PassQueen
a.k.a dumpsqueen.com

Zend PHP 5 Certification

Zend 200-500

Version Demo

Total Demo Questions: 25

Total Premium Questions: 259

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, PHP Basics	21
Topic 2, Functions	14
Topic 3, Data Format and Types	5
Topic 4, Strings	27
Topic 5, Arrays	23
Topic 6, Object Oriented Programming	52
Topic 7, Input/Output	39
Topic 8, Error Handling	3
Topic 9, Security	25
Topic 10, Databases	19
Topic 11, XML and Web Services	31
Total	259

QUESTION NO: 1

What type of class definition can be used to define multiple inheritance?

A. Class

B. Abstract

C. Interface

D. Final

ANSWER: C

Explanation:

Interface is correct because PHP does not permit a class to extend more than one parent class, but it does allow a class to implement multiple interfaces. An interface defines a contract: it declares the public methods that implementing classes must provide, without requiring those classes to share a single concrete parent implementation. A class can therefore combine capabilities specified by several independent interfaces using a comma-separated `implements` list, such as `class Report implements Renderable, Exportable`. This provides the PHP object model's supported mechanism for achieving the practical benefit commonly associated with multiple inheritance: one class can conform to multiple API contracts at the same time.

Interfaces may also build on multiple contracts themselves: an interface can extend more than one interface. This lets a broader contract aggregate several smaller, focused contracts while preserving clear method requirements for any implementing class. In PHP 5, this is especially important because native class inheritance is deliberately single inheritance. The PHP manual documents that a class may implement one or more interfaces and that interfaces can extend other interfaces. See the PHP manual on object interfaces and the PHP manual on inheritance.

QUESTION NO: 2

Which of the following are valid SoapClient calls? (Choose 2)

A. $client = new SoapClient("weather.wsdl");

B. $client = new SoapClient;

C. $client = new SoapClient(null, array("location" => "http://example.com/weather", "uri" => "http://test-uri.com/"));

D. $client = new SoapClient(null, array());

ANSWER: A C

Explanation:

`$client = new SoapClient("weather.wsdl");` is valid because the first constructor argument supplies a WSDL document. In WSDL mode, `SoapClient` reads the service definition from that file or URL, including the available operations, endpoint information, message formats, and namespace details. The optional second argument is not required when the WSDL provides the necessary service metadata. This is the standard SOAP client construction pattern for a WSDL-described service.

`$client = new SoapClient(null, array("location" => "http://example.com/weather", "uri" => "http://test-uri.com/"));` is also valid. Passing `null` as the WSDL explicitly selects non-WSDL mode. In that mode, PHP cannot discover service details automatically, so the options provide the required endpoint through `location` and the SOAP namespace through `uri`. With these values, the client has the information needed to construct and send SOAP requests without a WSDL file. The PHP SOAP extension documents the constructor's WSDL and non-WSDL forms, including the required non-WSDL configuration fields. See the PHP SoapClient constructor documentation and the PHP SOAP extension manual.

QUESTION NO: 3

Type hinting in PHP allows the identification of the following variable types: (Choose 2)

A. String

B. Integer

C. Array

D. Any class or interface type

ANSWER: C D

Explanation:

In the PHP 5 type-hinting model tested by Zend PHP 5 certification, a function or method parameter can be constrained to an `array` or to an object of a specified class or interface type. An `array` hint requires the caller to supply an array value. A class name hint requires an instance of that class, while an interface name hint accepts an object implementing that interface. These declarations allow PHP to validate an argument at the function boundary and help make an API's expected input explicit. For example, `function process(array $items)` declares an array parameter, and `function save(Persistable $item)` declares a parameter that must satisfy the `Persistable` interface. PHP's documentation describes these parameter constraints under class/interface type declarations and array type declarations. The relevant PHP manual material is available at PHP: Type Declarations and PHP: Type Declarations in Function Arguments.

QUESTION NO: 4

What function is ideal for outputting contents of a static file to screen?

A. file_get_contents()

B. readfile()

C. fread()

D. include()

E. require()

F. file()

ANSWER: B

Explanation:

`readfile()` is designed to read a file and write its contents directly to the output buffer, making it especially suitable for sending a static file's contents to the browser. It accepts a filename or supported stream URL and returns the number of bytes read, or `false` if it fails. Because the function streams the content as output rather than first requiring application code to retrieve and separately echo a complete string, it is a concise, purpose-built choice when the goal is simply to display or deliver a static file. For example, `readfile('document.txt');` outputs the contents of `document.txt`. In real download handling, applications commonly send appropriate HTTP headers, such as `Content-Type` and `Content-Disposition`, before calling `readfile()`, ensuring that the client interprets the static content as intended. PHP's documentation specifically describes `readfile()` as outputting a file, while also noting its support for stream wrappers where enabled. See the official PHP manual for readfile() and its related filesystem and stream configuration.

QUESTION NO: 5

Which of the following techniques ensures that a value submitted in a form can only be yes or no?

A. Use a select list that only lets the user choose between yes and no.

B. Use a hidden input field that has a value of yes or no.

C. Enable the safe_mode configuration directive.

D. None of the above.

ANSWER: D

Explanation:

“None of the above.” is correct because ensuring that a submitted value is restricted to `yes` or `no` requires validation performed by the server after the request is received. HTML form controls only influence the normal browser interface; they do not establish a trustworthy constraint on the HTTP request. A user can alter submitted parameters using browser developer tools, a custom client, or a manually constructed request. The application must therefore treat all incoming data as untrusted and explicitly allow only the expected values before using the value in business logic, storage, or output.

For example, PHP code can check whether the received value is exactly one of the permitted strings, such as with `in_array($value, array('yes', 'no'), true)`, and reject or normalize anything else. Alternatively, an application may use a boolean-oriented validation/filtering approach where appropriate. The key requirement is a server-side allowlist: the server, rather than the submitted form markup, makes the final decision about whether the input is valid. PHP’s documentation describes the filtering facilities for external input at PHP Filter extension; its security guidance also emphasizes validating external input at PHP Security.

QUESTION NO: 6

Which of the following keywords is not new in PHP 5?

A. implements

B. instanceof

C. static

D. abstract

ANSWER: C

Explanation:

`static` is the keyword that was not introduced by PHP 5. PHP already supported static local variables before PHP 5: a variable declared with `static` inside a function retains its value between calls to that function. This longstanding language feature made `static` an established PHP keyword prior to the PHP 5 release.

PHP 5 substantially expanded PHP’s object-oriented model, but that expansion did not make the `static` keyword itself new. In PHP 5, the keyword also gained important relevance in the enhanced object-oriented syntax, including static properties and methods, yet its pre-existing use for function-local static variables means it does not belong among keywords newly introduced in PHP 5. The PHP manual’s scope documentation describes static variables and their retained value across function calls, while the PHP 5 object model documentation provides the context for the major object-oriented additions in that version. See PHP manual: Variable scope and PHP manual: Objects (PHP 5).

QUESTION NO: 7

When checking whether two English words are pronounced alike, which function should be used for the best possible result?

A. levenshtein()

B. metaphone()

C. similar_text()

D. soundex()

ANSWER: B

Explanation:

`metaphone()` is correct because it converts an English word into a phonetic key intended to represent its pronunciation rather than its literal spelling. Two words can therefore be compared by generating their Metaphone keys and checking whether those keys match. This is particularly useful for names and words that sound alike despite having different character sequences.

PHP's documentation describes `metaphone()` as calculating the Metaphone key of a string and identifies Metaphone as a phonetic algorithm for English words. Its documentation also notes that the algorithm is a better version of the Soundex algorithm, making it the appropriate choice where the question asks for the best available result among these functions. The function accepts the word to encode and can optionally receive a maximum phoneme length; the resulting key is then suitable for pronunciation-oriented matching. For example, code can compare `metaphone($firstWord)` with `metaphone($secondWord)` to test whether the words map to the same phonetic representation.

See the PHP manual for metaphone() and its overview of phonetic string functions.

QUESTION NO: 8

Which of the following code snippets is correct? (Choose 2)

a)

interface Drawable {

abstract function draw();

}

b)

interface Point {

function getX();

function getY();

}

c)

interface Line extends Point {

function getX2();

function getY2();

}

d)

interface Circle implements Point {

function getRadius();

}

A. interface Drawable {abstract function draw();}

B. interface Point {function getX();function getY();}

C. interface Line extends Point {function getX2();function getY2();}

D. interface Circle implements Point {function getRadius();}

ANSWER: B C

Explanation:

The declaration `interface Point { function getX(); function getY(); }` is valid PHP interface syntax. An interface defines a contract: each method declaration states a public method that any implementing class must provide. In PHP 5, methods declared in an interface are implicitly public and abstract, so declarations with a method name, parameter list, and terminating semicolon correctly establish the required API. This makes `Point` a valid reusable contract for objects that expose coordinate values.

The declaration `interface Line extends Point { function getX2(); function getY2(); }` is also valid. PHP interfaces may extend another interface with `extends`. The derived interface inherits every method in the parent contract and may add further method requirements. Consequently, a class implementing `Line` must supply implementations for `getX()`, `getY()`, `getX2()`, and `getY2()`, which models a line as having two points. This is the intended mechanism for interface inheritance in PHP. See the PHP manual's Interfaces documentation and the PHP OOP basics documentation.

QUESTION NO: 9

After executing a query on a database server, PHP offers several functions to read the resulting lines, such as mysqli_fetch_assoc, pg_fetch_row, oci_fetch,etc.). If such functions do not return any rows, it means: (Choose 2)

A. a SELECT statement returned no rows

B. the transaction has been rolled back

C. the connection to the database server was disconnected during query execution

D. the query was too slow to execute

ANSWER: A C

Explanation:

A successful `SELECT` can legitimately produce an empty result set when no database records satisfy its search condition. In that situation, a row-fetching call has no next row available and returns its driver-specific end-of-result indication. For example, `mysqli_fetch_assoc()` returns `null` when there are no more rows in a result set, while older PHP database APIs commonly use `false` for an unavailable row. This is a normal outcome of querying and must be handled separately from processing a row.

A database connection being disconnected during query execution can also leave PHP without a usable result from which to fetch rows. Connection loss is a database-operation failure: the query may not complete, its result may not be delivered to the client, and fetch-related processing therefore cannot produce a row. Robust PHP code should check the result returned by the query operation, inspect the database driver's error reporting where appropriate, and handle connection failures before attempting to iterate over rows. The exact return value and error behavior vary by extension and PHP version, but the distinction between an empty result and a failed database operation is fundamental. See the PHP documentation for mysqli_result::fetch_assoc() and mysqli::query().

QUESTION NO: 10

What is the output of the following script?

```
1

2 function fibonacci ($x1, $x2)

3 {

4 return $x1 + $x2;
```

```
5 }
6
7 $x1 = 0;
8 $x2 = 1;
9
10 for ($i = 0; $i < 10; $i++) {
11 echo fibonacci($x1, $x2) . ',';
12 }
13 ?>
```

A. 1,2,3,4,5,6,7,8,9

B. 1,2,3,4,5,6,7,8,9,10,

C. 1,2,3,5,8,13,21,34,55,89,

D. 1,1,1,1,1,1,1,1,1,1,

ANSWER: D

Explanation:

The output is `1,1,1,1,1,1,1,1,1,1,`. Before the loop begins, `$x1` is assigned `0` and `$x2` is assigned `1`. Each of the ten loop iterations calls `fibonacci($x1, $x2)`. The function returns the sum of its two parameters, which is always `0 + 1`, or `1`. The expression concatenates that returned integer with a comma and sends the resulting text to output on every iteration.

Although the function is named `fibonacci`, it contains no assignment that updates either variable for a subsequent calculation. In PHP, function arguments are passed by value unless the parameter is explicitly declared by reference using `&`. Here, neither parameter is a reference, and the function merely calculates and returns a value. Consequently, the outer `$x1` and `$x2` variables retain their original values throughout the loop. Since the loop condition permits exactly ten executions, the same value and trailing comma are printed ten times. PHP's function documentation describes parameter passing and return values at PHP: Function arguments and PHP: Returning values.

QUESTION NO: 11

Which of the following functions are used to escape data within the context of HTML?

(Choose 2)

A. htmlentities()

B. addslashes()

C. stripslashes()

D. strip_tags()

E. htmlspecialchars()

ANSWER: A E

Explanation:

`htmlentities()` and `htmlspecialchars()` are PHP output-encoding functions intended for safely placing untrusted text into an HTML document. They transform characters that have special significance in markup into HTML entities, so

browser parsing treats the supplied value as visible text rather than as tags, attributes, or other markup syntax. This is an essential output-encoding step when displaying data originating from users, databases, request parameters, or other untrusted sources.

`htmlspecialchars()` encodes the principal HTML-sensitive characters, including the ampersand, angle brackets, quotation mark, apostrophe when the applicable flags are used, and greater-than sign. It is generally the preferred routine for ordinary HTML text output because it preserves non-special characters in their original form. `htmlentities()` also performs HTML-safe encoding and can encode a broader set of characters according to the selected document type and character encoding. Both functions should be called with an explicit character encoding such as `UTF-8`, and with quote encoding enabled when values may be emitted in quoted HTML attributes. PHP's documentation describes `htmlspecialchars()` as converting special characters to HTML entities and documents the broader conversion behavior of `htmlentities()`. See PHP: htmlspecialchars and PHP: htmlentities.

QUESTION NO: 12

How can the id attribute of the 2nd baz element from the XML string below be retrieved from the SimpleXML object found inside $xml?

One

Two

A. $xml->getElementById('2');

B. $xml->foo->bar->baz[2]['id'] C. $xml->foo->baz[2]['id']

C. $xml->foo->bar->baz[1]['id']

D. $xml->bar->baz[1]['id']

ANSWER: D

Explanation:

`$xml->bar->baz[1]['id']` retrieves the requested attribute when `$xml` is the SimpleXML object representing the document's `<foo>` root element. SimpleXML exposes child elements through property syntax, so `$xml->bar` reaches the `<bar>` child and `->baz` selects its collection of `<baz>` children. Collections in SimpleXML use zero-based numeric indexes: the first `<baz>` element is at index `0`, making the second one available at index `1`. Once that element has been selected, array syntax with the attribute name, `['id']`, accesses its `id` XML attribute. The resulting value is a SimpleXMLElement representing the attribute value and can be cast to a string when a plain PHP string is needed, for example `(string) $xml->bar->baz[1]['id']`, which yields `"2"`. PHP's SimpleXML documentation illustrates both accessing repeated child nodes by numeric offset and retrieving attributes using array notation. See the PHP SimpleXML basic usage examples and the SimpleXMLElement class reference.

QUESTION NO: 13

Which of the listed changes would you make to the following PHP 4 code in order to make it most compliant with PHP 5? (Choose 2)

class Car {

var $model;

function Car($model) {

$this->model = $model;

} function toString() {

return "I drive a $this->model.";

}}

$c = new Car('Dodge');

echo $c->toString();

?>

A. Change var to public or private

B. Change function Car to function __construct

C. Change "I drive a $this->model." to "I drive a {$this->model}."

D. Change function toString()to static function toString()

ANSWER: A B

Explanation:

PHP 5 introduced a substantially revised object model, so the most appropriate modernization is to use explicit property visibility and the standard constructor syntax. “Change var to public or private” is correct because PHP 5 supports visibility modifiers for class properties. Declaring `public $model;` preserves the practical accessibility of the original `var $model;` declaration, while `private $model;` is appropriate when the property should be encapsulated and accessed through class methods. Visibility makes the intended access contract explicit, which is a central PHP 5 object-oriented practice.

“Change function Car to function __construct” is also correct when expressed as `function __construct($model)`. PHP 5 recognizes `__construct()` as the constructor method and invokes it when an object is created with `new Car('Dodge')`. Using the magic constructor name avoids reliance on the older PHP 4 convention in which a method named after its class acts as a constructor. The string interpolation already works for an object property in this context, and the instance method needs access to `$this`, so it remains an ordinary non-static method. See the PHP manual on constructors and destructors and object properties and visibility.

QUESTION NO: 14

How many elements does the $matches array contain after the following function call is performed?

preg_match('/^(\d{1,2}([a-z]+))(?:\s*)\S+ (?=200[0-9])/', '21st March

2006', $matches);

A. 1

B. 2

C. 3

D. 4

ANSWER: C

Explanation:

The correct answer is **3**. When `preg_match()` succeeds without the `PREG_UNMATCHED_AS_NULL` or `PREG_OFFSET_CAPTURE` flags, it populates `$matches` with the complete matched text at index `0`, followed by one entry for each capturing parenthesized subpattern. Here, the outer capturing group `(\d{1,2}([a-z]+))` captures `21st`, and its nested capturing group `([a-z]+)` captures `st`. The complete pattern match is also stored, covering the date-prefix text through the space immediately before `2006`. Therefore, the array consists of the full match, the outer captured value, and the nested captured value.

The group written as `(?:\s*)` is explicitly non-capturing, so it participates in matching whitespace but does not create an entry in `$matches`. Likewise, `(?=200[0-9])` is a positive lookahead: it verifies that the next characters form a year from 2000 through 2009 without consuming them or adding a capture. PHP documents that the matches array contains the full pattern match and captured subpatterns; see PHP: preg_match() and PHP: Subpatterns.

QUESTION NO: 15

Given the following array:

$a = array(28, 15, 77, 43);

Which function will remove the value 28 from $a?

A. array_shift()

B. array_pop()

C. array_pull()

D. array_unshift()

ANSWER: A

Explanation:

`array_shift()` is correct because it removes and returns the first element of an array. In the supplied array, `28` is the first value: `$a = array(28, 15, 77, 43);`. Calling `array_shift($a)` therefore removes `28` directly from `$a`, leaving the array with the values `15`, `77`, and `43`. The function operates on the array itself because its array argument is passed by reference, so the change persists after the call. It also returns the removed value, meaning that `$removed = array_shift($a);` assigns `28` to `$removed` while updating `$a`. For numerically indexed arrays, PHP adjusts the remaining numeric keys after shifting so that they are reindexed starting at zero. This behavior precisely matches the requirement to remove the leading value from the given array. PHP's official manual documents that `array_shift()` shifts the first value off an array and shortens it by one element: PHP: array_shift. Additional array-function reference material is available in the PHP Arrays manual.

QUESTION NO: 16

What visibility denies access to properties and methods outside of the class?

A. static

B. protected

C. private

D. public

E. const

ANSWER: C

Explanation:

The `private` visibility modifier denies access to a class property or method from outside the class that declares it. A private member is intended for implementation details that only the class itself should read, modify, or invoke. For example, a class can keep internal state in a private property and expose controlled behavior through public methods, preserving encapsulation. In PHP, visibility applies to both properties and methods: when a member is declared `private`, code outside the declaring class cannot access it directly, including code in subclasses. The declaring class may, however, use its own private members normally through `$this` or self-references. This supports a clear class interface by separating externally usable behavior from internal mechanisms. The PHP language documentation defines private members as accessible only

from within the class that defines them. See the PHP manual's visibility documentation and its overview of classes and objects.

QUESTION NO: 17

What does the __FILE__ constant contain?

A. The filename of the current script.

B. The full path to the current script.

C. The URL of the request made.

D. The path to the main script.

ANSWER: B

Explanation:

The full path to the current script is correct. In PHP, `__FILE__` is a magic constant whose value is the full pathname and filename of the file in which it appears. Its value is determined by PHP while processing the source file, so it identifies the current file rather than relying on request information or the entry script. For example, if code containing `__FILE__` is located in `/var/www/app/includes/config.php`, evaluating the constant produces that file path. This behavior is particularly useful for building reliable paths relative to a source file, such as loading another file from the same directory with `dirname(__FILE__)` in PHP 5. PHP documentation also notes that, beginning with PHP 4.0.2, the value contains an absolute path where possible, with symlinks resolved. Because the constant reflects the file where it is written, it continues to identify an included file when that file is loaded by another script. See the PHP manual's documentation of predefined magic constants at https://www.php.net/manual/en/language.constants.magic.php and the PHP 5 language reference at https://www.php.net/manual/en/language.constants.predefined.php.

QUESTION NO: 18

When a transaction reports no affected rows, it means that: (Choose 2)

A. The transaction failed

B. The transaction affected no lines

C. The transaction was rolled back

D. The transaction was committed without error

E. For an UPDATE statement, matching rows may already contain the requested values.

ANSWER: B E

Explanation:

An affected-row count is a result reported by the data-modification statement executed within a transaction, rather than a status report for the transaction as a whole. A report of zero affected rows therefore means that the executed statement did not modify any rows. For example, an UPDATE may have had no matching records to change, and a DELETE may have found no records satisfying its condition.

For an UPDATE statement, zero affected rows can also occur when records match the WHERE condition but each matched row already contains the value being assigned. In the usual MySQL affected-row behavior, such rows are matched but not counted as changed, because no stored value was modified. Code should therefore treat an affected-row count as information about the statement's data changes, not as proof of transaction failure, rollback, or commit status. Transaction completion must be determined separately through the database API's transaction methods and error handling. PHP documents that `PDOStatement::rowCount()` returns the number of rows affected by DELETE, INSERT, or UPDATE

statements; MySQL documents the distinction between matched and changed rows for UPDATE statements. See PHP: PDOStatement::rowCount and MySQL: ROW_COUNT().

QUESTION NO: 19

An unbuffered query will: (Choose 2)

A. Return the first data faster

B. Return all data faster

C. Free connection faster for others scripts to use

D. Use less memory

ANSWER: A D

Explanation:

An unbuffered query returns rows to PHP as the database server sends them, rather than first transferring the entire result set into a client-side buffer. Consequently, **Return the first data faster** is correct: application code can begin fetching and processing the initial row as soon as it is available, which is useful when a query produces a large result set or when rows can be handled incrementally.

Use less memory is also correct because PHP does not need to retain the complete result set in memory before iteration begins. Memory consumption is therefore substantially lower for large results, since rows are processed one at a time or in small application-managed portions. In the PHP MySQL APIs relevant to Zend PHP 5, this behavior is represented by unbuffered result handling, such as `mysqli::use_result()` and the older `mysql_unbuffered_query()`. While consuming such a result, the connection remains occupied until all rows are fetched or the result is freed, so unbuffered operation is primarily a trade-off favoring early row availability and reduced client memory use.

See the PHP manual for mysqli::use_result() and mysql_unbuffered_query().

QUESTION NO: 20

Transactions can be used to: (Choose 2)

A. Recover from errors in case of a power outage or a failure in the SQL connection

B. Ensure that the data is properly formatted

C. Ensure that either all statements are performed properly, or that none of them are.

D. Recover from user errors

ANSWER: A C

Explanation:

Transactions provide a controlled unit of database work. “Ensure that either all statements are performed properly, or that none of them are.” describes transaction atomicity: related changes are treated as one operation. An application can issue several data-modification statements, then commit them together only after all required work has succeeded. If processing detects a failure, it can roll back the transaction, preventing a partial set of changes from becoming permanent.

“Recover from errors in case of a power outage or a failure in the SQL connection” is also a valid transaction benefit when the database uses a transactional storage engine. A committed transaction is designed to remain durable despite failures, while an uncommitted transaction can be rolled back during connection loss or crash recovery. This protects database consistency by avoiding partially completed work. PHP’s PDO transaction API exposes this model through `beginTransaction()`, `commit()`, and `rollBack()`; the database engine must support transactions for these guarantees to apply. See the PHP PDO transactions documentation and the MySQL InnoDB ACID model documentation.

QUESTION NO: 21

Which of the following is correct? (Choose 2)

1) A class can extend more than one class.

2) A class can implement more than one class.

3) A class can extend more than one interface.

4) A class can implement more than one interface.

5) An interface can extend more than one interface.

6) An interface can implement more than one interface.

A. 1)

B. 2)

C. 3)

D. 4)

E. 5)

F. 6)

ANSWER: D E

Explanation:

In PHP 5, a class declaration may implement a comma-separated list of interfaces. Therefore, a single class can commit to the public contracts defined by multiple interfaces at once, using syntax such as `class Example implements FirstInterface, SecondInterface {}`. The class must then provide implementations for all interface methods that it inherits, subject to the usual visibility and signature requirements. This supports composition of capabilities without requiring multiple class inheritance.

An interface may also extend one or more interfaces. PHP permits a comma-separated list after `extends`, allowing a child interface to aggregate the method contracts of several parent interfaces: `interface Combined extends FirstInterface, SecondInterface {}`. A class implementing that combined interface is consequently required to satisfy the methods inherited through each parent interface as well as methods declared directly by the combined interface. These rules are part of PHP's object-oriented interface model and let applications define reusable, layered contracts while retaining PHP's single-inheritance model for classes. See the PHP manual on object interfaces and class basics.

QUESTION NO: 22

Under which circumstances is the $_SESSION super-global available? (Choose 2)

A. If session_start() was called.

B. If session.auto_start INI setting is enabled.

C. Always available in PHP 5.

D. If a valid session id is passed via GET, POST or COOKIE.

E. If register_globals are enabled.

ANSWER: A B

Explanation:

The `$_SESSION` superglobal becomes available when PHP has started session handling for the current request. Calling `session_start()` explicitly performs that initialization: PHP obtains or creates the session identifier, initializes the session subsystem, reads any stored session data associated with that identifier, and populates `$_SESSION` as an array for application use. This is the normal approach when an application needs session data only on selected requests.

Enabling the `session.auto_start` INI directive is the other valid circumstance. With that directive enabled, PHP automatically starts a session at the beginning of every request, before the script executes. Consequently, scripts can access `$_SESSION` without making their own `session_start()` call. In both cases, the essential condition is that session initialization has occurred during the request; once it has, the session array is available for reading and writing. PHP's session documentation describes `session_start()` as creating a session or resuming the current one and initializing the session data. See PHP: session_start() and PHP session.auto_start configuration.

QUESTION NO: 23

What object method specifies post-serialization behavior for an object?

A. __sleep()

B. __wakeup()

C. __set_state()

D. __get()

E. __autoload()

ANSWER: B

Explanation:

`__wakeup()` specifies the object's behavior after it has been restored by `unserialize()`. PHP calls this magic method automatically after reconstructing the object and restoring its serialized property values. It is intended for post-unserialization initialization: for example, an object can re-establish a database or external-resource connection, restore invariants that cannot simply be represented as stored property values, rebuild derived internal data, or perform other setup required before the object is used again.

The wording "post-serialization" is commonly used in certification questions to mean the stage after serialized data has been converted back into a live object. In PHP's serialization lifecycle, `__wakeup()` is therefore the relevant hook. The method should be declared as a public instance method and normally has no parameters. Its role is especially important when an object has dependencies or runtime state that should not be persisted directly in the serialized representation but must be initialized once deserialization is complete.

PHP documents `__wakeup()` among its magic methods and states that it is invoked when an object is unserialized. See the PHP manual's magic-method documentation and the unserialize() documentation.

QUESTION NO: 24

Given the following code, what is correct?

```
function f(stdClass &$x = NULL) { $x = 42;

}

$z = new stdClass;

f($z);

var_dump($z);
```

A. Error: Typehints cannot be NULL

B. Error: Typehints cannot be references

C. Result is NULL

D. Result is object of type stdClass

E. Result is 42

ANSWER: E

Explanation:

Result is 42. The parameter declaration combines a class type hint, pass-by-reference syntax, and a default value of `NULL`. At the time `f($z)` is called, `$z` contains an instance of `stdClass`, so it satisfies the `stdClass` type requirement. Passing the parameter by reference means that `$x` becomes an alias for the caller's variable, `$z`, rather than a separate local copy of that variable.

Inside the function, assigning `42` to `$x` therefore replaces the value stored in the caller's variable as well. The type hint validates the argument when the function is entered; it does not permanently constrain subsequent assignments made through a reference within the function. Consequently, after the call returns, `$z` is an integer whose value is `42`, and `var_dump($z)` displays `int(42)`. PHP's documentation describes how reference parameters allow a function to modify the variable supplied by its caller, and its PHP 5 object-model documentation covers class type hints on function parameters. See PHP: Passing by Reference and PHP: Type Hinting.

QUESTION NO: 25

How can a PHP extension be loaded? (Choose 2)

A. ini_set("extension", "extension.so");

B. dl("extension.so");

C. extension_load("extension.so");

D. extension=extension.so inside php.ini

ANSWER: B D

Explanation:

PHP extensions can be loaded at PHP startup through the PHP configuration file by placing an `extension=extension.so` directive in `php.ini`. During initialization, PHP reads this directive, locates the shared extension module in the configured extension directory (or at the supplied path), and loads it before executing application code. This is the normal and persistent way to enable an extension for the applicable PHP SAPI, such as Apache or the CLI.

PHP 5 also provides `dl()` for loading a shared extension dynamically at runtime. Calling `dl("extension.so")` instructs PHP to load the named extension module for the current request or process, provided dynamic loading is enabled and the environment permits it. The extension must be a compatible shared module; it cannot be an arbitrary PHP file. In practice, configuration-based loading is generally preferred because it ensures the module is available consistently from startup, while `dl()` is intended for situations where runtime loading is supported and needed.

See the PHP documentation for dl() and the extension configuration directive.