
```

What does the browser do after sending data to localhost:4000?

## File uploads, continued

When a POST with a file is received, PHP populates $_FILES. Here's how PHP transforms the file data in the browser's POST request, which we dumped out with netcat:

```
array(1) {                          # NOTE: an array of arrays!
  ["f1"]=> array(5) {
    ["name"]=> string(5) "a.txt"
    ["type"]=> string(10) "text/plain"
    ["tmp_name"]=>
    string(45) "/Applications/XAMPP/xamppfiles/temp/
phpWgA5Ru"
    ["error"]=> int(0)
    ["size"]=> int(43) } }
```

What do we see?  Where are the file contents?

## Simple file display

upload2.php simply displays the contents of an uploaded file:

```
if ($_SERVER["REQUEST_METHOD"] == "GET") {
     echo "<form  enctype=multipart/form-data method=post
               action=upload2.php>
     File: <input name=f1 type=file>
     <input type=submit value=Upload></form>";
} else {
     $f = fopen($_FILES["f1"]["tmp_name"], "r");
     echo "<pre style='...[borders, padding, inline-block]...'>";
     while ($line = fgets($f)) {
      echo htmlspecialchars($line);  // Try it without this fcn...
      }
     echo "</pre><form><input type=submit value=Again>
               </form>";  // Note minimal form!
}
```

Run it!

## What lies outside <?php ... ?>

## PHP in 1996

Here's an example from php.net/manual/en/history.php.php:

```
<!--getenv HTTP_USER_AGENT-->
<!--ifsubstr $exec_result Mozilla-->
  Hey, you are using Netscape!<p>
<!--endif-->

<!--sql database select * from table where
                                          user='$username'-->
<!--ifless $numentries 1-->
  Sorry, that record does not exist<p>
<!--endif exit-->
  Welcome <!--$user-->!<p>
  You have <!--$index:0--> credits left in your account.<p>
```

Note that the PHP code is enclosed in HTML comments interleaved with literal text and markup.

## <?php ... ?> ("PHP tags")

We've learned to  start PHP programs with "<?php", but maybe you've occasionally forgotten to do that...

```
% cat hello-oops.php
echo "Hello, world!";
```

and gotten this:

```
% php hello-oops.php
echo "Hello, world!";
```

## PHP tags, continued

Early PHP only looked for code in HTML comments.

Modern PHP only looks for code in in <?php ... ?> blocks.

Text outside of <?php ... ?> is simply copied to standard output!

## PHP tags, continued

Here are two ways to write a program that shows what's in $_GET:

```
<?php
echo "Contents of \$_GET
<pre>";
var_dump($_GET);
echo "</pre>";
```

```
Contents of $_GET
<pre>
<?php var_dump($_GET); ?>
</pre>
```

dumpget.php and dumpgettags.php

Which is easier to read?

Conceptually, text outside <?php ... ?> is just echoed.

## PHP tags, continued

Which of these is easier to read? Which is faster?

```
<?php
echo "<!doctype html>
<title>1 to 10</title>
<ul>";

$i = 1;
while ($i <= 10) {
   echo "<li>$i";
   $i += 1;
    }

echo "</ul>";
```

```
<!doctype html>
<title>1 to 10</title>
<ul>

<?php
$i = 1;
while ($i <= 10) {?>
   <li><?php echo $i; ?><?php
   $i += 1;
   }
?>
</ul>
```

ten.php and tentags.php

## PHP tags, continued

Here's a slight variant that uses *alternative syntax for control structures* and a *short echo tag*:

```
<!doctype html>
<title>1 to 10</title>
<ul>
<?php
$i = 1;
while ($i <= 10): ?>          # colon instead of opening brace
   <li><?= $i ?> <?php        # <?= ... ?> for echo
   $i += 1;
endwhile;                     # endwhile instead of closing brace
?>
</ul>
```

## Best practice: don't end with ?>

It is considered a bad idea to have the PHP closing tag (the `?>`) at the end of the following program. Why?

```
<!doctype html>
<title>Testing</title>
<?php                                    # endtag.php
echo str_repeat("testing ", 100);
?>
```

## Cookies

## What is a cookie?

A cookie is a name/value pair that a site asks a browser to store on the site's behalf.

Whenever a browser hits a URL at a site, it sends the site's cookies in a Cookie header.

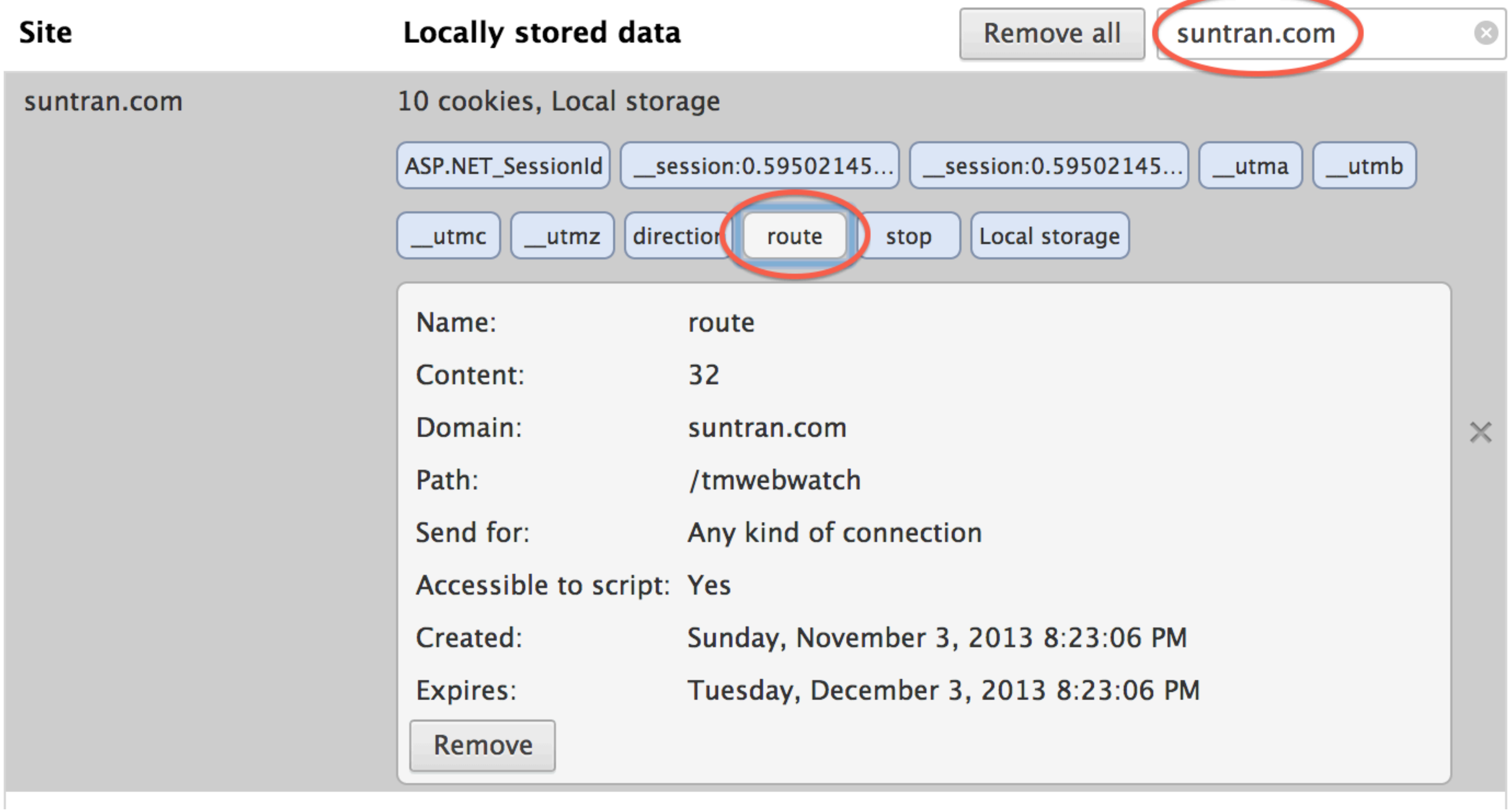
Cookies are another way to provide input to a web app.

Chrome lets us examine cookies.  Here are two ways:

> Hit chrome://settings/cookies#cookies
>
> In DevTools: Resources tab, then Cookies

## Example: SunTran cookies

Filtering my cookies with suntran.com shows this:



| Site | Locally stored data |
|---|---|
| suntran.com | 10 cookies, Local storage |

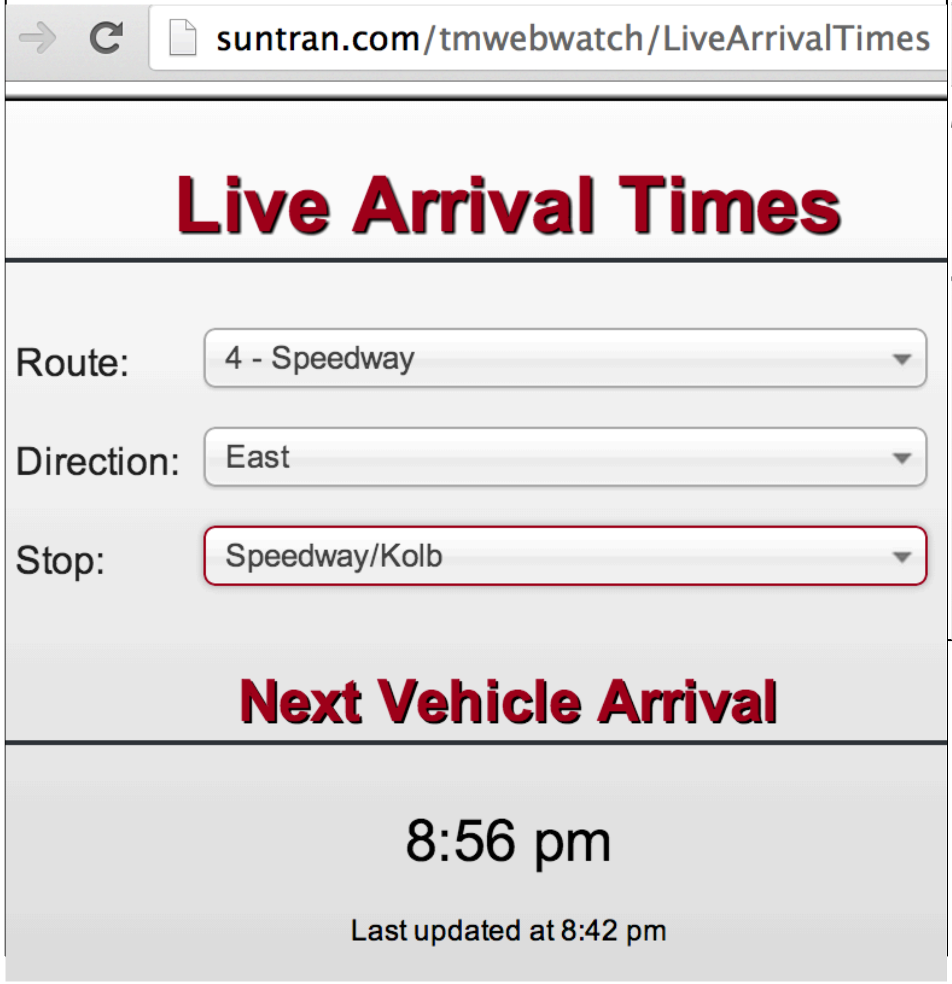
| | |
|---|---|
| Name: | route |
| Content: | 32 |
| Domain: | suntran.com |
| Path: | /tmwebwatch |
| Send for: | Any kind of connection |
| Accessible to script: | Yes |
| Created: | Sunday, November 3, 2013 8:23:06 PM |
| Expires: | Tuesday, December 3, 2013 8:23:06 PM |

What information is associated with the "route" cookie? How does "route" differ from other cookies?

Try the DevTools view, too!

# SunTran cookies, continued

With cookies present, hitting Live Arrival Times shows this:



suntran.com/tmwebwatch/LiveArrivalTimes

Live Arrival Times

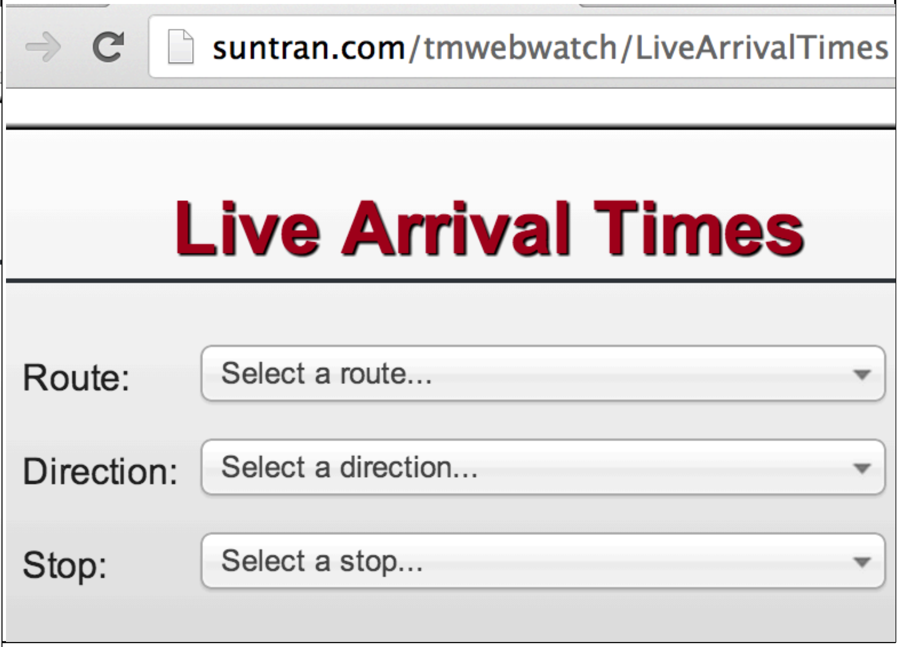
| | |
|---|---|
| Route: | 4 - Speedway |
| Direction: | East |
| Stop: | Speedway/Kolb |

Next Vehicle Arrival

8:56 pm

Last updated at 8:42 pm

If I delete the cookies, I see this:



suntran.com/tmwebwatch/LiveArrivalTimes

Live Arrival Times

| | |
|---|---|
| Route: | Select a route... |
| Direction: | Select a direction... |
| Stop: | Select a stop... |

Do cookies improve the user experience in this case?

## Cookies in headers

If I hit http://suntran.com/tmwebwatch/LiveArrivalTimes, here's the Cookie header that's **sent by the browser**:

```
GET /tmwebwatch/LiveArrivalTimes HTTP/1.1
....
Cookie: route=32; direction=2; stop=984;
__utma=243881607.944773664.1383537905.1383537905.
1383537905.1; __utmb=243881607.2.10.1383537905;
__utmc=243881607;
__utmz=243881607.1383537905.1.1.utmcsr=(direct)|
utmccn=(direct)|utmcmd=(none);
ASP.NET_SessionId=aplzjorggquuuzlwcsrvkrow; __session:
0.9430673203896731:enableLogin=true; __session:
0.9430673203896731:=http:
```

How many cookies are being sent?
How do cookies differ if we instead hit http://suntran.com?

## Cookies in headers, continued

If I hit http://cgi.cs.arizona.edu/classes/cs337/fall13/showcookies.php, here's the cookie header:

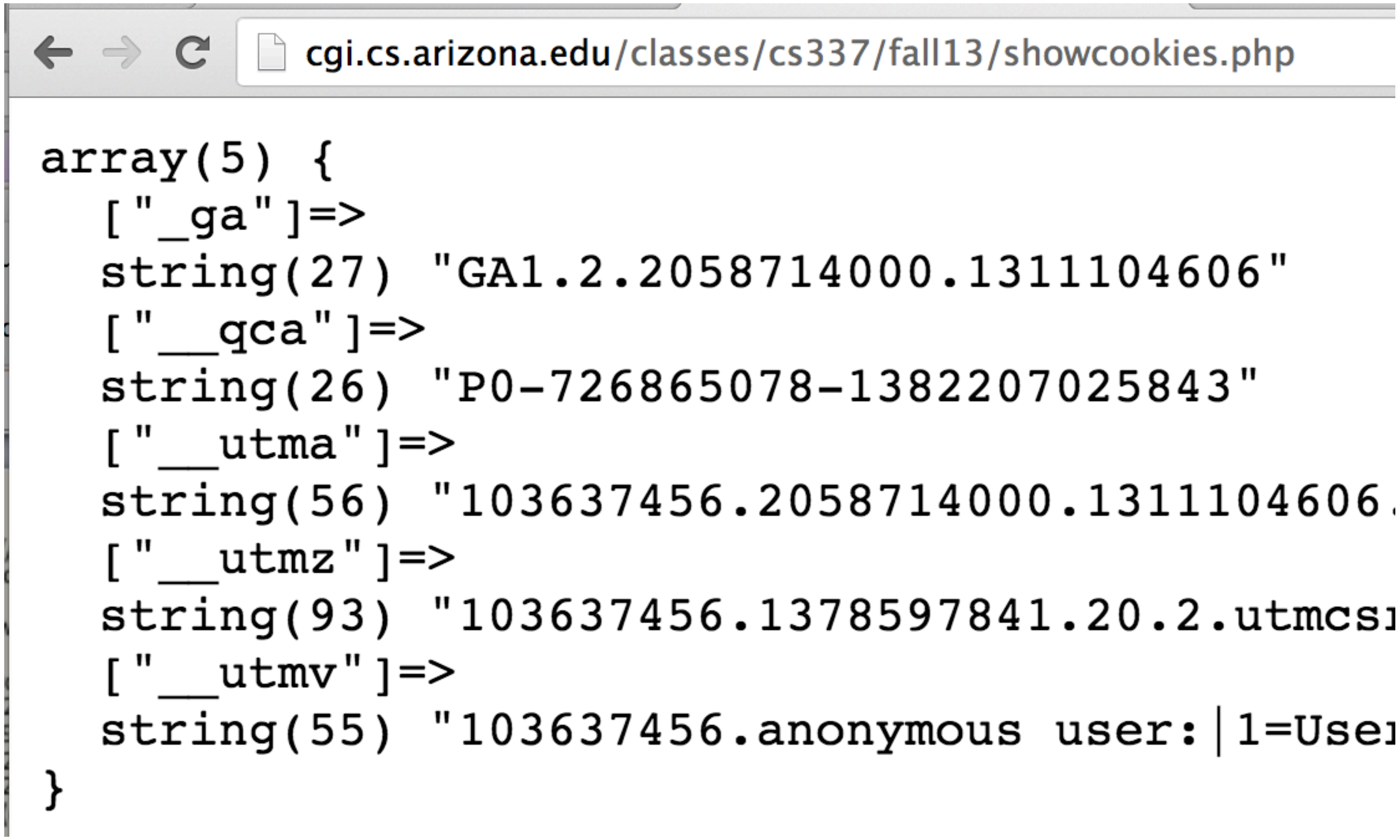
```
Cookie: _ga=GA1.2.2058714000.1311104606;
__qca=P0-726865078-1382207025843;
__utma=103637456.2058714000.1311104606.1383274970.138
3281857.50;
__utmz=103637456.1378597841.20.2.utmcsr=google|
utmccn=(organic)|utmcmd=organic|utmctr=(not%20provided);
__utmv=103637456.anonymous%20user%3A|1=User
%20roles=anonymous%20user=1;
__utma=110116631.537126843.1313707269.1383066322.1383
458132.74;
__utmz=110116631.1382291135.65.6.utmcsr=google|
utmccn=(organic)|utmcmd=organic|utmctr=catcard
```

_ga and the __ut* cookies are for Google Analytics. __qca is for QuantCast analytics.

## Examining cookies with PHP

When a PHP page is hit, $_COOKIE is populated with the cookies sent in the HTTP request's Cookie: header.

showcookies.php simply does var_dump($_COOKIE):



cgi.cs.arizona.edu/classes/cs337/fall13/showcookies.php

```
array(5) {
  ["_ga"]=>
  string(27) "GA1.2.2058714000.1311104606"
  ["__qca"]=>
  string(26) "P0-726865078-1382207025843"
  ["__utma"]=>
  string(56) "103637456.2058714000.1311104606.
  ["__utmz"]=>
  string(93) "103637456.1378597841.20.2.utmcs
  ["__utmv"]=>
  string(55) "103637456.anonymous user:|1=Use
}
```

## Creating a cookie

PHP's `setcookie(...)` function causes a Set-Cookie header to be generated.

Trivial example: (cookie1.php)

```
<!doctype html>
<title>Cookies</title>
<?php
setcookie("last_visit", time());
```

`time()` returns the number of seconds since the UNIX "epoch" (Jan 1 1970 00:00:00 GMT)

What can this cookie be used for?

## Creating a cookie, continued

Let's hit it with curl:

```
% curl -i http://localhost/c/cookie1.php
HTTP/1.1 200 OK
Date: Mon, 04 Nov 2013 05:15:10 GMT
Server: Apache/2.4.4 (Unix) ...
X-Powered-By: PHP/5.4.19
Set-Cookie: last_visit=1383542110
Content-Length: 39
Content-Type: text/html

<!doctype html>
<title>Cookies</title>
```

```
% cat cookie1.php
<!doctype html>
<title>Cookies</title>
<?php
setcookie("last_visit", time());
```

Will we be able to see this cookie in Chrome?

## Creating a cookie, continued

Let's do this with Chrome:

(1) Delete cookies for localhost

(2) Open Network tab in DevTools

(3) Hit localhost/c/cookie1.php

(4) Examine Request and Response headers for cookie.

(5) Hit cookie1.php

(6) Examine headers again

## Using a cookie

What does this program do?

```
<!doctype html>
<title>Cookies</title>
<?php
error_reporting(22519);

$cookie = $_COOKIE["last_visit"];
if ($cookie) {
   $elapsed = time() - $cookie;
   echo "You were last here $elapsed seconds ago.";
   }
else
   echo "Hmm... Is this your first time here?";

setcookie("last_visit", time());
```

Let's run it! (cookie2.php)

## The big picture

The PHP setcookie(...) function causes the <u>response</u> (the PHP output) to a request to have a Set-Cookie header.

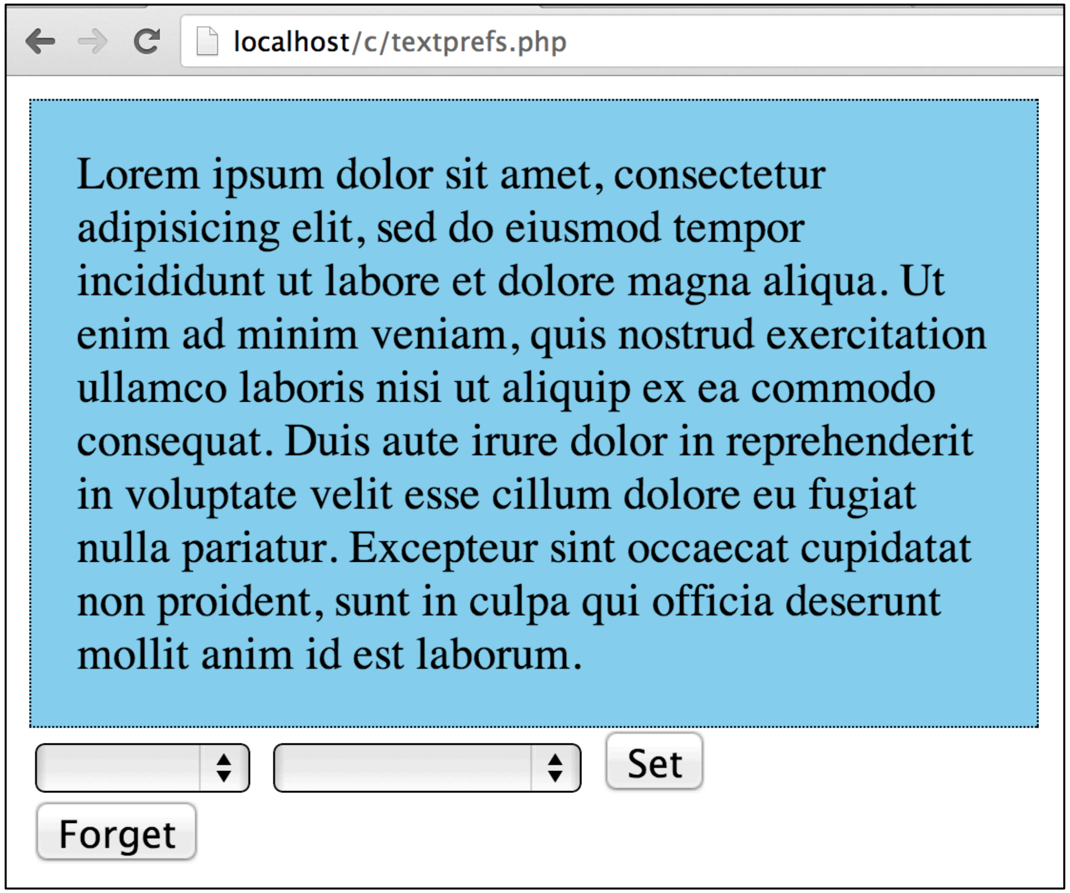
When a browser sends a request (a GET or POST) to a site, all* the cookies for the site are put in a single Cookie: header.

In other words...

> A web app at x.com tells the browser to store a cookie.
>
> When the browser sends a request to x.com, it sends all* the cookies for x.com, not knowing which, if any, the app being hit might use.

## Larger example: Text preferences

textprefs.php uses a cookie to hold size and background-color preferences.



localhost/c/textprefs.php

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Let run it!

## Text preferences, continued

Testing:

(1) Delete text_prefs cookie for localhost
(2) Hit textprefs.php.  Confirm white background.
(3) Set size and background.
(4) Close browser.
(5) Reopen and confirm size and background same.
(6) Open new tab, hit app, set new color and size.
(7) Reload in first tab, confirm color and size.
(8) Click Forget, then reload in second tab.

## Text preferences, continued

Here's the main logic.

```
error_reporting(22519); include("lorem.php");

$size = "10"; $bgcolor = "white";

$cookie = $_COOKIE["text_prefs"];

if ($_POST["size"]) {
    $size = $_POST["size"];
    $bgcolor = $_POST["bgcolor"];
    setcookie("text_prefs",                    // expire in 7 days
        "{$_POST['size']}/{$_POST['bgcolor']}", time() + 7*24*60*60);
    }
elseif ($_POST["forget"]) {
    setcookie("text_prefs", "", time() - 24*60*60); # delete with prior
time
}
elseif ($cookie) {
    list($size,$bgcolor) = explode("/", $cookie); # parallel assignment
    }

echo "<div style='font-size:{$size}px; background-color:$bgcolor'>";
echo lorem(); echo "</div>";
```

## Text preferences, continued

Here are the forms. Note that two forms are used—why?

```
<form action=textprefs.php method=post>
   <select name=size required>
      <option></option>
      <option value=16>Small</option>
      <option value=24>Medium</option>
      <option value=32>Large</option>
   </select>
   <select name=bgcolor required>
      <option></option>
      <option value=antiquewhite>Antique White</option>
      <option value=lightcoral>Light Coral</option>
      <option value=skyblue>Sky Blue</option>
   </select>
   <input type=submit name=submit value=Set>
</form>
<form action=textprefs.php method=post>
   <input type=submit name=forget value=Forget>
</form>
```

## Cookie details

Cookies are far from bulletproof:

Users can disable storage of cookies.

Users can delete cookies.

Users can hack cookies.

setcookie(...) has some additional arguments that we didn't talk about.