PHP FUNDAMENTALS

PHP is a well-known and used programming language that allows us to develop websites focusing on the server-side scripting and also can be embedded to HTML.

LANGUAGE REFERENCE (from PHP documentation)

* Basic Syntax
  + PHP Tags
* In PHP, whenever file is parsed, it always looks for the open and close tags which are <?php for opening tag and ?> for closing tag that starts PHP in executing the code between them. PHP allows this manner of parsing to embed all kinds of documents as all other outside the open and close tags is ignored by the PHP parser.

Sample code:

<?php

echo “PHP opening and closing tags”;

?>

* + Escaping from HTML
* All other codes outside the open and close tags will be ignored by PHP. PHP allowed this to embed in the HTML documents, one example is creating templates.

Sample code:

<p>This part of code is ignored by PHP.</p>  
<?php echo 'While this is going to be parsed.'; ?>  
<p>This will be also ignored by PHP.</p>

* Other examples are (1) Escape using the conditions and (2) Using the PHP and close tags. Samples of the two are the following respectively.

<?php if ($expression == true): ?>  
<?php else: ?>  
<?php endif; ?>

and

1.  <?php echo 'You can use this tag if the PHP code will served in XML or XHTML documents.; ?>  
  
2   <?php echo 'You can use this tag known as the “short echo tag”. It is enabled in PHP 5.4.0 or later versions of PHP.' ?>.  
  
3.  <? echo 'You can use this code if it is within the short tags, but will only work if the “short open tag” Is enabled.'; ?>  
  
4.  <script language="php">  
        echo 'You can also use this tag in processing instructions.';  
    </script>

    5.  <% echo 'You can also use ASP-style tags optionally'; %>  
    All the codes that are within this tag <%= $variable; %> is a shortcut for this code <% echo $variable; %>

* + Instruction Separation
* Like in other programming languages such as C and Perl, PHP also requires all the instructions that all of the statement will end with semicolon. The closing tag of a PHP block will automatically imply a semicolon, and you are no longer to input another semicolon to terminate the last line of the PHP code. The closing tag will include the immediately trailing new line of there is any.

Sample code:

<?php echo 'This is the first example; ?>  
  
<?php echo 'This is the second example' ?>  
  
<?php echo 'This is the third example that omits the last closing tag.';

* + Comments
* PHP styles for comments support C, C++, and Perl-style comment.
* The one-line comment, multi-line line comment and another one-line comment that just like a shell-style or a Perl-style comment.

Sample code:

<?php  
    echo 'Examples for different styles of comment in PHP.';

// This is a one-line c++ style comment  
    /\* This is a multi-line comment  
       Part of multi0line comment\*/  
    echo 'This is another example of comment style';  
    echo 'Last example'; # This is a one-line shell-style comment  
?>

* Types
  + Introduction
* PHP is supporting ten primitive types, and each primitive type are classified as scalar types, compound types, and the special types. PHP also introduces some of the pseudo-types for readability reasons.
* The four scalar types are Boolean, integer, float or double, and the string.
* The four compound types are array, object, callable, and iterable.
* The two special types are resource, and NULL.
* The pseudo-types are mixed, number, callback or callable, array|object, and void.
  + Boolean
* Boolean, like in Java, expresses a value of true or false
  + Integers
* Integers can also be specified in decimal, hexadecimal, octal, or binary digits that is optionally preceded by the + or – sign. In using the octal notation, the number must be preceded by *zero (0)*, using a hexadecimal notation, the number must be preceded by *0x*, and in using binary notation, the number must be preceded by *0b*.

Sample code:

<?php  
$a = 1234; // decimal number  
$a = -123; // a negative number  
$a = 0123; // number in octal notation  
$a = 0x1A; // number in hexadecimal notation $a = 0b11111111; // number in binary notation

?>

* + Floating point numbers
* Floating point number also known as float, double or real numbers are allowed to be specified by the following syntax.

Sample code:

<?php  
$a = 1.234;   
$b = 1.2e3;   
$c = 7E-10;  
?>

* + String
* String are series of characters that are same as byte. PHP supports 256-character set and do not support native Unicode. String can be specified in different ways such as “single quoted”, “double quoted”, heredoc syntax”, or “nowdoc syntax”. Example of each syntax are as follows respectively.

Sample code:

<?php  
echo 'this is an example of single quoted string';  
?>

If a string is in double-quotes, PHP will interpret the following for special characters.

\n linefeed

\r carriage return

\t horizontal tab

\v vertical tab

\e escape

\f form feed

\\ backslash

\$ dollar sign

\” double-quote

\[0-7]{1,3} sequence of characters that matches a regular expression

is a character in octal notation that overflows to fit in a byte.

\x[0-9A- sequence of characters that matches a regular expression is

Fa-f]{1,2} a character in hexadecimal notation.

\u{[0-9A- sequence of characters that matches a regular expression is

Fa-f]+} a Unicode that will output to the string as to codepoint’s

UTF-8 representation.

<?php  
class sample {  
    public $bar = <<<EOT  
bar  
EOT;  
}  
?>

<?php  
class foo {  
    public $bar = <<<'EOT'  
bar  
EOT;  
}  
?>

* + Arrays
* PHP array is an ordered map, it is a type that is associating values to keys. The type optimizes in many different uses that can be treated as an array, list or vector, hash table, dictionary, collection, stack, queue, and many more. It is also possible that values can also be another array, trees, and multidimensional arrays.
  + Objects
* In creating a new object, use the *new* statement in instantiating a class.

Sample code:

<?php  
class sample  
{  
    function do\_sample()  
    {  
        echo "Doing sample.";   
    }  
}  
  
$bar = new sample;  
$bar->do\_sample();  
?>

* + Iterable
* Iterable is a pseudo-type that accepts any array or object that implements the Traversable interface. Both of the types are iterable that uses foreach and can be used with yield from in a generator.
  + Resources
* Resource is a special variable that holds a reference to the external resource. Resources are used by the special functions. Resource variables holding a special handle to opened files, database connections, image canvas, etc.
  + NULL
* A null represents a variable without value. A variable is considered as null if the variable is assigned to a constant null, the variable has not set to any value, and the variable has been unset().
  + Callbacks / Callable
* Callbacks are denoted by callable type. Functions like call\_user\_func() or usort() accepts user-defined callback functions as a parameter. Callback functions are not only as simple functions, but also as object methods that includes static class methods.

Sample Callback function syntax:

<?php  
  
// An example callback function  
function my\_callback\_function() {  
    echo 'hello world!';  
}  
  
// An example callback method  
class MyClass {  
    static function myCallbackMethod() {  
        echo 'Hello World!';  
    }  
}  
  
// Type 1: Simple callback  
call\_user\_func('my\_callback\_function');  
  
// Type 2: Static class method call  
call\_user\_func(array('MyClass', 'myCallbackMethod'));  
  
// Type 3: Object method call  
$obj = new MyClass();  
call\_user\_func(array($obj, 'myCallbackMethod'));  
  
// Type 4: Static class method call (As of PHP 5.2.3)  
call\_user\_func('MyClass::myCallbackMethod');  
  
// Type 5: Relative static class method call

class A {  
    public static function who() {  
        echo "A\n";  
    }  
}  
  
class B extends A {  
    public static function who() {  
        echo "B\n";  
    }  
}  
  
call\_user\_func(array('B', 'parent::who')); // A  
  
// Type 6: Objects implementing \_\_invoke can be used as callables  
class C {  
    public function \_\_invoke($name) {  
        echo 'Hello ', $name, "\n";  
    }  
}  
  
$c = new C();  
call\_user\_func($c, 'PHP!');  
?>

* + Pseudo-types and variables
* Pseudo-types are keywords used in PHP to determine the types or values that argument can have.

**mixed**

it indicates that parameters may accept multiple types.

**number**

it indicates that parameter is either integer or float.

**callback**

callback pseudo types are used before callable type was introduced by PHP 5.4

**array|object**

it indicates that a parameter can either be an array or object.

**void**

as a return type, void return values are useless. It also means that function do not accept parameters.

* Variables
  + Basics
* Variables are represented by a dollar sign, followed by the variable name, and the variable name is case-sensitive.
  + Predefined Variables
* PHP has many predefined variables in any scripts which is runs. Many of the variables cannot be as full documented as the dependent upon what server is running, the version of the server, and some other factors. Some of the variables are not available when PHP is running on the command line.
  + Variable scope
* The variable scope is the context where it is defined. All variables of PHP have only a single scope.
* Constants
* Constant is a name or an identifier for a simple value. A constant cannot change its value while it is still in execution except for the magic constants which is actually not a constant. Constant is also a case-sensitive and its identifiers are always in uppercase.
  + Syntax
* Use the define() function to define a constant or use the *const* keyword outside its class definition. The define() allows a constant to be identified in an arbitrary execution, while the *const* keyword has a restriction that only a scalar data like Boolean, integer, float and string, can be contained in constants.
* The following sample codes are example of defining a constant and defining a constant with the use of the *const* keyword respectively.

Sample codes:

<?php  
define("CONSTANT", "Hello world.");  
echo CONSTANT; // outputs "Hello world."  
echo Constant; // outputs "Constant" and issues a notice.  
?>

<?php  
// Works as of PHP 5.3.0  
const CONSTANT = 'Hello World';  
echo CONSTANT;  
  
// Works as of PHP 5.6.0  
const ANOTHER\_CONST = CONSTANT.'; Goodbye World';  
echo ANOTHER\_CONST;  
  
const ANIMALS = array('dog', 'cat', 'bird');  
echo ANIMALS[1]; // outputs "cat"  
  
// Works as of PHP 7  
define('ANIMALS', array(  
    'dog',  
    'cat',  
    'bird'  
));  
echo ANIMALS[1]; // outputs "cat"  
?>

* + Magic Constants
* PHP has many predefined constants that runs to any of the script. Many of the constants are created by extensions and only be present to those available extensions either dynamic loading or its because they are compiled in.
* Some are few of the magical PHP constants.

\_ \_LINE\_ \_ current line number of the file

\_ \_FILE\_ \_ the full path and filename of the file with

symlinks resolved

\_ \_DIR\_ \_ directory of the file

\_ \_FUNCTION\_ \_ function name

\_ \_CLASS\_ \_ class name

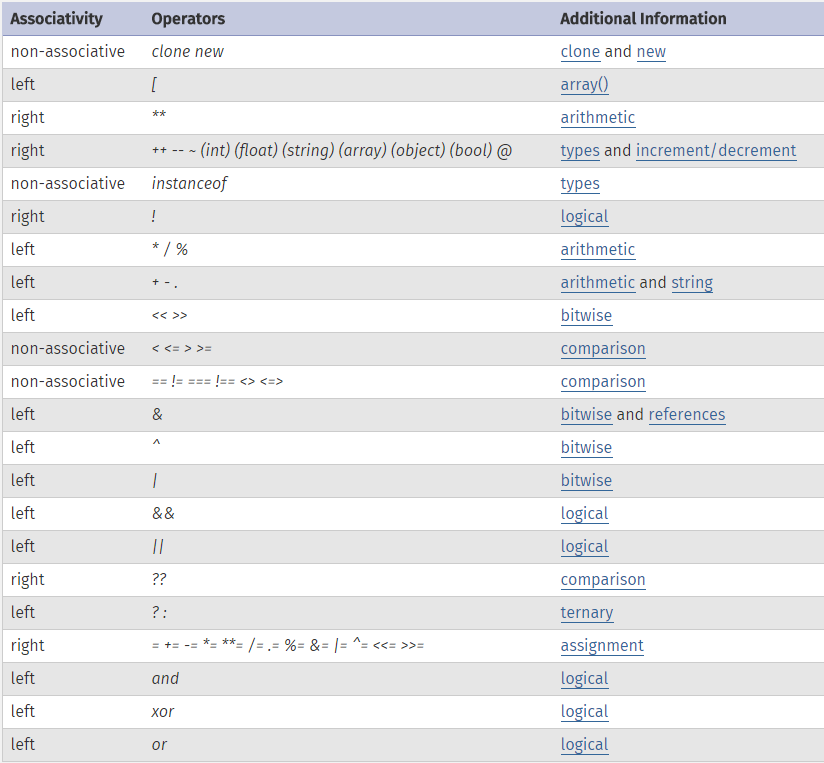
\_ \_TRAIT\_ \_ trait name

\_ \_METHOD\_ \_ class method name

\_ \_NAMESPACE\_ \_ name of the namespace

ClassName::class fully qualified class name

* Expressions
* The most important building blocks of PHP are the expressions because almost everything that is written in PHP is an expression. Expression is anything that has a value.
* Operators
* Operators are the ones that take one or more values or expression and yields another value. Operators are grouped according to the number of the values it takes. The unary operator takes only one value, binary operators take two values, and finally the ternary operator that takes three values.
  + Operator Precedence
* Operator precedence specifies on how tight it binds on two expressions together. The following list are the operator precedence.



* + Arithmetic Operators
* Arithmetic operators in PHP works just like the same as the basic arithmetic from school, it is just that a division operator will return a float unless the operands are integers and evenly divisible.
  + Assignment operator
* The basic assignment operator is “=”, means that the operand on the left will get the value of the expression on the right, that is “gets set to”. The value assigned is the value of the assignment expression.
  + Bitwise Operators
* It allows manipulating the specific bits in an integer.

|  |  |
| --- | --- |
| **Operator** | **Name** |
| & | and |
| | | or |
| ^ | Xor |
| ~ | not |
| << | shift left |
| >> | shift right |

* + Comparison Operators
* Comparison operators are operators comparing two values.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Description/Function** |
| $firstvalue == $secondvalue | equal | $firstvalue is equal to $secondvalue |
| $firstvalue === $secondvalue | identical | $firstvalue is equal to $secondvalue, and they have similar type |
| $firstvalue != $secondvalue | not equal | $firstvalue is not equal to $secondvalue |
| $firstvalue <> $secondvalue | not equal | $firstvalue is not equal to $secondvalue |
| $firstvalue !== $secondvalue | not identical | $firstvalue is not equal to $secondvalue, or they don’t have the same type |
| $firstvalue < $secondvalue | less than | $firstvalue is strictly less than $secondvalue |
| $firstvalue > $secondvalue | greater than | $firstvalue is greater less than $secondvalue |
| $firstvalue <= $secondvalue | less than or equal to | $firstvalue is less than or equal $secondvalue |
| $firstvalue >= $secondvalue | greater than or equal to | $firstvalue is greater than or equal $secondvalue |
| $firstvalue <==> $secondvalue | spaceship | An value less than, equal to, or greater than zero when $firstvalue is respectively less than, equal to, or greater than $secondvalue. |

* + Execution Operators
* Backticks (``) are the only execution operator that the PHP supports. PHP executes the content of backticks as a shell command. Backtick operator usage is identical to shell\_exec().

Sample code:

<?php

$output = `ls -al`;

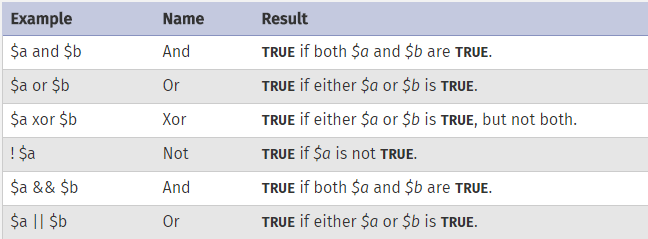
echo "<pre>$output</pre>";

?>

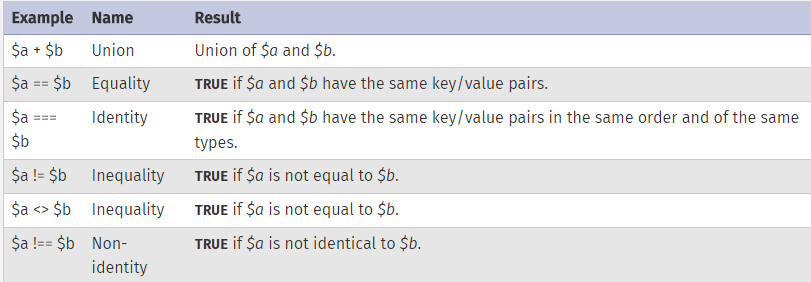
* + Incrementing/Decrementing Operators
* C-style pre-increment and post-increment are what PHP is supporting.

|  |  |  |
| --- | --- | --- |
| **Example** | **Name** | **Effect** |
| ++$a | pre-increment | Increments $a by one, then returns $a. |
| $a++ | post-increment | Returns $a, then increments $a by one |
| --$a | pre-decrement | Decrements $a by one, then returns $a. |
| $a-- | post-decrement | Returns $a, then decrements $a by one |

* + Logical Operators



* + Array Operators



* Control Structures
  + Introduction
  + if
  + else
  + elseif/else if
  + Alternative syntax for control structures
  + while
  + do-while
  + for
  + foreach
  + break
  + continue
  + switch
  + declare
  + return
  + require
  + include
  + require\_once
  + include\_once
  + goto
* [Predefined Variables](http://php.net/manual/en/reserved.variables.php)
  + [Superglobals](http://php.net/manual/en/language.variables.superglobals.php) — Superglobals are built-in variables that are always available in all scopes
  + [$GLOBALS](http://php.net/manual/en/reserved.variables.globals.php) — References all variables available in global scope
  + [$\_SERVER](http://php.net/manual/en/reserved.variables.server.php) — Server and execution environment information
  + [$\_GET](http://php.net/manual/en/reserved.variables.get.php) — HTTP GET variables
  + [$\_POST](http://php.net/manual/en/reserved.variables.post.php) — HTTP POST variables
  + [$\_FILES](http://php.net/manual/en/reserved.variables.files.php) — HTTP File Upload variables
  + [$\_REQUEST](http://php.net/manual/en/reserved.variables.request.php) — HTTP Request variables
  + [$\_SESSION](http://php.net/manual/en/reserved.variables.session.php) — Session variables
  + [$\_ENV](http://php.net/manual/en/reserved.variables.environment.php) — Environment variables
  + [$\_COOKIE](http://php.net/manual/en/reserved.variables.cookies.php) — HTTP Cookies
  + [$php\_errormsg](http://php.net/manual/en/reserved.variables.phperrormsg.php) — The previous error message
  + [$HTTP\_RAW\_POST\_DATA](http://php.net/manual/en/reserved.variables.httprawpostdata.php) — Raw POST data
  + [$http\_response\_header](http://php.net/manual/en/reserved.variables.httpresponseheader.php) — HTTP response headers
  + [$argc](http://php.net/manual/en/reserved.variables.argc.php) — The number of arguments passed to script
  + [$argv](http://php.net/manual/en/reserved.variables.argv.php) — Array of arguments passed to script
* [Context options and parameters](http://php.net/manual/en/context.php)
  + [Socket context options](http://php.net/manual/en/context.socket.php) — Socket context option listing
  + [HTTP context options](http://php.net/manual/en/context.http.php) — HTTP context option listing
  + [FTP context options](http://php.net/manual/en/context.ftp.php) — FTP context option listing
  + [SSL context options](http://php.net/manual/en/context.ssl.php) — SSL context option listing
  + [CURL context options](http://php.net/manual/en/context.curl.php) — CURL context option listing
  + [Phar context options](http://php.net/manual/en/context.phar.php) — Phar context option listing
  + [MongoDB context options](http://php.net/manual/en/context.mongodb.php) — MongoDB context option listing
  + [Context parameters](http://php.net/manual/en/context.params.php) — Context parameter listing
  + [Zip context options](http://php.net/manual/en/context.zip.php) — Zip context option listing
* [Supported Protocols and Wrappers](http://php.net/manual/en/wrappers.php)
  + [file://](http://php.net/manual/en/wrappers.file.php) — Accessing local filesystem
  + [http://](http://php.net/manual/en/wrappers.http.php) — Accessing HTTP(s) URLs
  + [ftp://](http://php.net/manual/en/wrappers.ftp.php) — Accessing FTP(s) URLs
  + [php://](http://php.net/manual/en/wrappers.php.php) — Accessing various I/O streams
  + [zlib://](http://php.net/manual/en/wrappers.compression.php) — Compression Streams
  + [data://](http://php.net/manual/en/wrappers.data.php) — Data (RFC 2397)
  + [glob://](http://php.net/manual/en/wrappers.glob.php) — Find pathnames matching pattern
  + [phar://](http://php.net/manual/en/wrappers.phar.php) — PHP Archive
  + [ssh2://](http://php.net/manual/en/wrappers.ssh2.php) — Secure Shell 2
  + [rar://](http://php.net/manual/en/wrappers.rar.php) — RAR
  + [ogg://](http://php.net/manual/en/wrappers.audio.php) — Audio streams
  + [expect://](http://php.net/manual/en/wrappers.expect.php) — Process Interaction Streams