Variable scope

- All <?php ?> blocks in the file share the same scope
  - This example prints the number 5 as expected

```
<?php $x = 5; ?>
<h3>And then some HTML</h3>
<?= $x ?>
```

Defining functions

- PHP functions follow c-style syntax
  - But with no explicit types

```
function pythagorean( $s1, $s2 ) {
   return sqrt( $s1*$s1 + $s2*$s2 );
}
```

  - No explicit parameter or return types
  - No return statement implies a return of NULL

Calling functions

- PHP function calls follow c-style syntax

```
$s3 = pythagorean( $s1, $s2 );
```

  - Number of parameters must match
  - Types are not enforced (obviously)

- Functions have global scope
  - Can be called from anywhere
  - Should be defined at the top-level of the file
  - Do *not* have to be defined before they are called

Default parameter values

- Function parameters may be given default values

```
function makecoffee($type = "cappuccino")
{
  return "Making a cup of $type.\n";
}
echo makecoffee();              # Making a cup of cappuchino.
echo makecoffee(null);          # Making a cup of .
echo makecoffee("espresso"); # Making a cup of espresso.
```

  – Only rightmost parameters can be default

Pass by reference

- PHP defaults to pass-by-value
- Pass by reference is supported

```
function add_some_extra( &$string )
{
    $string .= 'and something extra.';
}
$str = 'This is a string, ';
add_some_extra( $str );
echo $str;  # This is a string, and something extra.
```

  – But really only makes sense when using classes

Global variables

- Variables are assumed to exist in the scope they are used
  - There's no declaration to say otherwise

```
function example()
{
  print $a;      # will always print NULL
}
```

- To access a global variable, it must be declared as such

```
$a = 15;
function example()
{
  global $a;
  print $a;      # will print 15
}
```

Includes

- The *include* and *require* functions insert the contents of the specified file at that point
  - Which are then interpreted right there

```
include( "login.php" );
require( "header.php" );
```

- Using include/require encourages modularity
  - Put common functions in included files
  - Put header/footer/sidebar code in included files
- Commonly, use *include_once/require_once* to avoid the same file being included twice
  - Like the include guard problem in C++

Client-side redirects

- Asks the client to request a replacement page
- Send an HTTP header via PHP
  - PHP: header("Location: http://google.com");
- Or directly in HTML
  - <meta http-equiv="refresh“ content="0;url=http://google.com/">