PHP functions & files

CIS 1152 Adv Web Dev

Steve Ruegsegger

Modified with Permission by Peter Chapin

Overview

Goal: ability to write and call functions, and read/write data from/to files

Objectives:

1. Writing & calling your PHP functions
2. Variable scope
3. Call-by-value vs Call-by-reference
4. More function info
5. Using flat files (read & write)

1. PHP functions

Definition

- What is a function?
- When do we use them?

PHP function syntax

1. *Define* your function (declaration):

In PHP-land

```
<?php

function my_function() {
    <code here>
    }

?>
```

- Naming convention:
    - letters, digits, underscore.
    - cannot start with number
    - function names *are* case-sensitive

PHP function syntax

2. Calling the function:

- The name, inside a PHP block (PHP-land), is a *new* command
- Input parameters (arguments) are put in the trailing ()'s.
- Some functions return values and can be on the left side of the an equals sign
- Functions can also be smashed in a string concat (kinda cool).

Some examples:

```
    my_function();
    echo "you have ” . my_function() . " shopping items.";
    $y = my_function(4);
```

Function I/O: input/output

- Sometimes functions are just often-used shortcuts.
- But, usually, they *process* an input variable and *return* the output.
  - Input variables are called: arguments
  - Output variables are "returned" back to the original location.
- Format:

```
$output = myfunction( $input1, $input2 );
```

  - Output is to the left of the =
  - Function name is to the right
  - Inputs are in ( )

Returning a value

- We often/usually desire a function to return a value.
- The return value can be a string, Boolean, mathematical result. It can often be an error code.
- Use the `return` command.
- The `return` command also stops the function right then. *Nothing* after the return is executed.
- A `return` can be inside a conditional statement.

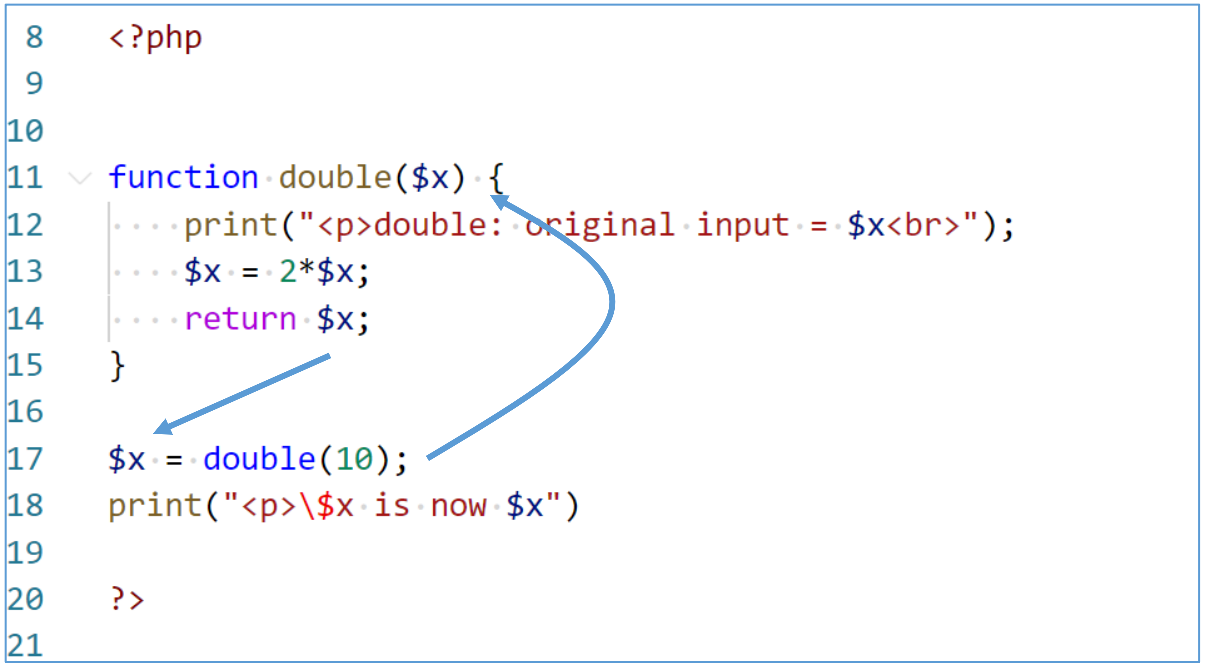
"only one way in; but many ways out"

```
function larger ($x, $y)
{
  if (!isset($x)||!isset($y))
    return false;
  else if ($x>=$y)
    return $x;
  else
    return $y;
}
```

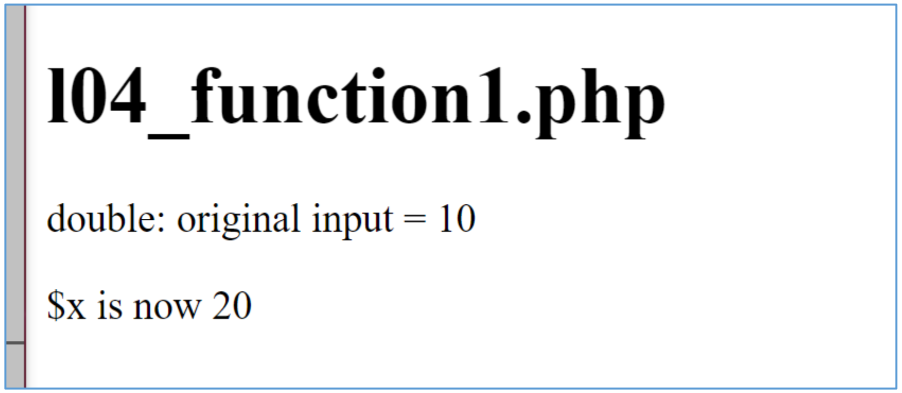
Simple example

l05_function1.php

- Line 17 sends a 10 to the function
- $x in line 11 is the function "argument"
- Line 13 doubles it
- Line 14 returns it back to line 17



```
 8  <?php
 9
10
11  function double($x) {
12      print("<p>double: original input = $x<br>");
13      $x = 2*$x;
14      return $x;
15  }
16
17  $x = double(10);
18  print("<p>\$x is now $x")
19
20  ?>
21
```

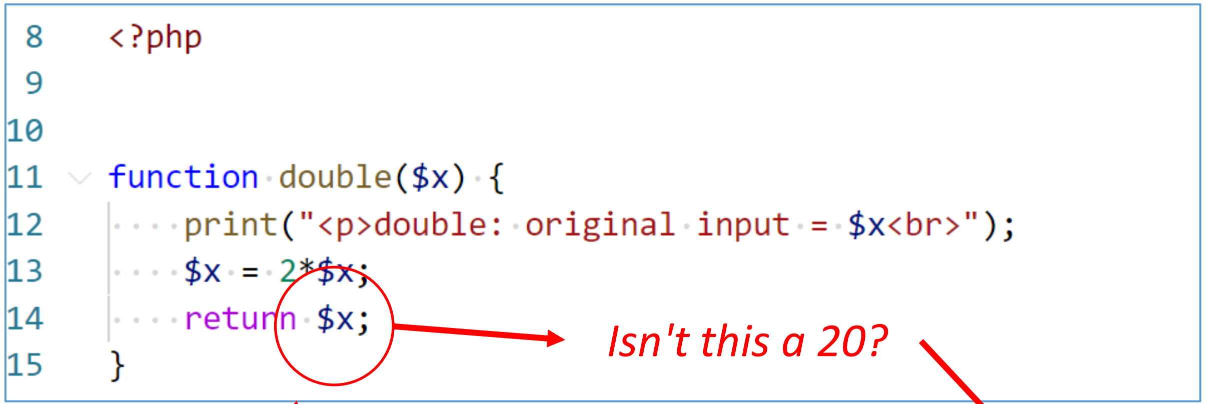


```
l04_function1.php

double: original input = 10

$x is now 20
```

Note: the $x in lines 11,13,14 are <u>not</u> the same $x in line 17!

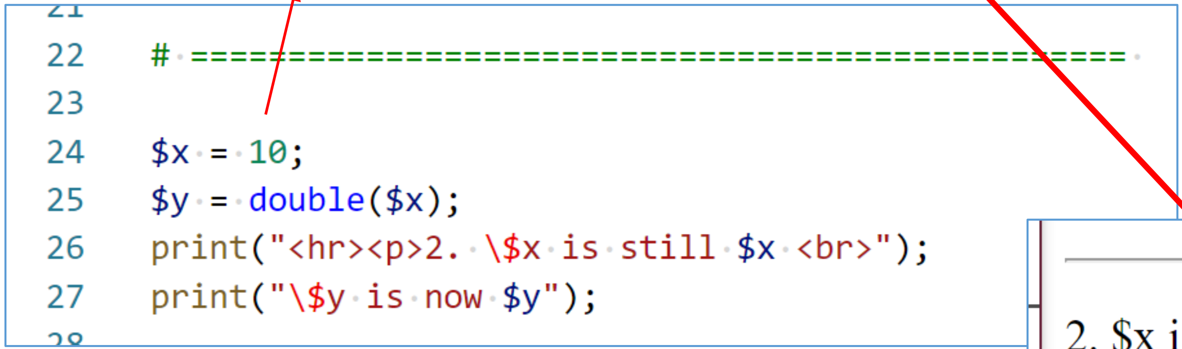
Variable names

- Don't be confused that all $x's are the same!
- They are not.

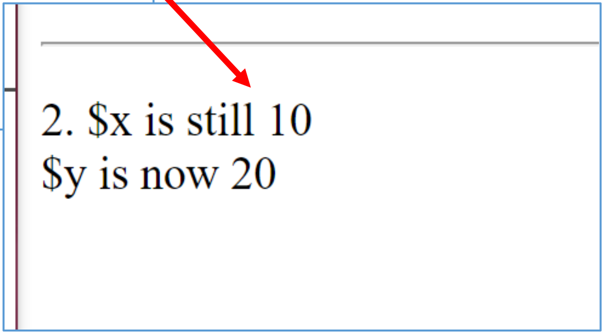


```php
<?php


function double($x) {
    print("<p>double: original input = $x<br>");
    $x = 2*$x;
    return $x;
}
```

Isn't this a 20?



```php
# ==============================================

$x = 10;
$y = double($x);
print("<hr><p>2. \$x is still $x <br>");
print("\$y is now $y");
```



```
2. $x is still 10
$y is now 20
```

We need to talk about 2 topics!

1. Variable scope
2. Call-by-value

2. Variable scope

Variable namespace

Variable scope

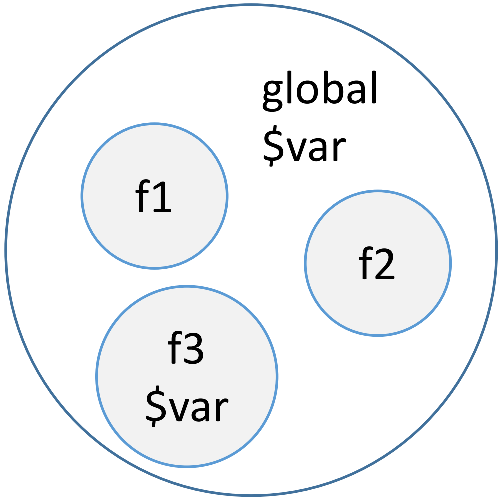
- What is variable scope?
- **scope** = where the variable is "*visible*" or "*usable*"
- General concept:

★ Variables *inside* a function should not affect the variables *outside* the function!

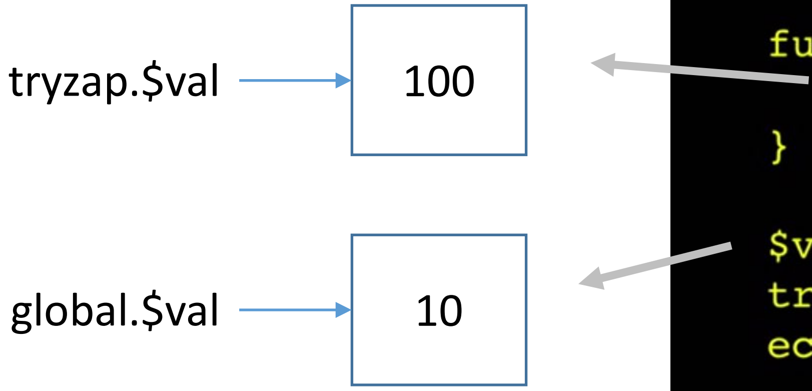
- List of specific rules (pg 148 in 5th ed Thomson)
  - Variables *declared* inside a function stay in the function (**local**)
  - Variables *declared* outside a function are **global**
  - Global variables do not automatically get into a function ★
  - Keyword `global` is required inside a function to access global variables from the function.

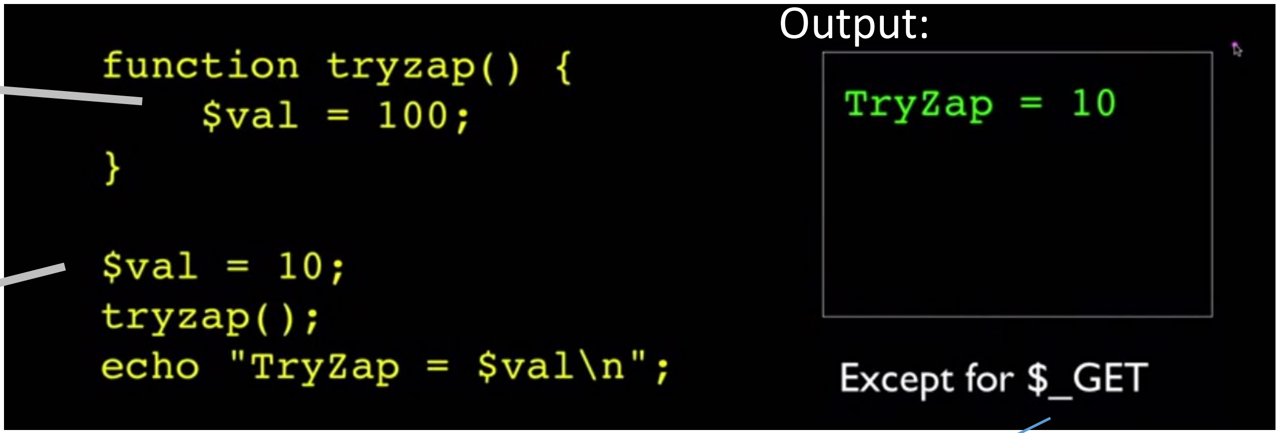
Namespace

- Where a function lives is also called the *namespace*.
  - Each function has it's own namespace.
  - Global is a namespace.
  - Think of a var *pythonicly*: namespace.$var



$val is "declared" in function tryzap().
See first bullet on prev page.





```
function tryzap() {
    $val = 100;
}

$val = 10;
tryzap();
echo "TryZap = $val\n";
```

Output:

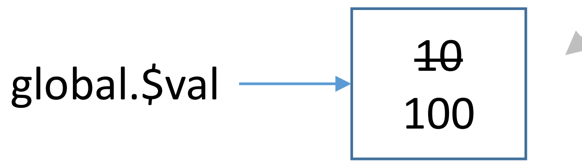
```
TryZap = 10
```

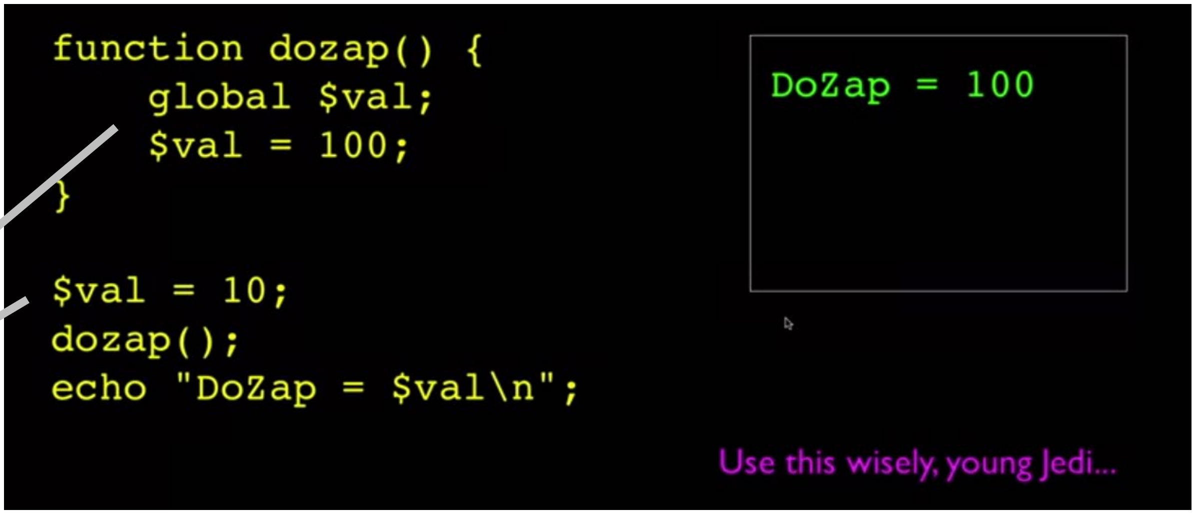
Except for $_GET

Note: $_GET is a "superglobal" array (more later)

Accessing global name space

- You *could* use the "global" command.
- However, in general, we don't like to use global variables. It's considered '*bad form*.' Use *sparingly*...



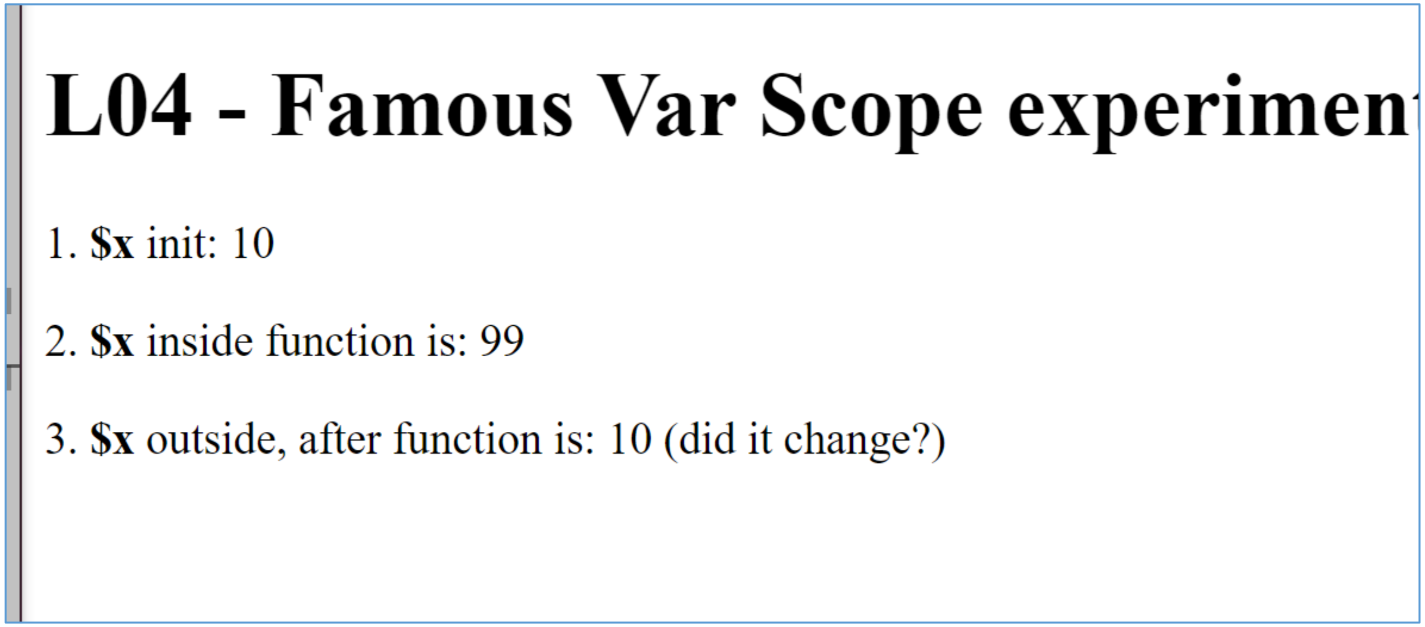


```
function dozap() {
    global $val;
    $val = 100;
}

$val = 10;
dozap();
echo "DoZap = $val\n";
```

```
DoZap = 100
```

examples

- You can experiment with l05_var_scope.php
- Also l05_global_var.php



L04 - Famous Var Scope experimen

1. $x init: 10

2. $x inside function is: 99

3. $x outside, after function is: 10 (did it change?)

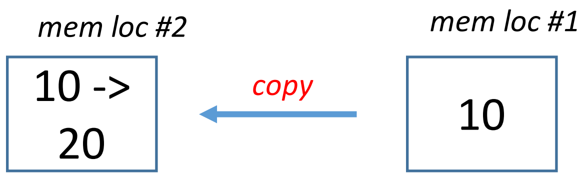
3. Two ways to send arguments

to a function()...

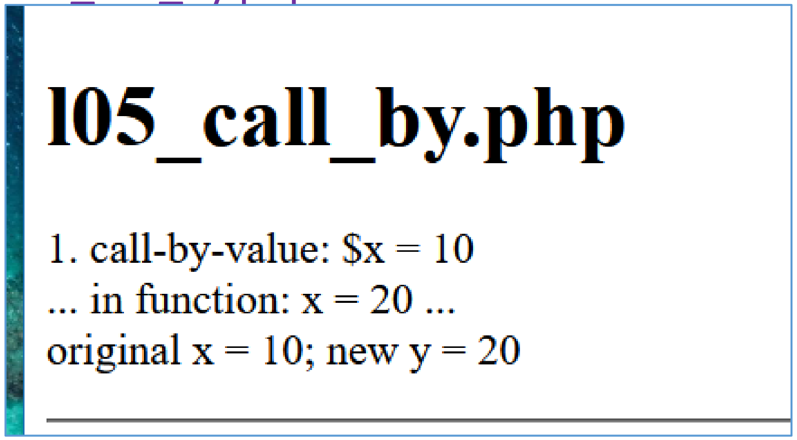
1. Call by value -- *default*
2. Call by reference -- *special*

1. Call by Value

- By default, PHP functions are "*Call by Value*"
- The input variables to a function are copies of the original. ☆
- That is, the *originals* are safe and sound in an *undisclosed location*.
- That is, functions do not change the original values of input parameters.
- "Return" the value to a new variable.

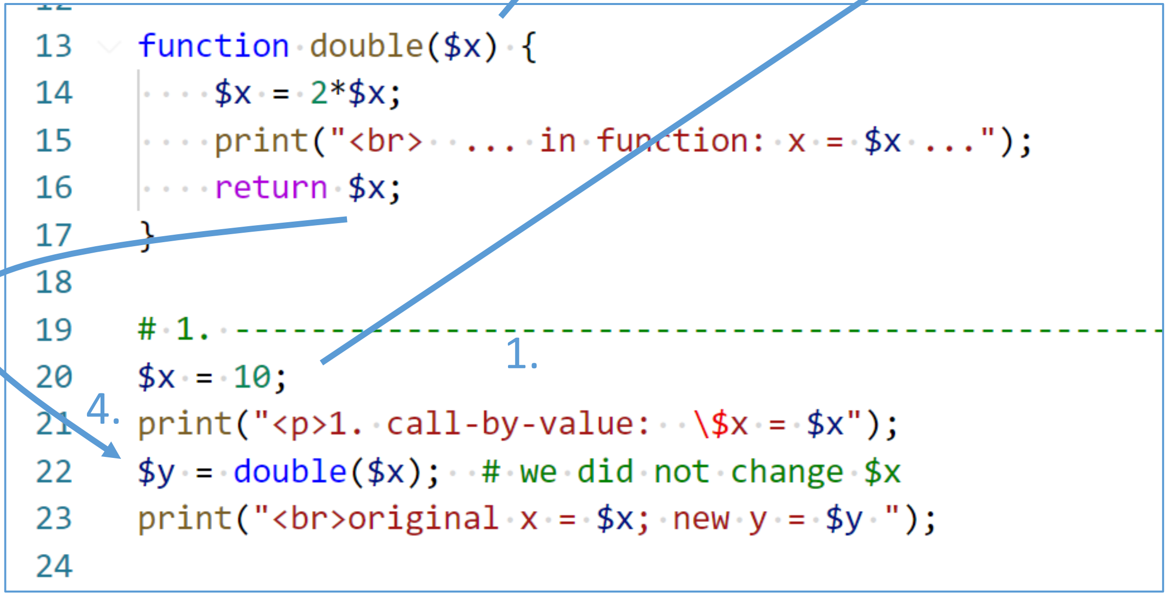


l05_call_by.php



l05_call_by.php

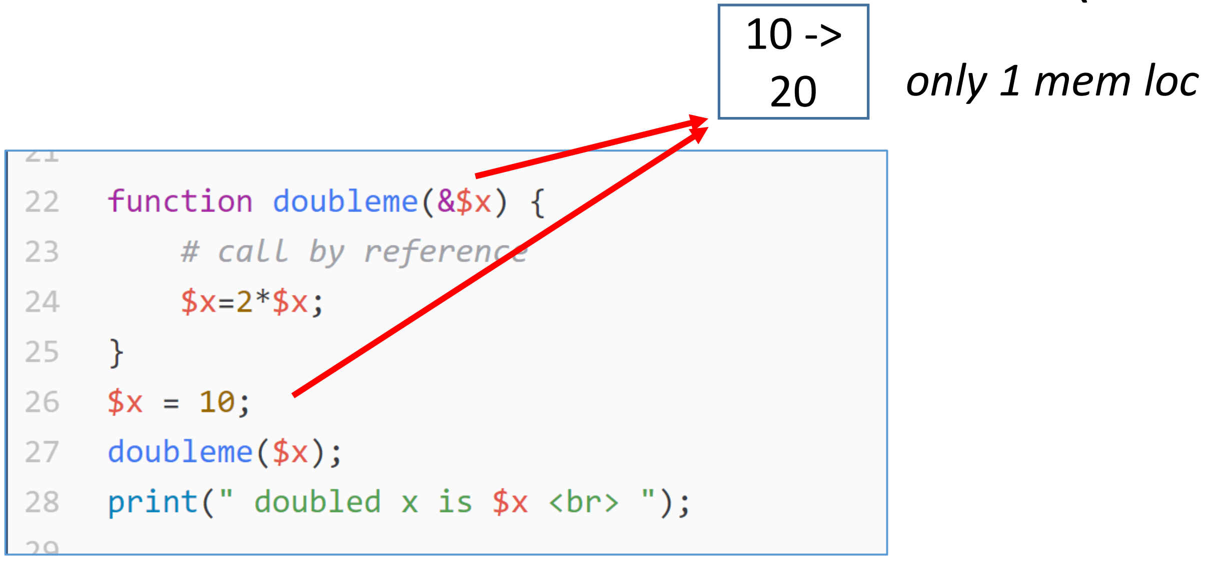
1. call-by-value: $x = 10
... in function: x = 20 ...
original x = 10; new y = 20



```php
function double($x) {
    $x = 2*$x;
    print("<br>  ... in function: x = $x ...");
    return $x;
}

# 1. ------------------------------------------
$x = 10;
print("<p>1. call-by-value:  \$x = $x");
$y = double($x);  # we did not change $x
print("<br>original x = $x; new y = $y ");
```

2. Call by reference

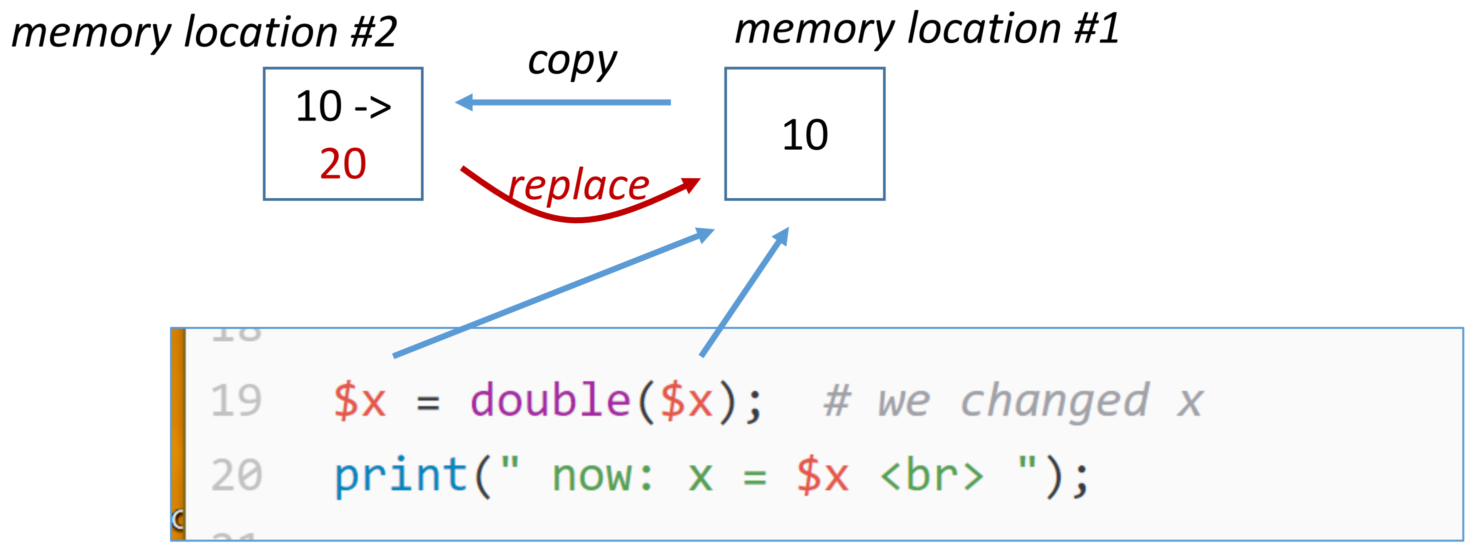
- PHP can also do "*call by reference*" **<u>if</u> you ask it to**
- A *reference* is simply the exact *memory* location. (A *pointer* to the *original* memory box.)
- If you pass a reference to a function, you pass the *exact* memory location → NO copy.
- Therefore, if you change the value in the function, you change the value in the main memory location of the original variable!
- A reference has an "&" in front of the var name. (from C++)



```php
function doubleme(&$x) {
    # call by reference
    $x=2*$x;
}
$x = 10;
doubleme($x);
print(" doubled x is $x <br> ");
```

What *is* good form?

- Globals are the *least* preferred.   *"use sparingly"*
- Call-by-reference, &$var, is OK.   *"not my pref."*
- Probably the <u>most preferred</u> is:
  pass in as a function parameter and **return** the updated value.
- This code *looks like* we changed $x directly, **but** we now know that we used a "local" variable in the process.



```
$x = double($x);   # we changed x
print(" now: x = $x <br> ");
```

4. More function info

Default values for input parameters

- Having a default value means that passing the parameter is optional.

```
function create_table2( $data, $border =1, $cellpadding = 4, $cellspacing = 4 )
{
  echo "<table border='$border' cellpadding = '$cellpadding'"
       ." cellspacing='$cellspacing'>";
  reset($data);
  $value = current($data);
  while ($value)
  {
     echo "<tr><td>$value</td></tr>\n";
     $value = next($data);
  }
  echo '</table>';
}
```

- Vars with defaults are *optional*
- Values can be overwritten in call
- Order matters

- *Calling* the function with parameter values:

```
create_table2($employees);
create_table2($customers, 1, 5, 5);
```

require() and include()

- A common practice is to put your *common* functions into a *library*.
  - *Library* means a separate file with <u>only</u> function(){} calls.
  - Headers, footers, mathematical functions, etc.
- Use require() or include() to read in those libraries of functions.
  - require() – will throw an error and *crash* if the library can't be found
  - include() will simply given a warning.
- *_once()
  - require_once() and include_once() functions will only read in the library file if it has not already been read in .

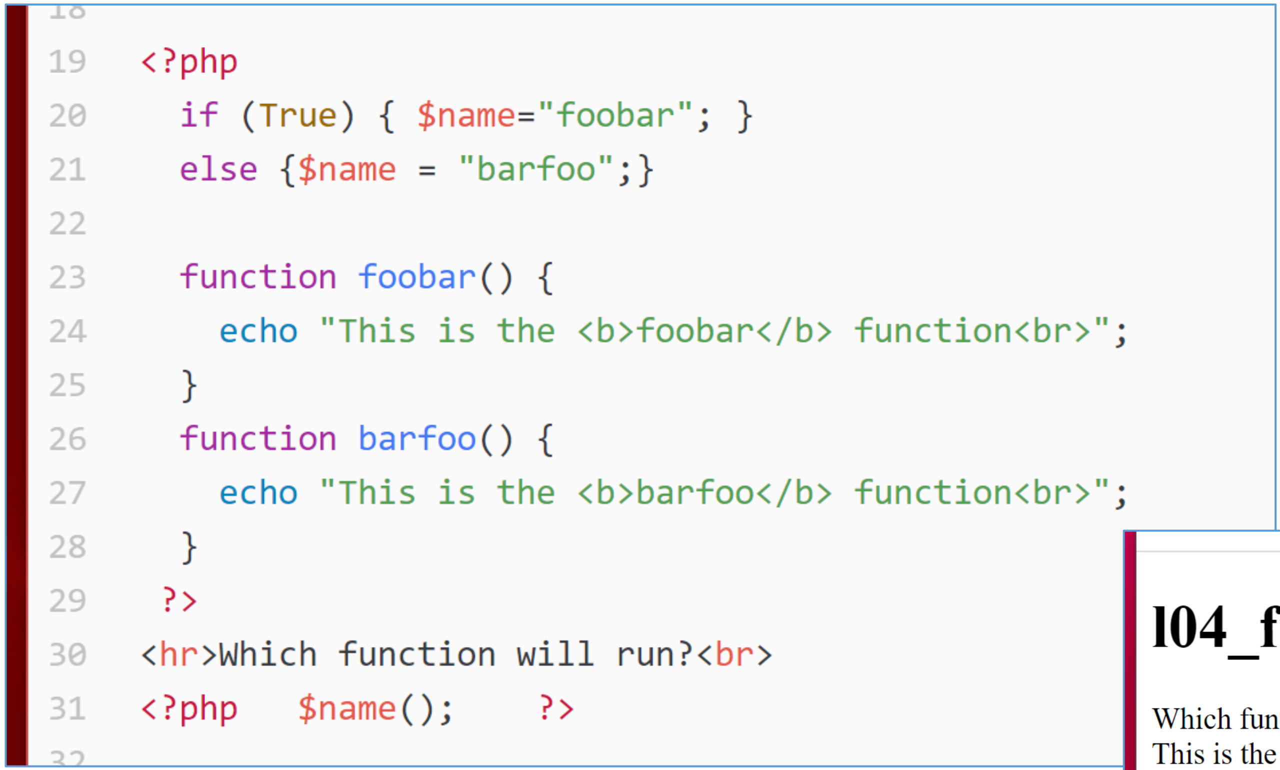
Examples:

- Use separate library files for repeated code

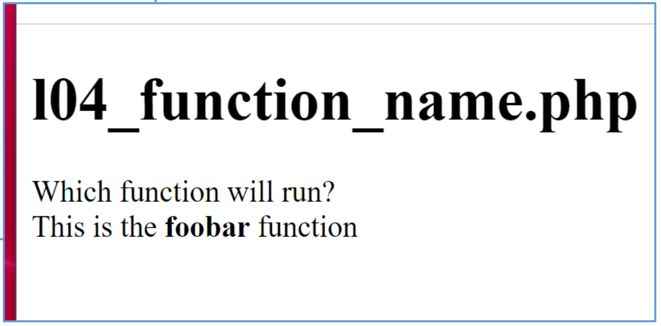
```
<?php
require "header.php";
require "nav.php";
?>

<!--  HTML here -->

<?php  require "footer.php"; ?>
```

Tricky -- Variable function name

- You can even call a function where the *name* of the function is in a variable.
- That way you can change the value of var $name, and then you will call a different function! Whoa. Cool.



```
<?php
  if (True) { $name="foobar"; }
  else {$name = "barfoo";}

  function foobar() {
    echo "This is the <b>foobar</b> function<br>";
  }
  function barfoo() {
    echo "This is the <b>barfoo</b> function<br>";
  }
 ?>
<hr>Which function will run?<br>
<?php   $name();    ?>
```



104_function_name.php

Which function will run?
This is the **foobar** function

5. File IO in PHP

A template and a few commands to memorize...

The File Template

- Learn (memorize) this ‘*template*’
  1. Open the file
  2. Flock the file
  3. Read or write to the file
  4. Unlock the file
  5. Close the file
- Open returns a file pointer
- Typical modes are: r, w, a

```
$fp = fopen("$root/orders/myorder.txt", 'r');
// read and process file
fclose($fp)
```

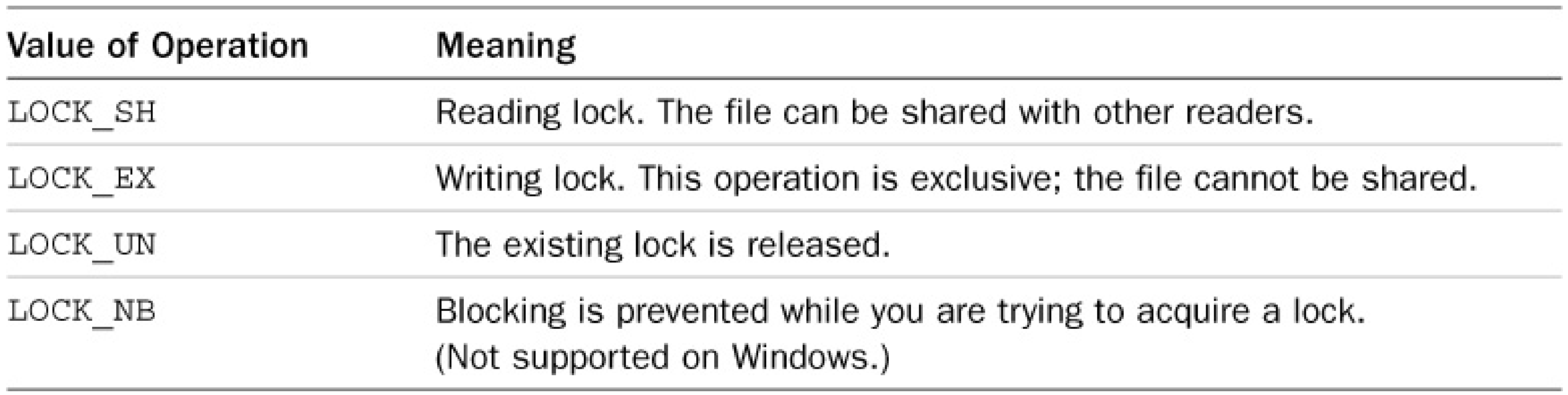
Opening a file

- `fopen()` takes a filename and a mode
- Filenames are usually in double quotes and can have variables
- `fopen` returns a file pointer
- Typical modes are: r, w, a
- It's good to error handle

```
$fp = fopen("$root/orders/myorder.txt", 'r');
if (!$fp) {
        echo "<p>File Error: Could not find file<br>";
        exit;
}
// lock
// read and process file
// unlock
fclose($fp)
```

Locking files

- Use the `flock()` command
- Syntax: `flock(fp, operation);`
- Call after opening and before closing the file
- PHP uses the correct OS calls
- Use LOCK_SH for reading and LOCK_EX when writing



Value of Operation	Meaning
`LOCK_SH`	Reading lock. The file can be shared with other readers.
`LOCK_EX`	Writing lock. This operation is exclusive; the file cannot be shared.
`LOCK_UN`	The existing lock is released.
`LOCK_NB`	Blocking is prevented while you are trying to acquire a lock. (Not supported on Windows.)

Reading in data from a file

There are a couple of options:

A. One line at a time:

- **fgets()** – one line into a string
- **fgetss()** – same as fgets, but strips out HTML tags
- **fgetcsv()** – returns an array splitting on commas

B. Read in the whole file:

- **readfile**(<filename>) – echos file to standard out (the web browser)
- **file_get_contents**(<filename>) – same as above, but returns one big string
- **fpassthru**(<fp>) – same as readfile(), but with file pointer as argument
- **file**(<filename>) – returns an array of strings, where each line in the file is an array element

Writing to a file

- Use the `fwrite()` command
- Syntax: `fwrite( $fp, $string );`
- Another option is `file_put_contents()` command
  - It has the file open and closed built-in
  - Syntax: `file_put_contents( filename, string );`

```
// open file 'w' mode
// lock file
$outputstr = "$name,$p1,$q1,$p2,$q2";
fwrite( $fp, $outputstr );
// unlock file
// close file
```

Example: read csv to an array

```
$fp = fopen("$root/apps/myorder", 'r');
if (!$fp) {
        echo "<p><b>Error.  Cannot open file.</b><br>";
        exit;
}
flock($fp, LOCK_SH);

print "<ul>";
while (!feof($fp)) {
        print "<li>";
        $order = fgetcsv( $fp, 0, "," );
        # $order is a 1D array
        foreach ($order as $o) {
                print "o=$o;";
        }
        print "<br>";
}

flock($fp, LOCK_UN);
fclose($fp);
print "</ul>";
```

Example: read in key=value pair

Data looks like:

```
key1=value1
key2=value2
```

```
$fp = fopen("./data.txt", 'r');
flock($fp, LOCK_SH);
if (!$fp) {
        echo "<p><b>Error.  Cannot open file.</b><br>";
        exit;
}
while (!feof($fp)) {
    # read in 1 line and strip ending return \n
    $line=rtrim(fgets($fp));
    list($key, $value) = explode("=",$line);
    #echo "debug: key=$key    value=$value<br>\n";

}
flock($fp, LOCK_UN);
fclose($fp);
```

Summary

1. Know how to write a custom PHP function
2. Understand local and global variable scope
3. Read in and write out text files for storing variables

Lab 4

Read from and write to files