What is PHP?

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use.
- PHP code are executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php"
- PHP can generate dynamic page content.
- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP is easy to learn and runs efficiently on the server side

---

1. The History of PHP

- PHP is an "HTML-embedded scripting language" primarily used for dynamic Web applications. The first part of this definition means that PHP code can be interspersed with HTML, making it simple to generate dynamic pieces of Web pages on the fly.
- PHP takes most of its syntax from C, Java, and Perl. It is an open source technology and runs on most operating systems and with most Web servers.
- PHP was written in the C programming language by Rasmus Lerdorf in 1994 for use in monitoring his online resume and related personal information.
- For this reason, PHP originally stood for "Personal Home Page". Lerdorf combined PHP with his own Form Interpreter, releasing the combination publicly as PHP/FI (generally referred to as PHP 2.0) on June 8, 1995.
- Two programmers, Zeev Suraski and Andi Gutmans, rebuilt PHP's core, releasing the updated result as PHP/FI 2 in 1997. The acronym was formally changed to PHP: HyperText Preprocessor, at this time. (This is an example of a recursive acronym: where the acronym itself is in its own definition.)
- In 1998, PHP 3 was released, which was the first widely used version. PHP 4 was released in May 2000, with a new core, known as the Zend Engine 1.0. PHP 4 featured improved speed and reliability over PHP 3. In terms of features, PHP 4 added references, the Boolean type, COM support on Windows, output buffering, many new array functions, expanded object-oriented programming, inclusion of the PCRE library, and more. Maintenance releases of PHP 4 are still available, primarily for security updates.
- PHP 5,Although previous major releases had enormous numbers of new library additions, version 5 contained improvements over existing functionality and added several features commonly associated with mature programming language architectures like vastly improved object-oriented capabilities,

Try/Catch exception handling, Improved XML and web services support etc.. The enhanced object-oriented capabilities introduced in PHP 5 resulted in an additional boost for the language.

- PHP 5.3, Although officially a point release, PHP 5.3 is actually the most significant upgrade to the language since the release of 5.0. Heralding a powerful array of new features including namespaces, late static binding, lambda functions and closures, a new MySQL driver, version 5.3 represents a serious step forward in PHP's evolution.
- PHP 6, A new major version of PHP known as PHP 6 has been concurrently developed alongside PHP 5.X for several years, with the primary goal of adding Unicode support to the language. However, in March, 2010 the development team decided to primarily focus on the 5.X series of releases. In fact, several features originally slated for PHP 6 have been integrated into 5.X releases. Although PHP 6 beta releases had previously been made available at **http://snaps.php.net**, at the time of this writing it appears as if those releases have been removed from the PHP website.

2. General Language Features:

Features of PHP

2.1 Practicality
2.2 Power
2.3 Possibility
2.4 Price

2.1 Practicality

From the very start, the PHP language was created with ***practicality*** in mind. The result is a language that allows the user to build powerful applications even with a minimum of knowledge.

For instance, a useful PHP script can consist of as little as one line; unlike C, there is no need for the mandatory inclusion of libraries. For example, the following represents a complete PHP script, the purpose of which is to output the current date, in this case one formatted like **September 23, 2007**:

<?php echo date("F j, Y"); ?>.

Another example of the language's penchant for compactness is its ability to nest functions. For instance, you can effect numerous changes

to a value on the same line by stacking functions in a particular order. The following example produces a string of five alphanumeric characters such as **a3jh8**:

```
$randomString = substr(md5(microtime()), 0, 5);
```

PHP is a *loosely typed* language, meaning there is no need to explicitly create, typecast, or destroy a variable, although you are not prevented from doing so. PHP handles such matters internally, creating variables on the fly as they are called in a script, and employing a best-guess formula for automatically typecasting variables. For instance, PHP considers the following set of statements to be perfectly valid:

```
<?php
$number = "5"; // $number is a string
$sum = 15 + $number; // Add an integer and string to produce integer
$sum = "twenty"; // Overwrite $sum with a string.
?>
```

2.2 Power

PHP's has ability to interface with databases, manipulate form information, and create pages dynamically, besides these PHP can also do the following

- Create and manipulate Adobe Flash and Portable Document Format (PDF) files.
- Evaluate a password for guess ability by comparing it to language dictionaries and easily broken patterns.
- Parse even the most complex of strings using the POSIX and Perl-based regular expression libraries.
- Authenticate users against login credentials stored in flat files, databases, and even Microsoft's Active Directory.
-

2.3 Possibility

PHP developers are rarely bound to any single implementation solution. On the contrary, a user is typically fraught with choices offered by the language.

For example, consider PHP's array of database support options. Native support is offered for more than 25 database products, including Adabas D, dBase, Empress, FilePro, FrontBase, Hyperwave, IBM DB2, Informix, Ingres, InterBase, mSQL, Microsoft SQL Server, MySQL, Oracle, Ovrimos, PostgreSQL, Solid, Sybase, Unix dbm, and Velocis

PHP's flexible string-parsing capabilities offer users of differing skill sets the opportunity to not only immediately begin performing complex string operations but also to quickly port programs of similar functionality (such as Perl and Python) over to PHP. In addition to almost 100 string-manipulation functions, Perl-based regular expression formats are supported.

PHP offers support for both procedural and Object oriented Programming

2.4 Price

PHP is available free of charge. i.e. PHP is a Open source software. In recent years, software meeting such open licensing qualifications has been referred to as *open source software*. Open source software and the Internet go together like bread and butter.

3. PHP BASICS

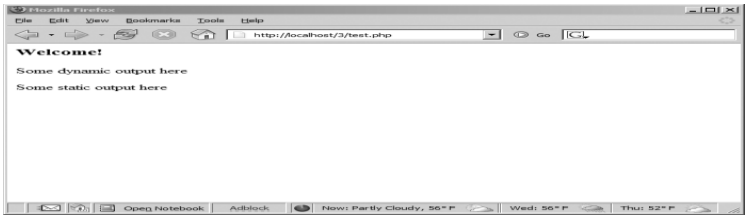
- **3.1 Code embedding web pages**
- **3.2 Commenting the Code**
- **3.3 Output Data to Browser**
- **3.4 Data Types**
- **3.5 Identifiers**
- **3.6 Variables**
- **3.7 Constants**
- **3.8 Expressions**
- **3.9 String Interpolation**
- **3.10 Arrays**

3.1 Code Embedding Web Pages

- One of PHP's advantages is that you can embed PHP code directly alongside HTML.
- The web server will send only those pages that have .php extension to the PHP Engine .
- It is highly inefficient for the engine to consider every line as a potential PHP command.
- Therefore, the engine needs some means to immediately determine which areas of the page are PHP-enabled.
- This is logically accomplished by delimiting the PHP code.
- There are four delimitation variants.
  - 3.1.1 Default Syntax
  - 3.1.2 Short –Tags
  - 3.1.3 Script
  - 3.1.4 ASP Style

3.1.1 Default syntax

The default delimiter syntax opens with <?php and concludes with ?>, like this:



3.1.2 Short-Tags

- For less motivated typists, an even shorter delimiter syntax is available. Known as *short-tags*.
- However, to use this feature, you need to enable PHP's short_open_tag directive.
- An example follows:

```
<?
print "This is another PHP example.";
?>
```

- When short-tags syntax is enabled and you want to quickly escape to and from PHP to output a bit of dynamic text, you can omit these statements using an output variation known as ***short-circuit syntax***

```
<?="This is another PHP example.";?>
```

This is functionally equivalent to both of the following variations:

```
<? echo "This is another PHP example."; ?>
<?php echo "This is another PHP example.";?>
```

3.1.3 Script

Certain Editors have problems dealing with PHP's more commonly used escape syntax variants. To Solve such issues, another delimiter variant, <script>, is offered.

```
<script language="php">
        print "This is another PHP example.";
</script>
```

3.1.4 ASP Style

- Microsoft ASP pages employ a delimiting strategy similar to that used by PHP.
- ASP Style includes opening dynamic syntax with <%, and concluding with %>.
- If you're coming from an ASP background and prefer to continue using this escape syntax, PHP supports it. Here's an example:

```
<%
        print "This is another PHP example.";
%>
```

- The ASP Style and Script delimiting variants are rarely used and should be avoided unless you have ample reason for doing so. **Caution** ASP Style syntax is no longer available as of PHP 5.3.

Embedding Multiple Code Blocks

You can escape to and from PHP as many times as required within a given page. For instance, the following example is perfectly acceptable:

```
<html>
    <head>
        <title><?php echo "Welcome to my web site!";?></title>
    </head>
    <body>
        <?php
            $date = "July 26, 2010";
        ?>
        <p>Today's date is <?=$date;?></p>
    </body>
</html>
```

3.2 Commenting Your Code

- The main purpose of comments is to serve as a note to the developer and the maintenance people
- A comment in PHP code is a line that is not read/executed as part of the program. Its only purpose is to be read by someone who is looking at the code.
- Comments can be used to
    - Let others understand what you are doing
    - Remind yourself of what you did.
- While there is only one type of comment in HTML, PHP has three types.
    1. Single-Line C++ Syntax
    2. Shell syntax
    3. Multiple-Line C Syntax

3.2.1 Single-Line C++ Syntax

The single line comment tells the interpreter to ignore everything that occurs on that line to the right of the comment. PHP supports C++ single-line comment syntax, which is prefaced with a double slash (//), like this:

```
<?php
// Title: My first PHP script
// Author: Jason Gilmore
echo "This is a PHP program.";
?>
```

3.2.2 Shell syntax

PHP also supports an alternative to the C++-style single-line syntax, known as *shell syntax*, which is prefaced with a hash mark (#).

```
<?php
# Title: My first PHP script
# Author: Jason Gilmore
echo "This is a PHP program.";
?>
```

3.2.3 Multiple-Line C Syntax

The multi-line PHP comment can be used to comment out large blocks of code or writing multiple line comments. The multiple line PHP comment begins with " /* " and ends with " */ ".

```
<?php
/* This Echo statement will print out my message to the
the place in which I reside on.  In other words, the World. */
echo "Hello World!";
/* ec
ho "My name is Humperdinkle!";
echo "No way! My name is Uber PHP Programmer!";
*/
?>
```

3.3 Output Data to Browser

PHP offers several methods for dynamic web sites to output data to the browser

3.3.1 The print() Statement
3.3.2 The echo() Statement
3.3.3 The printf() Statement
3.3.4 The sprintf() Statement

3.3.1 The print() Statement

- The print() statement outputs data passed to it . Its prototype looks like this:

int print(argument)

```
<?php
print("<p>I love the summertime.</p>");
?>
<?php
$season = "summertime";
print "<p>I love the $season.</p>";
?>
<?php
print "<p>I love the
summertime.</p>";
?>
```

- All these statements produce identical output:

I love the summertime

- The print() statement's return value is misleading because it will always return 1 regardless of Outcome

3.3.2 The echo() Statement

- Alternatively, use the echo() statement for the same purposes as print().
- **void echo(string argument1 [, ...string argumentN])**
- To use echo(), just provide it with an argument just as was done with print():
- **echo "I love the summertime.";**
- echo() is capable of outputting multiple strings

```
<? php
        $heavyweight = "Lennox Lewis";
        $lightweight = "Floyd May weather";
        echo $heavyweight, " and ", $lightweight, " are great fighters.";
?>
```

- This code produces the following:
  Lennox Lewis and Floyd May weather are great fighters.
- Executing the following (in my mind, more concise) variation of the above syntax produces the same output:
  echo "$heavyweight and $lightweight are great fighters.";

Note: Which is faster, echo() or print()? The answer is that the echo() function is a tad faster because it returns nothing, whereas print() will return 1 if the statement is successfully output.

3.3.3 The printf() Statement

- The printf() statement is ideal when you want to output a blend of static text and dynamic information stored within one or several variables.
- It's ideal for two reasons.
  - First, it neatly separates the static and dynamic data into two distinct sections, allowing for easy maintenance.
  - Second, printf() allows you to wield considerable control over how the dynamic information is rendered to the screen in terms of its type, precision, alignment, and position.
- Its prototype looks like this:
  integer printf(string format [, mixed args])
- For example, suppose you wanted to insert a single dynamic integer value into an otherwise static string:
  printf ("Bar inventory: %d bottles of tonic water.", 100);
- Executing this command produces the following:
  Bar inventory: 100 bottles of tonic water
- In this example, *%d* is a placeholder known as a *type specifier*, and the *d* indicates an integer value will be placed in that position.

Type Description

Type	Description
%b	Argument considered an integer; presented as a binary number
%c	Argument considered an integer; presented as a character corresponding to that ASCII value
%d	Argument considered an integer; presented as a signed decimal number
%f	Argument considered a floating-point number; presented as a floating-point number
%o	Argument considered an integer; presented as an octal number
%s	Argument considered a string; presented as a string
%u	Argument considered an integer; presented as an unsigned decimal number
%x	Argument considered an integer; presented as a lowercase hexadecimal number
%X	Argument considered an integer; presented as an uppercase hexadecimal number

Another Example: printf ("%d bottles of tonic water cost $%f", 100, 43.20);

Executing this command produces the following:

100 bottles of tonic water cost $43.200000

3.3.4 The sprintf() Statement

- The sprintf() statement is functionally identical to printf() except that the output is assigned to a string rather than rendered to the browser. The prototype follows:

  string sprintf(string format [, mixed arguments])

- An example follows:

```
$cost = sprintf("$%.2f", 43.2); // $cost = $43.20
```

3.4 PHP's Supported Data Types

3.4.1 Scalar Data Type
3.4.2 Compound Data Types
3.4.3 Converting Between Data Types Using Type Casting
3.4.4 Type-Related Functions
3.4.5 Type Identifier Functions

A *data type* is the generic name assigned to any data sharing a common set of characteristics.

3.4.1 Scalar Data Types

- *Scalar* data types ***are used to represent a single value***.
- Several data types fall under this category, including ***Boolean, integer, float, and string***.

Boolean:

- A Boolean represents two possible states: TRUE or FALSE.
- Alternatively, you can use zero to represent FALSE, and any nonzero value to represent TRUE.

```
$alive = false; // $alive is false.
$alive = 1; // $alive is true.
$alive = -1; // $alive is true.
$alive = 5; // $alive is true.
$alive = 0; // $alive is false.
```

Integer:

- An *integer* is representative of any whole number or, in other words, a number that does not contain fractional parts.
- PHP supports integer values represented in base 10 (decimal), base 8 (octal), and base 16 (hexadecimal) numbering systems

```
42          // decimal
-678900     // decimal
0755        // octal
0xC4E       // hexadecimal
```

Float:

- Floating-point numbers, also referred to as *floats*, *doubles*, or *real numbers*, allow you to specify numbers that contain fractional parts.
- Floats are used to represent monetary values, weights, distances etc ..
- PHP's floats can be specified in a variety of ways, several of which are demonstrated here:

```
4.5678
4.0
8.7e4
1.23E+11
```

String:

- Simply put, a string is a sequence of characters treated as a contiguous group.
- *Strings* are delimited by single or double quotes.
- The following are all examples of valid strings:

```
"PHP is a great language"
"Whoop-de-do"
'*9subway\n'
```

"123$%^789"

3.4.2 Compound Data Types

- *Compound data types* allow for multiple items of the same type to be aggregated under a single representative entity.
- The ***array and the object*** fall into this category.

Object:

- An object is a data type which stores data and information on how to process that data.
- In PHP, an object must be explicitly declared.
- First class of object must be declared
- For this, use the class keyword.
- A class is a structure that can contain properties and methods:

Example

```
<?php
class Car {
   function Car() {
      $this->model = "VW";
   }
}

// create an object
$herbie = new Car();

// show object properties
echo $herbie->model;
?>
```

Array:

- An Array is a variable which holds collection of homogeneous values.
- In the following example $cars is an array.
- The PHP var_dump() function returns the data type and value:

Example

```
<html>
    <body>
    <?php
        $cars = array("Volvo","BMW","Toyota");
        var_dump($cars);
        echo("<br>");
        $str = "hello";
        var_dump($str);
    ?>
    </body>
</html>
```

Output:

```
array(3) { [0]=> string(5) "Volvo" [1]=> string(3) "BMW" [2]=> string(6) "Toyota" }
string(5) "hello"
```

3.4.3 Converting Between Data Types Using Type Casting

- Converting values from one datatype to another is known as *type casting*.
- A variable can be evaluated once as a different type by casting it to another.
- This is accomplished by placing the intended type in front of the variable to be cast.
- A type can be cast by inserting one of the operators shown in below Table in front of the variable.

Type Casting Operators

(array)	Array
(bool) or (boolean)	Boolean
(int) or (integer)	Integer
(object)	Object
(real) or (double) or (float)	Float
(string)	String

Suppose to cast an integer as a double:

```
$score = (double) 13; // $score = 13.0
```

Type casting a double to an integer will result in the integer value being rounded down, regardless of the decimal value.

```
$score = (int) 14.8; // $score = 14
```

3.4.4 Type-Related Functions

- A few functions are available for both verifying and converting data types.

1) Retrieving Types

- The gettype() function returns the type of the provided variable.
- In total, eight possible return values are available: array, boolean, double, integer, object, resource, string, and unknown type.
- Its prototype follows:

string gettype(mixed var)

```
<html>
<body>
 <?php
        $str="hello";
        $n=10;
        echo "type of str is ",gettype($str);
        echo"</br>";
        echo "type of n is ",gettype($n);

 ?>
</body>
</html>
```

Output:

```
type of str is string
type of n is integer
```

2) Converting Types

- The settype() function converts a variable to the type specified by type.
- Seven possible type values are available: array, boolean, float, integer, null, object, and string.
- If the conversion is successful, TRUE is returned; otherwise, FALSE is returned.
- Its prototype follows:
  boolean settype(mixed var, string type)

```
<html>
<body>
 <?php
        $str="hello";
        $n=10;
        echo "type of str is ",gettype($str);
```

```
        echo"</br>";
        echo "type of n is ",gettype($n);
        settype($n,string);
        echo"</br>";
        echo "type of n is ",gettype($n);
    ?>
    </body>
    </html>
```

Output:

```
Type of str is string
Type of n is integer
Type of n is string
```

3.4.5 Type Identifier Functions

- A number of functions are available for determining a variable's type, including is_array(), is_bool(), is_float(), is_integer(), is_null(), is_numeric(), is_object(), is_resource(), is_scalar(), and is_string().
- Because all of these functions follow the same naming convention, arguments, and return values, their introduction is consolidated into a single example.
- The generalized prototype follows:

  boolean is_name(mixed var)
- All of these functions are grouped ultimately accomplishes the same task.
- Each determines whether a variable, specified by var, satisfies a particular condition specified by the function name.
- If var is indeed of the type tested by the function name, TRUE is returned; otherwise,FALSE is returned.
- An example follows:

```
<? php
$item = 43;
printf("The variable \$item is of type array: %d <br />", is_array($item));
printf("The variable \$item is of type integer: %d <br />", is_integer($item));
printf("The variable \$item is numeric: %d <br />", is_numeric($item));
?>
```

This code returns the following:

```
The variable $item is of type array: 0
The variable $item is of type integer: 1
The variable $item is numeric: 1
```

3.5 Identifiers

- *Identifier* is a general term applied to variables, functions, and various other user-defined objects.
- There are several properties that PHP identifiers must abide by:
  - An identifier can consist of one or more characters and must begin with a letter or an underscore. Furthermore, identifiers can consist of only letters, numbers, underscore characters, and other ASCII characters from 127 through 255.
  - ***Valid and Invalid Identifiers***

Valid	**Invalid**
my_function	This&that
Size	!counter
_someword	4ward
  - Identifiers are case sensitive. Therefore, a variable named $recipe is different from a variable named $Recipe, $rEciPe, or $recipE.
  - Identifiers can be any length. This is advantageous because it enables a programmer to accurately describe the identifier's purpose via the identifier name.
  - An identifier name can't be identical to any of PHP's predefined keywords.

3.6 Variables

3.6.1 Variable Declaration
3.6.2 Variable scope
3.6.3 Variable variables

3.6.1 Variable Declaration:

- A variable is a named memory location that contains data and may be manipulated throughout the execution of the program.
- A variable always begins with a dollar sign, $, which is then followed by the variable name.
- Variable names follow the same naming rules as identifiers.
- The following are all valid variables:

```
$color
$operating_system
$_some_variable
$model
```

- Note that variables are case senCsitive.
- For instance, the following variables bear no relation to one another:

```
$color
$Color
$COLOR
```

- Interestingly, variables do not have to be explicitly declared in PHP as they do in a language such as C.
- Rather, variables can be declared and assigned values simultaneously.
- Good programming practice dictates that all variables should be declared prior to use.
- After declaring the variables,values can be assigned to them.
- **Two methodologies are available for variable assignment:**
  - **by value and**
  - **by reference.**

Value Assignment

- Assignment by value simply involves copying the value of the assigned expression to the variable assignee.
- This is the most common type of assignment.
- A few examples follow:

```
$color = "red";
$number = 12;
$age = 12;
$sum = 12 + "15"; // $sum = 27
```

- Here, each of these variables possesses a copy of the expression assigned to it
- For example, $number and $age each possesses their own unique copy of the value 12.
- If you prefer that two variables point to the same copy of a value, you need to assign by reference.

Reference Assignment

- PHP 4 introduced the ability to assign variables by reference, which essentially means that you can Create a variable that refers to the same content as another variable does.
- Therefore, a change to any variable referencing a particular item of variable content will be reflected among all other variables referencing that same content.
- You can assign variables by reference by appending an ampersand (&) to the equal sign. Let's consider an example:

```
<? php
$value1 = "Hello";
$value2 =& $value1; // $value1 and $value2 both equal "Hello"
```

3.6.2 Variable Scope

- In PHP, variables can be declared anywhere in the script.
- The scope of a variable is the part of the script where the variable can be referenced / used.
- PHP variables can be one of four scope types:
  - **3.6.2.1 Local variables**
  - **3.6.2.2 Function parameters**
  - **3.6.2.3 Global variables**
  - **3.6.2.4 Static variables**

3.6.2.1 Local Variables

A variable declared in a function is considered *local*. That is, it can be referenced only in that function. Any assignment outside of that function will be considered to be an entirely different variable from the one contained in the function. Note that when you exit the function in which a local variable has been declared, that variable and its corresponding value are destroyed

```
$x = 4;
function assignx ()
{
        $x = 0;
        printf("\$x inside function is %d <br />", $x);
}
assignx();
        printf("\$x outside of function is %d <br />", $x);
```

$x inside function is 0
$x outside of function is 4

3.6.2.2 Function Parameters

In PHP, as in many other programming languages, any function that accepts arguments must declare those arguments in the function header. Although those arguments accept values that come from outside of the function, they are no longer accessible once the function has exited.

```
// multiply a value by 10 and return it to the caller
        function x10 ($value)
        {
        $value = $value * 10;
        return $value;
        }
```

Keep in mind that although you can access and manipulate any function parameter in the function in which it is declared, it is destroyed when the function execution ends.

3.6.2.3 Global Variables

In contrast to local variables, a *global* variable can be accessed in any part of the program. To modify a global variable, however, it must be explicitly declared to be global in the function in which it is to be modified. This is accomplished, conveniently enough, by placing the keyword global in front of the variable that should be recognized as global.

```
<html>
        <?php
                $x = 5;
                $y = 10;

        function myTest()
        {
```

```
        global $x,$y;
         $y = $x + $y;
    }
    myTest();
        echo $y; // outputs 15
    ?>
</html>
```

3.6.2.4 PHP The static Keyword

Normally, when a function is completed / executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.
To do this, use the **static** keyword when you first declare the variable:

Example

```
<? Php
    function myTest()
    {
    static $x = 0;
    echo $x;
     $x++;
    }
    myTest();
    myTest();
    myTest();
?>
```

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

3.6.3 Variable Variables

- On occasion, you may want to use a variable whose content can be treated dynamically as a variable in itself. Consider this typical variable assignment:
  $recipe = "spaghetti";
- Interestingly, you can treat the value spaghetti as a variable by placing a second dollar sign in front of the original variable name and again assigning another value:
  $$recipe = "& meatballs";
- This in effect assigns *& meatballs* to a variable named spaghetti. Therefore, the following two snippets of code produce the same result:
  echo $recipe $spaghetti;
- The result is the string *spaghetti & meatballs*.

3.7 Constants

- A *constant* is a value that cannot be modified throughout the execution of a program.
- Constants are particularly useful when working with values that definitely will not require modification, such as Pi (3.141592).
- Once a constant has been defined, it cannot be changed (or redefined) at any other point of the program.
- Constants are defined using the define() function.

Defining a Constant

- The define() function defines a constant by assigning a value to a name.
- boolean define(string name, mixed value)
  Eg : define("PI", 3.141592);
- The constant is subsequently used in the following listing:
  printf("The value of Pi is %f", PI);

```
$pi2 = 2 * PI;
printf("Pi doubled equals %f", $pi2);
```

- This code produces the following results:
  The value of pi is 3.141592.
  Pi doubled equals 6.283184.
- constants are global; they can be referenced anywhere in your script

3.8 Expressions

An *expression* is a phrase representing a particular action in a program. All expressions consist of at least one operand and one or more operators. A few examples follow:

```
$a = 5; // assign integer value 5 to the variable $a
$a = "5"; // assign string value "5" to the variable $a
$sum = 50 + $some_int; // assign sum of 50 + $some_int to $sum
$wine = "Zinfandel"; // assign "Zinfandel" to the variable $wine
$inventory++; // increment the variable $inventory by 1
```

3.8.1 Operands

Operands are the inputs of an expression

```
$a++; // $a is the operand
$sum = $val1 + val2; // $sum, $val1 and $val2 are operands.
```

3.8.2 Operators

An *operator* is a symbol that specifies a particular action in an expression. complete listing of all operators, ordered from highest to lowest precedence.

Operator	Purpose
new	Object instantiation
( )	Expression subgrouping
[ ]	Index enclosure
! ~ ++ --	Boolean NOT, bitwise NOT, increment, decrement
@	Error suppression
/ * %	Division, multiplication, modulus
+ - .	Addition, subtraction, concatenation
<< >>	Shift left, shift right (bitwise)
< <= > >=	Less than, less than or equal to, greater than, greater than or equal to
== != === <>	Is equal to, is not equal to, is identical to, is not equal to
& ^ |	Bitwise AND, bitwise XOR, bitwise OR
&& ||	Boolean AND, Boolean OR
?:	Ternary operator
= += *= /= .= %=&=|= ^= <<= >>=	Assignment operators
AND XOR OR	Boolean AND, Boolean XOR, Boolean OR
,	Expression separation

Operator Precedence

Operator precedence is a characteristic of operators that determines the order in which they evaluate the operands surrounding them.

PHP follows the standard precedence rules used in elementary school math class. Consider a few examples:

```
$total_cost = $cost + $cost * 0.06;
```

This is the same as writing

```
$total_cost = $cost + ($cost * 0.06);
```

because the multiplication operator has higher precedence than the addition operator.

PHP Operators

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators

3.9 String Interpolation

3.9.1 Double quotes
3.9.2 Escape sequences
3.9.3 Single quotes
3.9.4 Curly braces
3.9.5 Heredoc

To offer developers the maximum flexibility when working with string values, PHP offers a means for both literal and figurative interpretation

3.9.1 Double Quotes

Strings enclosed in double quotes are the most commonly used in PHP scripts because they offer the most flexibility. This is because both variables and escape sequences will be parsed accordingly.
Consider the following example:

```
<?php
$sport = "boxing";
echo "Jason's favorite sport is $sport.";
?>
```

This example returns the following:

```
Jason's favorite sport is boxing.
```

3.9.2 Escape Sequences

Escape sequences are also parsed. Consider this example:

```
<?php
$output = "This is one line.\nAnd this is another line.";
echo $output;
?>
```

Output:

```
This is one line. And this is another line.
```

In addition to the newline character, PHP recognizes a number of special escape sequences, all of which are

Recognized Escape Sequences

Sequence	Description
\n	Newline character
\r	Carriage return
\t	Horizontal tab
\\	Backslash
\$	Dollar sign
\"	Double quote
\[0-7]{1,3}	Octal notation
\x[0-9A-Fa-f]{1,2}	Hexadecimal notation

3.9.3 Single Quotes

Enclosing a string within single quotes is useful when the string should be interpreted exactly as stated. This means that both variables and escape sequences will not be interpreted when the string is parsed.

For example, consider the following single-quoted string:

print 'This string will $print exactly as it\'s \n declared.';

This produces the following:

This string will $print exactly as it's \n declared.

3.9.4 Curly Braces

While PHP is perfectly capable of interpolating variables representing scalar data types, you'll find that variables representing complex data types such as arrays or objects cannot be so easily parsed when embedded in an echo() or print() string. You can solve this issue by delimiting the variable in curly braces, like this:

echo "The capital of Ohio is {$capitals['ohio']}.";

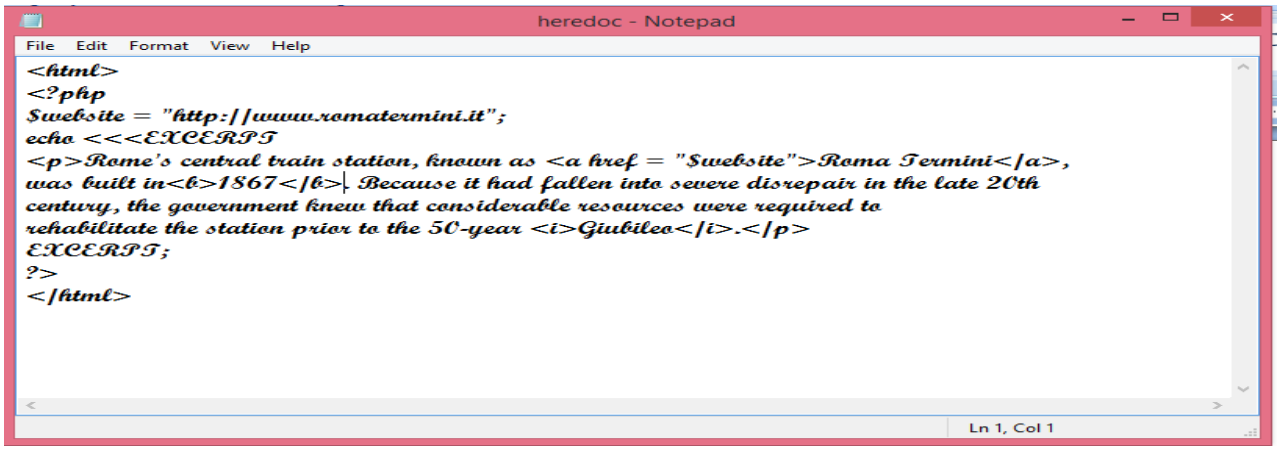
3.9.5 Heredoc

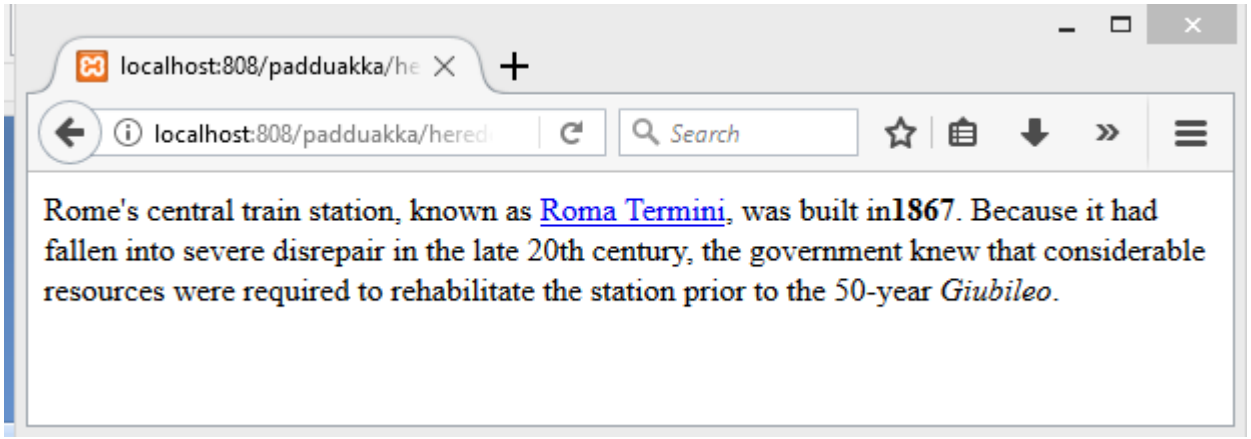
Heredoc syntax offers a convenient means for outputting large amounts of text. Rather than delimiting strings with double or single quotes, two identical identifiers are employed. An example follows:

```
<? php
        $website = "http://www.romatermini.it";
        echo <<<EXCERPT
        <p>Rome's central train station, known as <a href = "$website">Roma Termini</a>,
        was built in 1867. Because it had fallen into severe disrepair in the late 20th
        century, the government knew that considerable resources were required to
        rehabilitate the station prior to the 50-year <i>Giubileo</i>.</p>
        EXCERPT;
?>
```

Several points are worth noting regarding this example:

- The opening and closing identifiers (in the case of this example, EXCERPT) must be Identical. You can choose any identifier you please, but they must exactly match. The only constraint is that the identifier must consist of solely alphanumeric characters and underscores and must not begin with a digit or an underscore.
- The opening identifier must be preceded with three left-angle brackets (<<<).
- Heredoc syntax follows the same parsing rules as strings enclosed in double quotes. That is, both variables and escape sequences are parsed. The only difference is that double quotes do not need to be escaped.
- The closing identifier must begin at the very beginning of a line. It cannot be preceded with spaces or any other extraneous character. Furthermore, the presence of any spaces following the opening or closing identifier will produce a syntax error.





3.10 ARRAYs

3.10.1 What is an Array?

- Array is a variable that represents collection of homogeneous Data items.

3.10.2 Creating an Array

- Two ways of Creating Arrays are
  1) **To Create an Empty Array**
     $variable = array (); // to create an empty array
     Eg: $a = array ();
  2) **To Create and Initialize an Array**
     $variable = array (item1, item2,…,itemN);
     Eg: $a = array (10,20,30);

- Associative arrays are arrays that use named keys

$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

```
<?php
$age= array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter   is   " .   $age['Peter']   . "   years   old.";
?>
```

Output:

Peter is 35 years old.

3.10.3 Outputting an Array

3.10.3.1 Printing Arrays for Testing Purposes

- The most common way to output an array's contents is by iterating over each key and echoing the corresponding value.
- foreach statement does the trick nicely:

```
<html>
    <? php
            $a = array ();
            $a[0] = "10";
```

```
        $a[1]  = "20";
        $a[2] = "30";
        foreach ($a AS $i)
        {
                echo "{$i}<br/>";
        }
    ?>
</html>
```

OUTPUT:

10
20
30

3.10.3.1 Printing Arrays for Testing Purposes

- print_r() function is used to easily output their contents to the screen for testing purposes.
- Its prototype follows:

  boolean print_r(mixed *variable* [, boolean *return*])
- The print_r() function accepts a variable and sends its contents to standard output, returning TRUE on success and FALSE otherwise.
- The optional parameter return modifies the function's behavior, causing it to return the output to the caller, rather than send it to standard output.
- Therefore, to return the contents of the preceding $a array, just set return to TRUE:

Program :

```
<html>
    <? php
        $a = array();
        $a[0] = "10";
        $a[1]  = "20";
        $a[2] = "30";
        print_r($a);
    ?>
</html>
```

Output

Array ([0] => 10 [1] => 20 [2] => 30)

program

```
 <html>
    <?php
        $a = array();
        $a[0] = "10";
        $a[1]  = "20";
        $a[2] = "30";
        $res = print_r($a, TRUE);
        echo $res;
    ?>
</html>
```

Output

Array ([0] => 10 [1] => 20 [2] => 30)

10.3.4 Adding and Removing Array Elements

- **10.3.4.1 Adding a Value to the Front of an Array**
- **10.3.4.2 Adding a Value to the End of an Array**
- **10.3.4.3 Removing a Value from the Front of an Array**
- **10.3.4.4 Removing a Value from the End of an Array**

10.3.4.1 Adding a Value to the Front of an Array

- Adding a Value to the Front of an Array

- The array_unshift() function adds elements to the front of the array.
- All preexisting numerical keys are modified to reflect their new position in the array.
- Its prototype follows:
  int array_unshift(array *array*, mixed *variable* [, mixed *variable*...])
- The following example adds two states to the front of the $states array:

```
$states = array("Ohio", "New York");
array_unshift($states, "California", "Texas");
// $states = array("California", "Texas", "Ohio", "New York");
```

10.3.4.2 Adding a Value to the End of an Array

- The **array_push()** function adds a value to the end of an array, returning the total count of elements in the array after the new value has been added.
- You can push multiple variables onto the array simultaneously by passing these variables into the function as input parameters.
- Its prototype follows:
  int array_push(array *array*, mixed *variable* [, mixed *variable*...])
- The following example adds two more states onto the $states array:

```
$states = array("Ohio", "New York");
array_push($states, "California", "Texas");
// $states = array("Ohio", "New York", "California", "Texas");
```

10.3.4.3 Removing a Value from the Front of an Array

- The **array_shift()** function removes and returns the first item found in an array.
- Its prototype follows:
  mixed array_shift(array *array*)
- The following example removes the first state from the $states array:

```
$states = array("Ohio", "New York", "California", "Texas");
$state = array_shift($states);
// $states = array("New York", "California", "Texas")
// $state = "Ohio“
```

10.3.4.4 Removing a Value from the End of an Array

- The **array_pop()** function removes and returns the last element from an array.
- Its prototype follows:/
  mixed array_pop(array *array*)
- The following example removes the last state from the $states array:

```
$states = array("Ohio", "New York", "California", "Texas");
$state = array_pop($states);
// $states = array("Ohio", "New York", "California"
// $state = "Texas".
```

10.3.5 Locating Array Elements

- Several functions are available to search arrays in order to locate items of interest

10.3.5.1 Searching an Array
10.3.5.2 Searching Associative Array Keys
10.3.5.3 Searching Associative Array Values
10.3.5.4 Retrieving Array Keys
10.3.5.5 Retrieving Array values

10.3.5.1 Searching an Array

- The **in_array()** function searches an array for a specific value, returning TRUE if the value is found and FALSE otherwise.
- Its prototype follows:
  boolean in_array(mixed *needle*,)
- **I**n the following example, a message is output if a specified state (Ohio) is found in an array

```
$state = "Ohio";
$states = array("California", "Hawaii", "Ohio", "New York");
if(in_array($state, $states))
echo "Not to worry, $state is smoke-free!";
```

10.3.5.2 Searching Associative Array Keys

- The function **array_key_exists()** returns TRUE if a specified key is found in an array and FALSE otherwise.
- Its prototype follows:
  boolean array_key_exists(mixed *key*, array *array*)
- The following example will search an array's keys for Ohio, and if found, will output information Ohio joined the Union on March 1, 1803

```
$state["Delaware"] = "December 7, 1787";
$state["Pennsylvania"] = "December 12, 1787";
$state["Ohio"] = "March 1, 1803";
if (array_key_exists("Ohio", $state))
printf("Ohio joined the Union on %s", $state["Ohio"]);
```

- The following is the result:
  Ohio joined the Union on March 1, 1803

10.3.5.3 Searching Associative Array Values

- The array_search() function searches an array for a specified value, returning its key if located and FALSE otherwise.
- Its prototype follows:
  mixed array_search(value, array)
- The following example searches $state for a particular date (December 7), returning information about the corresponding state if located:

```
$state["Ohio"] = "March 1";
$state["Delaware"] = "December 7";
$state["Pennsylvania"] = "December 12";
$founded = array_search("December 7", $state);
if ($founded) printf("%s was founded on %s.", $founded,  $state[$founded]);
```

- The output follows:
  -Delaware was founded on December 7.

10.3.5.4 Retrieving Array Keys

- The **array_keys()** function returns an array consisting of all keys located in an array. Its prototype follows:
  array array_keys(array *array*)
- The following example outputs all of the key values found in the $state array:

```
$state["Delaware"] = "December 7, 1787";
$state["Pennsylvania"] = "December 12, 1787";
$state["New Jersey"] = "December 18, 1787";
$keys = array_keys($state);
print_r($keys);
```

- The output follows:
  Array ([0] => Delaware [1] => Pennsylvania [2] => New Jersey)

10.3.5.5 Retrieving Array values

- The **array_values()** function returns all values located in an array, automatically providing numeric indexes for the returned array.
- Its prototype follows:
  array array_values(array *array*)
- The following example will retrieve the population numbers for all of the states found in $population:

```
$population = array("Ohio" => "11,421,267", "Iowa" => "2,936,760");
print_r(array_values($population));
```

- This example will output the following:
  Array ([0] => 11,421,267 [1] => 2,936,760)

10.3.6 Traversing Arrays

10.3.6.1 Retrieving the Current Array Key
10.3.6.2 Retrieving the Current Array Value
10.3.6.3 Retrieving the Current Array Key and Value
10.3.6.4 Moving the Array Pointer
10.3.6.4.1 Moving the Pointer to the Next Array Position
10.3.6.4.2 Moving the Pointer to the First Array Position
10.3.6.4.3 Moving the Pointer to the Last Array Position

10.3.6.1 Retrieving the Current Array Key

- The **key()** function returns the key located at the current pointer position of the provided array. Its prototype follows:
  mixed key(array *array*)
- Example :

```
<html>
    <body>
        <? php
            $num = array (10=>"apple", 20=>"mango",30=>"orange");
            $key = key($num);
            print "Current key is $key";
        ?>
    </body>
</html>
```

- This returns the following: The Current key is 10

10.3.6.2 Retrieving the Current Array Value

- The **current()** function returns the array value residing at the current pointer position of the array. Its prototype follows: **mixed current(array *array*)**

```
<html>
    <body>
        <?php
            $num = array(10=>"apple",20=>"mango",30=>"orange");
            $val =current($num);
            print "Current key is $val";
        ?>
    </body>
</html>
```

This returns the following: The Current value is apple

10.3.6.3 Retrieving the Current Array Key and Value'

- **each () function returns the** current key/value pair from the array and advances the pointer one position.
- Its prototype follows:
  array each(array *array*)

PROGRAM

```
<html>
    <body>
        <?php
            $num = array(10=>"apple", 20=>"mango",30=>"orange");
            $key_val =each($num);
            print_r($key_val);
        ?>
    </body>
</html>
```

OUTPUT

Array ([1] => apple [value] => apple [0] => 10 [key] => 10)

10.3.6.4 Moving the Array Pointer

- Several functions are available for moving the array pointer.

10.3.6.4.1 Moving the Pointer to the Next Array Position

- The next() function returns the array value residing at the position immediately following that of the current array pointer. Its prototype follows:

  mixed next(array *array*)

10.3.6.4.2 Moving the Pointer to the First Array Position

- The **reset()** function serves to set an array pointer back to the beginning of the array. Its prototype follows: **mixed reset(array *array*)**

```
<?php
    $num = array("apple","mango","orange");
    $n = next($num);
    echo $n;//displays mango;
    $r = reset($num);//brings cursor to apple
    $c=current($num);//displays current value
    echo $c;
?>
```

10.3.6.4.3 Moving the Pointer to the Last Array Position

- The **end()** function moves the pointer to the last position of an array, returning the last element. Its prototype follows:
  mixed end(array *array*)
- The following example demonstrates retrieving the first and last array values:

```
$fruits = array("apple", "orange", "banana");
$fruit = reset(fruits); // returns "apple"
$fruit = end($fruits); // returns "banana"
```

10..3.7 Determining Array Size and Uniqueness

- A few functions are available for determining the number of total and unique array values.

10.3.7.1 Determining the Size of an Array
10.3.7.2 Counting Array Value Frequency
10.3.7.3 Determining Unique Array Values

10.3.7.1 Determining the Size of an Array

- The count() function returns the total number of values found in an array. Its prototype follows:

  integer count(array *array)*
- number of vegetables found in the $garden array:

```
$garden = array("cabbage", "peppers", "turnips", "carrots");
echo count($garden);
```

- This returns the following: **4**

10.3.7.2 Counting Array Value Frequency

- The array_count_values() function returns an array consisting of associative key/value pairs. Its prototype follows:

  array array_count_values(array *array*)
- Each key represents a value found in the input_array, and its corresponding value denotes the frequency of that key's appearance (as a value) in the input_array. An example follows:

```
$states = array("Ohio", "Iowa", "Arizona", "Iowa", "Ohio");
$stateFrequency = array_count_values($states);
print_r($stateFrequency);
```

- This returns the following:
  Array ([Ohio] => 2 [Iowa] => 2 [Arizona] => 1)

10.3.7.3 Determining Unique Array Values

- The **array_unique()** function removes all duplicate values found in an array, returning an array consisting of solely unique values.
- Its prototype follows:
  array array_unique(array *array)*
- An example follows:
  $states = array("Ohio", "Iowa", "Arizona", "Iowa", "Ohio");
  $uniqueStates = array_unique($states);
  print_r($uniqueStates);
- This returns the following:
  Array ([0] => Ohio [1] => Iowa [2] => Arizona)

3.10.8 Sorting Arrays

3.10.8.1 Reversing Array Element Order
3.10.8.2 Flipping Array Keys and Values
3.10.8.3 Sorting an Array
3.10.8.4 Sorting an Array in Reverse Order

3.10.8.1 Reversing Array Element Order

- The array_reverse() function reverses an array's element order. Its prototype follows:
  array array_reverse(array *array)*
- An example follows:
  $states = array("Delaware", "Pennsylvania", "New Jersey");
  print_r(array_reverse($states));
- This example returns the following:

Array ([0] => New Jersey [1] => Pennsylvania [2] => Delaware)

3.10.8.2 Flipping Array Keys and Values

- The array_flip() function reverses the roles of the keys and their corresponding values in an array. Its prototype follows:
  array array_flip(array *array*)
- An example follows:
  $state = array("Delaware", "Pennsylvania", "New Jersey");
  $state = array_flip($state);
  print_r($state);
- This example returns the following:
  Array ([Delaware] => 0 [Pennsylvania] => 1 [New Jersey] => 2)

3.10.8.3 Sorting an Array

- The sort() function sorts an array, ordering elements from lowest to highest value. Its prototype follows:
  void sort(array *array)*
- Consider an example. Suppose you want to sort exam grades from lowest to highest:
  $grades = array(42, 98, 100, 100, 43, 12);
  sort($grades);
  print_r($grades);
- The outcome looks like this:

Array ([0] => 12 [1] => 42 [2] => 43 [3] => 98 [4] => 100 [5] => 100)

3.10.8.4 Sorting an Array in Reverse Order

- The rsort() function is identical to sort(), except that it sorts array items in reverse (descending) order. Its prototype follows:
  void rsort(array *array* [, int *sort_flags*])

- An example follows:
  $states = array("Ohio", "Florida", "Massachusetts", "Montana");
  rsort($states);
  print_r($states);
- It returns the following:
  Array ([0] => Ohio [1] => Montana [2] => Massachusetts [3] => Florida)

3.10.9 Merging, Slicing, Splicing, and Dissecting Arrays

3.10.9.1 Merging Arrays
3.10.9.2 Combining Two Arrays
3.10.9.3 Slicing an Array
3.10.9.4 Splicing an Array
3.10.9.5 Calculating an Array Intersection
3.10.9.6 Calculating an Array Difference

3.10.9.1 Merging Arrays

- The array_merge() function merges arrays together, returning a single, unified array.
- The resulting array will begin with the first input array parameter, appending each subsequent array parameter in the order of appearance.
- Its prototype follows:
  array array_merge(array *array1*, array *array2* [, array *arrayN*])

```
<? php
        $face = array("J", "Q", "K", "A");
        $numbered = array("2", "3", "4", "5", "6", "7", "8", "9");
        $cards = array_merge($face, $numbered);
        foreach($cards AS $i)
                echo "$i</br>";
?>
```

3.10.9.2 Combining Two Arrays

- The array_combine() function produces a new array consisting of a submitted set of keys and corresponding values.
- Its prototype follows:
  array array_combine(array *keys*, array *values*)
- Both input arrays must be of equal size, and neither can be empty. An example follows:

```
$abbreviations = array("AL", "AK", "AZ", "AR");
$states = array("Alabama", "Alaska", "Arizona", "Arkansas");
$stateMap = array_combine($abbreviations,$states);
print_r($stateMap);
```

- This returns the following:
  Array ([AL] => Alabama [AK] => Alaska [AZ] => Arizona [AR] => Arkansas)

3.10.9.3 Slicing an Array

- The array_slice() function returns a section of an array based on a starting and ending offset value. Its prototype follows:
  array array_slice(array *array*, int *offset* [, int *length*)

```
$states = array("Alabama", "Alaska", "Arizona", "Arkansas",
"California", "Colorado", "Connecticut");
$subset = array_slice($states, 4);
print_r($subset);
```

- This returns the following:

Array ([0] => California [1] => Colorado [2] => Connecticut)

3.10.9.4 Splicing an Array

- The array_splice() function removes all elements of an array found within a specified range, returning those removed elements in the form of an array.
- Its prototype follows:

array array_splice(array *array*, int *offset* [, int *length* [, array *replacement*]])

```
$states = array("Alabama", "Alaska", "Arizona", "Arkansas",
"California", "Connecticut");
$subset = array_splice($states, 4);
print_r($states);
print_r($subset);
```

- This produces the following (formatted for readability):

```
Array ( [0] => Alabama [1] => Alaska [2] => Arizona [3] => Arkansas )
Array ( [0] => California [1] => Connecticut )
```

3.10.9.5 Calculating an Array Intersection

- The **array_intersect()** function returns a key-preserved array consisting only of those values present in the first array that are also present in each of the other input arrays. Its prototype follows:

  array array_intersect(array *array1*, array *array2* [, *arrayN*])
- The following example will return all states found in the $array1 that also appear in $array2 and $array

```
$array1 = array("OH", "CA", "NY", "HI", "CT");
$array2 = array("OH", "CA", "HI", "NY", "IA");
$array3 = array("TX", "MD", "NE", "OH", "HI");
$intersection = array_intersect($array1, $array2, $array3);
print_r($intersection);
```

- This returns the following:

```
Array ( [0] => OH [3] => H
```

3.10.9.6 Calculating an Array Difference

- Essentially the opposite of array_intersect(), the function array_diff() returns those values located in the first array that are not located in any of the subsequent arrays:

  array_diff(array *array1*, array *array2* [, arrayN])
- An example follows:

```
$array1 = array("OH", "CA", "NY", "HI", "CT");
$array2 = array("OH", "CA", "HI", "NY", "IA");
$array3 = array("TX", "MD", "NE", "OH", "HI");
$diff = array_diff($array1, $array2, $array3);
print_r($intersection);
```

- This returns the following:

```
Array ( [0] => CT )
```

3.10.10 Other Useful Array Function

- **3.10.10.1 Returning a Random Set of Keys**
- **3.10.10.2 Shuffling Array Elements**

3.10.10.1 Returning a Random Set of Keys

- The array_rand() function will return a random number of keys found in an array. Its prototype follows:

  mixed array_rand(array *array* [, int *num_entries*])
- Omitting the optional num_entries parameter, only one random value will be returned.
- tweak the number of returned random values by setting num_entries accordingly. An example follows:

```
$states = array("Ohio" => "Columbus", "Iowa" => "Des Moines",
"Arizona" => "Phoenix");
```

```
$randomStates = array_rand($states, 2);
print_r($randomStates);
```

- This returns the following (your output may vary):

Array ([0] => Arizona [1] => Ohio)

3.10.10.2 Shuffling Array Elements

- Shuffling Array Elements
- The shuffle() function randomly reorders an array. Its prototype follows:

  void shuffle(array *input_array*)
- Consider an array containing values representing playing cards:

```
$cards = array("jh", "js", "jd", "jc", "qh", "qs", "qd", "qc",
"kh", "ks", "kd", "kc", "ah", "as", "ad", "ac");
shuffle($cards);
print_r($positions);
```

- This returns something along the lines of the following (your results will vary because of the shuffle):

Array ([0] => js [1] => ks [2] => kh [3] => jd
[4] => ad [5] => qd [6] => qc [7] => ah
[8] => kc [9] => qh [10] => kd [11] => as
[12] => ac [13] => jc [14] => jh [15] => qs)