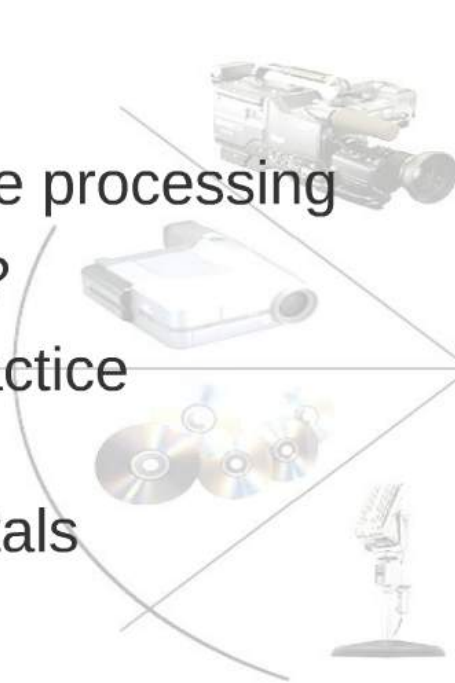


# Server-Side Scripting

- Servers
- Server-side processing
- Why PHP?
- SSP in practice
- PHP fundamentals

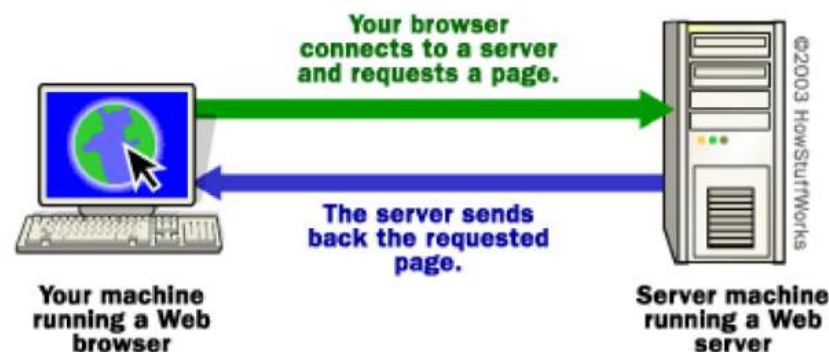


Multimedia  
Computing  
& the World-Wide Web

Server-Side Scripting  
Introduction to PHP

# Loading a Web Page

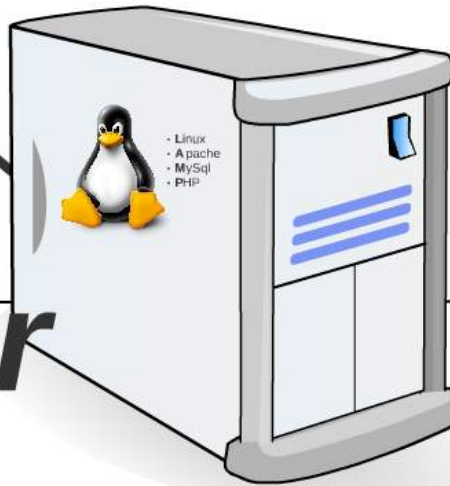
- ◆ User clicks on a link in a Browser (or types in a URL)
- ◆ Your browser forms a connection to a *Web server*, requests a page, and receives and renders it
  - a web server is a computer on the internet that hosts web documents
  - the term web server is also used to describe the software that hosts these documents





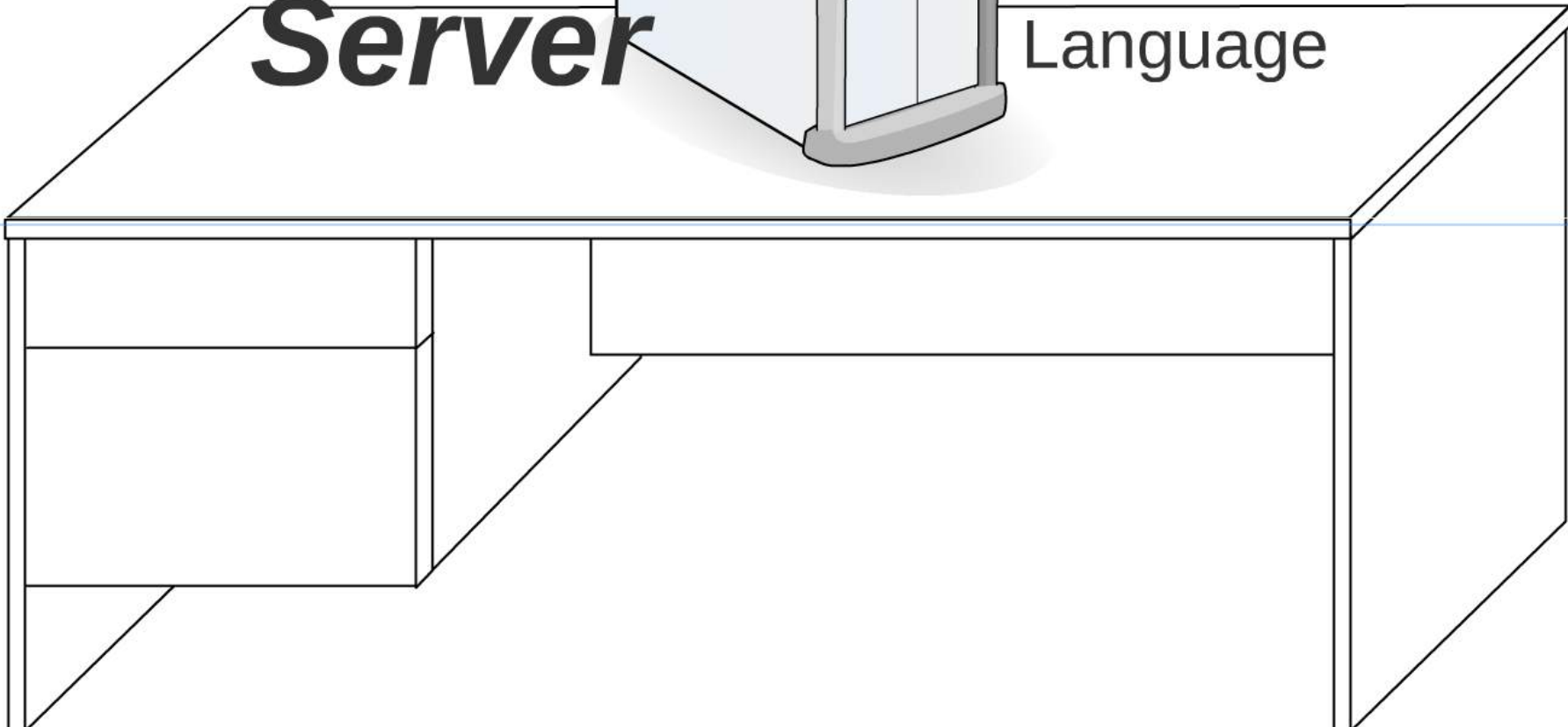
208.80.154.224

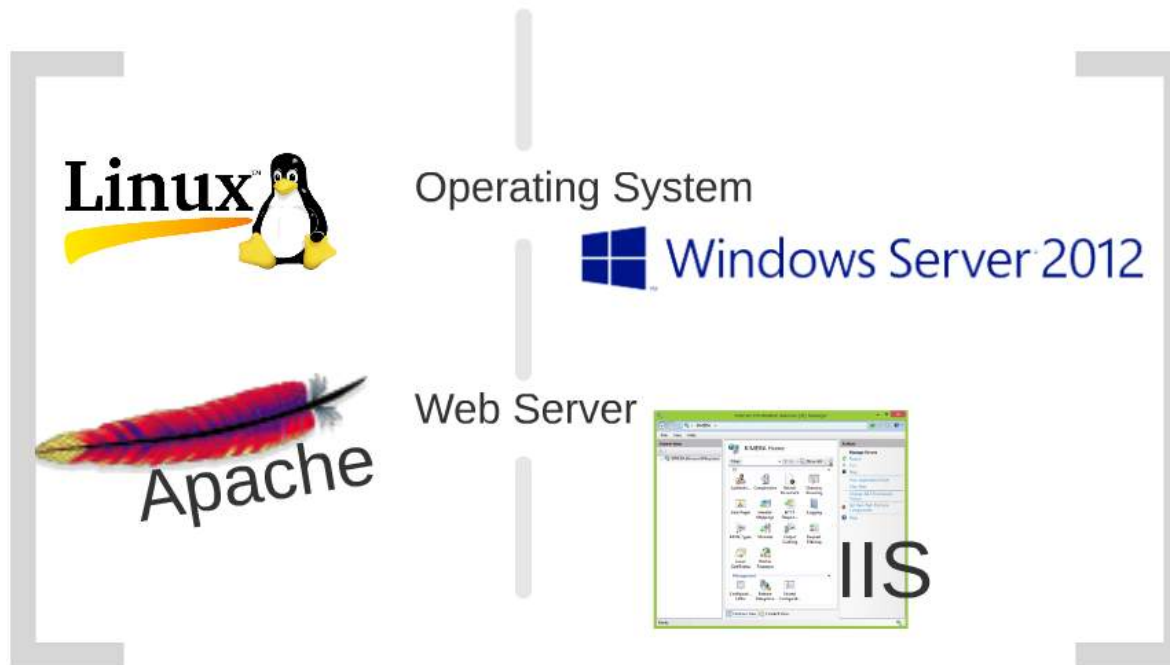
Port 80



Operating System  
Web Server  
Language

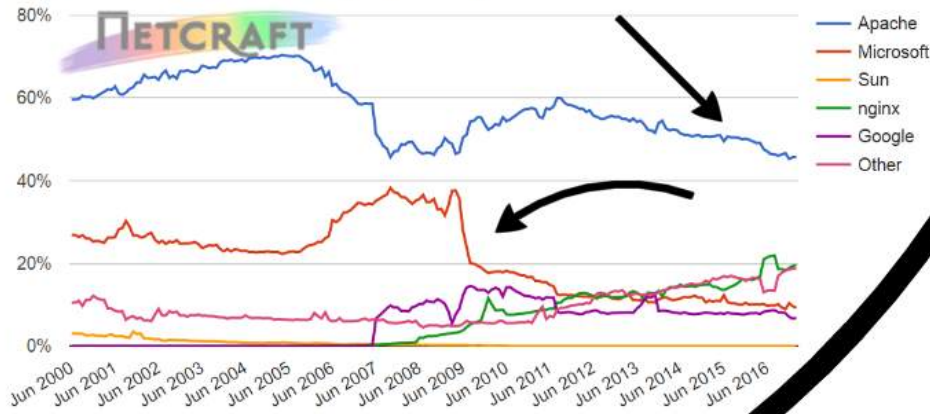
**Server**





# Web servers

Web server developers: Market share of active sites



# PHP

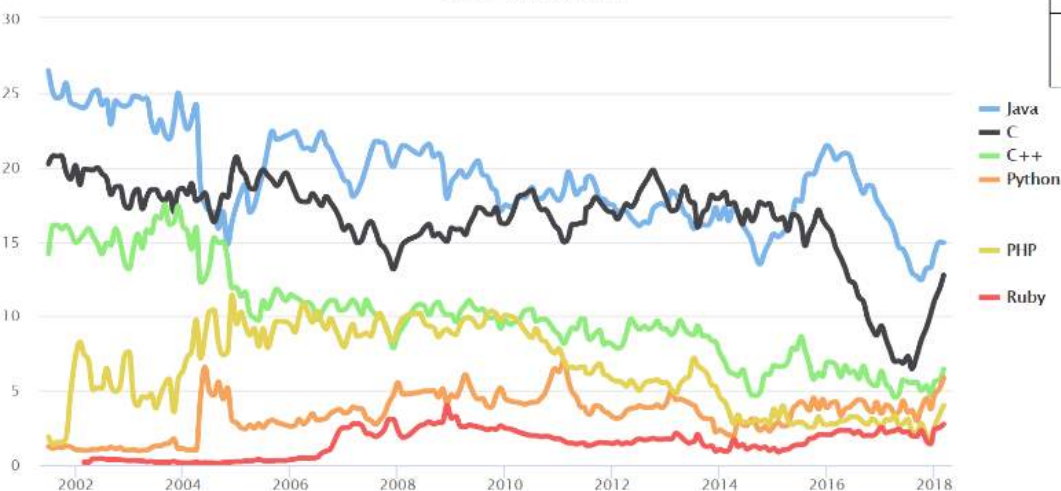
<b>PHP</b>	83.2%
<b>ASP.NET</b>	13.8%
<b>Java</b>	2.4%
<b>static files</b>	1.7%
<b>ColdFusion</b>	0.6%
<b>Ruby</b>	0.5%
<b>JavaScript</b>	0.4%
<b>Perl</b>	0.3%
<b>Python</b>	0.2%
<b>Erlang</b>	0.1%

W3Techs.com, 18 March 2018

Percentages of websites using various server-side programming languages  
Note: a website may use more than one server-side programming language

TIOBE Programming Community Index

Source: www.tiobe.com



number of skilled engineers world-wide, courses and third party vendors.

GitHub: no.5  
TIOBE: no.7  
StackOverflow: no.6  
Indeed: no.6



# PHP

## Pros



- Well documented
- Great for connecting with databases (especially MySQL)
- Perfectly designed for delivering web content

## Cons


- Doesn't scale well
- Can be difficult to debug
- messy code
- not particularly consistent

# The "Dynamic Web"

- ◆ the web is not static
  - reactive
  - customisable
  - gives different HTML under different conditions, e.g.
    - Google search results – these are generated dynamically
    - online banking – your bank details are different from everyone else



## Why use server side scripting? e.g. PHP

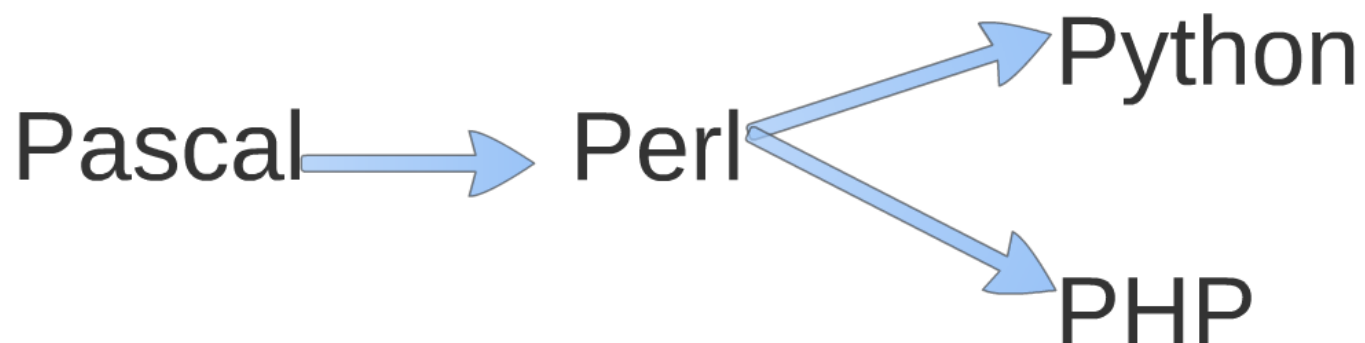


- 100% essential for user generated content
- virtually essential for database use with a website
- adds significant additional capabilities to websites
- significantly reduces maintenance of websites



# PHP

- ◆ PHP is a HTML-embedded scripting language.
- ◆ Much of its syntax is borrowed from C, Java and Perl with a couple of unique PHP-specific features thrown in
- ◆ The goal of the language is to allow web developers to write dynamically generated pages quickly and (fairly) painlessly



# What is PHP?

- ◆ PHP is an *embedded language*.
  - PHP code is written right into the web document
  - other languages used to make dynamic web pages are not embedded, but rather generate their own page  
e.g Java Servlets

# How do I make it work

- ◆ To make PHP work you must tell the web server two things:
  - that the file it's serving is a PHP file
    - usually this is done by naming the file *filename.php*
  - demarcate the areas that need to be interpreted as PHP instead of served verbatim as HTML

Filename	Filesize	Filetype	Last modified	Permissions	Owner/Group
..					
old stuff		File folder	17/01/2018 22:23:...	drwxrwxr-x	duncanw duncanw
foo.php	0	PHP File	18/03/2018 13:14:...	-rw-rw-r--	duncanw duncanw
index.html	3,229	Chrome HT...	26/02/2018 17:30:...	-rw-rw-r--	duncanw duncanw
js.html	928	Chrome HT...	21/02/2018 20:46:...	-rw-rw-r--	duncanw duncanw
monty.py	0	PY File	18/03/2018 13:14:...	-rw-rw-r--	duncanw duncanw
robot.mp3	2,177,704	MP3 Forma...	21/02/2018 20:32:...	-rw-rw-r--	duncanw duncanw

# What is PHP?

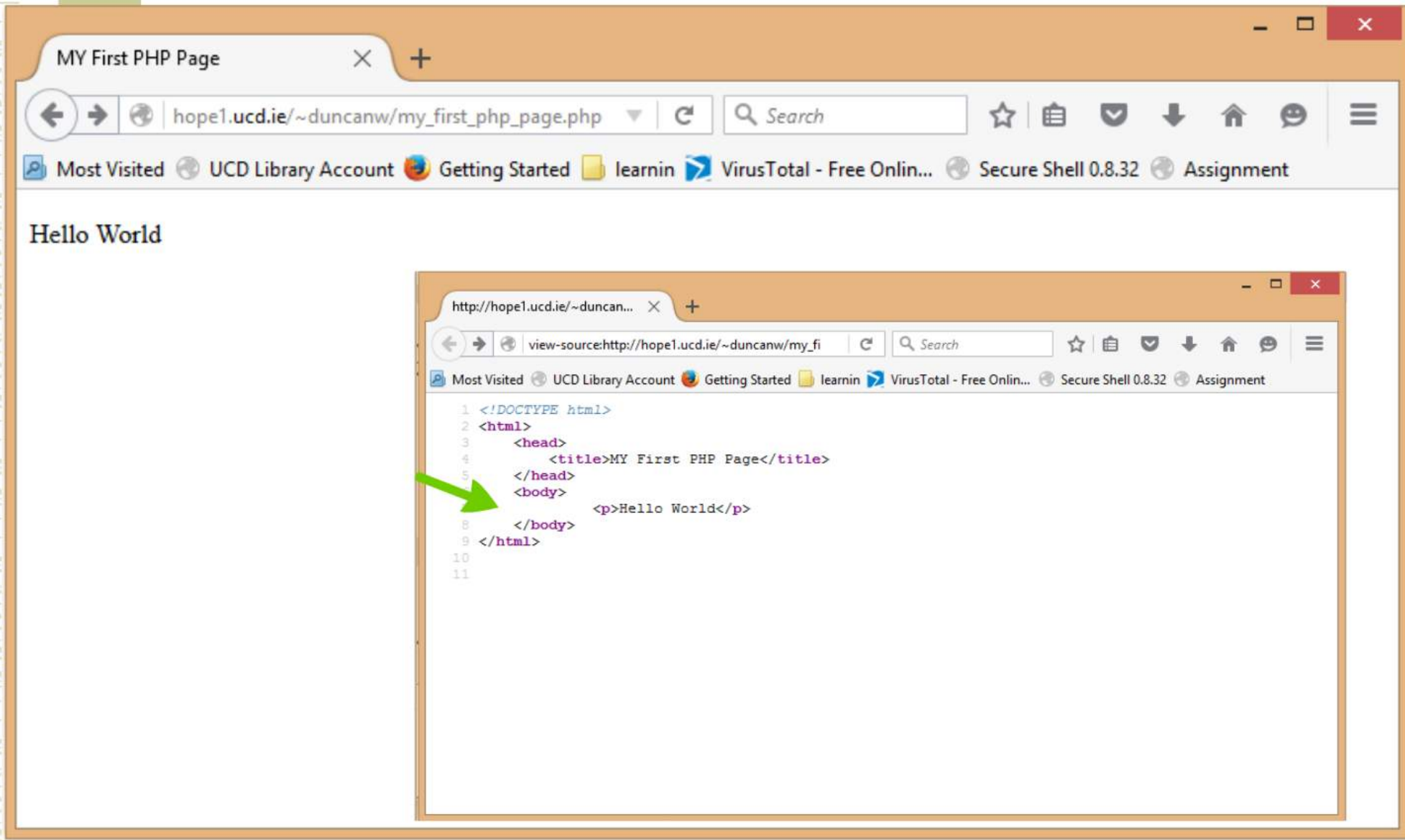
- when creating a PHP file, just name it foo.php instead of foo.html
  - lets the server know that it there may be some PHP code in here to be interpreted
- so the server will go through the file, looking for parts that it needs to pass to the PHP interpreter
  - it does this before serving the document
- those parts are marked with a special tag:  
`<?php code here ?>`

# Hello World – PHP

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>MY First PHP Page</title>
5   </head>
6   <body>
7     <?php
8     echo ("\t <p>Hello World</p> \n");
9     ?>
10  </body>
11 </html>
```

} *PHP*

# PHP Hello World Source View



# What happened?

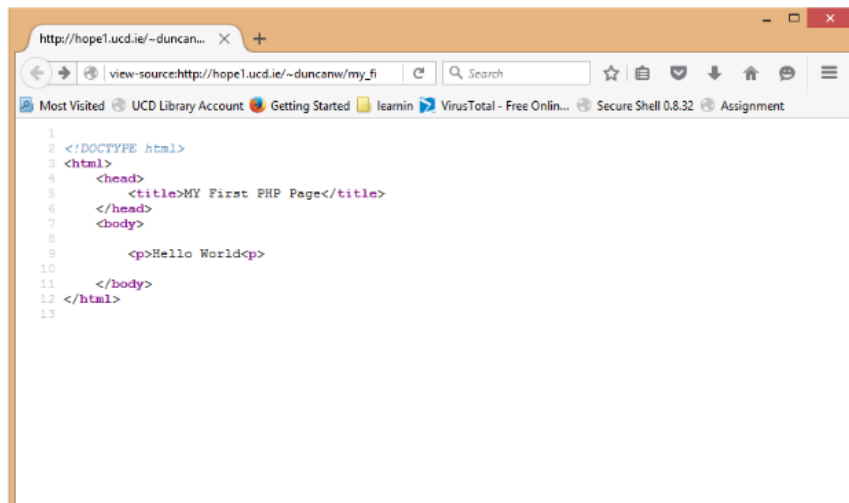
- ◆ The source view didn't contain any PHP
  - why not?
- ◆ Remember how JavaScript scripts are used to write HTML
  - The browser reads the HTML document and executes JavaScript as it appears in the document
    - i.e. `document.write("This is JS");`
- ◆ With server side scripting, the server reads the PHP document and executes anything contained within the `<?php ?>` tags



# What happened?

- ◆ This is because a PHP file doesn't look like a HTML file, and a browser wouldn't know what to do with it.
  - PHP is a **server-side scripting language**, because the server does all of the work
  - by contrast, JavaScript is a **client-side scripting language**, because the client -- the web browser -- is smart enough to understand it.

```
1 <?php
2 if (8<4)
3     print ("404 error: can't find maths");
4 else print("
5 <!DOCTYPE html>
6 <html>
7     <head>
8         <title>MY First PHP Page</title>
9     </head>
10    <body>
11
12        <p>Hello World</p>
13
14    </body>
15 </html>
16 ");
17 ?>
```



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>MY First PHP Page</title>
5 </head>
6 <body>
7
8 <p>Hello World</p>
9
10 </body>
11 </html>
```

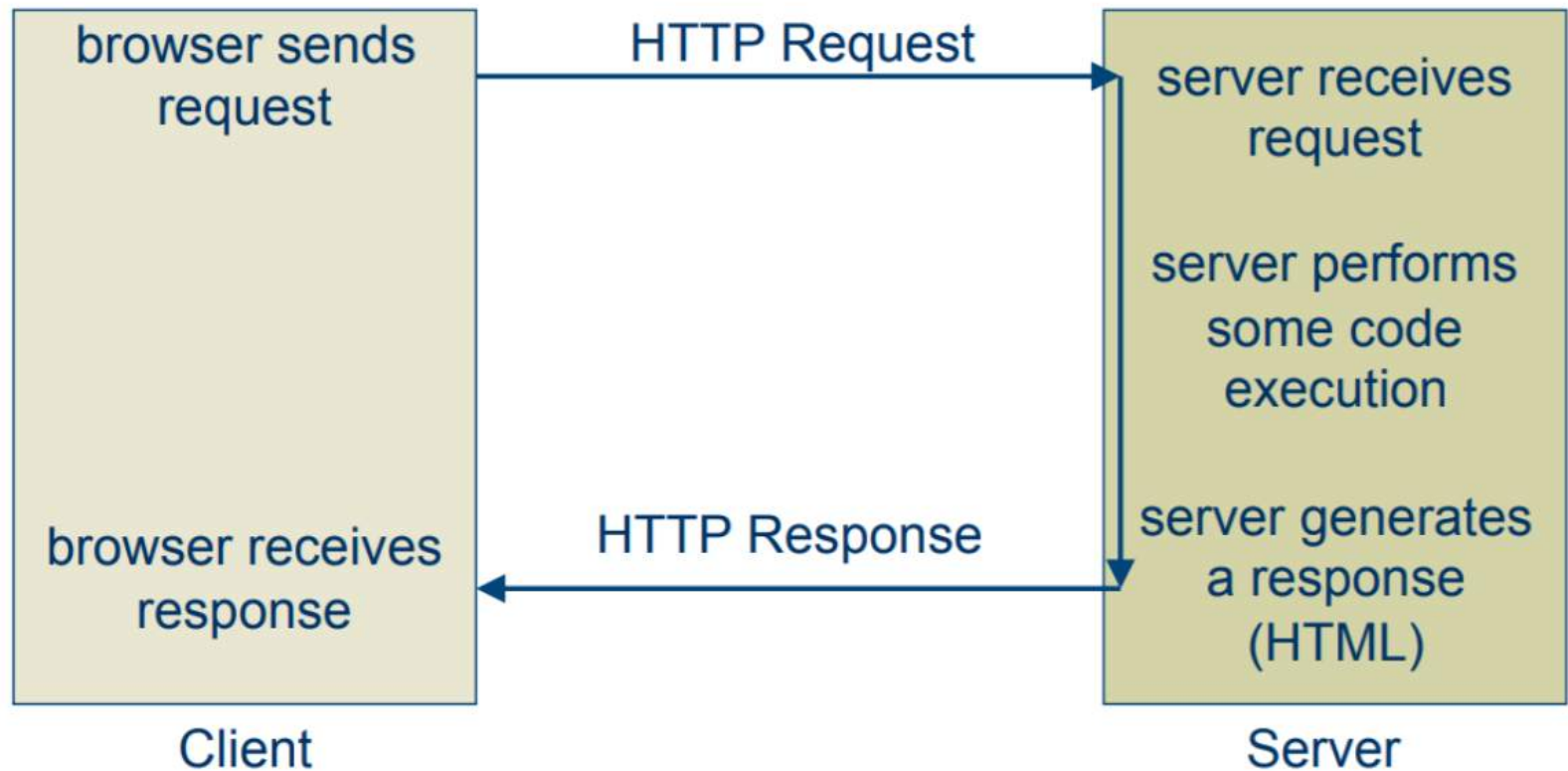


# Client side Versus Server Side



- ◆ Client side scripting
  - executed by the user's web browser
- ◆ Server-side scripting
  - executed by the web-server
  - code is not observable by the end user

# Server Side Execution



# A Simple Example

- ◆ Web counter
  - indicates the number of visitors, or hits, a particular webpage has received
- ◆ Server keeps track of the number of visitors
  - this knowledge can only be known by the server
    - since it is the one doing the counting
  - therefore server side processing is needed to perform the task
    - even if it's just to read a value from a file
    - and add one to the value in that file every time the page is requested

# Variables

- ◆ Variables begin with \$

```
$number = 1;
```

```
$name = "Mark";
```

- ◆ Loosely typed (like JavaScript, unlike Java)
- ◆ All statements end with a semi-colon (;)
- ◆ Variables may only be comprised of alpha-numeric characters and underscores. a-z, A-Z, 0-9, or \_
  - Variables must start with a letter
- ◆ Value of uninitialised variables depends on context
  - numeric context it is 0
  - string context it is empty string ""

# Assignment Operators

- ◆ Operators the same as Java (and C)

Operator	English
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

# Variable Scope


- ◆ The scope of a variable is the context within which it is defined
  - Usually just a single scope level between `<?php` and `?>`
- ◆ If you use functions and want access to global variables
  - Use `global` keyword



# echo

- ♦ To output anything, i.e. to insert strings or variables into HTML use `echo`

```
<?php
    $my_string = "My name is: ";
    $newline = "<br/>";
    $name = "Mark";
    echo $name;
    echo $newline;
    echo $my_string.$name.$newline;
?>
```

A diagram consisting of two blue arrows. One arrow originates from the variable `$name` in the line `echo $my_string.$name.$newline;` and points to the output `Mark` in the first line of the output. The second arrow originates from the variable `$newline` in the same line and points to the output `<br/>` in the second line of the output.

# Comparison Operators

`$x = 2;`

`$y = 3;`

Operator	English	Example	Result
<code>===</code>	Equal to (no type conversion)	<code>\$x===\$y</code>	False
<code>!==</code>	Not equal to (no type conversion)	<code>\$x!==\$y</code>	True
<code>&lt;</code>	Less than	<code>\$x&lt;\$y</code>	True
<code>&gt;</code>	More than	<code>\$x&gt;\$y</code>	False
<code>&lt;=</code>	Less than or equal to	<code>\$x&lt;=\$y</code>	True
<code>&gt;=</code>	More than or equal to	<code>\$x&gt;=\$y</code>	false

`<=>`

`$x<=>$y`

Spaceship

-1

(True)

# if and else

```
if($name == "Joe"){  
    echo "Hello Mr. Bloggs";  
} elseif($employee == "Bob"){  
    echo "Good Morning Sir!";  
} else {  
    echo "Good Morning";  
}
```

# Looping

- ◆ For loops

```
for ($i = 0; $i < 10; $i += 1) {  
    //do this code;  
}
```

- ◆ While loops

```
while ($statement == true) {  
    //do this code;  
}
```

# Arrays

- Are actually ordered maps.
- Keys :: values.
- Handily, PHP has the keys, by default as auto-incrementing integers: 0..n
- Accessing Array indexes can be done as in other languages. `$my_array[0]` is the first position, `$my_array[1]` is the second, and so on.
- However, indexing can be messed around with:  
0, 1, 2, 3, 8, 9, 10...
- Variable length

```
$intgr_array[0] = 1;  
$intgr_array[1] = 7;  
echo (intgr_array[0] + intgr_array[1]);
```

**outputs: 8**

```
$myArray = array('red','blue','green');  
$size = sizeof($myArray);  
echo ($size);
```

**outputs: 3**

# Arrays

Now the unusual stuff

Array of bytes. Doesn't care about type.

```
$foo = 1 + "bob3";  
// $foo is integer (1)
```

```
$foo = 1 + "10 Small Pigs";  
// $foo is integer (11)
```

```
$foo = 4 + "10.2 Little Piggies";  
// $foo is float (14.2)
```



```
<?php
$my_happy_array = array("happiness",);
var_dump($my_happy_array);
?>
```

```
array(1) {
  [0]=> string(9) "happiness"
}
```

```
<?php
```

```
$my_array = array("h", "a", 'p', 'p', 8 => "i", "n","e","s","s",);
```

```
var_dump($my_array);
```

```
?>
```

```
array(9) {
```

```
    [0]=> string(1) "h"
```

```
    [1]=> string(1) "a"
```

```
    [2]=> string(1) "p"
```

```
    [3]=> string(1) "p"
```

```
    [8]=> string(1) "i"
```

```
    [9]=> string(1) "n"
```

```
    [10]=> string(1) "e"
```

```
    [11]=> string(1) "s"
```

```
    [12]=> string(1) "s"
```

```
}
```

# Arrays

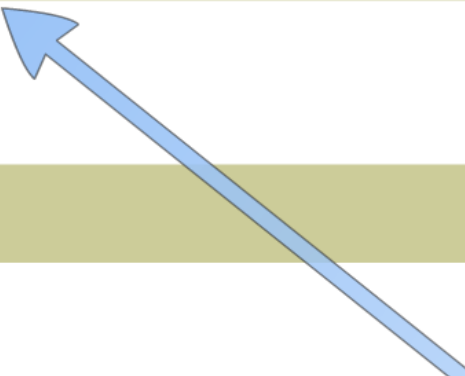
- ◆ Numerically indexed arrays

```
$firstnames[0] = "alice";  
$firstnames[1] = "bob";  
$firstnames[2] = "carol";
```

- ◆ To retrieve a value

```
echo $firstnames[2];
```

- "carol"



implicit 2 dimensional array  
(array of strings)

# Arrays

- ◆ Associative Arrays (like hash tables in Java)

```
$lastnames["alice"] = "jones";  
$lastnames["bob"] = "bones";  
$lastnames["carol"] = "gones";  


---

$age["alice"] = 24;  
$age["bob"] = 25;  
$age["carol"] = 26;
```

- ◆ To retrieve a value

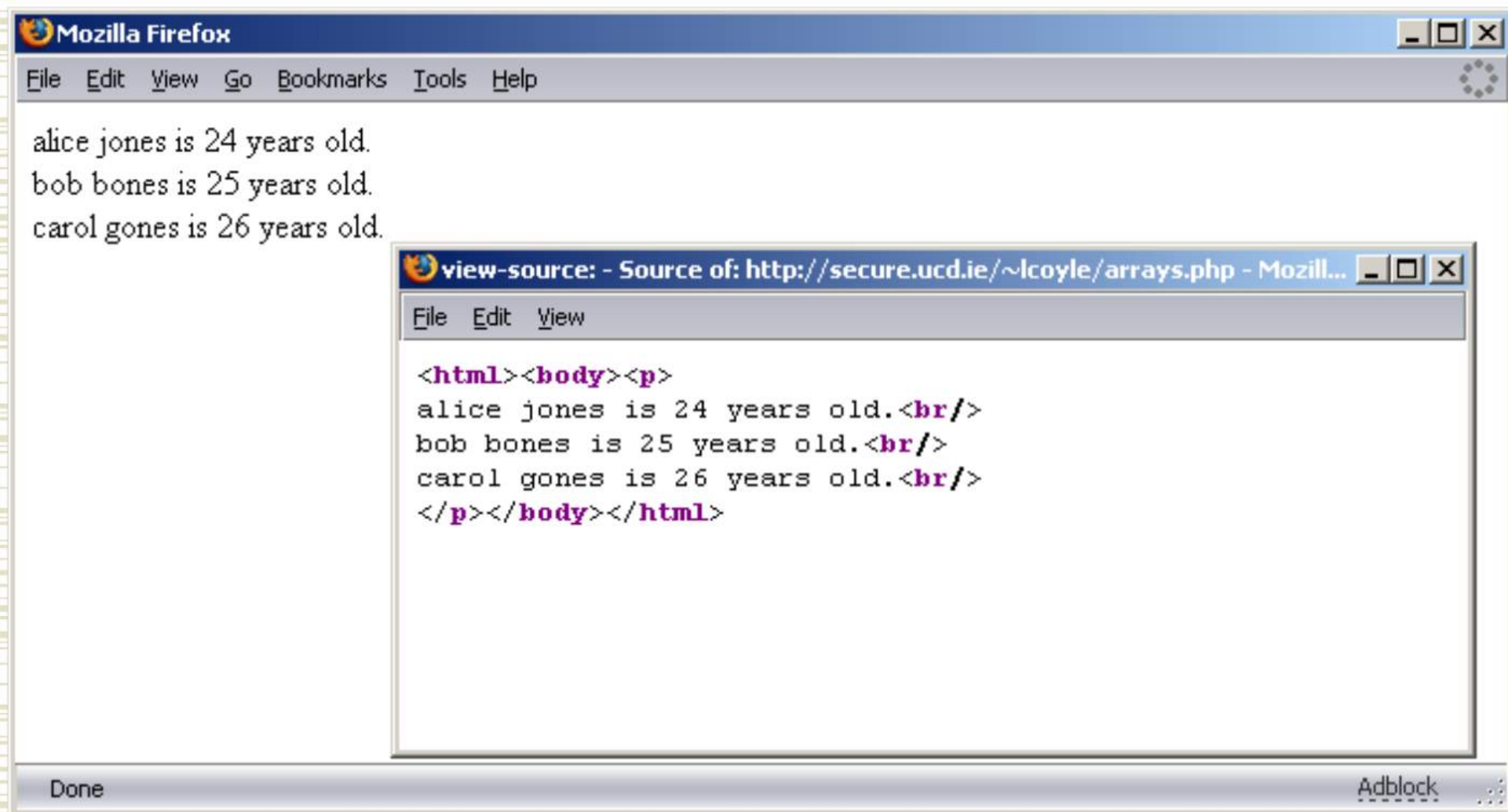
```
echo ($age["bob"]);  
■ 25
```

# Looping through Arrays

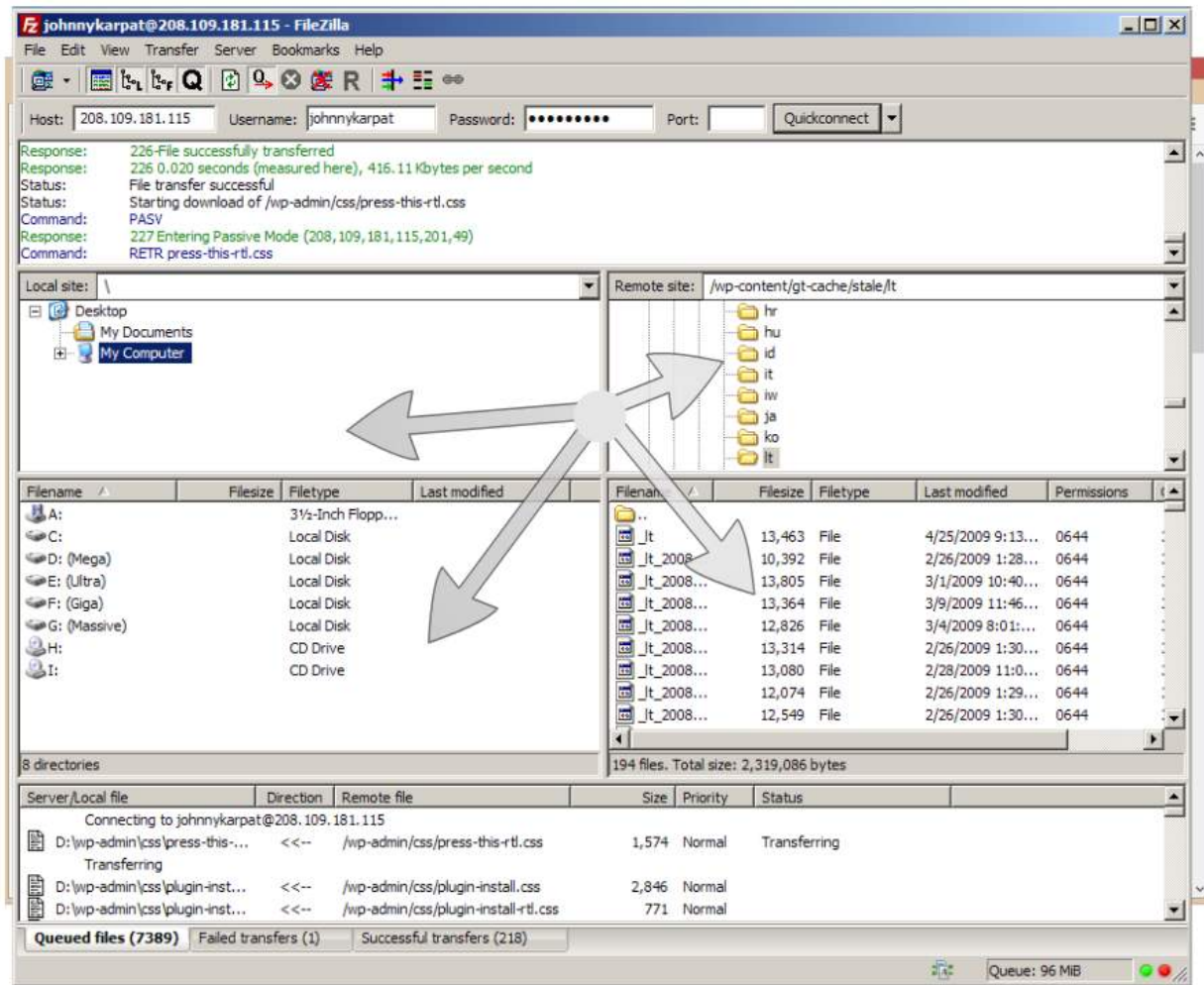
```
<html><body><p>
<?php
for($i = 0; $i < 3; $i+=1){
    $firstname = $firstnames[$i];
    echo $firstname." ".$lastnames[$firstname];
    echo " is ".$age[$firstname];
    echo " years old.<br/>\n";
}
?>
</p></body></html>
```

```
foreach ($lastnames as $i) {  
    echo ("last name is: ". $i."\n");  
}
```

# Arrays Source View







ftp://ftp.ncbi.nlm.nih.gov/  
ftp://ftp.software.ibm.com/



# Next Class



- Communicating with the web server
  - **POST** and **GET**
- Uses and examples of PHP
  - form processing
  - databases

## Resources:

- [php.net](http://php.net)
- [webmonkey.com](http://webmonkey.com)