PHP Arrays, Array Functions and Processing

Arrays Defined and Types

- In computer programming an array is a named, ordered collection of values that are all of the same data type.
- Arrays can hold primitive data types (int, float, char, boolean) or referenced data types (Strings, objects in general, arrays).
- Array come in three broad “types”:
  - Indexed
  - Associative
  - Multidimensional (layers of indexed/associative arrays)

Index vs. Associative

- An indexed array is one whose keys are numbered as integers, starting at *0* and goes up to *n – 1* where *n* is the number of elements in the array
- An associative array is one has string identifiers as keys, the each element of the array is a property-value pair

Array Functions: Informational

- is_array() - Finds whether a variable is an array
- count():Count all elements in an array, or something in an object
- sizeof():an alias of the count() function
- isset() - Determine if a variable is set and is not NULL
- array_keys() - Return all the keys or a subset of the keys of an array
- in_array() - Checks if a value exists in an array
- array_values() - Return all the values of an array
- array_key_exists() - Checks if the given key or index exists in the array
- array_search() - Searches the array for a given value and returns the corresponding key if successful

Array Functions: Transformative

- Transformative functions change (or transform) the array:
  - array_pop() - Pop the element off the end of array
  - array_push() - Push one or more elements onto the end of array
  - array_shift() - Shift an element off the beginning of array
  - array_unshift() - Prepend one or more elements to the beginning of an array
  - array_slice() - Extract a slice of the array
  - array_splice() - Remove a portion of the array and replace it with something else
  - array_chunk() - Split an array into chunks
  - array_merge() - Merge one or more arrays
  - unset() - Unset a given variable

Array Functions: Traversing

- reset() - Set the internal pointer of an array to its first element
- prev() - Rewind the internal array pointer
- current() - Return the current element in an array
- next() - Advance the internal array pointer of an array
- end() - Set the internal pointer of an array to its last element
- each() - Return the current key and value pair from an array and advance the array cursor
- foreach - a construct that provides an easy way to iterate over arrays

Array Iteration: foreach()

- foreach()- the *foreach* construct provides an easy way to iterate over arrays. NOTE: *foreach* works only on arrays and objects (causes an error otherwise).
- Takes two different forms:

  foreach (array_expression as $value)

  statement

  foreach (array_expression as $key => $value)

  statement
- EXAMPLES:

```
$colors = array("red","green","blue","yellow"); //indexed array

foreach ($colors as $value) {
  echo $value . "<br/>";
}


foreach ($_POST as $field => $data) {   //$_POST is an associative array
  echo "In the form submitted: property is " . $field . " that stores " . $data . "<br/>";
}
```

Other Array Related Functions

- func_get_args (): gets an array of the function's argument list.
- print_r() - Prints human-readable information about a variable

Variable Dumping for Debugging

- To quickly display any variable (arrays including), you can create and use a “dump” function:

```
function dump($arg)
{
    echo "<pre>";
    echo (is_array($arg))? print_r($arg): $arg;
    echo "</pre>";
}
```

PHP/PostGreSQL Array Functions

- **pg_fetch_result()** returns the value of a particular row and field (column) in a PostgreSQL result resource.
- **pg_fetch_row()** fetches one row of data from the result associated with the passed result resource.
- **pg_fetch_all()** fetches all rows from a result as an array
- **pg_fetch_array()** returns an array that corresponds to the fetched row (record).
- **pg_fetch_assoc()** returns an associative array that corresponds to the fetched row (records).

Prepared Statements: Simple Login

```
<?php
    //Connect to the database using predefined function
    $dbconn = db_connect();
    $login = $_POST['user_id'];
    $password = $_POST['password'];
    //Prepare a query for execution
    $result = pg_prepare($dbconn, "login_query", 'SELECT * FROM
    users WHERE user_id = $1 AND password = $2');
    //Execute the prepared query. Note that it is not necessary to
    // escape the strings in any way, i.e. they can contain single- and
    //double –quotes, and any special characters
    $result = pg_execute($dbconn, "login_query", array($login,
    $password));
    //After the execute, you can handle the results as you would any
    other
    //result set
?>
```

Prepared Statements: More Efficient

```
<?php
    //Connect to the database using predefined function
    $dbconn = db_connect();
    //Prepare a query for execution
    $result = pg_prepare($dbconn, "login_query", 'SELECT * FROM
    users WHERE user_id = $1 AND password = $2');
    //Execute the prepared query. Note that it is not necessary to
    // escape the strings in any way, i.e. they can contain single- and
    //double –quotes, and any special characters
    $result = pg_execute($dbconn, "login_query ",$_POST);
    //This will work, but you must ensure that the log id occurs in the
    //form before the password, to ensure the array elements are
    //passed in the correct order
?>
```

Prepared Statement: Slick

```
<?php
	//some where else validate the confirm password and remove it from the POSTed from array
	//e.g. unset($_POST['confirm_password']
	unset($_POST['confirm_password']);
	$conn = db_connect();
	$sql_update = "";
	$sql_insert1 = "";
	$sql_insert2 = "";
	$i = 1;
	foreach($_POST as $field=>$data)
	{
			$sql_update .= $field . "=$" . $i . ", ";
			$sql_insert1 .= $field . ", ";
			$sql_insert2 .= "$" . $i . ", ";
			$i++;  //iterate the number
	}
	$sql_insert1 = substr($sql_insert1, 0, (strlen($sql_insert1) - 2)); //remove trailing comma,
	$sql_insert2 = substr($sql_insert2, 0, (strlen($sql_insert2) - 2)); //remove trailing comma,
	$sql_insert = "INSERT INTO users (" . $sql_insert1 . " ) VALUES (" . $sql_insert2 . ")";
	$sql_update = substr($sql_update, 0, (strlen($sql_update) - 2)); //remove trailing comma,
	$sql_update = "UPDATE users SET ". $sql_update ." WHERE user_id = $".$i++;
	echo $sql_update . "<br/>"; //take the outputted Strings and place in db.php as prepared statements
	echo $sql_insert . "<br/>";
?>
```