PassQueen
a.k.a dumpsqueen.com

Zend PHP 5.3 Certification

Zend 200-530

Version Demo

Total Demo Questions: 30

Total Premium Questions: 304

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, PHP Basics	33
Topic 2, Functions	16
Topic 3, Data Format and Types	10
Topic 4, Object Oriented Programming	48
Topic 5, Security	28
Topic 6, XML and Web Services	31
Topic 7, Strings and Patterns	37
Topic 8, Arrays	19
Topic 9, I/O	55
Topic 10, Databases	19
Topic 11, Error Handling	7
Topic 12, Mix Questions	1
Total	304

QUESTION NO: 1

The function mysqli_affected_rows() can be used to perform which of the following actions? (Choose 2)

A. get the number of rows that are affected by SELECT statements

B. get the number of rows that are affected by UPDATE statements

C. get the number of rows that are affected by INSERT statements

D. get the number of rows in a result set

E. get the numbers of rows that are affected after committing a transaction using COMMIT

ANSWER: B C

Explanation:

`mysqli_affected_rows()` returns the number of rows affected by the most recently executed operation on a MySQLi connection. It is therefore appropriate for "get the number of rows that are affected by UPDATE statements" and "get the number of rows that are affected by INSERT statements." For an `UPDATE`, the value reports how many existing rows were changed by the statement. For an `INSERT`, it reports how many rows were inserted; this is particularly useful with multi-row inserts, where the count can be greater than one. The function is connection-specific, so it must be called using the same `mysqli` connection that executed the SQL statement, and it should be read after that statement has run. MySQLi exposes this information both through the procedural function and through the corresponding `$mysqli->affected_rows` property. The PHP manual documents that the returned value represents rows affected by the previous MySQL operation and specifically identifies data-changing statements such as `INSERT` and `UPDATE`. See the PHP manual for mysqli_affected_rows() and the MySQLi query documentation.

QUESTION NO: 2

Which functions can be used to inspect the response-header state in PHP? (Choose 2)

A. headers_sent()

B. headers_list()

C. headers_open()

D. headers_read()

E. response_headers()

ANSWER: A B

Explanation:

`headers_sent()` determines whether HTTP headers have already been sent. It is commonly used before attempting a redirect or other header operation, because header fields normally must be sent before output that begins the response body.

`headers_list()` returns a list of response headers that PHP has prepared to send. This can be useful for diagnostics when examining headers registered through `header()` or related functions. See the PHP documentation for headers_sent() and headers_list().

QUESTION NO: 3

Which of the following rules must every correct XML document adhere to? (Choose 2)

A. It has to be well-formed.

B. If it declares a DTD, it has to be valid according to that DTD.

C. It has to be associated to a DTD.

D. It may only contain UTF-8 encoded characters.

ANSWER: A B

Explanation:

An XML document must be well-formed. Well-formedness is the fundamental XML syntax requirement: the document has exactly one document element, tags are properly nested and closed, attribute values are quoted, element and attribute names follow XML naming rules, and entity references are properly formed. XML processors are required to check this basic structural correctness before an application can reliably consume the document.

Validity is an additional requirement that applies when a document is governed by a declared document type definition (DTD). In that situation, the XML document must conform to the declarations and content models defined by that DTD to be valid. Therefore, validity should be expressed conditionally rather than as an unconditional rule for every XML document: XML permits well-formed documents that have no DTD and are not validated against one. This distinction is important in PHP XML work because parsers such as DOM and SimpleXML can process well-formed XML without requiring a DTD, while validation is used when a schema or DTD contract must be enforced.

The XML Recommendation defines well-formed documents and separately defines validity constraints for documents with a DTD. See the W3C XML specification and PHP's DOMDocument::validate documentation.

QUESTION NO: 4 - (SIMULATION)

Which PHP function is used to validate whether the contents of $_FILES['name'] ['tem_name'] have really been uploaded via HTTP'?

ANSWER: See the explanation for the answer

Explanation:

The correct function is `is_uploaded_file()`.

Use the temporary uploaded-file path from `$_FILES['name']['tmp_name']`:

```
if (is_uploaded_file($_FILES['name']['tmp_name'])) {
    // The file was uploaded through HTTP POST.
}
```

`is_uploaded_file()` verifies that the specified file was genuinely uploaded via PHP's HTTP POST upload mechanism. (The question's `tem_name` appears to be a typo for `tmp_name`.)

QUESTION NO: 5

Which of the following statements about Reflection are correct? (Choose 2)

A. Since 5.1 reflection is an extension that can be disabled

B. Reflection is present in any installation of PHP 5 or later

C. Reflection only allows to reflect on built-in classes

D. Built-in classes can be reflected on the command line using php --rc

ANSWER: A D

Explanation:

In the PHP 5.3 context, Reflection is supplied as a standard PHP extension and is enabled by default, but it could be excluded when PHP was built by using the `--disable-reflection` configure switch. Therefore, an installation's Reflection support was configurable rather than an unavoidable feature of every possible PHP build. Reflection provides programmatic metadata about classes, methods, properties, functions, parameters, and extensions, making it useful for frameworks, dependency injection containers, documentation tooling, and runtime inspection.

The PHP command-line interface also provides reflection-oriented switches for inspecting internal symbols without writing a PHP script. In particular, `php --rc <classname>` displays information for a specified class available to that CLI installation, including built-in/internal classes. This is a convenient command-line interface to class reflection data and is especially useful when checking the methods, constants, properties, and inheritance details exposed by an installed PHP build. The CLI option is documented in the PHP command-line options reference: PHP Command Line Options. Reflection's API and the metadata it exposes are described in the PHP Reflection documentation.

QUESTION NO: 6

Which functions can validate or filter external input using the PHP filter extension? (Choose 2)

A. filter_var()

B. filter_input()

C. validate_input()

D. sanitize_var()

E. input_filter()

ANSWER: A B

Explanation:

`filter_var()` filters or validates a value that has already been obtained by the application. It accepts the value to process together with a filter constant, such as `FILTER_VALIDATE_EMAIL` or `FILTER_SANITIZE_STRING`.

`filter_input()` retrieves a value from a specified external-input source, such as `INPUT_GET`, `INPUT_POST`, or `INPUT_COOKIE`, and applies the requested filter. Both functions are part of the filter extension available in PHP 5.3. See the PHP documentation for filter_var() and filter_input().

QUESTION NO: 7

Which of the following statements about the SPL `Iterator` interface are correct? (Choose 2)

A. An object implementing Iterator can be traversed with foreach.

B. The valid() method determines whether the current position is valid.

C. The Iterator interface provides array offset access.

D. The current() method advances to the next element.

E. An Iterator implementation cannot provide keys.

ANSWER: A B

Explanation:

An object implementing `Iterator` can be traversed with `foreach`. The interface defines the methods PHP uses to control iteration: `current()`, `key()`, `next()`, `rewind()`, and `valid()`. During traversal, PHP calls these methods to obtain the current element, advance through the collection, and determine when iteration has ended.

The `valid()` method determines whether the current position contains an element that can be read. It returns a boolean result. The `current()` method returns the current value, while `key()` returns its key. See PHP: Iterator and PHP: Object Iteration.

QUESTION NO: 8

Which of the following statements about database connections are commonly true?

(Choose 2)

A. Database connections are closed after each SQL statement is executed

B. Database connections are closed at the end of each request

C. Database connections are only closed when the Web server shuts down

D. A single database connection may serve more than one PHP application at the same time

ANSWER: B C

Explanation:

Database connections are closed at the end of each request is commonly true for ordinary, non-persistent PHP database connections. PHP releases resources associated with a request during request shutdown, and database extensions normally close non-persistent connections at that time. This request-based lifecycle prevents a standard connection opened by one execution of a PHP script from remaining open indefinitely after that execution has completed.

Database connections are only closed when the Web server shuts down describes the usual lifecycle of a persistent connection. Persistent connections are retained by the PHP worker or web-server process so that a later request handled by that same process can reuse an already-established database session. Rather than being closed during each request's shutdown, such a connection remains available for reuse and is ultimately released when the owning process terminates, such as during a web-server shutdown or worker restart. PHP's documentation notes that persistent connections are not closed when script execution ends and are instead kept for reuse by later requests. See the PHP manual's discussion of persistent database connections and its description of MySQLi connection closing behavior.

QUESTION NO: 9

You want to test if a string matches a relatively complex pattern. Which of the following functions can you use? (Choose 2)

A. match()

B. preg_match()

C. ereg()

D. regex()

ANSWER: B C

Explanation:

`preg_match()` is appropriate because it performs regular-expression matching with PHP's Perl-Compatible Regular Expressions (PCRE) engine. It accepts a pattern and a subject string, returns `1` when a match is found, `0` when no match is found, and can also populate an array with matched text. PCRE supports the kinds of expressive constructs useful for relatively complex matching, including grouping, alternation, character classes, quantifiers, assertions, and modifiers. The PHP manual documents its pattern syntax and matching behavior at PHP: preg_match.

`ereg()` is also a valid answer specifically in the PHP 5.3 context of this certification question. It tests a subject string using POSIX extended regular expressions and returns whether a match occurred, with optional capture of matched subexpressions. Thus, it can be used to test strings against regular-expression patterns. PHP 5.3 deprecates the POSIX regex extension, so new code at that time should generally prefer `preg_match()`; nevertheless, `ereg()` remained available and functional in PHP 5.3. Its historical documentation, including the deprecation notice, is available at PHP: ereg.

QUESTION NO: 10

Which of the following statements about Reflection are correct? (Choose 2)

A. Since 5.1 reflection is an extension that can be disabled

B. Reflection is present in any installation of PHP 5 or later

C. Reflection only allows to reflect on built-in classes

D. Built-in classes can be reflected on the command line using `php --rc`

ANSWER: A D

Explanation:

Reflection was provided as an extension beginning with PHP 5.1. Although it is enabled by default in standard builds, it is optional at build time and can be excluded using the relevant configure setting. Therefore, its availability is not guaranteed in every PHP installation; it depends on how the runtime was compiled. The Reflection extension provides programmatic information about classes, interfaces, functions, methods, properties, parameters, extensions, and other language constructs. This includes both internal constructs supplied by PHP and constructs declared in application code.

The command-line interface also provides reflection-oriented information switches. The `--rc` command-line option displays information for a specified class, including an internal class when that class is available in the CLI runtime. For example, `php --rc DateTime` requests class information for `DateTime`. This is useful for quickly inspecting installed runtime APIs without writing a script. The PHP manual documents Reflection as an extension and notes its installation/build availability at PHP Reflection installation. The CLI option reference, including class-information switches, is available in the PHP command-line options documentation.

QUESTION NO: 11

CORRECT TEXT

Which SPL class implements fixed-size storage?

A. SplFixedArray

ANSWER: A

Explanation:

`SplFixedArray` is the SPL class designed for fixed-size indexed storage. Its constructor accepts an optional size, and the array's capacity can later be explicitly set with `setSize()`. Unlike a standard PHP array, it is intended to store values in a predefined numeric-index range rather than dynamically expanding as elements are appended. This makes it suitable when the number of slots is known in advance and fixed-size semantics are required. The class implements the standard SPL iteration and array-access interfaces, so its elements can be accessed with bracket notation and traversed in a familiar way, while preserving its fixed-size storage model. PHP's documentation describes `SplFixedArray` specifically as an array with a fixed number of elements and notes its memory-efficiency benefit compared with a normal PHP array. The Zend PHP 5.3 SPL material includes this class as the fixed-array implementation. See the PHP manual for SplFixedArray and the PHP SPL data structures overview.

QUESTION NO: 12

What is "instanceof" an example of:

A. a boolean

B. an operator

C. a function

D. a language construct

E. a class magic

ANSWER: B

Explanation:

`instanceof` is an operator in PHP. It is used to test an object's relationship to a class or interface at runtime. The expression places an object on the left side and a class name, interface name, or compatible object expression on the right side, and evaluates to a Boolean result. For example, `$service instanceof LoggerInterface` evaluates to `true` when `$service` is an instance of a class that implements `LoggerInterface`, including cases where the relationship is inherited through a parent class. This makes `instanceof` especially useful when code must safely branch based on the capabilities or type hierarchy of an object before calling type-specific behavior. Although evaluating the operator produces a Boolean value, its syntactic category is still an operator, specifically PHP's type-checking operator. PHP's language documentation describes `instanceof` in the operators section and demonstrates its use with classes, inheritance, and interfaces. The PHP manual also notes that the left operand must be an object expression for this kind of instance check. See the PHP type operators documentation and the PHP object-oriented programming documentation for the defined behavior.

QUESTION NO: 13

Which of the following is used to find all PHP files under a certain directory?

A. PHPIterator

B. RecursiveTreeIterator

C. RecursiveDirectoryIterator

D. SplTempFileObject

ANSWER: C

Explanation:

`RecursiveDirectoryIterator` is the appropriate SPL iterator for walking a directory tree. It extends `FilesystemIterator` and implements `RecursiveIterator`, so each directory encountered can provide its children for recursive traversal. In practical code, it is commonly supplied to `RecursiveIteratorIterator`, which flattens the nested directory structure into a sequence of `SplFileInfo` objects. The application can then inspect each pathname or extension and retain files whose names end in `.php`. For example, a recursive iterator can traverse a project root while a filename or extension test identifies PHP source files at every nesting level. This is preferable to manually opening each directory because SPL provides a standard object-oriented filesystem traversal API and file metadata through `SplFileInfo`. The PHP manual documents that `RecursiveDirectoryIterator` provides an interface for iterating recursively over filesystem directories, while `RecursiveIteratorIterator` is designed to iterate recursively over recursive iterators. See the PHP manual for RecursiveDirectoryIterator and the PHP manual for RecursiveIteratorIterator.

QUESTION NO: 14

Which piece of code will return the ASCII value of a character?

A. (int)'t';

B. ord('t');

C. to_ascii('t');

D. chr('t');

ANSWER: B

Explanation:

`ord('t');` is correct because PHP's `ord()` function returns the numeric value of the first byte of a string. For the single-character string `'t'`, it returns `116`, which is the ASCII code for the lowercase letter "t." This is the standard PHP function for converting a character represented by a single-byte string into its corresponding numeric byte value. In the PHP 5.3 context, ASCII characters are represented using their familiar ASCII byte values, so this expression directly produces the requested result. The function accepts a string argument and evaluates its first character, making it appropriate for character-to-code conversion. PHP documents `ord()` as returning the ASCII value of the first character of a string; its counterpart, `chr()`, converts a numeric value back to a one-character string. See the official PHP documentation for ord() and the related chr() function.

QUESTION NO: 15

What will the following code print?

echo addslashes('I am a **small** "HTML" string, which is

\'invalid\'.');

A. I am a **small** "HTML" string, which is 'invalid'.

B. I am a **small** \"HTML\" string, which is \'invalid\'.

C. I am a **small** \"HTML\" string, which is \\'invalid\\'.

D. I am a **small** \"HTML\" string, which is \\\'invalid\\\'.

E. I am a \small\<\/b\> "HTML" string, which is 'invalid'\<\/u\>.

F. I am a **small** \\\"HTML\\\" string, which is \'invalid\'.

ANSWER: F

Explanation:

`I am a <b>small</b> \\\"HTML\\\" string, which is \'invalid\'.` is printed. The argument to `addslashes()` is a single-quoted PHP string. Within a single-quoted string, a backslash only has special handling when it precedes another backslash or a single quote. Consequently, each `\"` sequence in the source is initially represented by two characters: a literal backslash followed by a double quote. The `\'` sequences, however, represent literal apostrophes in the resulting string value.

`addslashes()` then adds a backslash before single quotes, double quotes, backslashes, and NUL bytes. Each original backslash before `"HTML"` is escaped to two backslashes, and the double quote is itself escaped with one more backslash. This produces three consecutive backslashes before each double quote. The apostrophes around `invalid` were not preceded by literal backslashes in the input value, so each receives one backslash from `addslashes()`. The HTML-like tags are ordinary characters and remain unchanged. See the PHP manual for addslashes() and single-quoted string syntax.

QUESTION NO: 16

Which of the following code snippets writes the content of the “source.txt” to “target.txt”?

A. file_put_contents("target.txt", fopen("source.txt", "r"));

B. file_put_contents("target.txt", readfile("source.txt"));

C. file_put_contents("target.txt", join(file("source.txt"), ""));

D. file_put_contents("target.txt", file_get_contents("source.txt"));

E. $handle = fopen("target.txt", "w+"); fwrite($handle, file_get_contents("source.txt")); fclose($handle);

ANSWER: A C D E

Explanation:

`file_put_contents("target.txt", fopen("source.txt", "r"));` is valid because `file_put_contents()` accepts a stream resource as its data argument. When given the resource returned by `fopen()`, it reads the stream’s remaining contents and writes them to the destination file. This approach is useful when the source is already represented by an open stream.

`file_put_contents("target.txt", join(file("source.txt"), ""));` also writes the source text. `file()` reads the file into an array whose elements are its lines, and `join()` concatenates those lines with an empty separator. The resulting string is then written to `target.txt`.

`file_put_contents("target.txt", file_get_contents("source.txt"));` directly obtains the complete source file as a string and supplies that string to `file_put_contents()`. The explicit `fopen()`, `fwrite()`, and `fclose()` sequence likewise reads the source string, writes it through a writable target handle, and closes that handle to release the resource. See the PHP documentation for file_put_contents() and file().

QUESTION NO: 17

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 functions intended for encoding untrusted text before it is included in HTML output. This output encoding prevents text that contains characters with special meaning to an HTML parser, such as angle brackets, quotation marks, and ampersands, from being interpreted as markup or attribute syntax. Instead, the browser receives the applicable HTML character references and renders the submitted content as text.

`htmlspecialchars()` converts the characters that are most significant in HTML: `&`, `<`, `>`, and, when the relevant quote flags are used, single and double quotation marks. It is the usual focused choice for safely displaying plain text in an HTML element or attribute, with an appropriate character encoding specified. `htmlentities()` also performs HTML-safe encoding and additionally converts many characters that have named or numeric HTML entities. Therefore, it is likewise an HTML-context escaping function. PHP documents both functions as converting applicable characters to HTML entities; see

the PHP htmlspecialchars() manual and the PHP htmlentities() manual. In practice, escaping must occur at output time and must match the specific output context; these functions address HTML text and suitable quoted HTML-attribute contexts.

QUESTION NO: 18

Which of the following statements are NOT true?

A. SimpleXML allows removal of attributes.

B. SimpleXML allows addition of new attributes.

C. SimpleXML allows removal of nodes.

D. SimpleXML allows addition of new nodes.

E. None of the above

ANSWER: E

Explanation:

None of the above is correct because each preceding statement describes a supported SimpleXML capability in PHP 5.3. A `SimpleXMLElement` object can be modified by adding child elements with `addChild()` and adding attributes with `addAttribute()`. These methods extend the XML tree directly and return the newly created element where applicable. SimpleXML also permits removal through PHP's `unset()` construct. For example, an attribute accessed using array-style syntax can be removed with `unset($xml['id'])`, while a child node can be removed by unsetting the relevant child-property index, such as `unset($xml->item[0])`. Consequently, SimpleXML supports both addition and removal for attributes and nodes, so there is no untrue statement among the four listed capabilities. The PHP manual documents `SimpleXMLElement::addChild()` and `SimpleXMLElement::addAttribute()` as the APIs for creating XML nodes and attributes; its SimpleXML examples also demonstrate modifying the object model using normal PHP access syntax. See the PHP manual for addChild() and the PHP manual for addAttribute().

QUESTION NO: 19

Can a private static member be accessed from a public static method of the same class?

A. Yes

B. No

C. Only if the class is defined as an abstract

ANSWER: A

Explanation:

Yes is correct. In PHP, `private` visibility restricts access to the class in which a property or method is declared, rather than restricting access based on whether the calling method is static or public. Therefore, a public static method belongs to the same class scope and may read or modify a private static property, or invoke a private static method, using `self::` (or the class name where appropriate). For example, a class can declare `private static $count = 0;` and expose a `public static function increment()` method that performs `self::$count++;`. External code can call the public method, but direct access to the private member remains unavailable outside that declaring class. This is a common encapsulation pattern: the class publishes a controlled static API while retaining implementation details and state as private. PHP's visibility rules apply consistently to static members, and the method's public visibility only controls whether callers outside the class may invoke that method. See the PHP documentation on object visibility and static keyword and properties.

QUESTION NO: 20

What XML component does the following XPath query try to match?

//foo[bar/@id=5]

A. bar element with an id attribute whose value is equal to 5

B. foo element containing a child node bar tag

C. id attribute whose value is equal to 5

D. foo element with a child note bar whose id attribute is equal to 5

E. all of the foo elements that have a child node bar, whose id attribute is equal to 5

ANSWER: E

Explanation:

The expression `//foo[bar/@id=5]` selects every `foo` element found at any applicable descendant location from the current context that satisfies the predicate in brackets. The predicate `bar/@id=5` is evaluated relative to each candidate `foo` element. In that relative location, `bar` denotes a child `bar` element, and `@id` denotes that child element's `id` attribute. The comparison is true when at least one such attribute has a value equal to the numeric value `5`, so the result consists of the matching `foo` elements themselves, rather than the nested `bar` elements or their attributes. Thus, "all of the foo elements that have a child node bar, whose id attribute is equal to 5" correctly describes both the node selected by the location path and the condition imposed by its predicate. XPath predicates filter the node set produced by the preceding step; they do not change the selected node type to a node mentioned inside the predicate. This behavior is defined by the XPath location-path and predicate model described in the W3C XPath 1.0 Recommendation; see also the PHP DOMXPath documentation for XPath queries in PHP.

QUESTION NO: 21

Given the following code, what will be the value of $a?

$a = array('a', 'b');

array_push($a, array(1, 2));

A. array('a', 'b', 1, 2)

B. array(1, 2, 'a', 'b')

C. array(array(1, 2), 'a', 'b')

D. None of the above

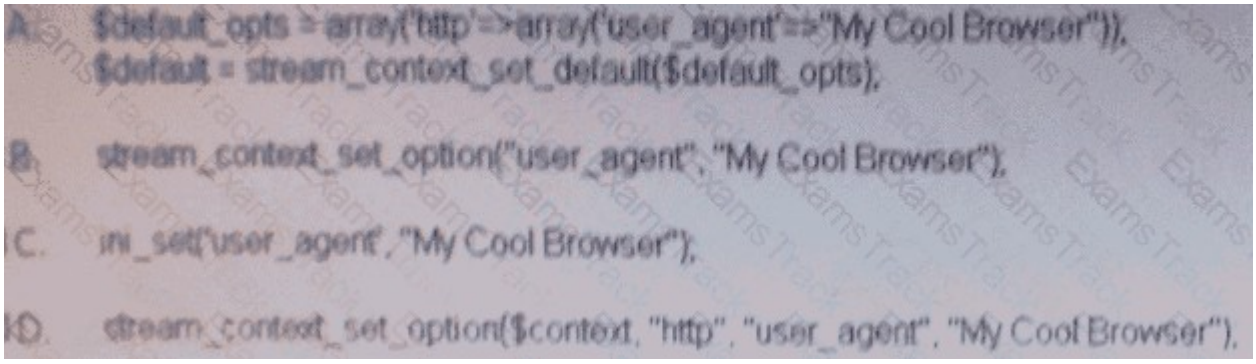
ANSWER: D

Explanation:

`None of the above` is correct because `array_push()` appends each argument supplied after the target array as one new element at the end of that target array. In this code, `$a` initially contains two string elements: `'a'` at index `0` and `'b'` at index `1`. The second argument passed to `array_push()` is itself an array, `array(1, 2)`. PHP therefore appends that entire array as a single element; it does not automatically unpack or merge its members into `$a`. After the call, the effective value is `array('a', 'b', array(1, 2))`, with the nested array at index `2`. This distinction is important when working with PHP arrays: pushing an array produces a multidimensional array, whereas combining the individual values would require passing them as separate arguments or using an appropriate array-combination operation. The PHP manual describes `array_push()` as pushing one or more elements onto the end of an array and provides examples of appended values. See PHP: array_push() and PHP: Arrays.

QUESTION NO: 22

When retrieving data from URLs, what are valid ways to make sure all file_get_contents calls send a certain user agent string? (Choose 2)



A. Set the `user_agent` configuration directive with `ini_set()`.

B. Option B

C. Option C

D. Set an HTTP `user_agent` value in the default stream context using `stream_context_set_default()`.

ANSWER: A D

Explanation:

Setting the `user_agent` configuration directive with `ini_set('user_agent', '...')` is valid because PHP's HTTP stream wrapper uses that directive as its default User-Agent value when no per-request HTTP context value is supplied. Since the directive is changed for the running script, subsequent URL reads performed through `file_get_contents()` can inherit the specified agent string.

Setting a default stream context with `stream_context_set_default()` is also valid. A default context applies its options to stream operations that do not receive an explicit context resource. Supplying an HTTP context containing a `user_agent` value therefore establishes the desired User-Agent for URL retrieval calls throughout the remainder of the request. This is particularly useful where changing each individual `file_get_contents()` invocation is impractical. PHP documents `user_agent` as an HTTP-context option and documents default contexts as the context used when no explicit context is provided. See the PHP HTTP context options documentation and the stream_context_set_default() documentation.

QUESTION NO: 23 - (SIMULATION)

Which SPL class implements fixed-size storage?

ANSWER: See the explanation for the answer

Explanation:

The SPL class that implements fixed-size storage is `SplFixedArray`.

Example:

```
$array = new SplFixedArray(10); // exactly 10 slots
$array[0] = 'value';
```

Unlike a normal PHP array, an `SplFixedArray` has a predefined size (though it can later be changed with `setSize()`).

QUESTION NO: 24

Which methods can be used to overload object properties? (Choose 2)

A. set(), get()

B. __set(), __get()

C. __put(), __receive(), __exists()

D. set(), get(), isset()

E. __isset(), __unset()

ANSWER: B E

Explanation:

PHP property overloading is implemented through special magic methods that intercept interaction with inaccessible object properties, meaning properties that are not visible in the current scope or do not exist. `__set()` and `__get()` provide the core write and read hooks: PHP calls `__set()` when code assigns a value to an inaccessible property, and calls `__get()` when code reads one. This lets a class validate, transform, store, or dynamically calculate property values while presenting normal property syntax to callers.

`__isset()` and `__unset()` are also property-overloading methods. They handle the corresponding language constructs when they operate on inaccessible properties: `__isset()` determines whether such a property should be considered set, and `__unset()` performs custom removal or cleanup behavior. Together, these methods cover reading, writing, testing, and unsetting overloaded properties. PHP requires magic methods to be declared `public`, and their names use the double-underscore prefix defined by the language. See the PHP manual's Overloading documentation and its reference on magic methods.

QUESTION NO: 25

What can NOT be used to send a cookie from within a PHP application?

A. header()

B. $_COOKIE

C. setcookie()

D. setrawcookie()

ANSWER: B

Explanation:

`$_COOKIE` cannot be used to send a cookie because it is a PHP superglobal containing cookie values supplied by the client in the current HTTP request. PHP populates it from the request's `Cookie` header, so it represents information already received by the application rather than an API for adding response headers. Assigning a value to `$_COOKIE` only changes the corresponding value in PHP memory for the remainder of that request; it does not instruct the browser to store anything and does not generate a `Set-Cookie` response header. To send a cookie, PHP must emit a `Set-Cookie` header before output is sent. PHP provides dedicated cookie functions for this purpose, and HTTP headers can also be sent directly when constructed correctly. The PHP manual describes `$_COOKIE` as an associative array of variables passed to the current script via HTTP cookies, confirming its role as request input. See the PHP documentation for $_COOKIE and the PHP setcookie() documentation.

QUESTION NO: 26

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 either during PHP startup through configuration or, where permitted, dynamically while a script is running. The `extension=extension.so` directive in `php.ini` is the standard persistent configuration method for a shared extension on Unix-like platforms. PHP reads this directive when it starts, loads the specified shared object, and makes the extension's functions and classes available to requests handled by that PHP installation. The exact filename and extension suffix are platform-dependent; for example, Windows commonly uses a `.dll` file. The extension must be installed in PHP's configured extension directory, or its path must otherwise be resolvable by PHP.

The `dl("extension.so")` function is PHP's runtime loading mechanism for a shared extension. It loads a dynamically loadable module for the current process, subject to PHP build and configuration restrictions. In particular, dynamic loading must be enabled through `enable_dl`; it is commonly unavailable in restrictive server SAPIs and is not a replacement for normal server-level configuration. For this reason, placing an extension directive in the active PHP configuration is generally the preferred operational approach, while `dl()` represents the supported programmatic mechanism when the environment allows it. See the PHP manual for dl() and the extension configuration directive.

QUESTION NO: 27

Which session function can help to avoid session fixation?

A. session_is_registered()

B. session_register()

C. session_unregister()

D. session_regenerate_id()

E. None of the above.

ANSWER: D

Explanation:

`session_regenerate_id()` is the appropriate function because it replaces the current session identifier with a newly generated identifier while retaining the session data by default. Session fixation occurs when an attacker causes or persuades a victim to use a session ID known to the attacker; if the application continues using that identifier after authentication, the attacker may be able to access the authenticated session. Regenerating the ID immediately after a successful login, privilege change, or other authentication-state transition prevents the pre-authentication identifier from remaining the identifier for the authenticated session. In PHP 5.3, calling `session_regenerate_id(true)` also requests deletion of the old session data, which is commonly appropriate after login when the application has considered concurrent requests and session-storage behavior. The PHP manual specifically describes this function as updating the current session ID with a newly generated one, and its session-security guidance recommends regenerating IDs periodically and on authentication-related state changes. See the PHP manual for session_regenerate_id() and the PHP session security management guidance.

QUESTION NO: 28

Which string will be returned by the following function call?

$test = '/etc/conf.d/wireless';

substr($test, strrpos($test, '/'));

A. ""

B. "/wireless"

C. "wireless"

D. "/conf.d/wireless"

E. "/etc"

ANSWER: B

Explanation:

The returned string is `"/wireless"`. The `strrpos()` function finds the numeric position of the last occurrence of a substring within a string. In this path, the final slash is the slash immediately before `wireless`. `strrpos($test, '/')` therefore returns that slash's zero-based offset.

That offset is passed as the second argument to `substr()`. When `substr()` receives only a starting offset and no length, it returns every character from that offset through the end of the string. Because the offset points to the slash itself, rather than the character after it, the slash is included in the result. The expression consequently extracts the final path component while retaining its leading directory separator: `/wireless`.

This behavior is defined by PHP's string-function documentation: strrpos() returns the position of the last occurrence of the needle, and substr() returns the portion of a string beginning at the specified offset when no length is supplied.

QUESTION NO: 29

In a shared hosting environment, session data can be read by PHP scripts written by any user. How can you prevent this?

A. Store session data in a different location with session. save_Path .

B. Store session data in a database.

C. Enable safe_mode .

D. Set session.name to something unique.

ANSWER: B

Explanation:

Store session data in a database. This prevents exposure caused by a shared filesystem session directory, such as a common temporary directory that is accessible to scripts running under multiple hosting accounts. PHP can use a custom session save handler to store and retrieve session records through database queries rather than writing serialized session files to a location shared by unrelated users. The application should connect using database credentials limited to its own session table or database, and that database account must not be available to other hosting users. With those access controls in place, another user's PHP script cannot simply open and inspect session files belonging to the application.

PHP provides the `session_set_save_handler()` API for registering callbacks that implement session storage operations, including reading, writing, destroying, and garbage collection. A database-backed handler also allows the application to control record lifetime and clean up expired sessions consistently. Session identifiers must still be protected with secure cookie settings and regenerated when appropriate, but moving the session payload behind database authorization addresses the stated cross-user file-reading risk. See the PHP documentation for session_set_save_handler() and the Sessions extension.

QUESTION NO: 30

You'd like to use the class MyDBConnection that's defined in the MyGreatFrarnework\GreafDatabaseAbstractionLayer namespace, but you want to minimize *as much as possible* the length of the class name you have to type. What would you do?

A. Import the MyGreatFramework namespace

B. Import the MyGreatFrarnework\GreafDatabaseAbstractionLayer namespace

C. Alias MyGreatFrarnework\GreafDatabaseAbstractionLayer\MyDBConnection to a shorter name

D. Alias MyGreatFrarnework\GreafDatabaseAbstractionLayer to a shorter name

ANSWER: C

Explanation:

Alias MyGreatFrarnework\GreafDatabaseAbstractionLayer\MyDBConnection to a shorter name is correct because PHP namespace imports operate on names, including class names, and may assign an alias with the `as` keyword. For example, placing `use MyGreatFrarnework\GreafDatabaseAbstractionLayer\MyDBConnection as DB;` in the file's outer scope allows later references to use simply `DB`. Since the requirement is to minimize the amount typed as much as possible, the alias itself can be chosen to be very short, such as a single meaningful character where appropriate. The imported name is resolved by PHP at compile time, so this is a naming convenience rather than a runtime wrapper, subclass, or modification to the original class. It also keeps the source explicit about which fully qualified class is being imported while making repeated instantiation, type hints, static calls, and other references concise. PHP's namespace documentation describes importing and aliasing names with `use`, including class aliases: PHP Manual: Importing Namespaces. The grammar and behavior of `use` declarations are also specified in the PHP language specification: PHP Manual: Namespaces.