

# Web Scripting using PHP

## Server side scripting

### So what is a Server Side Scripting Language?

- Programming language code embedded into a web page

PERL
<b>PHP</b>
PYTHON
ASP

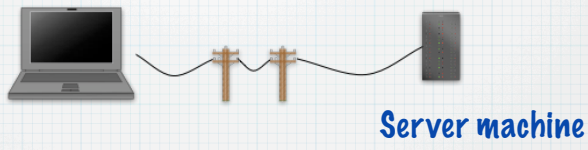
### Different ways of scripting the Web

- Programming language code embedded into a web page

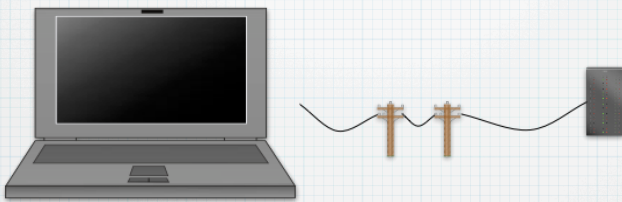
No scripting (plain markup)
Client Side scripting
Server Side scripting
Combination of the above (AJAX)

## No Scripting example - how it works...

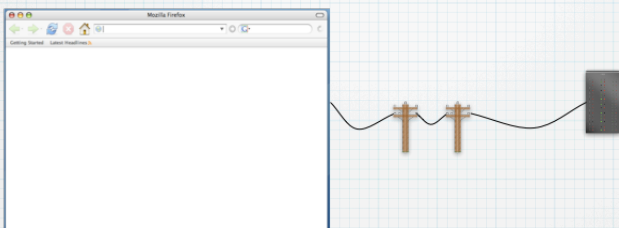
User on a machine somewhere

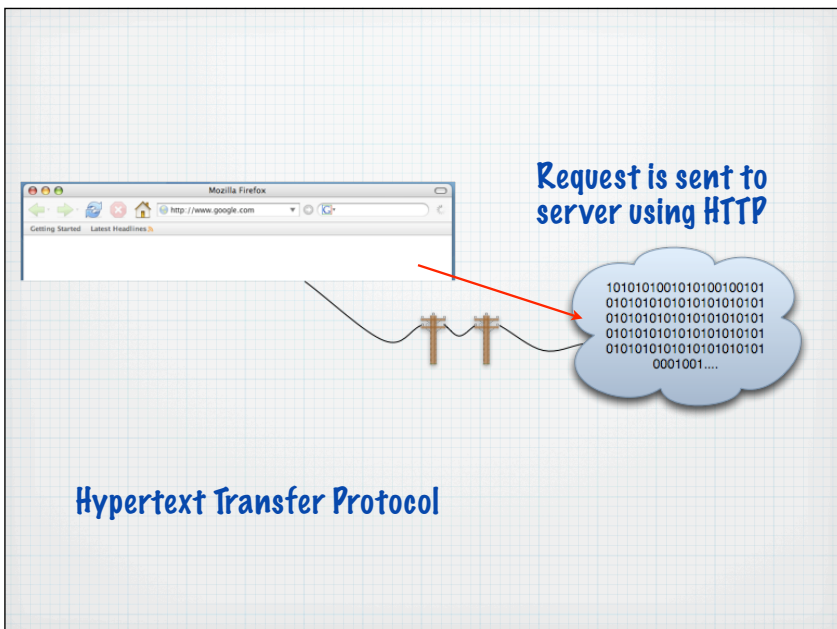
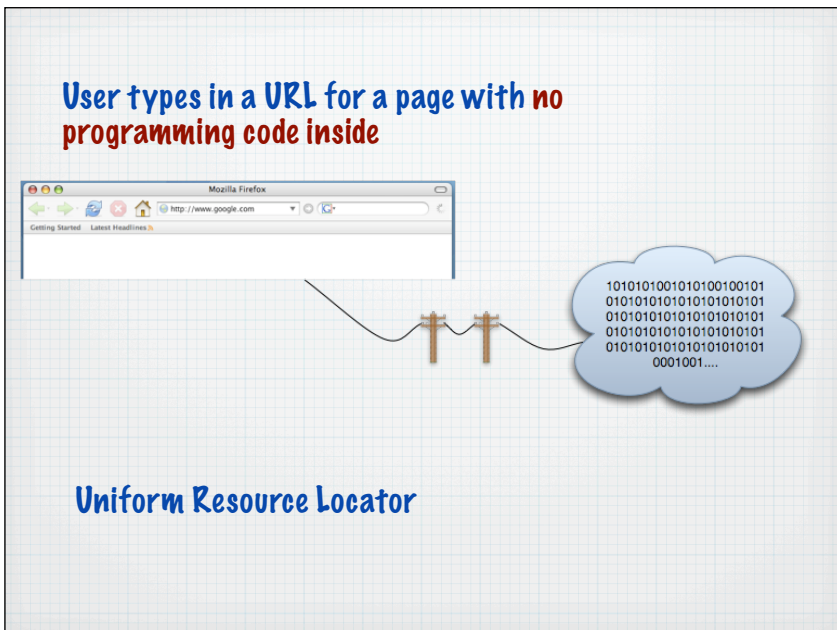
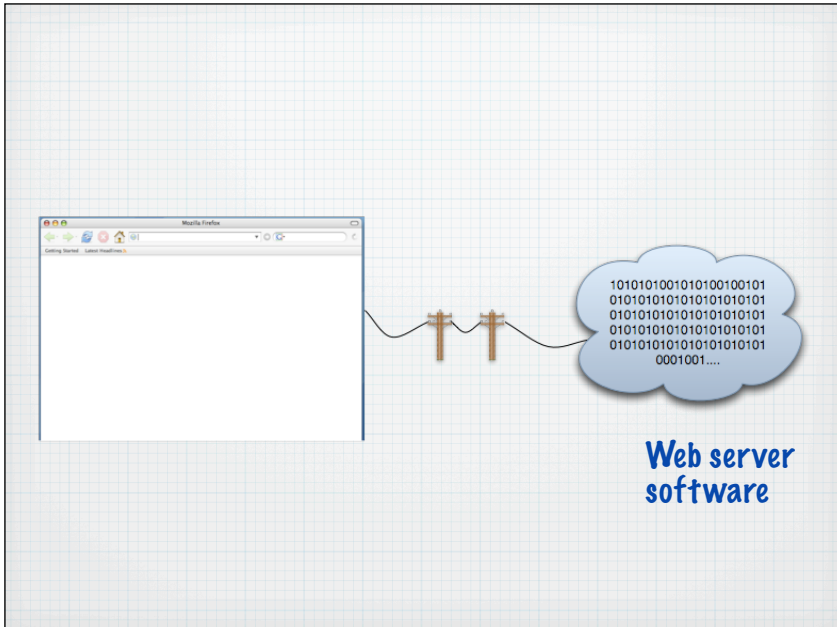


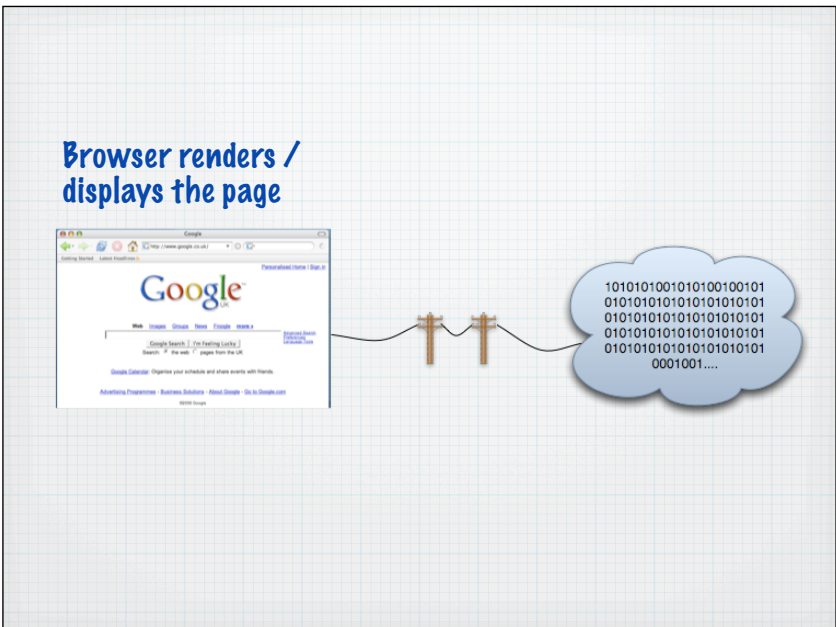
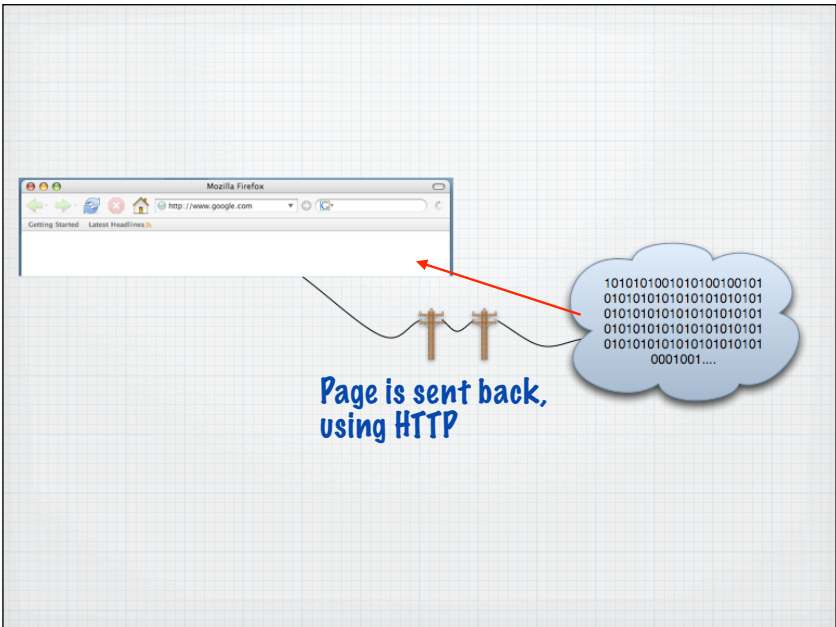
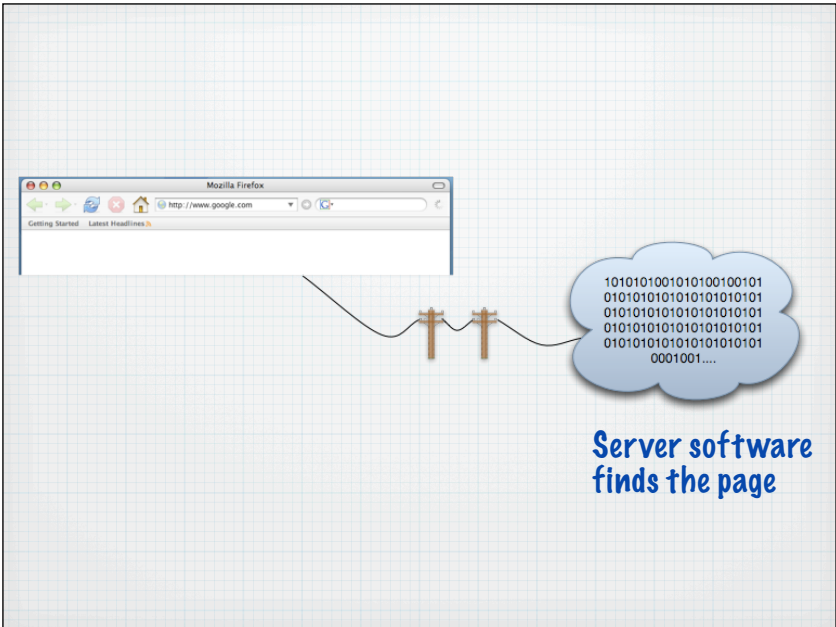
Being more specific...



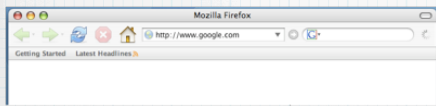
Web Browser software





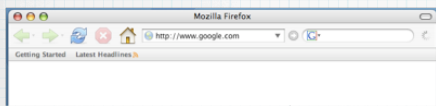


## Server Side scripting



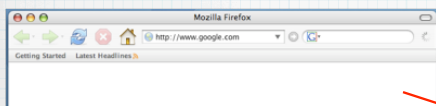
10101001010100100101  
01010101010101010101  
01010101010101010101  
01010101010101010101  
01010101010101010101  
0001001....

## User types in a URL for a page with PHP code inside

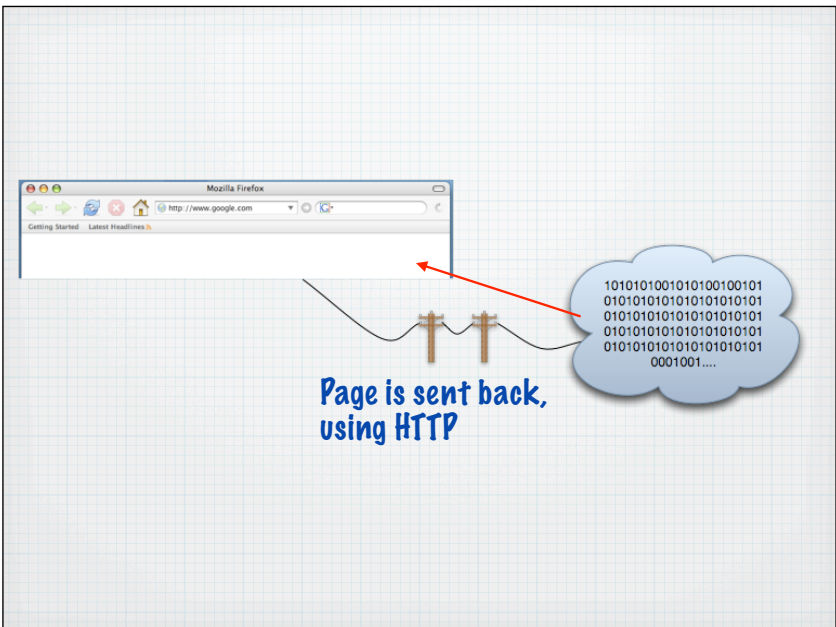
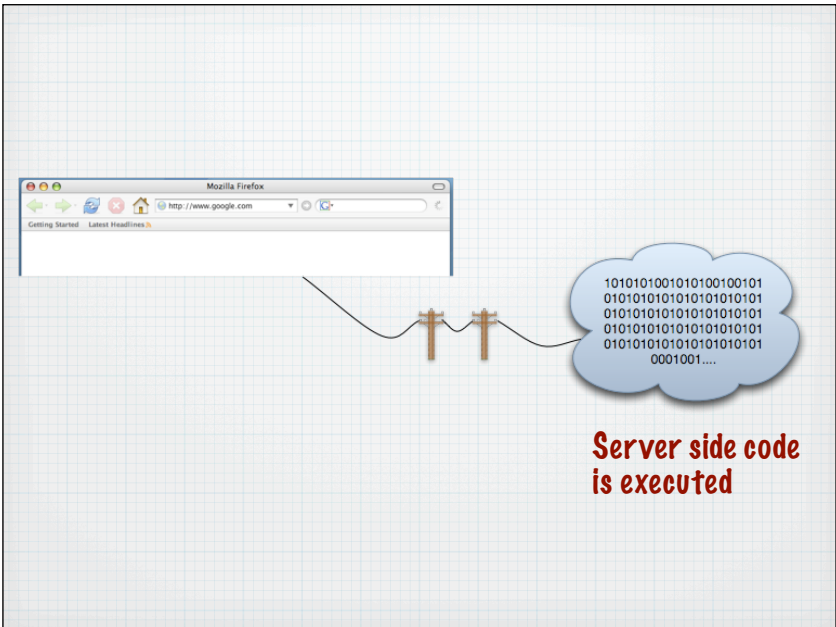
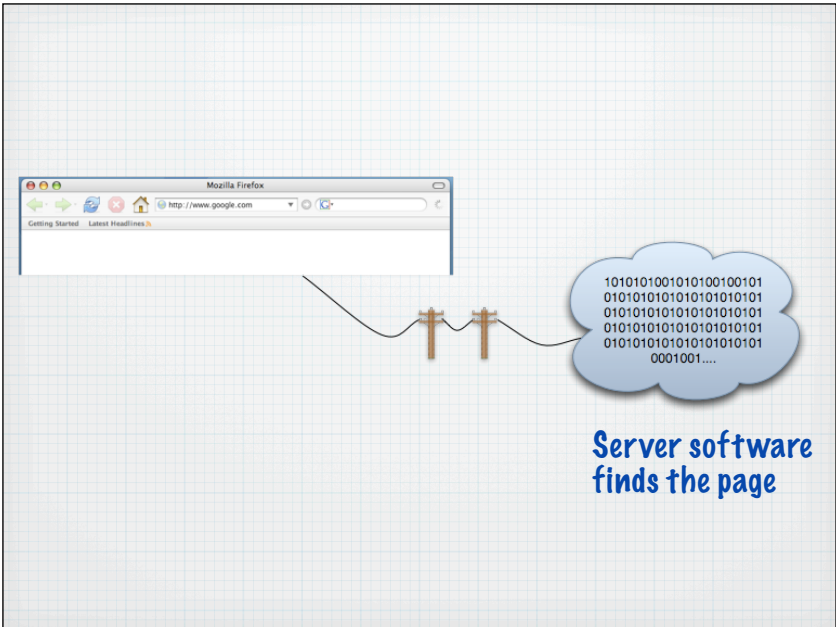


10101001010100100101  
01010101010101010101  
01010101010101010101  
01010101010101010101  
01010101010101010101  
0001001....

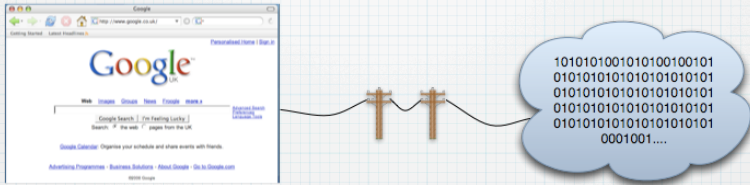
## Request is sent to server using HTTP



10101001010100100101  
01010101010101010101  
01010101010101010101  
01010101010101010101  
01010101010101010101  
0001001....



## Browser renders / displays the page



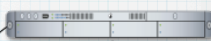
## Server side scripting languages

- Executes in the server
- Before the page is sent from server to browser
- Server side code is **not** visible in the client
- Server side code can access resources on the server side

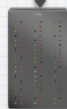
## Browser



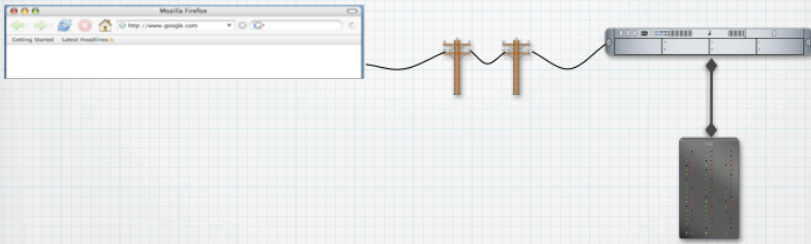
## Web server



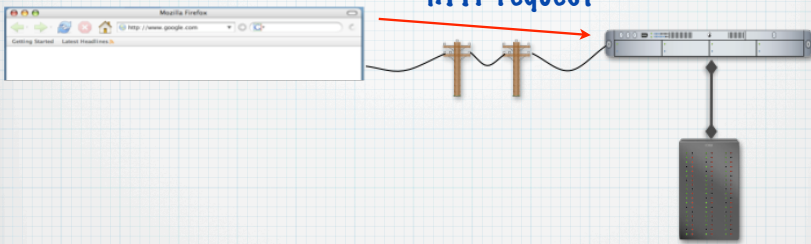
## Database server



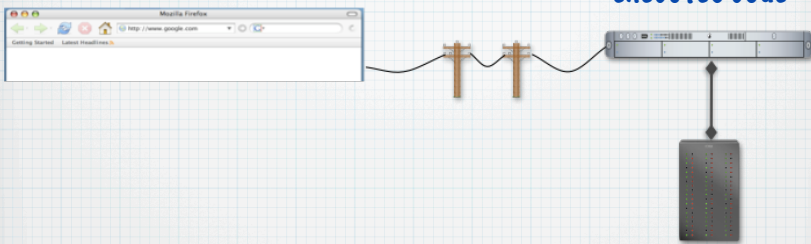
How many items in stock?



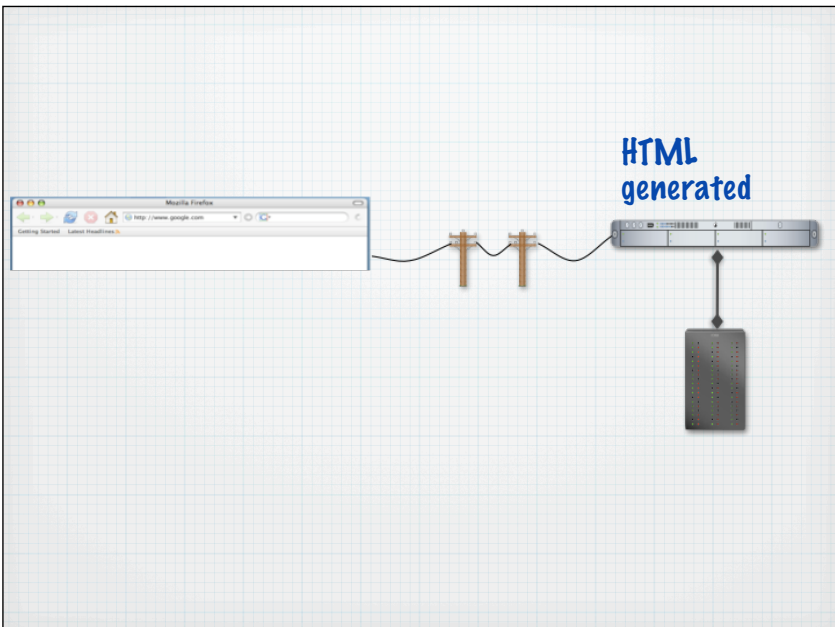
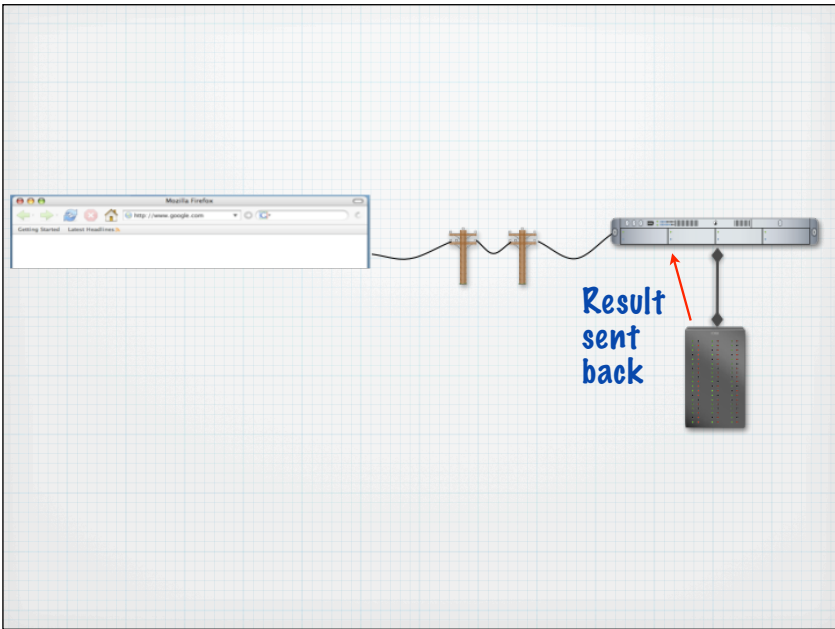
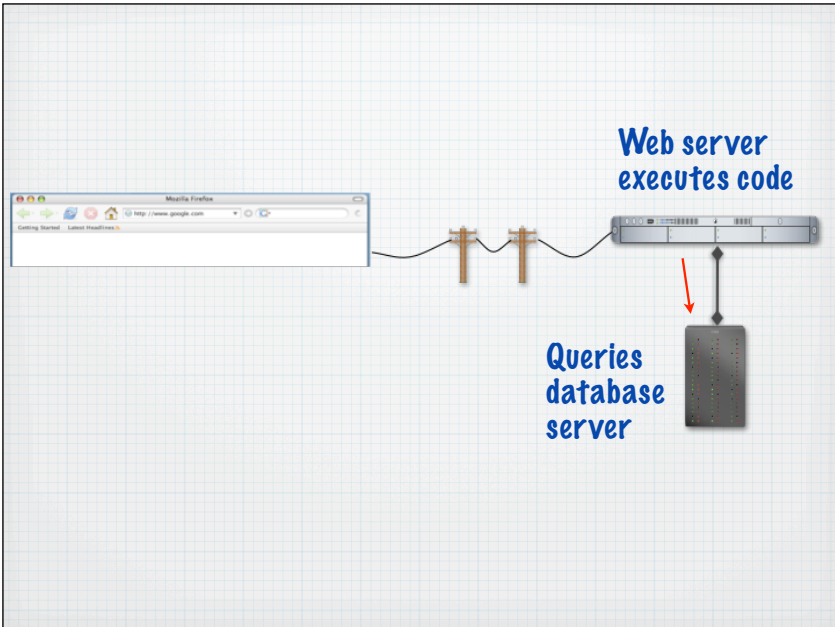
HTTP request

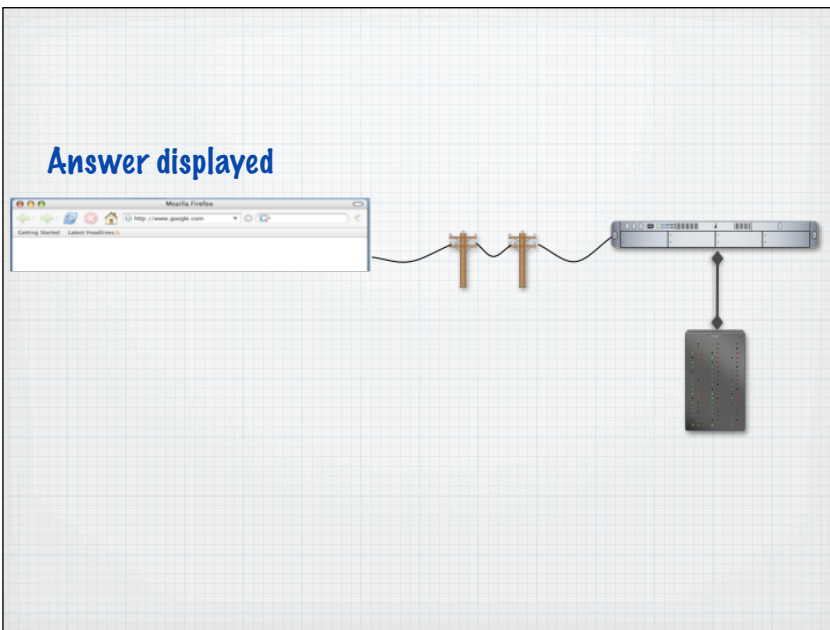
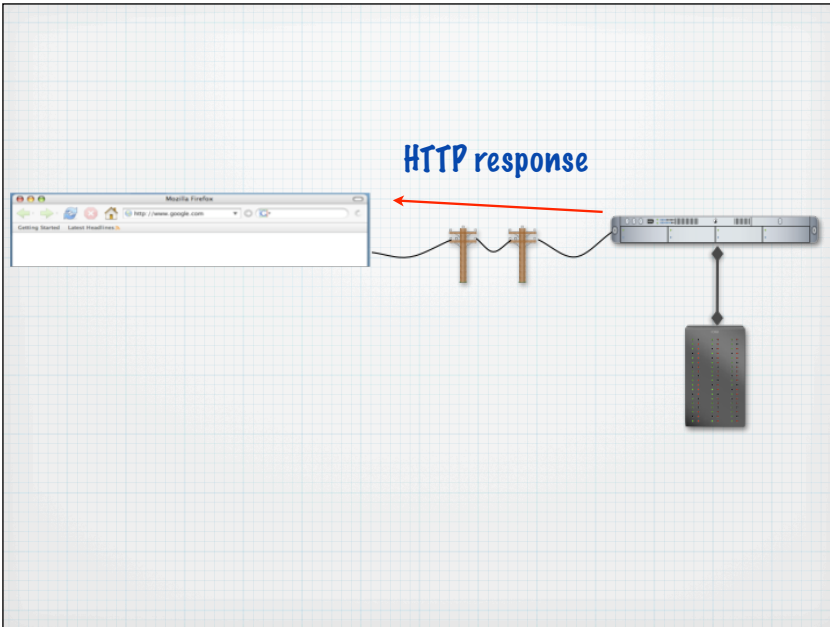


Web server executes code





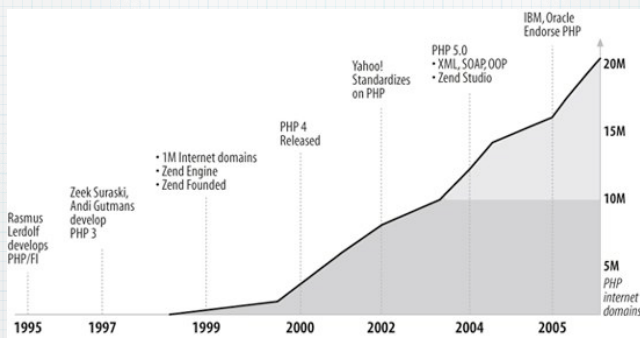




**So why PHP?**

PERL
<b>PHP</b>
PYTHON
ASP

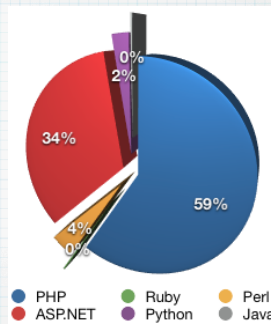
## PHP usage ... php 1 to php 5



- Source: [PHP programming 2nd Ed.](#)

## PHP compared to others ...

Difficult to compute - this from 6 million domains



PHP	3998425	59%
ASP.NET	2294166	34%
Perl	259931	4%
Python	159475	2%
Java	18065	0%
Ruby	16539	0%

- Source: <http://phpadvent.org/2010/usage-statistics-by-ilia-alshanetsky>

## PHP compared to others ...

But ...

PHP	3998425	59%
ASP.NET	2294166	34%
Perl	259931	4%
Python	159475	2%
Java	18065	0%
Ruby	16539	0%

ASP.NET reported by many passive domains (i.e. installed but not used)

PERL, Python and Java can be used to create stand alone programs so general usage statistics probably higher

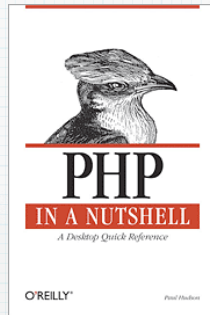
## Books - core syntax



Programming PHP, Second Edition

By Kevin Tatroe, Rasmus Lerdorf,  
Peter MacIntyre  
Second Edition April 2006

**\*\* Recommended**



PHP in a Nutshell

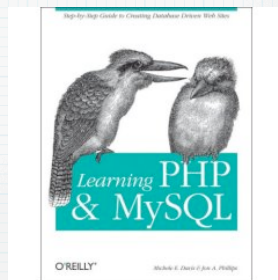
By Paul Hudson  
First Edition October 2005

## Books - learning / tutorial based



Learning PHP 5

By David Sklar  
First Edition June 2004



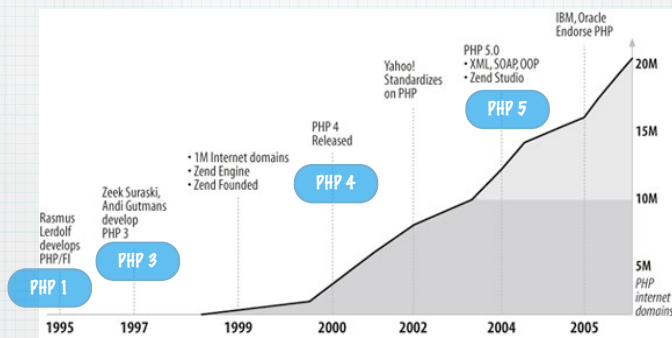
Learning PHP and MySQL

By Michele Davis, Jon Phillips  
First Edition June 2006

## Other texts..

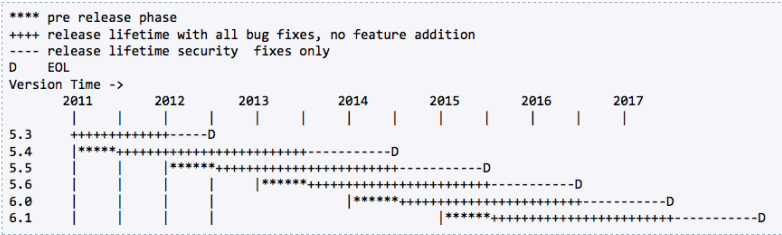
- There are other publishers / texts (trade books)
- Look for books that cover PHP 5
- Open source, server side languages can rapidly develop
- Features added or deprecated rapidly

## PHP development



- 5 versions in 10 years

## PHP development



- perhaps ...

<https://wiki.php.net/rfc/releaseprocess>

## Language basics

- Embedding PHP in Web pages
- Whitespace and Line breaks
- Statements and semicolons
- Case sensitivity
- Comments
- Literals
- Identifiers
- Keywords
- Data types

Much of this material is explained in **PHP programming 2nd Ed. Chap 1 & 2**

## Embedding PHP in web pages

```
<?php  
statement;  
statement;  
statement  
?>
```

Use `<?php` and `?>` to surround the php code

## Embedding PHP in web pages

```
<?php  
statement;statement; statement;  
    statement;  
statement;statement;  
?>
```

In general whitespace doesn't matter

Use indenting and separate lines to create readable code

## The legendary Hello World program

```
<!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.01  
Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">  
<html>  
<head>  
<title>This is the first PHP program</title>  
</head>  
<body>  
<p>  
<?php  
print "Hello World!";  
?>  
</p>  
</body>  
</html>
```

## The legendary Hello World program

```
<!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.01
  Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>This is the first PHP program</title>
</head>
<body>
<p>
<?php
print "Hello World!";
?>
</p>
</body>
</html>
```

## The legendary Hello World program

```
<!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.01
  Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>This is the first PHP program</title>
</head>
<body>
<p>
<?php
print "Hello World!";
?>
</p>
</body>
</html>
```

print a value to the  
output

Here the value is a  
sequence of chars  
indicated by start and  
end quotes

## Other ways to embed PHP

<? and ?>	S&MML style	Some older text books use this - deprecated
<% and %>	Microsoft ASP style	Some HTML editors use this for color syntax hints
<script language ="php"> and </script>	Echoes client side scripting embedding	Some strict HTML editors may respect this

The preferred method is <?php and ?>

## PHP can be put anywhere..

All the php blocks are processed  
before the page is sent

```
<html>
<?php ... ?>
<head>
<?php ... ?>
<title>... <?php ... ?> ...</title>
</head>
<body>
<p>
<?php ... ?>
</p>
</body>
</html>
```

## PHP can be put anywhere.. but works in sequence

```
<html>
<?php ... ?>
<head>
<?php ... ?>
<title>... <?php ... ?> ...</title>
</head>
<body>
<p>
<?php ... ?>
</p>
</body>
</html>
```

Starting at the top

Working down to the bottom

## Statements and semicolons

```
<?php
statement;
statement;
statement
?>
```

Use ; to separate  
statements

; optional here as end of  
the php block (probably  
best to put it in)

Make this a rule - Put at the end of every statement



## All of these would work the same way..

```
<?php statement; statement;statement ?>
```

```
<?php  
statement; statement;statement;  
?>
```

```
<?php  
statement;  
statement;  
statement;  
?>
```

This is the best way of  
laying the code out

## Case Sensitivity

Case insensitive	Case sensitive
built in constructs and keywords	names we make up

## Case insensitivity

```
<!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.01  
Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">  
<html>  
<head>  
<title>This is the second PHP program</title>  
</head>  
<body>  
<?php  
print "<h1>Welcome to my website</h1>";  
PRINT "<p>This is my web site, which is constructed";  
prINT " from some HTML and PHP</p>";  
?>  
</body>  
</html>
```

The same built in  
command

## Case insensitivity

```
<!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>This is the second PHP program</title>
</head>
<body>
<?php
print "<h1>Welcome to my website</h1>";
PRINT "<p>This is my web site, which is constructed";
prINT " from some HTML and PHP</p>";
?>
</body>
</html>
```

## Case sensitivity - names we define are case sensitive

```
$value
$VALUE
$valUE
```

← Three different names

PHP requires a \$ before names we define -  
more on this in a minute ...

## Comments

Many different ways to add comments

Comment	Source	Action
//	C++	Comments to EOL
#	Unix shell scripting	Comments to EOL
/* and */	C	Comments out a block

## Comments

```
<?php
php statement; // A comment here
php statement; # Another comment here

/* A series of lines
with comments ignored by the PHP processor
*/
php statement;
?>
```

## Comments

```
<?php
php statement; // A comment here
php statement; # Another comment here

/* A series of lines
with comments ignored by the PHP processor
*/
php statement;
?>
```

**Everything in red is ignored by the PHP interpreter**

## Language basics

- Embedding PHP in Web pages ✓
- Whitespace and Line breaks ✓
- Statements and semicolons ✓
- Case sensitivity ✓
- Comments ✓
- Literals
- Identifiers
- Keywords
- Data types

## Literals

A data value that appears directly in the program

2001	An integer
0xFE	Hexadecimal number
1.4142	Float
"Hello World"	String
'Hi'	String
true	Bool
null	built in 'no value' symbol

## Identifiers

Identifiers (or names) in PHP must -

Begin with an ASCII letter (uppercase or lowercase)

or begin with the underscore character `_`

or any character between ASCII 0x7F to 0xFF

followed by any of these characters and the digits 0-9

## Variables

Variables in PHP are identifiers prefixed by `$`

`$bill`  
`$value_count`  
`$anothervalue3`  
`$THIS_IS_NOT_A_GOOD_IDEA`  
`$_underscore` } **Valid**

**Invalid** { `$not valid`  
`$[`  
`$3wa`

## Variables

We use variables for items of data that will change as the program runs

Choose a sensible name and have as many as you like

```
    $total_income
$bill
    $salary      $month
$total
    $percentage_increase
```

## Variables

Your choice .. but be consistent

Camel case uses upper case for words in the name (apart from the first char)

```
$totalBill      $firstDayOfTheMonth
$percentageIncrease
```

<http://en.wikipedia.org/wiki/CamelCase>

## Variables

When we declare a variable, a space is reserved and labelled for that item (in memory)

```
$bill
```



```
$bill
```

## Variables

To give it a value, use the equals sign

¢bill = 42



## Variables

To give it a value, use the equals sign

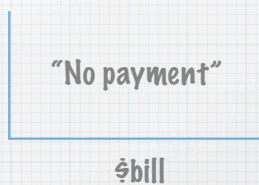
¢bill = 57.98



## Variables

To give it a value, use the equals sign

¢bill = "No payment"



## Variables

If a value is changed, the old value is overwritten

```
$bill = 42;  
$bill = 58;
```



## Variables

Sometimes we use the old value to recalculate the new value

```
$bill = 42;  
$bill = $bill*2;
```



## Variables

Some languages are very strict about what kinds of data are stored in variables - PHP doesn't care

```
$bill=42;
```

Stores an integer

```
$bill=42;  
$bill="Now its a string";
```

Overwrites with a string

```
print $bill;
```

Whoops - made a mistake  
but it still works

## Variables

Some languages are very strict about what kinds of data are stored in variables - PHP doesn't care

```
$bill=42;
```

Stores an integer

```
$bill=42;  
$bill="Now its a string";
```

Overwrites with a string

```
print $bill;
```

Whoops - made a mistake  
but it still works

## Case sensitivity

```
$value=56;  
$VALUE=78;  
$vaLUE=89;
```

Three different  
variables

PHP uses \$ before the identifier to indicate a  
variable

## Case sensitivity

```
...  
<body>  
<p>  
<?php  
$value=56;  
$VALUE=78;  
$vaLUE=89;  
  
print '$value has a value of ' ;  
print $value;  
print ', VALUE has a value of ' ;  
print $VALUE;  
print ', $vaLUE has a value of ' ;  
print $vaLUE;  
>  
</p>  
</body>  
</html>
```



## Case sensitivity

```
...
<body>
<p>
<?php
$value=56;
$VALUE=78;
$vaLUE=89;

print '$value has a value of ' ;
print $value;
print ', VALUE has a value of ' ;
print $VALUE;
print ', $vaLUE has a value of ' ;
print $vaLUE;
?>
</p>
</body>
</html>
```

## Constants

Referred to by their identifier and set using **define()**

```
define ('BESTLANGUAGE', "PHP");
print BESTLANGUAGE;
```

Traditionally constants have **UPPER CASE IDENTIFIERS**

## Keywords

Reserved by the language for core functionality

Also - can't use a built in function name as a variable

__CLASS__	Declare	extends	print()
__FILE__	Default	final	private
__FUNCTION__	die()	for	protected
__LINE__	do	foreach	public
__METHOD__	echo()	function	require()
Abstract	Else	global	require_once()
And	elseif	if	return()
array()	empty()	implements	static
As	enddeclare	include()	switch
Break	endfor	include_once()	throw
Case	endforeach	interface	try
catch	endif	isset()	unset()
cfunction	endswitch	list()	use
Class	endwhile	new	var
clone	eval()	old_function	while
Const	exception	Or	xor
Continue	exit()	php_user_filter	

## Data types

PHP provides 8 types

scalar (single-value)	compound
integers	arrays
floating-point	objects
string	
booleans	

Two are special - resource and NULL

## Integers

Whole numbers - range depends on the C compiler that PHP was made in (compiled in)

Typically +2,147,483,647 to -2,147,483,647

Octal 0755

Hexadecimal 0xFF

Larger integers get converted to floats automatically

## Floating-Point Numbers

Real numbers - again range is implementation specific

Typically 1.7E-308 to 1.7E+308 with 15 digits of accuracy

Examples 3.14, 0.017, -7.1, 0.314E1, 17.0E-3

## Strings

### Delimited by either single or double quotes

```
'here is a string'  
"here is another string"
```

## Strings - single quotes

### You can use single quotes to enclose double quotes

```
$outputstring='He then said "Goodbye" and left';
```

### Useful for easily printing HTML attributes

```
$outputstring='<a href="http://www.bbc.co.uk">BBC</a>';
```

## Strings - double quotes

### You can use double quotes to enclose single quotes

```
$outputstring="He then said 'Goodbye' and left";
```

### Variable are expanded within double quotes

```
$name="Barry";  
print "<p>We can use variable expansion when we print using double quotes - hello $name.</p>";  
print '<p>But it does not work with single quotes - hello $name</p>';
```

## Strings - double quotes

You can use double quotes to enclose single quotes

```
$outputstring="He then said 'Goodbye' and left";
```

Variables are expanded within double quotes

```
$name="Barry";  
print "<p>We can use variable expansion when we print using double quotes - hello $name.</p>";  
print "<p>But it does not work with single quotes - hello $name</p>";
```

## Strings - double quotes

Double quotes also support a variety of string escapes

\"	Double quotes
\n	Newline
\r	Carriage return
\t	Tab
\\	Backslash
\\$	Dollar sign
\{	Left brace
\}	Right brace
\[	Left bracket
\]	Right bracket
\0 through \777	ASCII character represented by octal value
\x0 through \xFF	ASCII character represented by hex value

## Strings - double quotes

Remember that the HTML source is manipulated by the PHP

```
print "He then said 'Goodbye' and left \n";  
print "leaving in a hurry";
```

Produces 1 line not 2 in the rendered HTML

So where is the \n ?

## Strings - double quotes

Remember that the HTML source is manipulated by the PHP

```
print "He then said 'Goodbye' and left \n";  
print "leaving in a hurry";
```

Produces 1 line not 2 in the rendered HTML

So where is the \n ?

## Strings - HTML

Its HTML that must be used to change the display

```
print "<p>He then said 'Goodbye' and left  
</p><p>driving off in a hurry.</p>";
```

## Strings - HTML

Its HTML that must be used to change the display

```
print "<p>He then said 'Goodbye' and left  
</p><p>driving off in a hurry.</p>";
```

## Boolean

PHP has special reserved words for **true** and **false**

```
$sunIsShining=true;  
$needACoat=false;
```

No quotes required - more on this later

## Operator precedence

Heavily  
borrowed  
from C /  
Perl

p - precedence

a - associativity

N - non-associative

R - Right to Left

L - Left to Left

P	A	Operator	Additional Information
19	N	new	Create new object
18	R	[	Array subscript
17	R	!	Logical NOT
	R	~	Bitwise NOT
	R	++	Increment
	R	--	Decrement
	R	(int) (float) (string) (array) (object)	Cast
	R	@	Inhibit errors
16	L	*	Multiplication
	L	/	Division
	L	%	Modulus
15	L	+	Addition
	L	-	Subtraction
	L	.	String concatenation
14	L	<<	Bitwise shift left
	L	>>	Bitwise shift right
13	N	<, <=	Less than, less than or equal
	N	>, >=	More than, more than or equal
12	N	==	Value equality
	N	!=, <>	Inequality
	N	===	Type and value equality
	N	!==	Type and value inequality
11	L	&	Bitwise AND
10	L	^	Bitwise XOR
9	L		Bitwise OR
8	L	&&	Logical AND
7	L		Logical Or
6	L	?:	Conditional operator
5	L	=	Assignment
	L	= += -= *= /= .= %= &=  = ^= <<= >>=	Assignment with operation
4	L	and	logical AND
3	L	xor	Logical XOR
2	L	or	Logical OR
1	L	,	List separator

## Operators

Standard arithmetic operators: +, -, \*, /, % ..

Concatenation operator: .

```
$outputstring="He then said ".$quote;
```

Any non-string value is converted to a string before the concatenation.

## Operators

```
$aBool=true;
$aInt=156;
$aFloat=12.56;
$anotherFloat=12.2E6;
$massiveFloat=12.2E-78;
print "The bool printed looks like this: ".$aBool."<br />";
print "The int printed looks like this: ".$aInt."<br />";
print "The (smaller) float printed looks like this: ".$aFloat."<br />";
print "The larger float printed looks like this: ".$anotherFloat."<br />";
print "The even larger float printed looks like this: ".$massiveFloat."<br />";
```

## Operators

```
$aBool=true;
$aInt=156;
$aFloat=12.56;
$anotherFloat=12.2E6;
$massiveFloat=12.2E-78;
print "The bool printed looks like this: ".$aBool."<br />";
print "The int printed looks like this: ".$aInt."<br />";
print "The (smaller) float printed looks like this: ".$aFloat."<br />";
print "The larger float printed looks like this: ".$anotherFloat."<br />";
print "The even larger float printed looks like this: ".$massiveFloat."<br />";
```