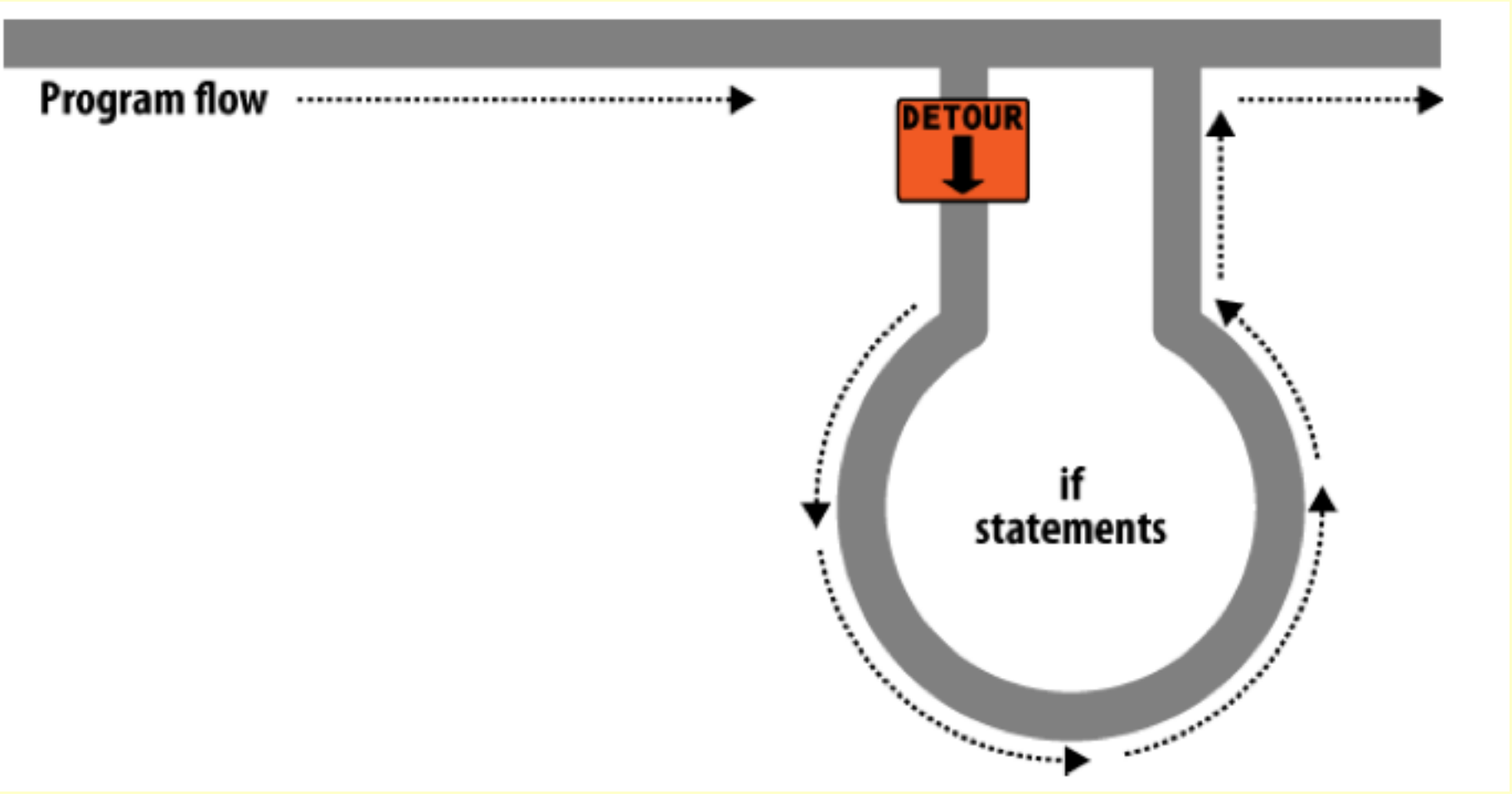
DT228/2 Web Development

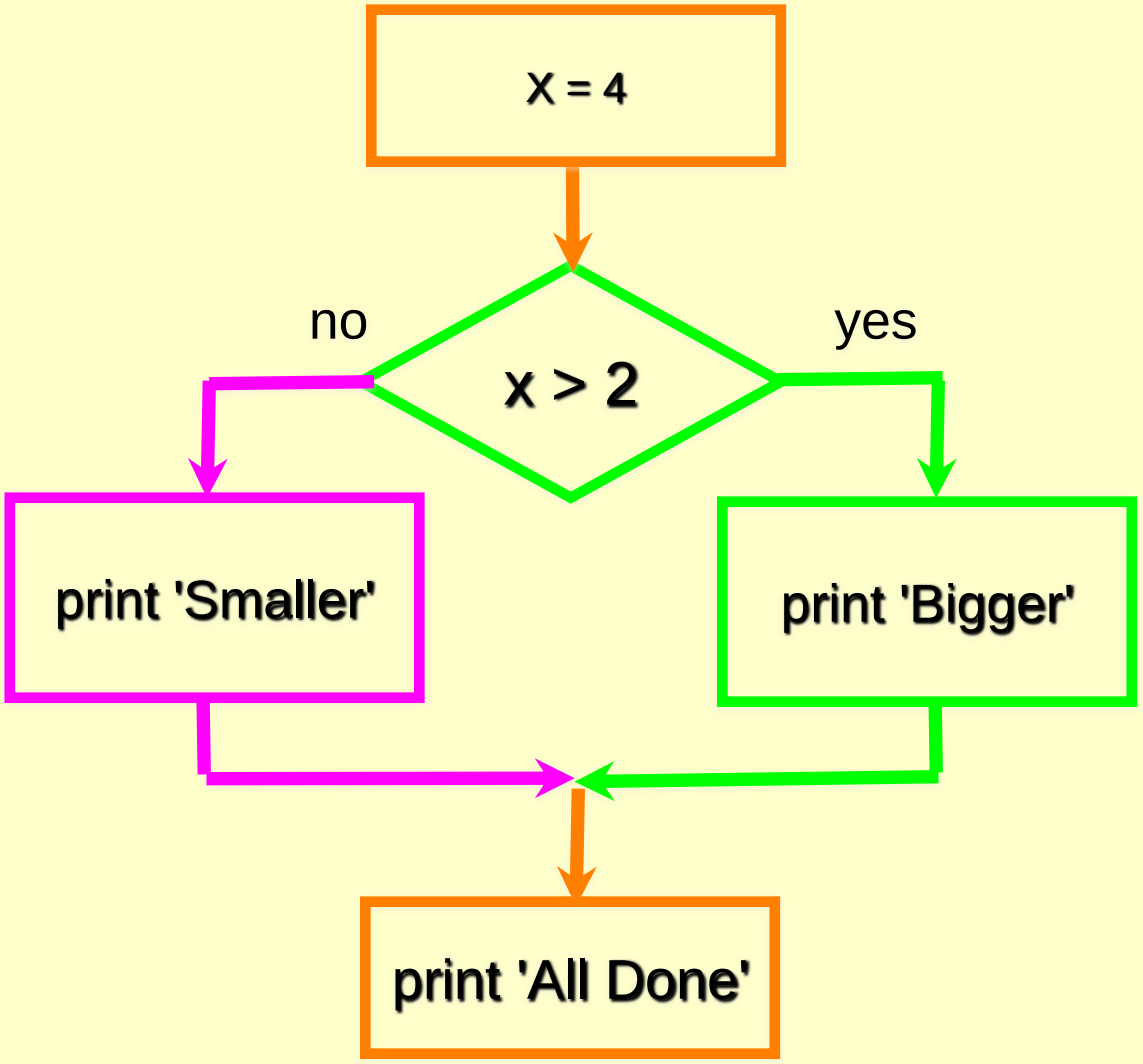
Basic PHP 2

Control Structures



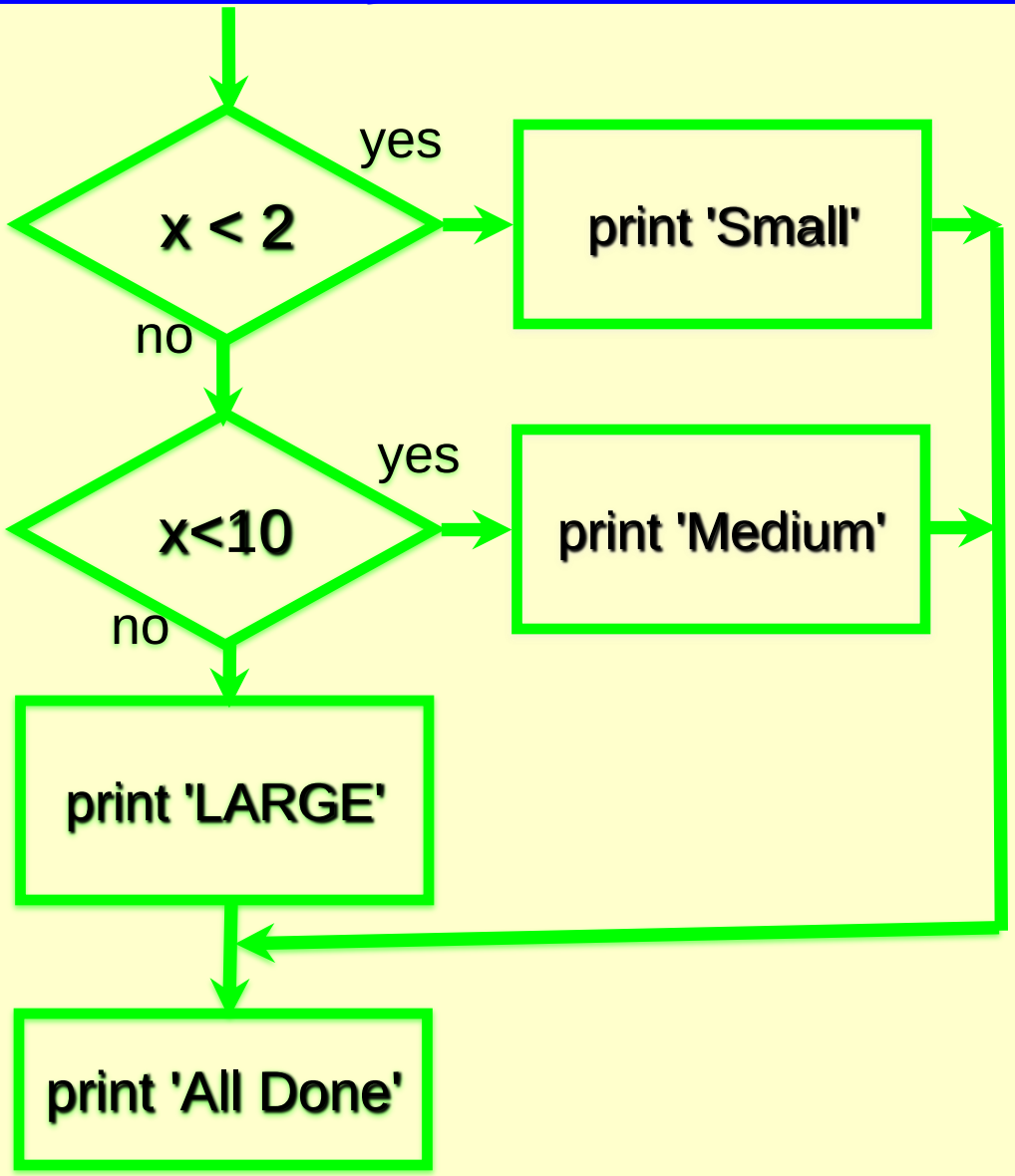
Two-way using else

```
$x = 4;
if ($x > 2) {
print "Bigger\n";
}
else {
print "Smaller\n";
}
print "All done\n";
```



Multi-way

```
$x = 7;
if
( $x < 2 ) {
        print "Small\n";
} elseif( $x < 10 ) {
        print "Medium\n";
} else {
        print "LARGE\n";
}
print "All done\n";
```



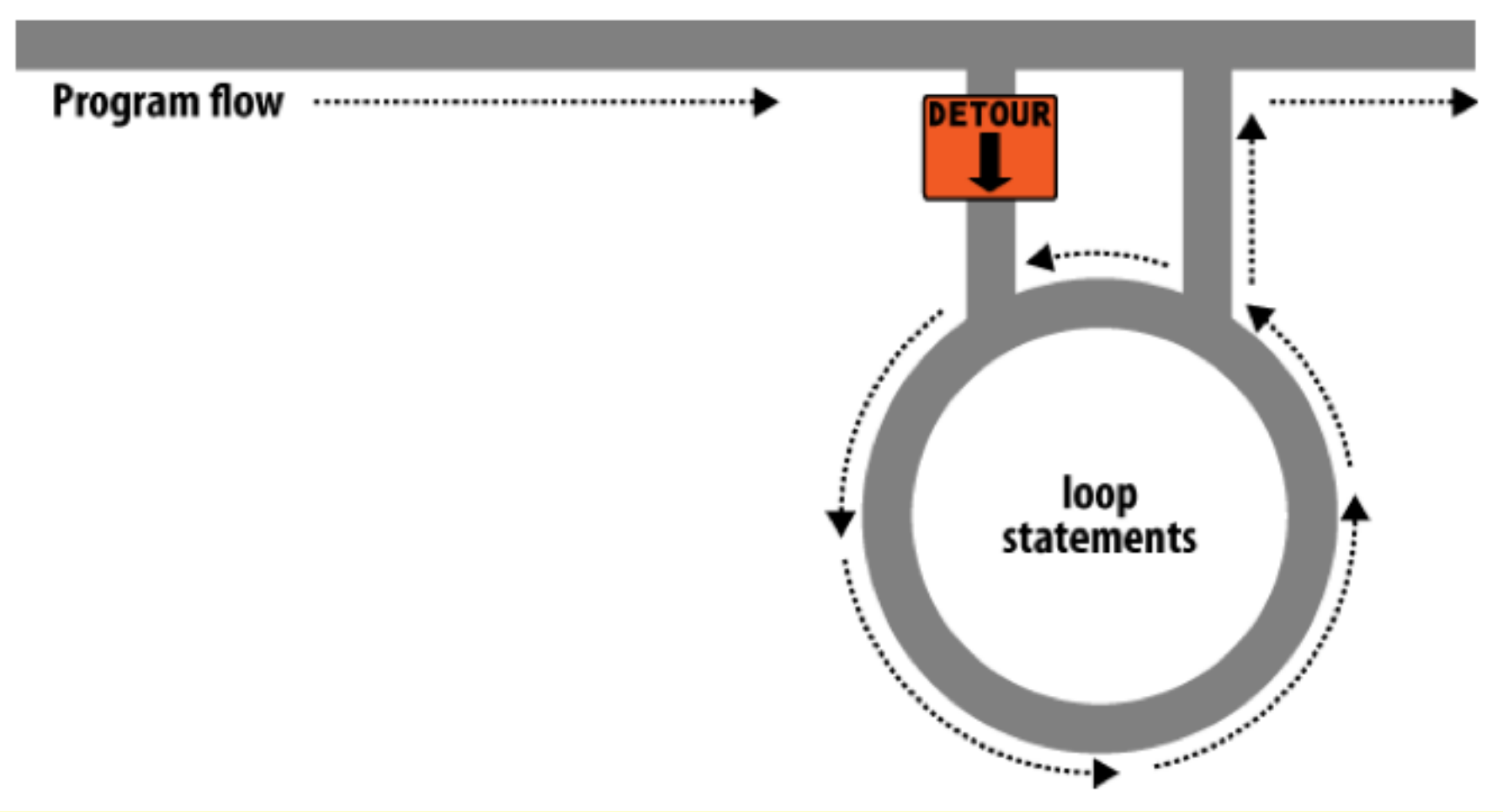
Curly Braces are not Required

```
if ($page == "Home") echo "You selected Home";
elseif ($page == "About") echo "You selected About";
elseif ($page == "News") echo "You selected News";
elseif ($page == "Login") echo "You selected Login";
elseif ($page == "Links") echo "You selected Links";
```

```
if ($page == "Home") { echo "You selected Home"; }
elseif ($page == "About") { echo "You selected About"; }
elseif ($page == "News") { echo "You selected News"; }
elseif ($page == "Login") { echo "You selected Login"; }
elseif ($page == "Links") { echo "You selected Links"; }
```

```
switch ($page)
{
        case "Home":
                echo "You selected Home";
                break;
        case "About":
                echo "You selected About";
                break;
        case "News": echo "You selected News";
                break;
        case "Login": echo "You selected Login";
                break;
        case "Links": echo "You selected Links";
                break;
}
```

Looping Structures



```
$fuel = 10;
while ( $fuel > 1) {
        print "Vroom vroom\n";
}
```

A while loop is a "zero-trip" loop with the test at the top. before the first iteration starts. We hand construct the iteration variable to implement a counted loop.

```
$fuel = 10;
while ( $fuel > 1) {
        print "Vroom vroom\n";
        $fuel = $fuel -1;
}
```

```
$count = 1;
do {
    echo "$count times 5 is " . $count * 5;
    echo "\n";
} while (++$count <= 5);
```

A do-while loop is a "one-trip" loop with the test at the bottom after the first iteration completes.

```
1 times 5 is 5
2 times 5 is 10
3 times 5 is 15
4 times 5 is 20
5 times 5 is 25
```

```
for( $count=1 ;  $count<=6 ;  $count++ ) {
    echo "$count times 6 is " . $count * 6;
    echo "\n";
}
```

A for loop is the simplest way to construct a counted loop.

```
1 times 6 is 6
2 times 6 is 12
3 times 6 is 18
4 times 6 is 24
5 times 6 is 30
6 times 6 is 36
```

Looping Through an Array

```
<?php
        $stuff = array("name" => "Liu",
                                "course" => "DT228");
        foreach ( $stuff as $k => $v ) {
                echo "Key=", $k ," Val=", $v ,"\n";
        }
?>
```

```
Key= name Val= Liu
Key= course Val= DT228
```

Looping Through an Array

```
<?php
        $stuff = array("Liu, "DT228");
        foreach ( $stuff as $k => $v ) {
                echo "Key=", $k ," Val=", $v ,"\n";
        }
?>
```

```
Key= 0 Val= Liu
Key= 1 Val= DT228
```

Looping Through an Array

```
<?php
        $stuff = array("Liu, "DT228");
        for ($i=0; $i < count( $stuff ) ; $i++) {
                echo "Index=",$i," Val=", $stuff [$i],"\n";
        }
?>
```

```
Index= 0 Val= Liu
Index= 1 Val= DT228
```

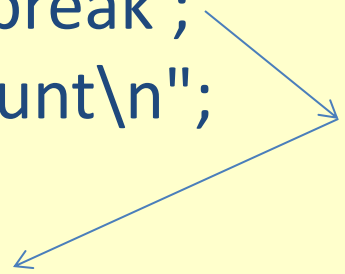
Loop Controls

- Like many C-inspired languages, PHP has two control structures that work within a loop
  - break - exit the loop immediately
  - continue - finish the current iteration and jump to the next iteration, starting at the top of the loop

Breaking Out of a Loop

- The break statement ends the current loop and jumps to the statement immediately following the loop
- It is like a loop test that can happen anywhere in the body of the loop

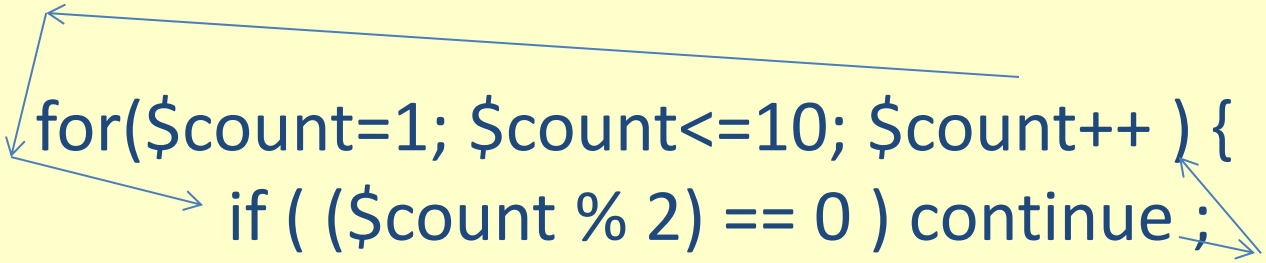
```
for($count=1; $count<=600; $count++ ) {
if ( $count == 5 ) break ;
echo "Count: $count\n";
}
echo "Done\n";
```



```
Count: 1
Count: 2
Count: 3
Count: 4
Done
```

Finishing an Iteration with continue

- The continue statement ends the current iteration and jumps to the top of the loop and starts the next iteration



```
for($count=1; $count<=10; $count++ ) {
        if ( ($count % 2) == 0 ) continue ;
        echo "Count: $count\n";
}
echo "Done\n"
```

```
Count: 1
Count: 3
Count: 5
Count: 7
Count: 9
Done
```

Conversion / Casting

- As PHP evaluates expressions, at times values in the expression need to be converted from one type to another as the computations are done.

- PHP does aggressive implicit type conversion (casting)

- You can also make type conversion (casting) explicit with casting operators.

Casting

```
$a = 56; $b = 12;
$c = $a / $b;
echo "C: $c\n";
$d = "100" + 36.25 + TRUE;
echo "D: ". $d . "\n";
echo "D2: ". (string) $d . "\n";
$e = (int) 9.9 - 1;
echo "E: $e\n";
$f = "sam" + 25;
echo "F: $f\n";
$g = "sam" . 25;
echo "G: $g\n"
```

In PHP, division forces operands to be floating point. PHP converts expression values silently and aggressively

C:

C: 4.66666666667
D: 137.25
D2: 137.25
E: 8
F: 25
G: sam25

Explicit Casting

Cast type	Description
`(int) (integer)`	Cast to an integer by dropping the decimal portion
`(bool) (boolean)`	Cast to a Boolean
`(float) (double) (real)`	Cast to a floating-point number
`(string)`	Cast to a string
`(array)`	Cast to an array
`(object)`	Cast to an object

PHP Casting Example

```
$x = "100" + 25;
echo "X: $x\n";
$y = "100" . 25;
echo "Y: $y\n";
$z = "sam" + 25;
echo "Z: $z\n";
```

```
X: 125
Y: 10025
Z: 25
```

```
echo "A".FALSE."B\n";
echo "X".TRUE."Y\n";
```

```
AB
X1Y
```

The concatenation operator tries to convert its operands to strings, TRUE becomes an integer 1 and then becomes a string. FALSE is "not there" it is even "smaller" than zero. At least when it comes to width.

PHP Functions

Why Functions?

- PHP has lots of built-in functions that we use all the time
- We write out own functions when our code reaches a certain level of complexity

To function or not to function…

- Organize your code into “paragraphs” - capture a complete thought and “name it”
- Don’t repeat yourself - make it work once and then reuse it
- If something gets too long or complex, break up logical chunks and put those chunks in functions
- Make a library of common stuff that you do over and over – perhaps share this with your friends...

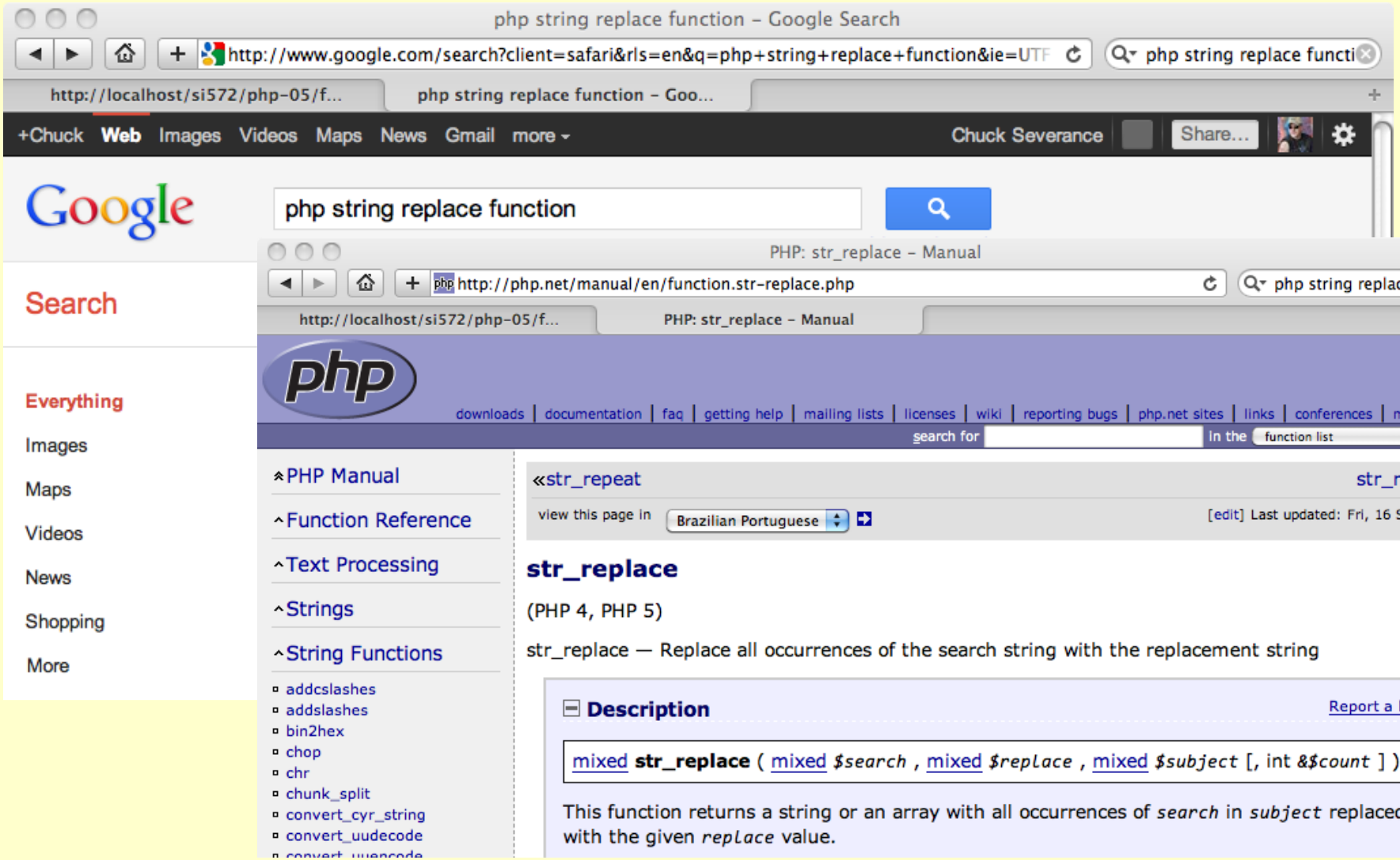
Built-In Functions …

•Much of the power of PHP comes from its built-in Functions

```
echo strrev(" .dlrow olleH");
echo str_repeat("Hip ", 2);
echo strtoupper("hooray!");
echo "\n";
```

Hello world. Hip Hip HOORAY!

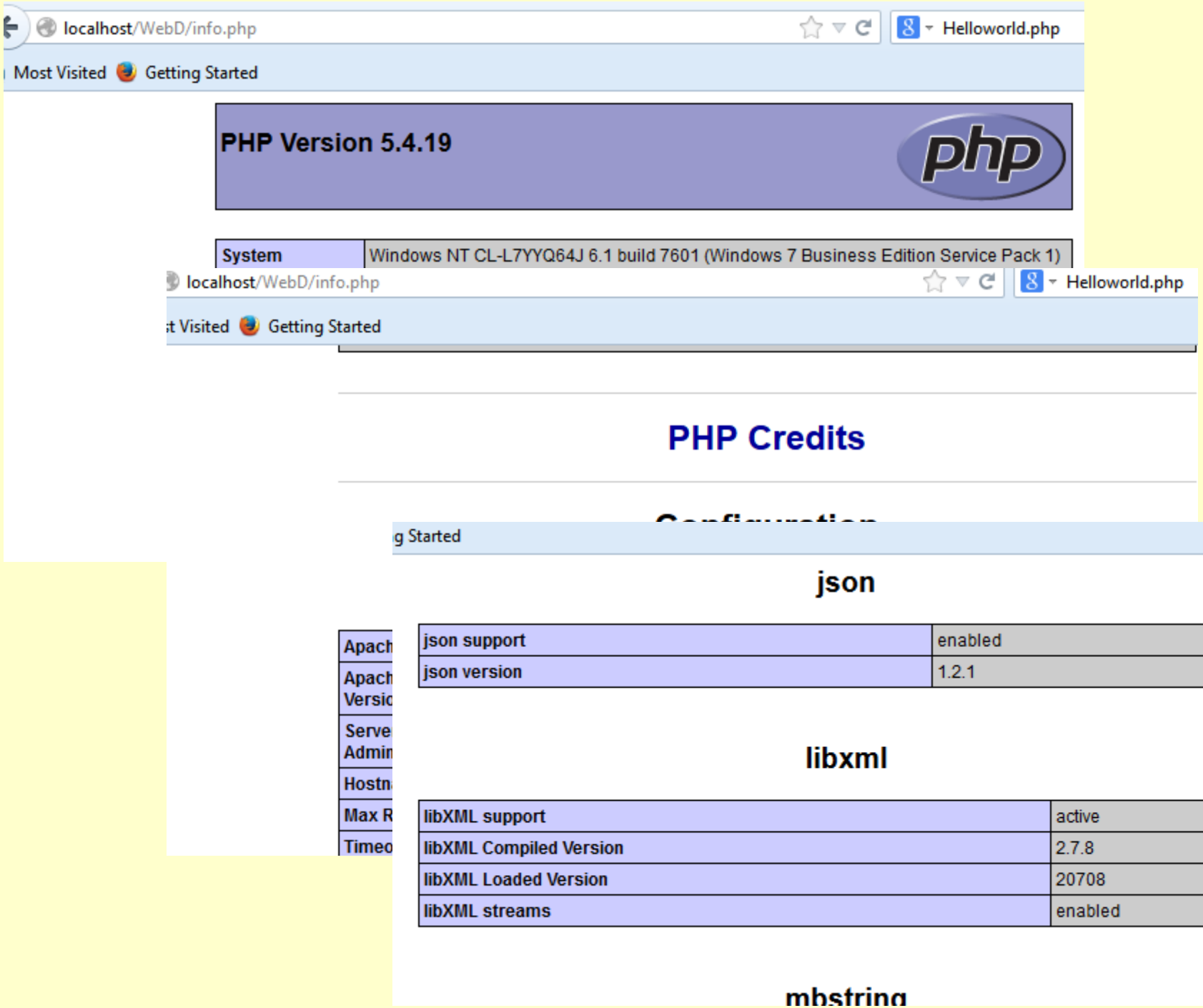
PHP Documentation - Google



One Heck of a Function

- PHP is a very configurable system and has lots of capabilities that can be plugged in.
- The phpinfo() function prints out the internal configuration capabilities of your particular PHP installation

```
<?php
        phpinfo();
?>
```


localhost/WebD/info.php
Helloworld.php
Most Visited
Getting Started
PHP Version 5.4.19
System
Windows NT CL-L7YYQ64J 6.1 build 7601 (Windows 7 Business Edition Service Pack 1)
localhost/WebD/info.php
Helloworld.php
Getting Started
PHP Credits
Configuration
Started
json
json support
enabled
json version
1.2.1
libxml
libXML support
active
libXML Compiled Version
2.7.8
libXML Loaded Version
20708
libXML streams
enabled
mbstring

Defining Your Own Functions

- We use the function keyword to define a function, we name the function and take optional argument variables. The body of the function is in a block of code{ }

```
function
greet() {
    print "Hello\n";
}
greet();
greet();
greet();
```

```
Hello
Hello
Hello
```

Return Values

- Often a function will take its arguments, do some computation and return a value to be used as the value of the function call in the calling expression. The return keyword is used for this.

```
function greeting() {
return "Hello";
}
print greeting() . " Glenn\n";
print greeting() . " Sally\n";
```

```
Hello Glenn
Hello Sally
```

Arguments

- Functions can choose to accept optional arguments. Within the function definition the variable names are effectively "aliases" to the values passed in when the function is called

```
function howdy($lang) {
    if ( $lang == 'es' ) return "Hola";
    if ( $lang == 'fr' ) return "Bonjour";
    return "Hello";
}
print howdy('es') . " Glenn\n";
print howdy('fr') . " Sally\n";
```

```
Hola Glenn
Bonjour Sally
```

Choosing Function Names

- Much like variable names - but do not start with a dollar sign
    - Start with a letter or underscore - consist of letters, numbers, and underscores (_)
- Avoid built in function names

Call By Value

- The argument variable within the function is an "alias" to the actual variable
- But even further, the alias is to a *copy* of the actual variable in the function call

```
function double($alias) {
    $alias = $alias * 2;
    return $alias;
}
$val = 10;
$dval = double($val);
echo "Value = $val Doubled = $dval\n"
```

Value = 10 Doubled = 20

Call By Reference

- Sometimes we want a function to change one of its arguments - so we indicate that an argument is "by reference " using (&)

```
function triple(&$realthing) {
      $realthing = $realthing * 3;
}
$val = 10;
triple($val);
echo "Triple = $val\n";
```

Triple = 30

Variable Scope

- In general, variable names used inside of function code, do not mix with the variables outside of the function. They are walled-off from the rest of the code. This is done because you want to avoid "unexpected" side effects if two programmers use the same variable name in different parts of the code.
- We call this "name spacing" the variables. The function variables are in one "name space" whilst the main variables are in another "name space“
- Like little padded cells of names - like silos to keep things separate.

Normal Scope (isolated)

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

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

TryZap = 10

Global Scope (common)

```
function dozap() {
    global $val;
    $val = 100;
}
$val = 10;
dozap();
echo "DoZap = $val\n";
```

DoZap = 100

Programming in Multiple Files

- When your programs get large enough, you may want to break them into multiple files to allow some common bits to be reused in many different files.

```
<!DOCTYPE html>
<head>
<?php include("header.php"); ?>
</head>
<body>
<?php include("nav.php"); ?>
<div id="main">
.
.
.
```

Including files in PHP

- include "header.php"; - Pull the file in here
- include_once "header.php"; - Pull the file in here unless it has already been pulled in before
- require "header.php"; - Pull in the file here and die if it is missing
- require_once "header.php"; - You can guess what this means...
- These can look like functions - require_once("header.php");

Coping with Missing Bits

- Sometimes depending on the version or configuration of a particular PHP instance, some functions may be missing. We can check that.

```
if ( function_exists("array_combine"))
{
echo "Function exists";
}
else
{
echo "Function does not exist";
}
```