ZCE 2017-PHP Exam - 01

December 8, 2020

1 Andrew Beak - PHP 7 Zend Certification Study Guide (Ace the ZCE 2017-PHP Exam) - Introduction + Chapter 1

1.1 Introduction

Welcome to what I hope is an accessible reference that helps you quickly find and learn relevant facts about the PHP language. I'm writing it with the following readers in mind:

- Intermediate PHP programmers with two or three years of experience who are hoping to sit for the Zend 2017 certification exams in the PHP 7.1 language.
- Programmers who are proficient in another language but want a quick reference book to dive into PHP.

This book is specifically not an introduction to programming and no attempt is made to introduce basic topics such as what a variable is. It is purely a reference to learn the idiosyncrasies of PHP and serve as a guide toward certifying.

Additionally, it does not intend to replace the PHP manual but rather to focus the reader's attention on aspects of PHP relevant to the Zend certification exams. To that end, I make liberal use of footnotes linking to manual pages and other reference pages.

This guide cannot possibly cover the full depth of the PHP manual, and you cannot consider yourself prepared until you have worked through the manual pages. It is essential to follow the footnote links, read the page, and even follow some links through the PHP manual related to the topic.

This book focuses on PHP 7.1 and in order to keep focused on the exam syllabus does not attempt to exhaustively explain what features are available in which version of PHP. You can assume that if a feature is referenced in this book then it is available in PHP 7.1.

You won't need to know when a feature was introduced into PHP, but you will need to know what features are deprecated in PHP 7.1. I've tried to keep that in mind when giving information about the availability of a feature.

Wherever possible, I'll flag where something has changed from PHP 5.6, especially where it is not backward compatible.

This book tries to avoid having an opinion about coding standards such as PSR2, but it is strongly recommended that a practicing developer learn these in parallel. You won't be asked about them on the exam and so I won't teach them here.

1.2 The Zend Certification Syllabus

[Contents omited]

Zend certification is one of the few industry certifications that is widely recognized as an established benchmark for proficiency in PHP. Part of why this certification is so well respected is that it is not easy to attain. To pass your exam, you need to have an in-depth knowledge of a wide area of topics in PHP programming.

This book aims to highlight the topics that are important in the certification examination. I've included a bunch of quizzes that test the topics, and if you notice any gaps in your knowledge, you should make sure to go carefully through the relevant sections in the manual.

The actual exam consists of about 70 questions drawn from a pool. Questions will vary in difficulty and will sometimes require you to apply the principles from more than one area of the syllabus to solve them.

There are ten topics covered in the examination. Zend places varying amount of importance to the topics in the exam. You can find a list of exam topics on the Zend web site.

You'll notice that this guide has a chapter for each of the topics in the list:

Very Important	Average Importance	Lower Importance
PHP basics	Functions	Databases and SQL
Object-Oriented PHP	Web Features	Data Format and Types
Security	Strings	Input-Output
	Arrays	
	Error Handling	

The sections on PHP basics, object-oriented programming, and security are regarded as being the most important and are given the most weight in the exam.

There are a lot of tricky questions in the PHP Basics section, so don't be fooled by the inclusion of the word "basics" in the topic title. It's one of the three most important topics in the exam syllabus, so don't skimp on the time you spend finding the gaps in your knowledge.

The sections on databases, data formats and types, and input-output are considered less important. You'll still need to know them, however, as exam questions will often rely on a broad knowledge of PHP and may require you to apply knowledge from different topics.

The exam has a mixture of multiple choice and free text questions. You'll be required to code short snippets to answer some questions.

Although you won't be required to have an encyclopedic knowledge of the PHP manual, you will be asked questions about function parameters, what function to use in a particular situation, common classes, exceptions, and other elements of PHP.

Unless they note otherwise, all questions assume that you are working with PHP 7.1 that has been configured with the recommended settings. You can assume that error display is set on and that error reporting is set to E_ALL .

You can mark questions for review and come back to them later. Don't spend more time than is

allocated to a question; That just makes the following questions more difficult because you have less time!

Finally, there is no negative marking in the exam. You won't be penalized for an incorrect answer, so taking a guess works in your favor.

2 1. PHP Basics

2.0.1 Introduction

This chapter focuses on the nitty gritty details of PHP as a language. This is a big chapter and it covers one of the most important aspects of the exam. The best advice that I can give is not to skim over it. There are a lot of places where PHP has some peculiarities and even with a few years of experience you might not have encountered them all.

2.0.2 Basic Language Features

All statements in PHP must be terminated with a semicolon. An exception to this is if the statement happens to be the last statement before a closing tag.

Whitespace has no semantic meaning in PHP. There is no need to line code up, but most coding standards enforce this to improve code readability. Whitespace may not appear in the middle of function names, variable names, or keywords.

Multiple statements are allowed on a single line.

Code blocks are denoted by the brace symbols { }.

PHP language construct and function names are not case-sensitive, but variable and constant names are.

[3]:
```
echo "Hello World"; // works
$variable = "Hello World";
assert($variable !== "Hello World", 'esto es un mensaje de error en aserción');
echo $VARIABLE; // won't work
```

[3]:
```
Hello World
```

[3]:
```
ErrorException with message 'estoes un mensaje de error en
aserción'
```

2.0.3 Inserting PHP into Web Pages

Although PHP is a general scripting language and can be used for other purposes, it is most commonly deployed as a server-side scripting language used for building web pages.

The PHP parser does not parse anything that is not included in the tags that indicate PHP script. Content outside of PHP tags is simply output without inspection. This allows PHP to be embedded in HTML.

There are several ways to delimit PHP script, but only the first two in this table are commonly used:

Type	Open	Close	Note
Standard			
Echo	<?=	?>	
Short	<?	?>	Deprecated
ASP	<%	%>	Deprecated

The echo tag allows you to easily echo a PHP variable and the shortened tag makes your HTML document easier to read. It is commonly used in templates where you want to output several values into various positions on a page. Using the short syntax keeps your template code much neater.

2.1 Language Constructs

Language constructs are different from functions in that they are baked right into the language.

Language constructs can be understood directly by the parser and do not need to be broken down. Functions, on the other hand, are mapped and simplified to a set of language constructs before they are parsed.

Language constructs are not functions, and so cannot be used as a callback function. They follow rules that are different from functions when it comes to parameters and the use of parentheses.

For example, echo doesn't always need parentheses when you call it and, if you call it with more than one argument, then you can't use parentheses.

[ ]:
```
echo "hello\r\n"; // one parameter, no brackets
// two parameters, brackets (syntax error)
// echo('hello', 'world');
// two parameters, no brackets
echo 'hello', 'world'; // no concatena
echo 'hello ' . 'world'; // el . concatena
```

Furthermore, echo does not return a value, whereas every function will always return a value (or null).

The PHP Manual page on reserved keywords has a complete list, but here are some of the constructs that you should be familiar with:

Construct	Used For
assert	Debug command to test a condition and do something if it is not true.
echo	Outputting a value to stdout.
print	Outputting a value to stdout.
exit	Optionally outputting a message and terminating the program.
die	This is an alias for exit.
return	Terminates a function and returns control to the calling scope, or if called in the global scope, terminates the program.

Construct	Used For
include	Includes a file and evaluates it. PHP will issue a warning if the file is not found or cannot be read.
include_once	If you specify include_once then PHP will make sure that it includes the file only once.
require	PHP will include a file and evaluate it. If the file is not found or cannot be read, then a fatal error will be generated.
require_once	As for include_once, but a fatal error will be generated instead of a warning.
eval	The argument is evaluated as PHP and affects the calling scope.
empty	Returns a Boolean value depending on whether the variable is empty or not. Empty variables include null variables, empty strings, arrays with no elements, numeric values of 0, a string value of 0, and Boolean values of false.
isset	Returns true if the variable has been set and false otherwise.
unset	Clears a variable.
list	Assigns multiple variables at one time from an array.

One possible tricky exam question that might come up is in understanding the small difference between print and echo. The echo construct does not return a value, not even null, and so is not suitable for use inside an expression. The print construct will however return a value.

The reason not to use include_once() and require_once() all the time is a performance issue. PHP tracks a list of files that has been included to support the functionality of these functions. This requires memory so these functions are rather used when they are necessary and not in favor of include or require.

2.2 Comments

There are three styles to mark comments:

[ ]:
```
# Perl style comments

// C style comments

/*

  Multiline comment

*/
```

API documentation can additionally conform to external standards such as those used by the **PHPDocumentor** project. This tool examines your API style comments and automatically creates documentation for you.

API documentation looks very similar to multiline comments:

[ ]:
```
/**

   API documentation has two asterisks, this is not a PHP

  syntax distinction, but is just a convention.

*/
```

2.3 Representing Numbers

There are four ways in which an integer may be expressed in a PHP script:

Notation	Example	Note
Decimal	1234	
Binary	0b10011010010	Identified by leading 0b or 0B
Octal	02322	Identified by leading 0
Hexadecimal	0x4D2	Identified by leading 0x or 0X

Floating point numbers (called doubles in some other languages) can be expressed either in standard decimal format or in exponential format.

Form	Example
Decimal	123.456
Exponential	0.123456e3 or 0.123456E3

Note

The letter "e" in the exponential form is case-insensitive, as are the other letters used in the integer formats.

[ ]:
```
echo 1234;
echo 0B10011010010;
echo 02322;
echo 0x4D2;
```

[ ]:
```
echo 123.456;
echo 0.123456e3;
echo 0.123456E3;
```

2.4 Variables

In this section, I'm going to be focusing on how PHP handles variables. I'm assuming that you've had enough experience with PHP that I don't need to explain what variables are or how to use them. We'll be looking at the various types of variables PHP offers, how to change the type of a variable, and how to check if a variable is set or not.

2.4.1 Variable Types

PHP is a loosely typed language. It is important not to think that PHP variables don't have a type. They most definitely do, it's just that they may change type during runtime and don't need their type to be declared explicitly when initialized.

PHP will implicitly cast the variable to the data type required for an operation. For example, if an operation requires a number, such as the addition (+) operation, then PHP will convert the operands into a numeric format.

You'll be introduced to type juggling in the **"Casting Variables"** section and you'll need to know the rules PHP follows when changing a variable type. For now, you just need to know that PHP variables have a type, that type can change, and although you can explicitly change the type PHP does this implicitly for you.

PHP has three categories of variable—scalars, composite, and resources. A scalar variable is one that can only hold one value at a time. Composite variables can contain several values at a time.

A resource variable points to something not native to PHP like a handle provided by the OS to a file or a database connection. These variables cannot be cast.

Finally, PHP has the null type, which is used for variables that have not had a value set to them. You can also assign the null value to a variable, but you cannot cast to a null type in PHP 7.1.

Scalar Types There are four scalar types:

Type	Alias	Contains
Boolean	bool	True or False
Integer	int	A signed numeric integer
Float		A signed numeric double or float data
String		An ordered collection of binary data

Some types have aliases. For example, consider this code that shows that bool is an alias for boolean:

[ ]:
```
$a = (boolean)true;
$b = (bool)true;
var_dump($a === $b);// bool(true)
var_dump($a !== $b); // bool(false)
```

Strings in PHP are not simply a list of characters. Internally PHP strings contain information about their length and are not null terminated. This means that they may contain binary information such as an image file that has been read from disk. In other words, *PHP strings are binary safe.*

Composite Types There are two composite types: **arrays** and **objects**. Each of these has its own section in this book.

Casting Variables This is a very important section of understanding PHP and even very experienced developers may not be aware of some of the rules that PHP uses to cast variables.

PHP implicitly casts variables to the type required to perform an operation.

It is also possible to explicitly cast variables using one of two options:

- Use a casting operator
- Use a PHP function

Casting operators are used by putting the name of the data type you want to cast into brackets before the variable name. For example:

[ ]:
```
$a = '123';    // $a is a string
var_dump($a);
$a = (int) $a;   // $a is now an integer
var_dump($a);
$a = (bool) $a;   // $a is now Boolean and is true
var_dump($a);
die;
```

You can cast a **variable to null**, as in the following example, but this behavior is deprecated in PHP 7.2 so you shouldn't do it even though PHP 7.1 supports it.

[ ]:
```
$a = "Hello World";
$a = (unset)$a; // Deprecated in PHP 7.2
var_dump($a);  // NULL
```

There are also PHP functions that will convert a variable to a data type. These are named in a way that is self-documenting: **floatval, intval, strval & boolval**.

Additionally, the **intdiv** function will potentially cast a double to an integer when it returns the integer result of dividing two integers.

You can also call the settype function on a variable that takes the desired data type as a second argument.

There are some rules that need to be remembered regarding how variables are cast in PHP. You should read the manual page on type juggling carefully, because there are many trips and traps in type juggling. Also make sure that you read the pages linked to from the type juggling page.

Instead of exhaustively listing the rules, I'll focus on some of the rules that may be counter-intuitive or are commonly mistaken.

Casting from *float to integer* does not round the value up or down, but rather truncates the decimal portion.

[ ]:
```
$a = 1234.56; // float
var_dump((int)$a);  // 1234 (not 1235)
$a = -1234.56;
echo (int)$a;  // -1234
```

Some general rules for casting to Boolean are that:

- Empty arrays and strings are cast to false.

- Strings always evaluate to Boolean true unless they have a value that's considered "empty" by PHP.
- Strings containing numbers evaluate to true if the number is not zero. Recall that such strings return false when the empty() function is called on them.
- Any integer (or float) that is non-zero is true, so negative numbers are true.

Objects can have the magic method ___toString() defined on them. This can be overloaded if you want to have a custom way to cast your object to string. We look at this in the section on "Casting Objects to String".

Converting a string to a number results in 0 unless the string begins with valid numeric data (see the PHP Manual for more detail). By default, the variable type of the cast number will be integer, unless an exponent or decimal point is encountered, in which case it will be a float.

Here is an example script that shows some string conversions:

[ ]:
```
$examples = [
  '12',
  '12.30',
  '7.2e2'
];

foreach ($examples as $example) {
  $result = 0+$example;
  var_dump($result);
}
```

Floats and Integers Be very careful when casting between floats and integers. The PHP Manual has a very good example of how internal implementation details of numeric types can have counter-intuitive results:

[ ]:
```
echo (int) ( (0.1+0.7) * 10 ); // 7
echo (int) ( (0.1+0.5) * 10); // 6
```

One would expect the first example to display 8, but in fact the internal floating-point representation is just slightly less than 8.

When PHP converts a floating point number to integer it rounds toward zero, so it becomes 7.

The reason behind this is that some numbers are rational in base 10 but are irrational in base 2. Although 0.7 can be expressed as a rational number in base 10, when expressed in base 2 it is irrational. Because there are a limited number of bits available to store the number, it is inevitable that some loss of precision will occur.

PHP integers are always signed. The range of values that an integer can take will depend on the system PHP is running on.

You can determine the size of an integer in bytes at runtime by querying the constant PHP_INT_SIZE. The constants PHP_INT_MAX and PHP_INT_MIN will give you the maximum and minimum values that can be stored in an integer, respectively. There are similar constants for other numeric types. They are listed in the PHP Manual page on reserved constants.

Caution

You should not rely on floats precision up to the last digit.

You should avoid testing floats directly for equality and rather test if they are the same up to a given degree of precision, as in this example:

[ ]:
```
$pi = 3.14159625;
$indiana = 3.2;
$epsilon = 0.01; // degree of error
if(abs($pi - $indiana) < $epsilon) {
  echo "Those values look the same to me";
} else {
  echo "Those values are different";
}
```

This code is checking if the values are the same to five degrees of precision. This script will output Those values are different because the difference is greater than the degree of error that we defined.

2.4.2 Naming Variables

PHP variables begin with the dollar symbol $ and PHP variable names adhere to the following rules:

- Names are case sensitive
- Names may contain letters, numbers, and the underscore character
- Names may not begin with a number

Coding conventions differ on the use of camelCase, StudlyCase, or snake_case, but all of these formats are valid PHP variable name formats.

PHP also allows for variable variable names. This is best illustrated by example:

[ ]:
```
$a = 'foo';
$$a = 'bar'; // $a is 'foo', so variable $foo is set
echo $foo;  // bar
```

PHP 7 will always evaluate access strictly left to right. Older versions had a complicated set of rules to determine how it would evaluate this sort of syntax. Happily, PHP 7 is simpler and consistent and I won't worry about explaining older versions.

Here is a more complicated example that illustrates how PHP evaluates from left to right:

[ ]:
```
$a = 'foo';
$$a = 'Murky code';
var_dump($foo);
die;
```

There are several caveats to using variable variable names. They could impact on your code security and can also make your code a little murky to read.

2.4.3 Checking If a Variable Has Been Set

The command **isset()** will return true if a variable has been set and false otherwise. It is preferable to use this function instead of checking if the variable is null because it won't cause PHP to generate a warning.

The command **empty()** will return true if a variable is not set and will not generate a warning. This is not a bulletproof way to test if a variable is set.

Note

Remember that the string "0" is considered empty, but is actually set.

Variables become unset when they become out of scope and you can use the command **unset()** to manually clear a variable. We'll see later in the book that the garbage collector is responsible for freeing up the memory allocated to variables that have been unset.

[ ]:
```
$a = '';
var_dump(isset($a));
var_dump(empty($a));

$b = 'hola mundo';
var_dump(isset($b));
var_dump(empty($b));

unset($a);
var_dump(isset($a));
var_dump(empty($a));
echo $a;

die;
```

Data Type	Category	Description
int	Scalar	Integer
float	Scalar	Floating-point number
bool	Scalar	Boolean value
string	Scalar	Series of characters
array	Composite	Collection of values
object	Composite	User-defined data type
resource	Special	External resource
callable	Special	Function or method
null	Special	No value

2.5 Constants

are similar to variables but are immutable. They have the same naming rules as variables, but by convention will have uppercase names.

They can be defined using the define function as shown:

[ ]:
```
define('PI', 3.142);
echo PI;
define('UNITS', ['MILES_CONVERSION' => 1.6, 'INCHES_CONVERSION' => '2.54']);
echo "5km in miles is " . 5 * UNITS['MILES_CONVERSION'];
```