Unit 2

STRING, ARRAY, MYSQL, DB CONNECTIVITY

Strings in PHP

- Strings are sequences of characters that can be treated as a unit — assigned to variables, given as input to functions, returned from functions, or sent as output to appear on your user's web page.
- The simplest way to specify a string in PHP code is to enclose it in quotation marks, whether single quotation marks (‘) or double quotation marks (“), like this:
  - $my_string = ‘A literal string’;
  - $another_string = “Another string”;

- If you enclose a string in single quotation marks, almost no interpolation will be performed; if you enclose it in double quotation marks, PHP will splice in the values of any variables you include, as well as make substitutions for certain special character sequences that begin with the backslash (\) character.

```
$statement = 'everything I say';
$question_1 = "Do you have to take $statement so literally?\n<BR>";
$question_2 = 'Do you have to take $statement so literally?\n<BR>';
echo $question_1;
echo $question_2;
```

- Interpolation with curly braces
  - $plan1 = "I will play $sport1ball in the summertime"; //wrong
  - $plan1 = "I will play {$sport1}ball in the summertime"; //right

Characters and string indexes

- You can retrieve the individual characters of a string by including the number of the character, starting at 0, enclosed in curly braces immediately following a string variable.
- These characters will actually be one-character strings.

```
$my_string = “Doubled”;
for ($index = 0; $index < 7; $index++) {
    $string_to_print = $my_string{$index};
    print(“$string_to_print$string_to_print”);
}
```

String operators

- PHP offers two string operators: the dot (.) or concatenation operator and the .= concatenating assignment operator.
- The concatenation operator, when placed between two string arguments, produces a new string that is the result of putting the two strings together in sequence. For example:
  - $my_two_cents = “I want to give you a piece of my mind “;
  - $third_cent = “ And another thing”;
  - print($my_two_cents . “…” . $third_cent);
  - How many string are there?

Concatenation and assignment

- PHP has a shorthand operator (.=) that combines concatenation with assignment. The following statement:
  - $my_string_var .= $new_addition;
- is exactly equivalent to:
  - $my_string_var = $my_string_var . $new_addition;

The heredoc syntax

- PHP offers another way to specify a string, called the *heredoc syntax*.
- *This syntax turns out to be extremely useful for specifying large* chunks of variable-interpolated text, because it spares you from the need to escape internal quotation marks.
- It is especially useful in creating pages that contain HTML forms.
- The operator in the heredoc syntax is <<<. What is expected immediately after this is a label (unquoted) that indicates the beginning of a multiline string.
- PHP will continue including subsequent lines in this string until it sees the same label again, beginning a line.
- The ending label may optionally be followed by a semicolon but by nothing else.

```
$my_string_var = <<<EOT
Everything in this rather unnecessarily wordy
ramble of prose will be incorporated into the
string that we are building up inevitably, inexorably,
character by character, line by line, until we reach that
blessed final line which is this one.
EOT;
```

- Interpolation of variables happens exactly the same way as with double-quoted strings. The nice thing about heredoc, though, is that quote signs can be included without any escaping and without prematurely terminating the string. Here's another example:

```
echo <<<ENDOFFORM
<FORM METHOD=POST ACTION="{$_ENV['PHP_SELF']}">
<INPUT TYPE=TEXT NAME=FIRSTNAME VALUE=$firstname>
<INPUT TYPE=SUBMIT NAME=SUBMIT VALUE=SUBMIT>
</FORM>
ENDOFFORM;
```

String Functions

- Inspecting strings:
- strlen(): Using the strlen() function (the name is short for *string length).*

```
$short_string = “This string has 29 characters”;
print(“It does have “ . strlen($short_string) . “ characters”);
for ($index = 0; $index < strlen($short_string); $index++)
print($short_string{$index});
```

Finding characters and substrings

- strpos(): the strpos() function finds the numerical position of a particular character in a string, if it exists.
  - $twister = “Peter Piper picked a peck of pickled peppers”;
  - print(“Location of ‘p’ is “ . strpos($twister, ‘p’) .’
’);
  - print(“Location of ‘q’ is “ . strpos($twister, ‘q’) .’
’);
- the strpos() function is case sensitive.
- The strpos() function can also be used to search for a substring rather than a single character, simply by giving it a multicharacter string rather than a single-character string.
- You can also supply an extra integer argument specifying the position to begin searching forward from.

- Searching in reverse is also possible, using the strrpos() function. Unlike with strpos(), the string searched for must have only one character.
  - $twister = “Peter Piper picked a peck of pickled peppers”;
  - printf(“Location of ‘p’ is “ . strrpos($twister, ‘p’) .’
’);
  - Location of ‘p’ is 40

Comparison and searching

- The simplest method to find an answer of comparision is to use the basic comparison operator (==), which does equality testing on strings as well as numbers.
- The most basic workhorse string-comparison function is strcmp(). It takes two strings as arguments and compares them byte by byte until it finds a difference.
- It returns a negative number if the first string is less than the second and a positive number if the second string is less. It returns 0 if they are identical.
- The strcasecmp() function works the same way, except that the equality comparison is case insensitive. The function call strcasecmp("hey!", "HEY!") should return 0.

Searching

- The strstr() function takes a string to search in and a string to look for (in that order).
- If it succeeds, it returns the portion of the string that starts with (and includes) the first instance of the string it is looking for.
- If the string is not found, a false value is returned
  - $string_to_search = “showsuponceshowsuptwice”;
  - $string_to_find = “up”;
  - print(“Result of looking for $string_to_find” .
  - strstr($string_to_search, $string_to_find) . “
”);

- The strstr() function also has an alias by the name of strchr().
- Other than the name, the two functions are identical. Just as with strcmp(), strstr() has a case insensitive version, by the name of stristr().

Function	Behavior
`strlen()`	Takes a single string argument and returns its length as an integer.
`strpos()`	Takes two string arguments: a string to search, and the string being searched for. Returns the (0-based) position of the beginning of the first instance of the string if found and a false value otherwise. It also takes a third optional integer argument, specifying the position at which the search should begin.
`strrpos()`	Like `strpos()`, except that it searches backward from the end of the string, rather than forward from the beginning. The search string must only be one character long, and there is no optional position argument.
`strcmp()`	Takes two strings as arguments and returns 0 if the strings are exactly equivalent. If `strcmp()` encounters a difference, it returns a negative number if the first different byte is a smaller ASCII value in the first string, and a positive number if the smaller byte is found in the second string.
`strcasecmp()`	Identical to `strcmp()`, except that lowercase and uppercase versions of the same letter compare as equal.
`strstr()`	Searches its first string argument to see if its second string argument is contained in it. Returns the substring of the first string that starts with the first instance of the second argument, if any is found — otherwise, it returns false.
`strchr()`	Identical to `strstr()`.
`stristr()`	Identical to `strstr()` except that the comparison is case independent.

Substring selection

- Many of PHP's string functions have to do with slicing and dicing your strings.
- By *slicing, we mean* choosing a portion of a string;
- By *dicing, we mean selectively modifying a string*.
- *Keep in mind that* (most of the time) even dicing functions do not change the string you started out with. Usually, such functions return a modified copy, leaving the original argument intact.

- The most basic way to choose a portion of a string is the substr() function, which returns a new string that is a subsequence of the old one.
- As arguments, it takes a string (that the substring will be selected from), an integer (the position at which the desired substring starts), and an optional third integer argument that is the length of the desired substring.
- If no third argument is given, the substring is assumed to continue until the end.

- For example, the statement:
- echo(substr(“Take what you need, and leave the rest behind”, 23));
  - prints the string leave the rest behind, whereas the statement:
- echo(substr(“Take what you need, and leave the rest behind”, 5, 13));
  - prints what you need — a 13-character string starting at (0-based) position 5.
- Both the start-position argument and the length argument can be negative, and in each case the negativity has a different meaning.
- If the start position is negative, it means that the starting character is determined by counting backward from the end of the string, rather than forward from the beginning.
- A negative-length argument means that the final character is determined by counting backward from the end rather than forward from the start position.

- Here are some examples, with positive and negative arguments:
- $alphabet_test = “abcdefghijklmnop”;
  - print(“3: “ . substr($alphabet_test, 3) . “
”);
  - print(“-3: “ . substr($alphabet_test, -3) . “
”);
  - print(“3, 5: “ . substr($alphabet_test, 3, 5) . “
”);
  - print(“3, -5: “ . substr($alphabet_test, 3, -5) . “
”);
  - print(“-3, -5: “ . substr($alphabet_test, -3, -5) . “
”);
  - print(“-3, 5: “ . substr($alphabet_test, -3, 5) . “
”);
- This gives us the output:
  - 3: defghijklmnop
  - -3: nop
  - 3, 5: defgh
  - 3, -5: defghijk
  - -3, -5:
  - -3, 5: nop

String cleanup functions

- The functions chop(), ltrim(), and trim() are really used for cleaning up untidy strings. They trim whitespace off the end, the beginning, and the beginning and end, respectively, of their single string argument.
  - $original = “ More than meets the eye “;
  - $chopped = chop($original);
  - $ltrimmed = ltrim($original);
  - $trimmed = trim($original);
  - print(“The original is ‘$original’
”);
  - print(“Its length is “ . strlen($original) . “
”);
  - print(“The chopped version is ‘$chopped’
”);
  - print(“Its length is “ . strlen($chopped) . “
”);
  - print(“The ltrimmed version is ‘$ltrimmed’
”);
  - print(“Its length is “ . strlen($ltrimmed) . “
”);
  - print(“The trimmed version is ‘$ltrimmed’
”);
  - print(“Its length is “ . strlen($trimmed) . “
”);

- In addition to spaces, these functions remove whitespace like that denoted by the escape sequences \n, \r, \t, and \o

String replacement

- The str_replace() function enables you to replace all instances of a particular substring with an alternate string.
- It takes three arguments: the string to be searched for, the string to replace it with when it is found, and the string to perform the replacement on.

```
$first_edition = “Burma is similar to Rhodesia in at least one way.”;
$second_edition = str_replace(“Rhodesia”, “Zimbabwe”, $first_edition);
$third_edition = str_replace(“Burma”, “Myanmar”, $second_edition);
print($third_edition);
```

- This replacement will happen for all instances found of the search string.

- str_replace() picks out portions to replace by matching to a target string; by contrast, substr_replace() chooses a portion to replace by its absolute position.
- The function takes up to four arguments: the string to perform the replacement on, the string to replace it with, the starting position for the replacement, and (optionally) the length of the section to be replaced
- print(substr_replace(“ABCDEFG”, “-“, 2, 3));
- you are allowed to replace a substring with a string of a different length.
- If the length argument is omitted, it is assumed that you want to replace the entire portion of the string after the start position.

- The substr_replace() function also takes negative arguments for starting position and length, which are treated exactly the same way as in the substr() function.
- It is important to remember with both str_replace and substr_replace that the original string remains unchanged by these operations
- The strrev() function simply returns a new string with the characters of its input in reverse order.
- The str_repeat() function takes a string argument and an integer argument and returns a string that is the appropriate number of copies of the string argument tacked together.

Function	Behavior
`substr()`	Returns a subsequence of its initial string argument, as specified by the second (position) argument and optional third (length) argument. The substring starts at the indicated position and continues for as many characters as specified by the length argument or until the end of the string, if there is no length argument. A negative position argument means that the start character is located by counting backward from the end, whereas a negative length argument means that the end of the substring is found by counting back from the end, rather than forward from the start position.
chop(), or rtrim()	Returns its string argument with trailing (right-hand side) whitespace removed. Whitespace is a blank space, `\n`, `\r`, `\t`, and `\0`.
`ltrim()`	Returns its string argument with leading (left-hand side) whitespace removed.
`Trim()`	Returns its string argument with both leading and trailing whitespace removed.
`Str_ replace()`	Used to replace target substrings with another string. Takes three string arguments: a substring to search for, a string to replace it with, and the containing string. Returns a copy of the containing string with *all* instances of the first argument replaced by the second argument.
`Substr_ replace()`	Puts a string argument in place of a position-specified substring. Takes up to four arguments: the string to operate on, the string to replace with, the start position of the substring to replace, and the length of the string segment to be replaced. Returns a copy of the first argument with the replacement string put in place of the specified substring. If the length argument is omitted, the entire tail of the first string argument is replaced. Negative position and length arguments are treated as in `substr()`.

Case functions

- strtolower() The strtolower() function returns an all-lowercase string. It doesn't matter if the original is all uppercase or mixed.

```
<?php
$original = "They DON'T KnoW they're  SHOUTING";
$lower = strtolower($original);
echo $lower;
?>
```

- strtoupper() The strtoupper() function returns an all-uppercase string, regardless of whether the original was all lowercase or mixed:

```
<?php
$original = "make this link stand out";
echo("<B>strtoupper($original)</B>");
?>
```

- ucfirst() The ucfirst() function capitalizes only the first letter of a string:

```
<?php
$original = “polish is a word for which pronunciation depends on capitalization”;
echo(ucfirst($original));
?>
```

- ucwords() The ucwords() function capitalizes the first letter of each word in a string:

```
<?php
$original = “truth or consequences”;
$capitalized = ucwords($original);
echo “While $original is a parlor game, $capitalized is a town in New Mexico.”;
?>
```

Printing and output

- PHP also offers printf() and sprintf(), which are modeled on C functions of the same name.

```
<pre>
<?php
    $value = 3.14159;
    printf("%f,%10f,%-010f,%2.2f\n",
    $value, $value, $value, $value);
?>
</pre>
```

gives us:

3.141590, 3.141590,3.141590000000000, 3.14

Learning Arrays

- PHP arrays can store data of varied types and automatically organize it for you in a large variety of ways.
- An array is a collection of variables indexed and bundled into a single, easily referenced super variable that offers an easy way to pass multiple values between lines of code, functions, and even pages.

What Are PHP Arrays?

- PHP arrays are *associative arrays*. The *associative part means* that arrays store element values in association with key values rather than in a strict linear index order.
- For example, storage is as simple as this:
  - $state_location[‘San Mateo’] = ‘California’;
- which stores the element ‘California’ in the array variable $state_location, in association with the lookup key ‘San Mateo’. After this has been stored, you can look up the stored value by using the key, like so:
  - $state = $state_location[‘San Mateo’];

- Similarly, if you want to associate a numerical ordering with a bunch of values, all you have to do is use integers as your key values, as in:
  - $my_array[1] = “The first thing”;
  - $my_array[2] = “The second thing”; // and so on …

Creating Arrays

- There are three main ways to create an array in a PHP script:
  - by assigning a value into one (and thereby implicitly creating it),
  - by using the array() construct, and
  - by calling a function that happens to return an array as its value.
- **Direct assignment:** The simplest way to create an array is to act as though a variable is already an array and assign a value into it, like this:

```
$my_array[1] = “The first thing in my array that I just made”;
```

- **The array() construct:** It creates a new array from the specification of its elements and associated keys.
- In its simplest version, array() is called with no arguments, which creates a new empty array.
- In its next simplest version, array() takes a comma separated list of elements to be stored, without any specification of keys.
- The result is that the elements are stored in the array in the order specified and are assigned integer keys beginning with zero.
  - $fruit_basket = array(‘apple’, ‘orange’, ‘banana’, ‘pear’);

- causes the variable $fruit_basket to be assigned to an array with four string elements, with the indices 0, 1, 2, and 3, respectively.
- The same effect could also have been accomplished by omitting the indices in the assignment, like so:
  - $fruit_basket[] = ‘apple’;
  - $fruit_basket[] = ‘orange’;
- Specifying indices using array(): array() offers us a special syntax for specifying what the indices should be. Instead of element values separated by commas, you supply key/value pairs separated by commas, where the key and value are separated by the special symbol =>.
- Consider the following statement:
  - $fruit_basket = array(0 => ‘apple’, 1 => ‘orange’, 2 => ‘banana’, 3 => ‘pear’);

- Or it can be
  - $fruit_basket = array('red' => 'apple', 'orange' => 'orange', 'yellow' => 'banana', 'green' => 'pear');
- To recover the name of the yellow fruit, for example, we just evaluate the expression:
  - $fruit_basket['yellow'] // will be equal to 'banana'
- you can create an empty array by calling the array function with no arguments. For example:
  - $my_empty_array = array();
- creates an array with no elements.

- **Functions returning arrays:** This may be a user defined function, or it may be a built-in function that makes an array via methods internal to PHP.
- Many database-interaction functions, for example, return their results in arrays that the functions create on the fly.
- Other functions exist simply to create arrays that are handy to have as grist for later array-manipulating functions.
- One such is range(), which takes two integers as arguments and returns an array filled with all the integers (inclusive) between the arguments. In other words:
  - $my_array = range(1,5);
  - is equivalent to:
  - $my_array = array(1, 2, 3, 4, 5);

Retrieving Values

- **Retrieving by index:** The most direct way to retrieve a value is to use its index. If we have stored a value in $my_array at index 5, $my_array[5] should evaluate to the stored value.
- **The list() construct:** It used to assign several array elements to variables in succession. Suppose that the following two statements are executed:
  - $fruit_basket = array('apple', 'orange', 'banana');
  - list($red_fruit, $orange_fruit) = $fruit_basket;
- This will assign the string 'apple' to the variable $red_fruit and the string 'orange' to the variable $orange_fruit.
- The variables in list() will be assigned to elements of the array in the order they were originally stored in the array.

Multidimensional Arrays

- Facilitates storing arrays of arrays!
- Extends beyond the associative array to store multiple sets of associative arrays

```
<?php
   $person = array(
       array("name" => "John", "age" => 19),
       array("name" => "Mary", "age" => 30),
       array("name" => "Aine", "age" => 23)
       );
?>
```

- each of the array elements store starting at an index value of 0
- for instance:
  - echo $person[0];
  - echo $person[1];
  - and so on and on and on!
- foreach ($person as $p) {

```
while (list($n, $a) = each ($p)) {
        echo “Name: “.$n.”\n Age: ”.$a.”<br/>”;
        }
}
```

- Output:

  Name: John

  Age: 19

  Name: Mary

  Age: 30

  Name: Aine

  Age: 23

Inspecting Arrays

Function	Behavior
`is_array()`	Takes a single argument of any type and returns a true value if the argument is an array, and false otherwise.
`count()`	Takes an array as argument and returns the number of nonempty elements in the array. (This will be 1 for strings and numbers.)
`sizeof()`	Identical to `count()`.
`in_array()`	Takes two arguments: an element (that might be a value in an array), and an array (that might contain the element). Returns `true` if the element is contained as a value in the array, `false` otherwise. (Note that this does not test for the presence of keys in the array.)
`isset($array[$key])`	Takes an *array*`[`*key*`]` form and returns `true` if the key portion is a valid key for the array. (This is a specific use of the more general function `isset()`, which tests whether a variable is bound.)

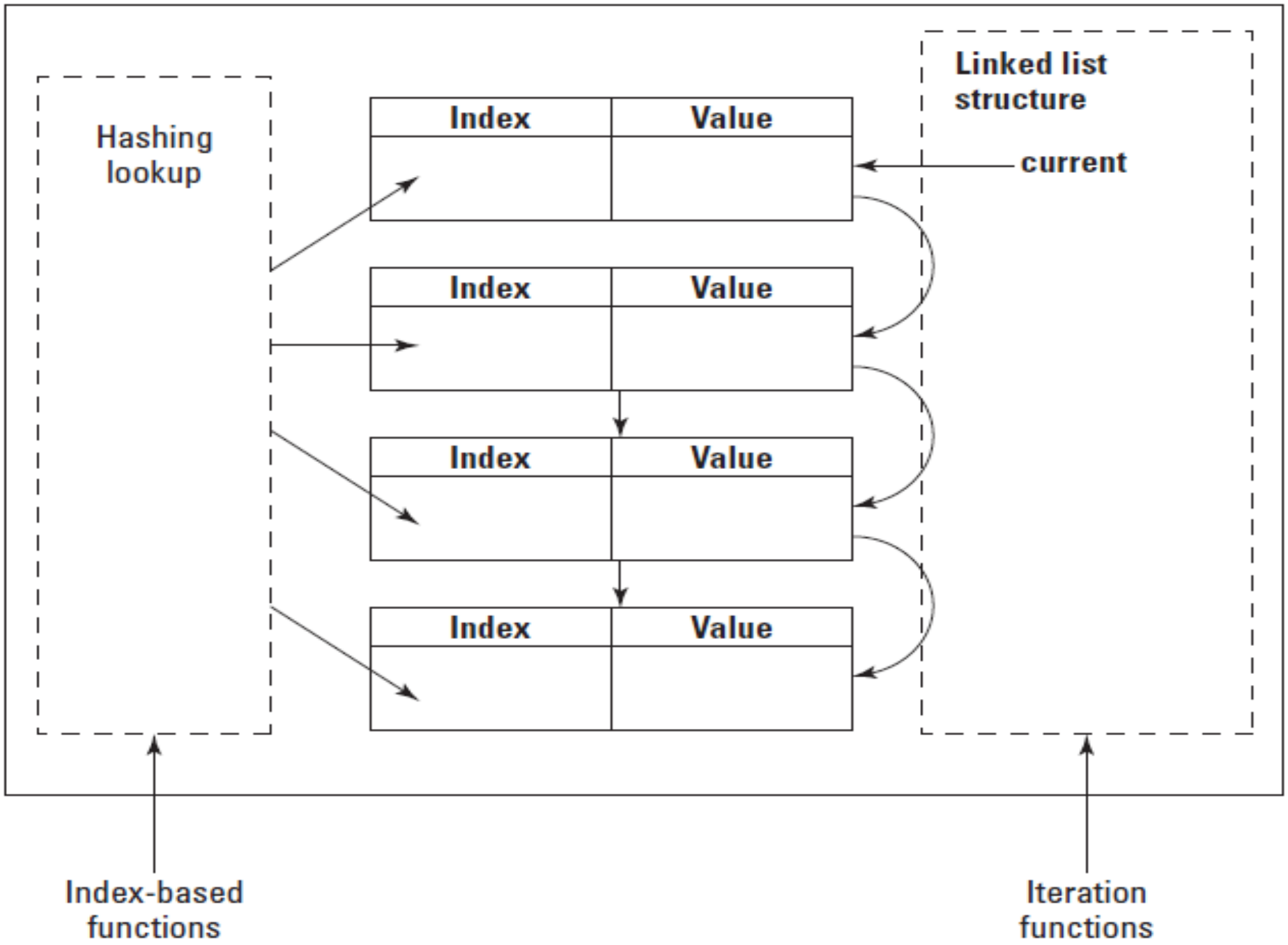
Deleting from Arrays

- Deleting an element from an array is simple, exactly analogous to getting rid of an assigned variable.
- Just call unset(), as in the following:
  - $my_array[0] = 'wanted';
  - $my_array[1] = 'unwanted';
  - $my_array[2] = 'wanted again';
  - unset($my_array[1]);
- Note that this is *not the same as setting the contents to an empty value. If, instead of calling* unset(), we had the following statement:
  - $my_array[1] = ' ';
- at the end we would have three stored values ('wanted', '', 'wanted again') in association with three keys (0, 1, and 2, respectively).

Iteration

- Iteration constructs help us do this by letting us step or loop through arrays, element by element or key by key.
- In addition to storing values in association with their keys, PHP arrays silently build an ordered list of the key/value pairs that are stored, in the order that they are stored.
- The reason for this is to support operations that iterate over the entire contents of an array.
- Each stored key/value pair points to the next one, and one side effect of adding the first element to an array is that a current pointer points to the very first element, where it will stay unless disturbed by one of the iteration functions

Internal structure of an array



foreach method

```
foreach ($array_variable as $value_variable) {
   // .. do something with the value in $value_variable
}


foreach ($array_variable as $key_var => $value_var)
   {
   // .. do something with $key_var and/or $value_var
}
```

Iterating with current() and next()

- foreach, is really only good for situations where you want to simply loop through an array's values.
- For more control, let's look at current() and next().
- The current() function returns the stored value that the current pointer points to.
- When an array is newly created with elements, the element pointed to will always be the first element.
- The next() function first advances that pointer and then returns the current value pointed to.
- If the next() function is called when the current pointer is already pointing to the last stored value and, therefore, runs off the end of the array, the function returns a false value.

- **Starting over with reset()**
  - The reset() function gives us a way to “rewind” that pointer to the beginning — it sets the pointer to the first key/value pair and then returns the stored value.
  - We can use it to make our printing function more robust by replacing the call to current() with a call to reset().
- **Reverse order with end() and prev():**
  - There are also the functions prev(), which moves the pointer back by one, and end(), which jumps the pointer to the last entry in the list.

- **Extracting keys with key():**
  - The keys are also retrievable from the internal linked list of an array by using the key() function.
  - $current_key = key($city_array);
- **Walking with array_walk():**
  - It lets you pass an arbitrary function of your own design over an array, doing whatever your function pleases with each key/value pair.
  - The array_walk() function takes two arguments: an array to be traversed and the name of a function to apply to each key/value pair.

```
function print_value_length($array_value,  array_key_ignored)
{
    $the_length = strlen($array_value);
    print("The length of $array_value is $the_length<BR>");
}
array_walk($major_city_info, 'print_value_length');
```

Learning PHP Number Handling

- Numerical Types: PHP has only two numerical types: *integer (also known as long), and double* (aka *float).*
- PHP does automatic conversion of numerical types.
- In situations where you want a value to be interpreted as a particular numerical type, you can force a typecast by prepending the type in parentheses, such as:
  - (double) $my_var
  - (integer) $my_var
- Or you can use the functions intval() and doubleval(), which convert their arguments to integers and doubles, respectively.

Arithmetic operators

Operator	Behavior	Examples
+	Sum of its two arguments.	4 + 9.5 evaluates to 13.5
-	If there are two arguments, the right-hand argument is subtracted from the left-hand argument. If there is just a right-hand argument, then the negative of that argument is returned.	50 - 75 evaluates to -25 - 3.9 evaluates to -3.9
*	Product of its two arguments.	3.14 * 2 evaluates to 6.28
/	Floating-point division of the left-hand argument by the right-hand argument.	5 / 2 evaluates to 2.5
%	Integer remainder from division of left-hand argument by the absolute value of the right-hand argument. (See discussion in the following section.)	101 % 50 evaluates to 1 999 % 3 evaluates to 0 43 % 94 evaluates to 43 -12 % 10 evaluates to -2 -12 % -10 evaluates to -2

- Incrementing or decrementing operators:

```
$count = 0;
$result = $count++;
print("Post ++: count is $count, result is $result<BR>");
$count = 0;
$result = ++$count;
print("Pre ++: count is $count, result is $result<BR>");
$count = 0;
$result = $count--;
print("Post --: count is $count, result is $result<BR>");
$count = 0;
$result = --$count;
print("Pre --: count is $count, result is $result<BR>");
```

- **Assignment operators:**
  - =, +=, -=,*=,/=
- **Precedence and parentheses:**
  - Arithmetic operators have higher precedence (that is, bind more tightly) than comparison operators.
  - Comparison operators have higher precedence than assignment operators.
  - The *, /, and % arithmetic operators have the same precedence.
  - The + and – arithmetic operators have the same precedence.
  - The *, /, and % operators have higher precedence than + and –.
  - When arithmetic operators are of the same precedence, associativity is from left to right (that is, a number will associate with an operator to its left in preference to the operator on its right).

Comparison operators

- The < (less than) operator is true if its left-hand argument is strictly less than its right-hand argument but false otherwise.
- The > (greater than) operator is true if its left-hand argument is strictly greater than its right-hand argument but false otherwise.
- The <= (less than or equal) operator is true if its left-hand argument is less than or equal to its right-hand argument but false otherwise.
- The >= (greater than or equal) operator is true if its left-hand argument is greater than or equal to its right-hand argument but false otherwise.
- The == (equal to) operator is true if its arguments are exactly equal but false otherwise.
- The != (not equal) operator is false if its arguments are exactly equal and true otherwise. This operator is the same as <>.
- The === operator (identical to) is true if its two arguments are exactly equal and of the same type.
- The !== operator (not identical to) is true if the two arguments are not equal or not of the same type.

Simple Mathematical Functions

Function	Behavior
`floor()`	Takes a single argument (typically a double) and returns the largest integer that is less than or equal to that argument.
`ceil()`	Short for ceiling — takes a single argument (typically a double) and returns the smallest integer that is greater than or equal to that argument.
`round()`	Takes a single argument (typically a double) and returns the nearest integer. If the fractional part is exactly 0.5, it returns the nearest even number.
`abs()`	Short for absolute value — if the single numerical argument is negative, the corresponding positive number is returned; if the argument is positive, the argument itself is returned.
`min()`	Takes any number of numerical arguments (but at least one) and returns the smallest of the arguments.
`max()`	Takes any number of numerical arguments (but at least one) and returns the largest of the arguments.

Randomness

- There are two random number generators (invoked with rand() and mt_rand(), respectively)

```
$length = strlen($string);
$position = mt_rand(0, $length - 1);
return($string[$position]);
```

Randomness

Function	Behavior
`srand()`	Takes a single positive integer argument and seeds the random number generator with it.
`rand()`	If called with no arguments, returns a "random" number between 0 and `RAND_MAX` (which can be retrieved with the function `getrandmax()`). The function can also be called with two integer arguments to restrict the range of the number returned — the first argument is the minimum and the second is the maximum (inclusive).
`getrandmax()`	Returns the largest number that may be returned by `rand()`. This number is limited to 32768 on Windows platforms.
`mt_srand()`	Like `srand()`, except that it seeds the "better" random number generator.
`mt_rand()`	Like `rand()`, except that it uses the "better" random number generator.
`mt_getrandmax()`	Returns the largest number that may be returned by `mt_rand()`.

Unit 2

MySQL Database Integration

What Is a Database?

- *A database is a collection of data. The term database usually indicates that* the collection of data is stored on a computer.
- Databases implemented through a computer are created within software.
- That software, commonly known as a database application, controls how the actual data is stored nd retrieved.
- Some database applications include Microsoft Access and OpenOffice.org's Base. Sometimes, databases are stored in a central location and managed by a database server. A database server is a database application built with multiple users in mind.

- Most of the time when programming PHP you'll be accessing a database server. Some database servers include PostgreSQL, MySQL, Microsoft's SQL Server, and the Oracle suite of databases.
- Database servers usually have one or more distinct APIs for programmatically creating, accessing, managing, searching, and replicating the data they hold.
- It is through the API that you connect to and work with data stored in database servers when using PHP

Why a Database?

- Maintainability and scalability
- Portability
- Avoiding awkward programming
- Searching

PHP-Supported Databases

- Popular Databases for PHP Web Application Development
  - *MySQL*
  - *PostGreSQL*
  - *SYBASE*
  - *IBM-DB2*
  - *Oracle Database*

Other Supported DB	
Cubrid	MaxDB
DB++	Mongo
dBase	mSQL
filePro	Ovrimos SQL
FireBird/InterBase	Paradox
FrontBase	SQLite
Informix	SQLite3
Ingres	SQLSRV
	Tokyo Tyrant

Our Focus: MySQL

- MySQL, (officially pronounced my- S - Q - L and not "mysequel"), is an incredibly popular and powerful RDBMS.
- MySQL provides one of the letters in the ubiquitous acronym "LAMP," which is an abbreviation for Linux, Apache, MySQL, PHP/Perl/Python. MySQL has become so popular for several reasons.
  - First, MySQL is free (as in price), although the licensing has changed (discussed later).
  - Second, MySQL is also stable, meaning that it's not prone to crashing even under load.
  - Third, MySQL is lightweight, meaning that it doesn't require many resources to install or run.
  - Fourth, MySQL is fast and easy to use.
  - Finally, MySQL is powerful, with all of the features required for web applications.

Relational Databases and SQL

- SQL is the language of relational databases.
- After you learn SQL, you will be able to interact with numerous databases across all platforms.
- SQL is a standard under both the American National Standards Institute (ANSI) and the Equipment Managers Council of America (ECMA).

The Workhorses of SQL

- The basic logical structure of a SQL database is very simple. A given SQL installation can usually contain multiple databases — for instance, one for customer data and one for product data.
- Each database contains a number of tables.
- Each table is made up of carefully defined columns, and every entry can be thought of as an added record or row.
- Four basic four commands of the database are SELECT, INSERT, UPDATE, and DELETE.
- These four SQL statements is that they manipulate only database *values,* not the structure of the database itself.
- In other words, you can use these commands to add data but not to make a database; you can get rid of every piece of data in a database, but the shell will still be there

- To make up new databases, you need to use other commands such as DROP, ALTER, and CREATE.

Database design

- Select, Insert, Update, Delete
- Create, Drop, alter,

Privileges and Security

- Setting database permissions:
  - The most fundamental rule of database use is to give each user or group only the minimum permissions necessary to do what needs to be done.
  - A typical database permissions package might be something like:
    - Web visitor: SELECT only
    - Contributor: SELECT, INSERT, and maybe UPDATE
    - Editor: SELECT, INSERT, UPDATE, and maybe DELETE and maybe GRANT
    - Database Administrator: SELECT, INSERT, UPDATE, DELETE, GRANT, and DROP
- DROP in particular is the nuclear bomb of SQL because it allows you to blow away an entire table or database with a single command.
- In many databases, including MySQL, passwords are encrypted using a different algorithm from system passwords

- **Database usernames and passwords should not be identical to system usernames and passwords.**
- Keep database passwords outside the web area.
- It's a good idea to separate passwords from the web pages that use them.
- With PHP's include()/ include_once() and require()/require_once() functions, it's very easy to drop in text from another file at runtime.

- A good example is a directory above or outside of your web document root or in a home directory.
- Taking the database variables out of PHP files is also good for other reasons. If you have many PHP scripts using the same database, they can all use the same password file.
- When you suspect the password has been compromised, or when you change the password on a regular schedule, you need only alter one script for all the files to be updated.
- Learn to make backups: The biggest part of database security may be backing up.
- Take an hour to learn the best way to back up data in your particular database (for example, via the mysqldump command in MySQL), and then schedule regular backups right away.
- Even better, with a little foresight you can also set up an automatic database backup schedule.

Integrating PHP and MySQL

- The basic command to initiate a MySQL connection is
  - mysql_connect($hostname, $user, $password);

  if you're using variables, or
  - mysql_connect('localhost', 'root', 'sesame');

  if you're using literal strings.
- The password is optional, depending on whether this particular database user requires one (it's a good idea). If not, just leave that variable off.
- The corresponding mysqli function is mysqli_connect, which adds a fourth parameter allowing you to select a database in the same function you use to connect.

- PHP offers two different ways to connect to MySQL server: **MySQLi** (Improved MySQL) and **PDO** (PHP Data Objects) extensions.
- While the PDO extension is more portable and supports more than twelve different databases, MySQLi extension as the name suggests supports MySQL database only.
- MySQLi extension however provides an easier way to connect to, and execute queries on, a MySQL database server.
- Both PDO and MySQLi offer an object-oriented API, but MySQLi also offers a procedural API which is relatively easy for beginners to understand.

Connecting to MySQL Database Server

- In PHP you can easily do this using the mysqli_connect() function. All communication between PHP and the MySQL database server takes place through this connection.
- $link = mysqli_connect("hostname", "username", "password", "database");
- $mysqli = new mysqli("hostname", "username", "password", "database");
- $pdo = new PDO("mysql:host=hostname;dbname=database", "username", "password");
- Example
- The default username for MySQL database server is root and there is no password.

- The connection to the MySQL database server will be closed automatically as soon as the execution of the script ends. However, if you want to close it earlier you can do this by simply calling the PHP mysqli_close() function.
- mysqli_close($link);

Making MySQL Queries

- A database query from PHP is basically a MySQL command wrapped up in a tiny PHP function called mysqli_query().
- This is where you use the basic SQL workhorses of SELECT, INSERT, UPDATE, and DELETE.
- The function mysqli_query takes as arguments a link identifier and the query string.
- It returns a TRUE (nonzero) integer value if the query was executed successfully *even if no rows were affected.*
- It returns a FALSE integer if the query was illegal or not properly executed for some other reason.

- If your query was an INSERT, UPDATE, DELETE, CREATE TABLE, or DROP TABLE and returned TRUE, you can now use mysql_affected_rows to see how many rows were changed by the query.
- This function optionally takes a link identifier, which is only necessary if you are using multiple connections.
- It *does not* take the result handle as an argument! You call the function like this, without a result handle:
- $affected_rows = mysqli_affected_rows();
- If your query was a SELECT statement, you can use mysqli_num_rows($result) to find out how many rows were returned by a successful SELECT.

- Create DB & Table
- Insert / Multiple Insert
- Insert using Form

Fetching Data Sets

- It would be logical to assume that the result of a query would be the desired data, but that is not correct.
- The result of a PHP query is an integer representing the success or failure or identity of the query.
- What actually happens is that a mysqli_query() command pulls the data out of the database and sends a receipt back to PHP reporting on the status of the operation.
- At this point, the data exists in a purgatory that is immediately accessible from neither MySQL nor PHP — you can think of it as a staging area of sorts.
- The data is there, but it's waiting for the commanding officer to give the order to deploy.
- It requires one of the mysqli_fetch functions to make the data fully available to PHP.

- The fetching functions are as follows:
  - mysqli_fetch_row: Returns row as an enumerated array
  - mysqli_fetch_object: Returns row as an object
  - mysqli_fetch_array: Returns row as an associative array
  - mysqli_result: Returns one cell of data
- The most general one is mysqli_fetch_row, which can be used something like this:

```
$query = "SELECT ID, LastName, FirstName
FROM users WHERE Status = 1";
$result = mysqli_query($query);
while ($name_row = mysqli_fetch_row($result)) {
print("{$name_row[0]} {$name_row[1]}
  {$name_row[2]}<BR>\n");
}
```

- This code will output the specified rows from the database, each line containing one row

- The most useful fetching function, mysqli_fetch_array, offers the choice of results as an associative or an enumerated array — or both, which is the default.
- This means you can refer to outputs by database field name rather than number:

```
$query = "SELECT ID, LastName, FirstName FROM users
   WHERE Status = 1";
$result = mysql_query($query);
while ($row = mysqli_fetch_array($result)) {
   echo "{$row['ID']}, {$row['LastName']},
   {$row['FirstName']}<BR>\n";
}
```

Getting Data about Data

- PHP offers extensive built-in functions to help you learn the name of the table in which your data resides, the data type handled by a particular column, or the number of the row into which you have just inserted data.
- With these functions, you can effectively work with a database about which you know very little.
- The MySQL metadata functions fall into two major categories:
  - Functions that return information about t ■■ he previous operation only
  - Functions that return information about the database structure in general

- A very commonly used example of the first type is mysqli_insert_id(), which returns the auto incremented ID assigned to a row of data you just inserted.
- A commonly used example of the second type is mysqli_field_type(), which reveals whether a particular database field's data must be an integer, a varchar, text, or what have you.
  - Rather than returning the MySQL type, it returns the PHP data type.

Multiple Databases

Building in Error Checking

- The main error-checking function is actually called die().
- die() is not a MySQL-specific function — the PHP manual lists it in "Miscellaneous Functions."
- It simply terminates the script (or a delimited portion thereof) and returns a string of your choice.

```
mysql_query("SELECT * FROM mutual_funds
WHERE code = '$searchstring'")
or die("Please check your query and try again.");
```

- Other built-in means of error-checking are error messages. These are particularly helpful during the development and debugging phase, and they can be easily commented out in the final edit before going live on a production server.
- MySQL error messages no longer appear by default. If you want them, you have to ask for them by using the functions mysql_errno() (which returns a code number for each error type) or mysql_error() (which returns the text message).
- Then you can send them to a custom error log by using the error_log() function:

```
if (!mysql_select_db($bad_db)) {
print(mysql_error());
}
```

An example of good error checking is:

```
function printError($errorMesg) {
    printf("<B>%s </B><BR>\n", $errorMesg);
}
function notify($errorMesg) {
    mail(webmaster@example.com,"An Error has occurred at
    example.com", $errorMesg)
}
if ($link = mysql_connect("host","user","pass")) {
    // Things to do if the connection is successful
} else {
    printError("Sorry for the inconvenience; but we are unable
    to process your request at this time. Please check back
    later");
    notify("Problem connecting to database in $SCRIPT_NAME at
    line 12 on date('Y-m-D')");
}
```

MySQL data types

HTML Tables and Database Tables

- the function to display the contents of a couple of tables
  - HTML Tables
- Displaying column headers
- Error checking
- Complex queries, sub queries and joins.
- Creating the sample tables
  - Insert queries

Integrating Web Forms and Databases

- There are a few PHP-specific points to brush up on:
  - You must use extra caution when using any data that comes from a visitor's web browser. Never use unfiltered data in a database query.
  - Always, always, *always use a NAME for every data entry element* (INPUT, SELECT, TEXTAREA, and so on). These NAME attributes will become PHP variable names — you will not be able to access your values if you do not use a NAME attribute for each one.
  - A form field NAME does not need to be the same as the corresponding database field name.
  - The VALUE can be set to data you wish to display in the form.
  - Remember that you can pass hidden variables from form to form (or page), using the HIDDEN data entry elements. This practice has negative security implications, so don't use it to store sensitive data and always validate the data you receive in a HIDDEN element.

Self-Submission

- Self-submission refers to the process of combining one or more forms and form handlers in a single script, using the HTML FORM standard to submit data to the script one or more times.
- Another situation in which self-submission is a win occurs when you need to submit the same form more than once. Say that you are applying for auto insurance online, and you need to give the particulars of three or four different cars.
- The single most important thing to remember about self-submitting forms is: *The logic comes before the display.*

- To prepare your HTML forms to work smoothly with PHP, you need to follow a few simple rules.
- Never use data that comes from the user directly in a database call or query. This means using the mysqli_real_escape_string() function on any $_POST, $_GET, and $_COOKIE values.
- Remember always to name every single form element