

# Introduction to PHP

CIS 1152 Adv Web Dev

Lecture 2

Steve Ruegsegger

# Overview

**Goal:** Starting concepts of the PHP scripting language

## Objectives:

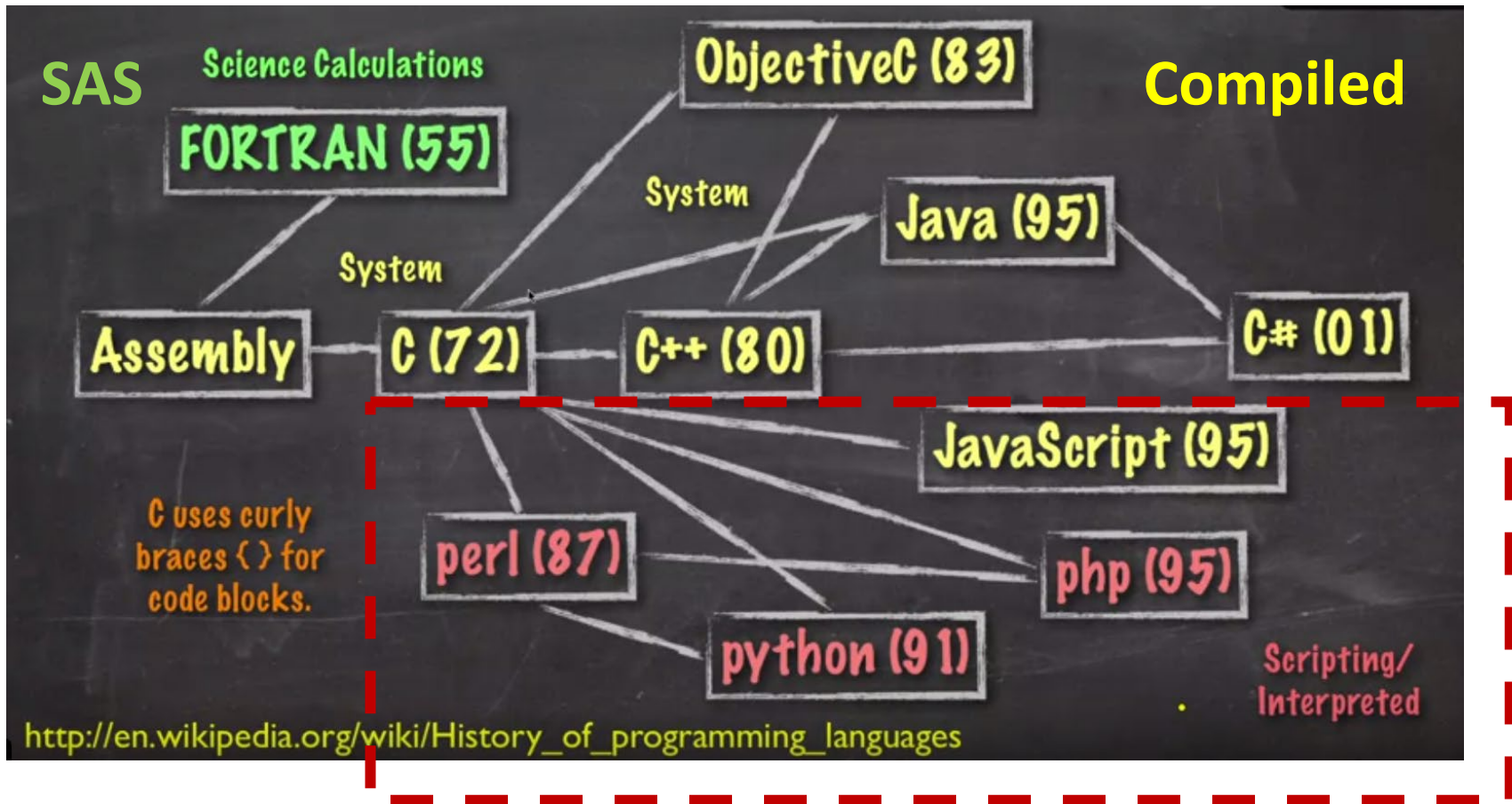
1. PHP init
2. Variables
3. Expressions and uses of variables: numeric, string
4. PHP Comments

# Init

PHP blocks

# You are here...

*Compiled vs Interpreted (script)*



# Hello world

- The **first program** anyone writes in a new language:

Required tags {

```
<?php  
    print "Hello world!";  
?>
```

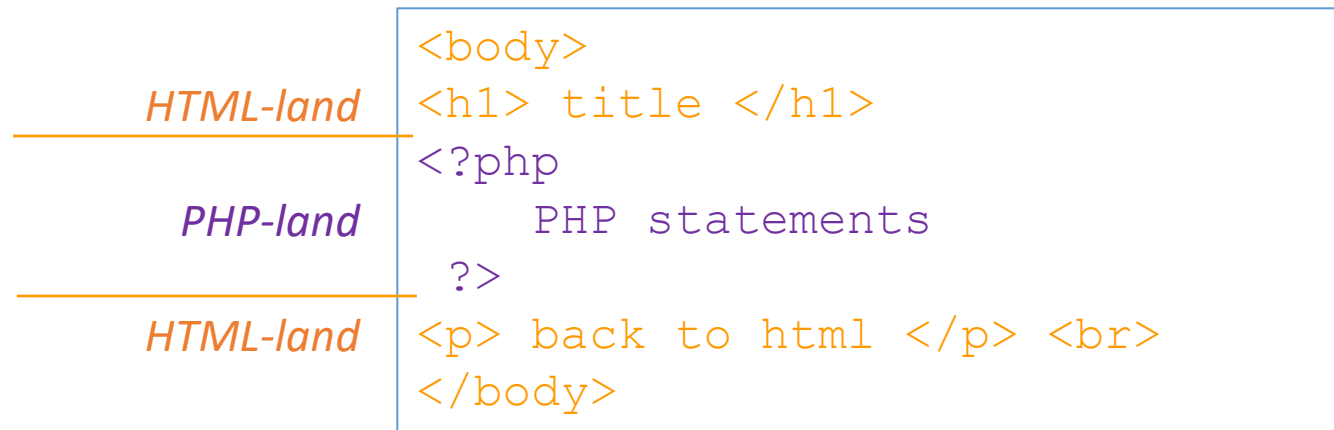
- PHP code is inserted in *blocks*!
- The PHP blocks can be *anywhere* within the normal HTML document
- PHP blocks are evaluated before the HTML is sent!
- When you see a PHP block, just remember that the PHP-engine will execute and replace it **before** sending to client.

# PHP Basics

- a) PHP code is put into blocks which are intermingled with the HTML tags
- b) PHP file must be saved with an extension of `.php` in order to be processed by the scripting engine.
- c) PHP code is server-side only. The PHP-engine replaces the PHP blocks with the results of the PHP logic
- d) PHP code is never sent to a *client's* Web browser. The user never sees it.
- e) PHP statements must end with a semicolon (;)
- f) PHP commands are case insensitive:  
print same as PRINT

# Creating PHP Code Blocks

- `<? ?>` is an generic HTML "**server-side**" block
- Outside of a block, is regular old HTML (the default)
- Code declaration blocks are separate sections in a Web page document that are *interpreted* by the scripting engine (server) before going to the client
- `<?php ?>` is the block which tells the **PHP** engine to run (execute). All other code is HTML and ignored by PHP engine.



# Displaying Script Results

- Any output the PHP block is added to the rest of the HTML around it and sent to the client browser (firefox)
- *i.e. PHP writes out HTML code*
- The server interprets the PHP commands, where the main purpose is to write out HTML code to the client
- ★ *• This is code writing code!*
- The `echo` and `print` commands of PHP write out HTML

```
<h1> This is a title </h1>
<?php
    echo "<p> a new paragraph </p>";
?>
<p> back to html </p>
```



# Multiple PHP Blocks

Multiple PHP blocks can be embedded *all over* the html document.

```
HTML-land <body>
           <h1>Multiple Script Sections</h1>
           <h2>First Script Section</h2>
           <?php
PHP-land   echo "<p>Output from the first script section.</p>";
           ?>
HTML-land <h2>Second Script Section</h2>
           <?php echo "<p>Output from the second script
PHP-land   section.</p>";?>
HTML-land </body>
```

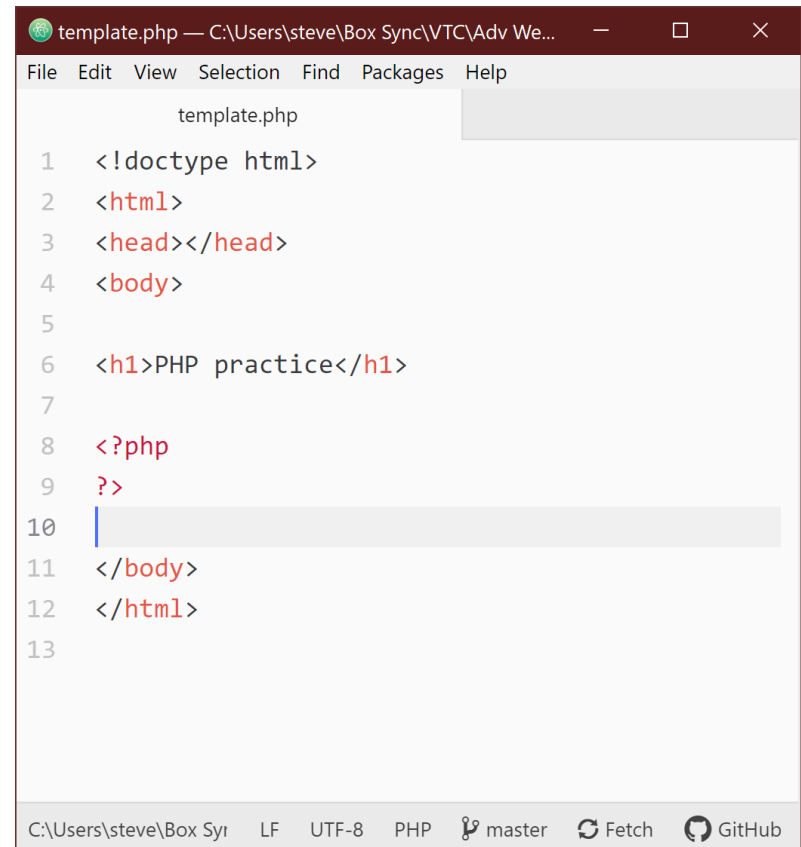
# HTML Template

- Since PHP is embedded within HTML, I recommend that we all create a simple HTML template for practicing PHP.

*HTML-land*

*PHP-land*

*HTML-land*



```
template.php — C:\Users\steve\Box Sync\VTC\Adv We... — □ ×
File Edit View Selection Find Packages Help
template.php
1 <!doctype html>
2 <html>
3 <head></head>
4 <body>
5
6 <h1>PHP practice</h1>
7
8 <?php
9 ?>
10
11 </body>
12 </html>
13
C:\Users\steve\Box Syr LF UTF-8 PHP master Fetch GitHub
```

# Whitespace

- whitespace = spaces, tabs newlines
- They are all ignored
- Use them for nice formatting

```
<?php print "Hello "; print "world!"; ?>
<p>    this
is a
Test in line 1 </p>
<p> line2 </p>
```

# PHP output – echo or print

- There are actually quite a few output *formats*:

## Output

- **echo** is a language construct - can be treated like a function with one parameter. Without parentheses, it accepts multiple parameters.
- **print** is a function - only one parameter, but parentheses are optional so it can look like a language construct.

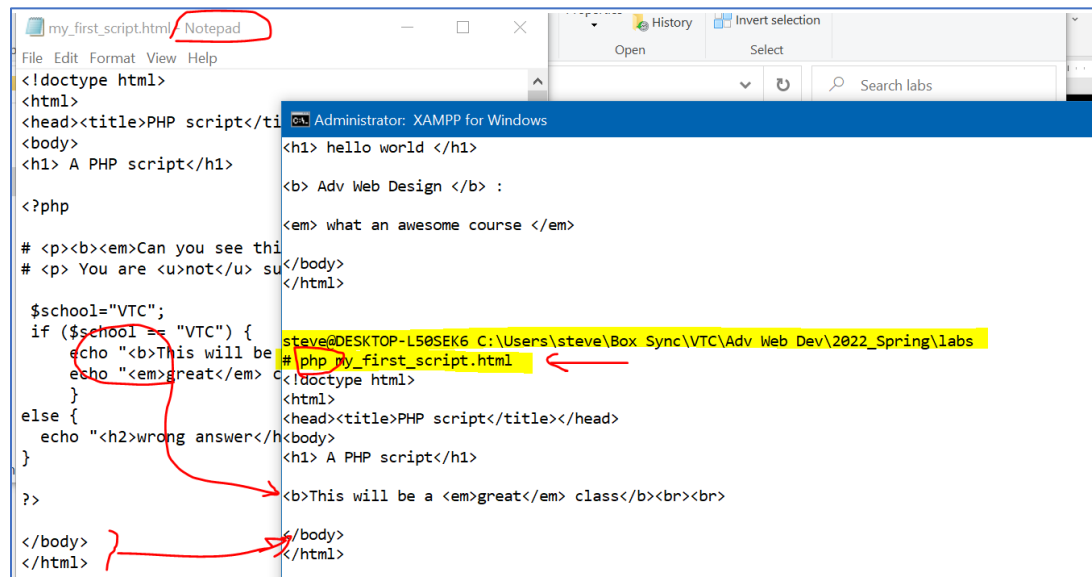
```
<?php
    $x = "15" + 27;
    echo $x;
    echo("\n");
    echo $x, "\n";
    print $x;
    print "\n";
    print($x);
    print("\n");
?>
```

**FYI:** echo is from bash, print is from perl/python

# Running, writing, testing PHP

- There is not really an IDE. I would argue it doesn't make sense anyway.
- You will **always** write your PHP in a text file editor.
- To run:
  1. The normal way is to use the apache web server.
  2. Or, you may run that script from a CLI; i.e. OS prompt.

*class demo needed*



The screenshot shows a Notepad window titled 'my\_first\_script.html' containing PHP code. The code includes an HTML header, a body with 'A PHP script', and a PHP block with an if-statement that checks if the variable '\$school' is 'VTC'. If true, it outputs 'This will be great'; otherwise, it outputs 'wrong answer'. The terminal window shows the output of the script: 'hello world', 'Adv Web Design', and 'what an awesome course'. The terminal prompt is 'steve@DESKTOP-L50SEK6 C:\Users\steve\Box\_Sync\VTC\Adv Web Dev\2022\_Spring\labs'. A red arrow points from the terminal output to the corresponding PHP code in the Notepad window.

```
my_first_script.html Notepad
File Edit Format View Help
<!doctype html>
<html>
<head><title>PHP script</title>
<body>
<h1> A PHP script</h1>

<?php
# <p><b><em>Can you see this?</em></b>
# <p> You are <u>not</u> sure?
}

$school="VTC";
if ($school == "VTC") {
    echo "<b>This will be great</b>";
}
else {
    echo "<h2>wrong answer</h2>";
}
?>

</body>
</html>

Administrator: XAMPP for Windows
<h1> hello world </h1>
<b> Adv Web Design </b> :
<em> what an awesome course </em>

steve@DESKTOP-L50SEK6 C:\Users\steve\Box_Sync\VTC\Adv Web Dev\2022_Spring\labs
# php my_first_script.html
<!doctype html>
<html>
<head><title>PHP script</title></head>
<body>
<h1> A PHP script</h1>

<b>This will be a <em>great</em> class</b><br><br>
</body>
</html>
```

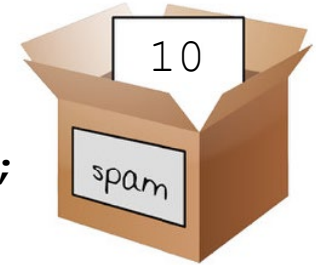
# Variables

*The Missing Link* -- ch27, pg 158 -- "Data Storage"

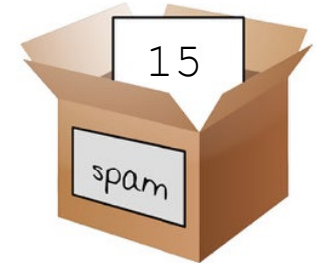
# What is a variable?

- Stores a value in memory
- That location might be address:  
0x62A86D153
- Fortunately, that memory location is given a name – very helpful for us programmers
- That memory location stays the same...
- ...but the content within that memory location changes throughout the program.
- The *expression* (Right of =) is evaluated first, then the results put into variable location (Left)

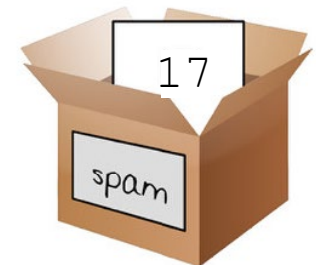
```
$spam=10;
```



```
$spam=10+5;
```



```
$spam=$spam+2;
```



# Variable names (a.k.a. identifier)

- The values stored in computer memory are called **variables**
- The values, or data, contained in variables are classified into *categories* known as **data types**
- The name you assign to a variable is called an **identifier**
- Rules for PHP variable names (*identifiers*):
  - must begin with a dollar sign (\$),
  - may not include a number or underscore as the *first* character,
  - cannot include spaces, and
  - is *case sensitive*
- *e.g.*: \$x, \$var, \$VAR, \$last\_name, \$lastName (all different)
- Most of the time, forgetting the "\$" start will end in a parse error. But there are some instances when PHP sees a non-\$-start id as a constant or keyword. Be careful.



# Variable name limits

- Review: to display variables in string context
  - separated by commas
  - or put right in double quotes

```
echo "<p>The legal voting age is ", $VotingAge, ".</p>";  
echo "<p>The legal voting age is $VotingAge. </p>";
```

- If you need to *smash* a character right next to a variable name, you can use {} around the *identifer*:

```
echo "that is ${name}'s book";  
echo "new filename is ${fname}_lab3.doc";
```

# demo

## *l02\_variables*

```
7  <?php
8
9  $x = 10;
10 $y = 5;
11 $z = $x + $y;
12 print("<p> x = $x <br> y = $y <br> z = $z");
13
14 $x = $x + $y;
15 print("<p> new x = $x");
16 print("<br> z = $z (unchanged)");
17
18 ?>
```

# Working with Data Types

- A **data type** is the specific category of information that a variable contains
- Data types that can be assigned only a single value are called **primitive types**

*The Missing Link: An Introduction to Web Development and Programming*

Chapter 27

- [Booleans](#) Can have a value of 0 or 1
- [Integers](#) Whole numbers (1, 3, 20, etc.)
- [Floating point numbers](#) Decimal values (1.33, 34.2325)
- [Strings](#) Contain any number of characters
- [Arrays](#) Structured lists of information
- [Objects](#) Collections of related variables and functions
- [Resources](#) Special variables that hold reference points to things like files
- [NULL](#) An empty (unused or unset) variable
- [Callbacks](#) A mechanism to reference a function declared elsewhere

Data Type	Description
Integer numbers	The set of all positive and negative numbers and zero, with no decimal places
Floating-point numbers	Positive or negative numbers with decimal places or numbers written using exponential notation
Boolean	A logical value of “true” or “false”
String	Text such as “Hello World”
NULL	An empty value, also referred to as a NULL value

**Table 1-1** Primitive PHP data types

## Tests:

```
is_int(), is_float(), is_string(), is_bool()  
is_null(), is_scalar(), is_array(), is_numeric()
```

# Data type assumption or typecasting

- PHP has *very aggressive* type **conversion**
  - PHP uses the **operator** to *figure out* your meaning
  - i.e. `$x = "15" + 27`
  - Because `+` is a numeric operator, the `"15"` string is converted to 15 numeric.
  - (Python, bash, java, C would *never* do this...)
- Rather than have PHP "convert" based on operator; you *can* Type Cast (-- it's better practice anyway)
  - `intval()`
  - `floatval()`
  - `strval()`
- Or you can use C-style typecasting
  - `$amount = (float)$input ;`

# String variables

- Put in quotes:

```
$name = "Steve";
```

- Strings can span multiple lines! (can contain \n's)

```
$header = "<html>
<head>
<title> foobar </title>
</head>
";
```



- Concatenate operator is the dot (.)

```
$address2 = $city . ', ' . $state;
```



*3 separate strings: 2 are variables*

# Quoting Strings

- Two kinds of quotes.
- Quotes are like *functions*
  1. Double quotes evaluate the string inside
  2. Single quotes are treated as literals.
- That is,
  1. *Double* quotes *look inside* the quotes and **change** any identifiers (i.e variable names with \$'s) to the value
  2. *Single* quotes don't. A \$ is just a dollar sign. *Literally*.

```
echo "his name is $name"; → his name is John  
echo 'his name is $name'; → his name is $name
```

# Escapes

- If you don't want the double quotes to evaluate something – then you *escape* it
- The *escape* character is a backslash \
- The character to the right of the \ is interpreted differently.

- Escapes quotes

```
# I really want double quotes to be printed  
echo "Define the word \"$word\" please";
```

```
# this also works:
```

```
echo "Define the word '$word'";
```

- Escape dollar signs

```
echo "your total bill is \$ $total. <br>\n";
```

## A Microsoft GOTCHA!

- So-called “Smart Quotes” **don't** work in PHP.
- You need DUMB QUOTES like "
- You can either change the quotes in the source, or change them in the ascii text editor .php file.
  - But you must change them!
  - For this PPT, I "turned-off" smart quotes



# Using variables

String functions and numeric expressions

# Some string functions

- `strlen()` – length
- `strtolower()` – all lower case
- `strtoupper()` – all upper case
- `ucwords()` – first letter of each word upper case
- `ucfirst()` – capitalize first word of string

```
$name=ucwords( $first." ".$last );  
echo "today is ", ucfirst($day);  
$ans = strtolower($ans);
```

# Constants

- A **constant** contains information that does not change and cannot change
- It's *immutable*
- Constant names do not begin with a dollar sign (\$)
- Constant names (usually?) use all UPPERCASE letters
- Use the `define()` function to create a constant  

```
define("CONSTANT_NAME", value);
```
- The value you pass to the `define()` function can be a text string, number, or Boolean value

# Numeric Data Types

- Numeric operators: + - \* /
- PHP does not require you to define INT or FLOAT
  - It will check and use the appropriate operations

```
$total = $quantity * $price;  
$total = $total + ($total * $taxrate);  
echo "total price is \$ $total <br>\n";
```

# Operators

Operator	Name	Example
+	Addition	$\$a + \$b$
-	Subtraction	$\$a - \$b$
*	Multiplication	$\$a * \$b$
/	Division	$\$a / \$b$
%	Modulus	$\$a \% \$b$

## Cool Combined Operators (contractions)

Operator	Use	Equivalent To
+=	$\$a += \$b$	$\$a = \$a + \$b$
-=	$\$a -= \$b$	$\$a = \$a - \$b$
*=	$\$a *= \$b$	$\$a = \$a * \$b$
/=	$\$a /= \$b$	$\$a = \$a / \$b$
%=	$\$a \% = \$b$	$\$a = \$a \% \$b$
.=	$\$a .= \$b$	$\$a = \$a . \$b$

See *The Missing Link* -- ch 28, pg 167

# Auto-increment

- Super helpful little function.
  - `$var++` means add 1 to the current value
  - *i.e.* `$ var = $var + 1;`
- Auto-increment can happen *within* other commands, like echo and loops.
- Timing:
  - `$var++` increments AFTER being used
  - `++$var` increments BEFORE being used


```
$student = 100;  
echo "The next student is: ", $student++ , "<br>\n";  
echo "The next student is: ", $student++ , "<br>\n";  
echo "The final student is: $student "<br>\n";
```

```
The next student is: 100  
The next student is: 101  
The final student is: 102
```

# Some number functions

- `round()` – will round the specified digit. The default is the ones place
- `number_format()` – will print the number as a string in a pleasant format, with commas.

```
$a = 3.14159;  
$b = 238391873494.98484984;  
echo "<br>";  
echo "a = ", round($a), "<br>";  
echo "a2 = ", round($a,2), "<br>";  
echo "b = ", number_format($b), "<br>";  
echo "b2 = ", number_format($b,2), "<br>";
```



```
a = 3  
a2 = 3.14  
b = 238,391,873,495  
b2 = 238,391,873,494.99
```

demo: `l02_expressions`

# PHP Comments

2 types



# Comments

- Comments are nonprinting lines placed in code that do not get executed, but provide *very* helpful information, such as:
  - The name of the script
  - Your name and the date you created the program
  - Notes to yourself
  - Instructions to future programmers who might need to modify your work
- *Two types*
  1. Block comments -- spans multiple lines
  2. Single line -- only the “rest” of that line

# Comments

## 1. **Line comments** hide a *single* line of code

```
statement; // comment
```

```
statement; # comment
```

## 2. **Block comments** hide *multiple* lines of code

```
/*    lots  
      of  
      lines in this comment  
*/
```

*Why 2 types?*

# Comments

- Use line comments:

- for comments after statements

```
$x = split($address,1); # this is the house number
```

- a short description of the line of code

```
// this is after the Submit button  
If ($ENV{'Submit'} eq true) {
```

- Use block comments:

- Long header comments
- Commenting out large sections of code, which could even have line comments nested

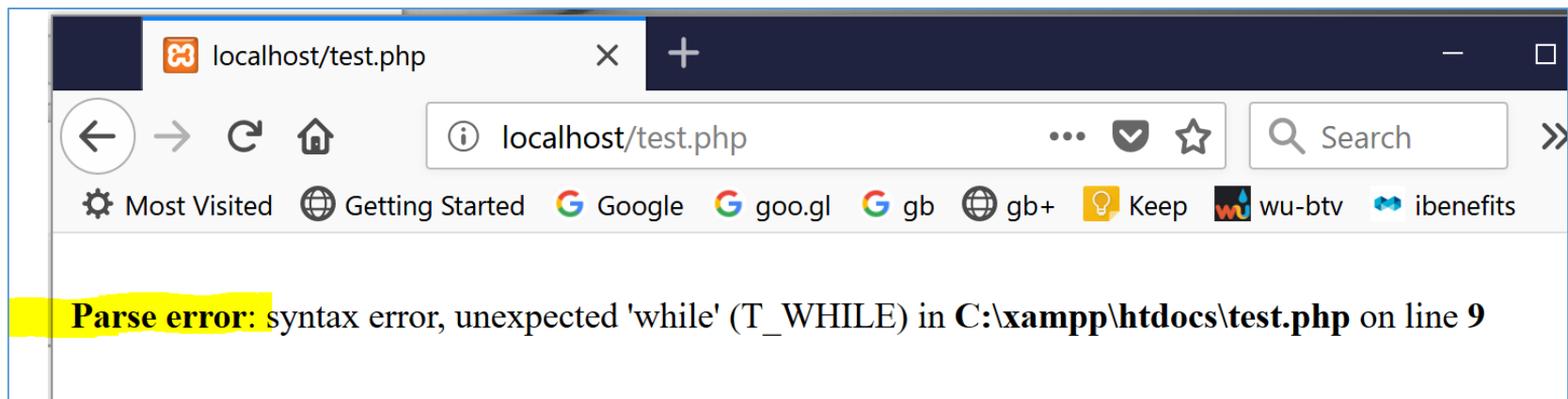

- **Note: You cannot *nest* block comments!**

```
/* this will not work  
  
# don't do this  
$name = 'bob'; /* smith */  
  
*/
```

# PHP errors

- The Missing Link-- ch 27, p153
- Let's sabotage a program.
- *Ack, missing the semicolon*

```
$k=1  
print "k = $k <br>\n";
```



# Lab 1 this week

- Install XAMPP (the server)
- Install and setup your dev *environment*
  - Atom, Microsoft Visual Studio Code, Brackets
- Write some simple PHP scripts and HTML pages
- Go for it. Enjoy!
- Take snapshots as you go along.
  - Not of the entire screen! Just the section of interest.
  - Store the pictures "*somewhere*"
  - Integrate the pictures with text in your report "**later**"
  - Format the report (text/pictures) as *the last thing* right before turning it in.
  - i.e. **Content first -- Format last**