PHP II

`print_r`

- `print_r` is a function that recursively prints any object passed to it
  - It is specifically designed to produce output that is readable by humans
  - Very helpful for debugging
- An optional second parameter can be set to `true` to force `print_r` to return a string rather than printing directly

In [ ]:
```
print_r(explode(',', "John,Paul,George,Ringo"));
```

In [ ]:
```
print_r(explode(',', "John,Paul,George,Ringo"),true);
```

Arrays

- Arrays in PHP can either be
  - associative
  - indexed
- Arrays are heterogeneous in PHP
- Indexed using square brackets
  - indexing starts at 0

Array Syntax

- Arrays are created by calling the special `array` function
- To initialize an indexed style array, pass the values separated by commas

```
$an_array = array(1,2,3,"Hello")
```

- To intialize an associatve array, pass the key-value pairs separated by commas
  - Key-value pairs are specified using the sytnax `key => value`

```
$assoc_array = array('course'=>'CMSC 433', 'title'=>'Scripting Languages');
```

In [ ]:

```
$assoc_array = array('course'=>'CMSC 433',
                     'title'=>'Scripting Languages',
                     'time'=> 11.5);
echo print_r($assoc_array,true);
```

Adding to an Array

- To push a new element on the end of an array, assign the new element to the array indexed with empty square brackets

```
$an_array[] = 5;
```

- To add a new element and a specific key, assign into the array indexed at that new key

```
$an_array['new_key'] = 10;
```

- Either of these methods can be used with a previsouly undeclared variable to both create an array and insert one element into it

In [ ]:

```
$some_numbers = array(0,4,6,1,4,9,0,22);
$some_numbers[] = 100;
echo print_r($some_numbers,true)
```

In [ ]:

```
$ascii = array('A' => 65, 'B' => 66, 'a' => 97);
$ascii['b'] = 98;
echo print_r($ascii,true);
```

In [ ]:

```
$ascii = array('A' => 65, 'B' => 66, 'a' => 97);
$ascii[] = 98;
echo print_r($ascii,true);
```

Array Functions

- For a full list of array functions, see the PHP manual
- Some example functions:
  - range
  - count
  - list
  - array_slice
  - array_chunk

Array Functions

- For a full list of array functions, see the PHP manual
- Some example functions:
  - array_key_exists
  - in_array
  - shuffle
  - array_sum
  - sort

In [ ]:
```
echo print_r(range(1,10) ,true);
```

In [ ]:
```
## Range in PHP must take two parameters!
echo print_r(range(10),true);
```

In [ ]:
```
echo print_r(range(1,10,4),true);
```

In [ ]:
```
count(range(1,10));
```

In [ ]:

```
$info = array('coffee', 'brown', 'caffeine');
list($drink, $color, $power) = $info;
echo $drink;
echo $color;
echo $power;
```

In [ ]:
```
$my_range = range(1,10);
echo print_r(array_slice($my_range,3,7),true);
```

In [ ]:
```
echo print_r(array_slice($my_range,-4,6),true);
```

In [ ]:
```
echo print_r(array_chunk($my_range,2),true);
```

In [ ]:
```
array_key_exists('Dog',range(0,10));
```

In [ ]:
```
array_key_exists('Dog',array('Dog' => 40, 'Cat' => 20))
```

In [ ]:
```
in_array(10,range(0,10));
```

In [ ]:
```
in_array(10,range(0,5));
```

In [ ]:
```
in_array(10,array('ten' => 10, 'twenty' => 20));
```

In [ ]:
```
in_array(0,array('ten' => 10, 'twenty' => 20));
```

In [ ]:
```
shuffle(range(1,10));
```

In [ ]:
```
shuffle($my_range);
echo print_r($my_range,true);
```

In [ ]:
```
array_sum(array(1,2,3,"5",2.4));
```

Extract and Compact

- `extract` and `compact` are two functions that allow you to convert from and to arrays respecitively
- Rather than return anything, `extract` creates new variables with names based on the keys in the array
- `compact` on the other hand takes a list of strings, and looks for variables with those names, creating an array out of them
    - How would you write your own compact function?

In [ ]:

```
$info = array('school'=>'UMBC',
              'department'=>'CSEE',
              'building'=>'ITE');
extract($info);
echo print_r($info,true);
echo "The $department department of $school is located in $building";
```

In [ ]:

```
$name = 'Raúl González Blanco';
$position = "Forward";
$nationality = "Spanish";
$player = compact('name','position','nationality');
echo print_r($player,true);
```

Multi-Dimensional Arrays

- Mutli-Dimensional arrays are fairly well supported in PHP
  - You could make an array of arrays manually
- Just like with a one-dimensional array, you can assign directly to mutliple indices with out ever declaring something as an array

In [ ]:

```
$matrix[0][] = 10;
$matrix[1][1] = 20;
$matrix[1]['Key'] = 30;
$matrix[1][] = 3;
$matrix[][][] = 40;
#shift($matrix,10);
echo print_r($matrix,true);
```

Loops

- PHP has all the standard looping constructs
  - While
  - Do-While
  - For
- It also has a specific `foreach` construct

```
foreach($array as $element){
}
```

In [ ]:

```
$i = 10;
while($i > 0)
{
    echo $i;
    $i--;
}
```

In [ ]:

```
$i = 10;
do
{
    echo $i;
    $i--;
}while($i > 0); #Note the Semicolon
```

In [ ]:

```
for($i = 0; $i < 10; $i++){
echo $i;
}
```

In [ ]:

```
$to_loop = array('a', 'b', 'c', 'd', 'e');
foreach($to_loop as $el){
echo $el;
}
```

In [ ]:

```
$to_loop = array('a'=>97, 'b'=>98, 'c'=>99, 'd'=>100, 'e'=>101);
foreach($to_loop as $el){
echo $el;
}
```

In [ ]:

```
$to_loop = array('a'=>97, 'b'=>98, 'c'=>99, 'd'=>100, 'e'=>101);
foreach($to_loop as $k => $el){
echo $k, $el;
}
```

Function Definitions

- Function in PHP are declared using the keyword `function`
  - The parameters also use the `$` notation
- Function definitions can appear anywhere in a file
  - Must be defined before they are used
- Support returning by reference

In [ ]:

```
function my_first_function(){
    echo "Hello from inside a function";
}
my_first_function();
my_first_function();
```

In [ ]:

```
function someMath($number1, $number2){
    echo $number1 * $number2;
}
someMath(1,2);
```

In [ ]:

```
function someMath2($number1, $number2){
    return $number1 * $number2;
}
someMath2(1,2);
```

Scope Refresher

- Remember in PHP scope is global, except in functions where it is local
  - In functions, things in global scope aren't in scope either
- We can modifiy the scope of a variable using two different keywords
  - `global`
  - `static`

In [ ]:

```
function scope_tester() {
        $a .= " World!";
        echo "a in scope_tester(): $a";
    }
$a = "Hello";
echo "a before scope_tester(): ", $a;
scope_tester();
echo "a after scope_tester(): ", $a;
```

In [ ]:

```
function scope_tester2() {
        $a .= " World!";
        echo "a in scope_tester(): $a";
    }
global $a;
$a = "Hello";
echo "a before scope_tester2(): ", $a;
scope_tester2();
echo "a after scope_tester2(): ", $a;
```

In [ ]:

```
function scope_tester3() {
        global $a;
        $a .= " World!";
        echo "a in scope_tester(): $a";
    }
$a = "Hello";
echo "a before scope_tester3(): ", $a;
scope_tester3();
echo "a after scope_tester3(): ", $a;
```

In [ ]:

```
function counter() {
        static $n = 0;
        $n++;
        return $n;
    }
echo counter();
echo counter();
echo counter();
```

Pass-by-Value and Pass-by-reference

- By default PHP passes parameters by value
  - Even arrays
  - This is fine in most instances
- If you want your function to modify a variable (like shuffle does), append an ampersand `&` before the variable in the definition
  - Can't pass literals by reference

In [ ]:

```
function change_array($arr){
    $arr[0] = 20;
    echo $arr[0];
}
$my_array = range(0,10);
echo $my_array[0];
change_array($my_array);
echo $my_array[0];
```

In [ ]:

```
function change_array_correct(&$arr){
    $arr[0] = 20;
    echo $arr[0];
}
$my_array = range(0,10);
echo $my_array[0];
change_array_correct($my_array);
echo $my_array[0];
```

In [ ]:
```
change_array_correct(range(0,10));
```

Default Parameters

- PHP allows default parameters, following the conventions of most languages that do so
  - Optional parameters must appear after all non-optional parameters
  - Parameters not pass will use their default value

In [ ]:

```
function say_hello($who = "World"){
    echo "Hello $who!";
}
say_hello();
say_hello("Class");
```

In [ ]:

```
function custom_range($end, $start = 0){
return range($start,$end);
}
echo print_r(custom_range(4),true);
echo print_r(custom_range(4,1),true);
```

Return

- Just as with parameters, PHP allows values to be returned by either copy or reference
  - By default, values that are returned are copied
- To return by reference, place an amperstand (`&`) before the function name in the declaration

```
function &name(....)
{
}
```

In [ ]:

```
function getFredReturnByValue() {
    global $names;
    return $names[0];
}

function & getFredReturnByRef() {
    global $names;
    return $names[0];
}

$names = array("Fred", "Barney", "Wilma", "Betty");
```

In [ ]:

```
$fred = getFredReturnByValue();
echo $fred;
$fred = 'Bam Bam';
$fred = getFredReturnByValue();
echo $fred;
```

In [ ]:

```
$fred =& getFredReturnByRef();
echo $fred;
$fred = 'Bam Bam';
$fred =& getFredReturnByRef();
echo $fred;
```

Varags

- To use a variable number of arguments in PHP, don't change the function signature at all
- Inside the function, call the function `func_get_args`
    - This returns an array holding the additional arguments

In [ ]:

```
function maximum() {
    $args = func_get_args();
    $max = $args[0];
    for($i = 1, $n = count($args); $i < $n; $i++) {
      $max = $args[$i] > $max ? $args[$i] : $max;
    }
    return $max;
  }
echo maximum(1,2,300,4,5,6);
```

Anonymous Functions

- Since PHP 5.3, truly anoymous functions have been supported
- To write one, provide the parameters directly after the `function` keyword
    - This can be saved to a variable if you'd like, or passed directly to another function

In [ ]:

```
$max = function($a,$b){
    return ($a > $b)?$a:$b;
}; #Note the semicolon
$max(1,2);
```

Reading Files

- PHP file reading can be done in a traditional manner using a file pointer
  - `fopen(file_name)`
- PHP also has numerous special functions meant to help with reading files
  - `file_get_contents(name)`
  - `file(name)`

In [ ]:

```
if($fp = fopen('hello.sh','r')){
    while($line = fgets($fp)){
        echo htmlspecialchars($line);
    }
}else{
    echo "Can't read file";
}
```

In [ ]:
```
htmlspecialchars(file_get_contents('hello.sh'));
```

In [ ]:

```
$contents = file('hello.sh');
echo print_r($contents,true);
```

Writing Files

- Just as with reading files, PHP allows files to be written using
  - The traditional file handle way
  - A simplified function
    - `file_put_contents` (Since PHP 5 on)

In [ ]:

```
if($fp = fopen('hello.txt','w')){
    fwrite($fp, "Hello");
    fwrite($fp, " World\n");
    fwrite($fp, "You need to include new lines explicitly\n");
    fclose($fp);
}else{
    echo "Can't open file";
}
```

In [ ]:

```
if($fp = fopen('hello.txt','a')){
    fwrite($fp, "Hello");
    fwrite($fp, " World\n");
    fwrite($fp, "You need to include new lines explicitly\n");
    fclose($fp);
}else{
    echo "Can't open file";
}
```

In [ ]:
```
file_put_contents("hello.txt","This\nWrites\nIt\nAll at once!")
```

In [ ]:
```
file_put_contents("hello.txt","This\nWrites\nIt\nAll at once!",FILE_APPEND)
```

In [ ]:
```
file_put_contents("range.txt",range(0,20));
```

Locking Files

- In general, you should expect your PHP script to be running multiple copies of itself at once
  - Lots of people are going to be on your website constantly
- When you lock a file in PHP, it only prevents other PHP processes from accessing it

In [ ]:

```
if($fp = fopen('hello.txt','a')){
    if(flock($fp,LOCK_EX)){
    fwrite($fp, "Hello");
    fwrite($fp, " World\n");
    fwrite($fp, "You need to include new lines explicitly\n");
    flock($fp,LOCK_UN);
    fclose($fp); }
    else{
     echo "Couldn't get lock";
    }
}else{
    echo "Can't open file";
}
```

In [ ]:
```
file_put_contents("range.txt",range(0,20),LOCK_EX);
```

Externalizing Code

- To pull in code from another location, there are several functions PHP provides
  - `include(file_name)` - Includes and evaluates the file
  - `require(file_name)` - Includes and evaluates the file, killing the script if unable to do so
  - `include_once(file_name)` - Only include and evaluate this file one time
  - `require_once(file_name)` - Only require this file one time

In [ ]:
```
include 'header.php'; #same as include('header.php')
```

In [ ]:
```
include 'I dont exist';
```

In [ ]:
```
require 'I dont exist';
```

In [ ]:
```
include_once 'header.php';
include_once 'header.php';
```