Anki Cards: PHP Core Terminology

Core Terminology ?

1. In web PHP, a *single execution* of code to respond to an HTTP request is often called a {{c1::request}} (or script run).
2. PHP runs via an {{c1::interpreter}} (the PHP engine), not by producing a native binary like C/C++.
3. PHP is {{c1::server-side}}: the browser receives the {{c2::output}} (HTML/JSON), not the {{c3::PHP source}}.
4. Built-in arrays that are always available (e.g., `$_GET`, `$_POST`, `$_SERVER`) are called {{c1::superglobals}}.
5. PHP's automatic conversion between types (e.g., string ↔ int) is {{c1::type juggling}}.
6. `declare(strict_types=1);` enables {{c1::strict typing}} behavior for {{c2::scalar type hints}}.
7. **Validation** asks: {{c1::"Is this allowed?"}}; invalid input is typically {{c2::rejected}}.
8. **Sanitization** asks: {{c1::"Can we make this safe/clean?"}}; input is {{c2::transformed}}.
9. **Escaping** asks: {{c1::"How do I safely output this in a context?"}} (HTML/attr/JS/URL), and is done at {{c2::output time}}.
10. **XSS** happens when unescaped output lets an attacker run {{c1::JavaScript}} in the victim's browser.
11. **CSRF** is a {{c1::forged request}} problem; typical defense is a per-request {{c2::token/nonce}}.
12. **SQL injection** is prevented by using {{c1::prepared statements}} instead of unsafe string concatenation.
13. `require` is {{c1::fatal}} if the file is missing; `include` emits a {{c2::warning}} and continues.
14. Composer-style class loading is {{c1::autoloading}}, commonly via {{c2::PSR-4}}.
15. A {{c1::namespace}} prevents naming collisions by qualifying names like `MyApp\Foo`.
16. {{c1::Dependency injection}} means {{c2::passing dependencies in}} rather than creating them inside the class/function.
17. {{c1::PDO}} is PHP's standard DB interface and supports {{c2::prepared statements}}.
18. In WordPress, a hook is a {{c1::callback point}}: an {{c2::action}} "does something," a {{c3::filter}} "modifies a value."

Daily PHP Constructs (“Commands”) ?

19. `echo` outputs {{c1::strings}} (and can output multiple args separated by commas).
20. `print` is like echo but returns {{c1::1}} (so it’s usable in expressions).
21. `var_dump($x)` shows both {{c1::type}} and {{c2::value}} (great for debugging).
22. `print_r($x, true)` returns the output as a {{c1::string}} when the second argument is {{c2::true}}.
23. `die()` / `exit()` {{c1::stops execution}} immediately (often after a redirect).
24. `include_once` / `require_once` ensure a file is included at most {{c1::once}} per request.

Control Flow (If / Switch / Match / Loops) ?

25. `if (...) {}` runs only when the condition is {{c1::true}}.
26. `switch` typically needs `{{c1::break}}` to avoid fall-through into the next case.
27. `match (...) { ... }` is an {{c1::expression}} that {{c2::returns a value}} (unlike `switch`).
28. `match` uses {{c1::strict comparisons}} (no type juggling like loose `switch` cases can do).
29. `foreach ($arr as $value)` iterates over the array’s {{c1::values}}.
30. `foreach ($arr as $k => $v)` gives both the {{c1::key}} and the {{c2::value}}.
31. `break` exits the {{c1::current loop/switch}}; `continue` skips to the {{c2::next iteration}}.
32. A `do { ... } while (...);` loop runs the body at least {{c1::once}}.

Functions & Organization ?

33. A function can define a default parameter like `function f($x = 123)`, meaning it’s {{c1::optional}} when calling.
34. `return` exits a function and optionally provides a {{c1::value}}.
35. `global $x;` accesses a variable from the {{c1::global scope}} (best used {{c2::sparingly}}).
36. `static $x = 0;` inside a function persists {{c1::between calls}} during the same request.
37. A function with a return type `: int` promises it will return an {{c1::integer}} (or throw).
38. In modern PHP, use `{{c1::strict_types}}` when you want stricter scalar parameter/return behavior.

Error Handling & Exceptions ?

39. `try { ... } catch (Throwable $e) { ... }` catches both {{c1::Exception}} and many {{c2::Error}} types.
40. `finally { ... }` runs whether an exception was {{c1::thrown}} or not (good for cleanup).
41. `throw new Exception('msg');` {{c1::raises}} an exception to be handled by a caller.
42. If an exception is not caught, it typically causes a {{c1::fatal error}} and aborts the request.

OOP: Classes, Visibility, Inheritance ?

43. `new ClassName()` creates an {{c1::object instance}}.
44. `public` members are accessible {{c1::everywhere}}; `protected` inside {{c2::class + subclasses}}; `private` only inside the {{c3::declaring class}}.
45. `extends` means {{c1::inheritance}}; `implements` means fulfilling an {{c2::interface contract}}.
46. `$this->` accesses the {{c1::current object}} instance members.
47. `self::` refers to the {{c1::current class}}; `parent::` refers to the {{c2::parent class}}.
48. A `trait` is a mechanism for {{c1::code reuse}} across classes (without inheritance).
49. An `abstract` class cannot be {{c1::instantiated}} directly.
50. An `interface` defines {{c1::method signatures}} that implementing classes must provide.

Variables, Types, Operators ?

51. PHP variables start with a {{c1::$}} sign.
52. `define('APP_ENV', 'dev')` defines a {{c1::constant}} at runtime; `const` defines a constant at {{c2::compile time}} (and can be used in classes).
53. Scalar types: {{c1::int}}, {{c2::float}}, {{c3::string}}, {{c4::bool}}.
54. `null` represents an {{c1::absence}} of value.
55. `.` is {{c1::string concatenation}} in PHP.
56. `.=` performs concatenation and {{c1::assignment}} in one step.
57. `==` is {{c1::loose comparison}} (type juggling); `===` is {{c2::strict}} (type + value).
58. The “spaceship” operator `<=>` returns {{c1::-1}}, {{c2::0}}, or {{c3::1}} for ordering comparisons.
59. Null coalescing `??` uses the right-hand side only if the left is {{c1::null or undefined}}.
60. Nullsafe `?->` stops and returns {{c1::null}} if the left side is {{c2::null}}.

61. `&&` / `||` are {{c1::short-circuit}} boolean operators.
62. The ternary `cond ? a : b` picks {{c1::a}} when cond is true, else {{c2::b}}.

Strings ?

63. In single quotes `'...'`, variables are generally {{c1::not interpolated}}.
64. In double quotes `"..."`, variables like `$name` are {{c1::interpolated}}.
65. `strlen($s)` returns the string length in {{c1::bytes}} (multibyte text may need {{c2::mb_strlen}}).
66. `strpos($haystack, $needle)` returns the position or {{c1::false}} (so use `=== false` checks).
67. `trim($s)` removes whitespace from the {{c1::start and end}} of a string.
68. `explode(',', $s)` converts a string into an {{c1::array}}.
69. `implode(',', $arr)` converts an array into a {{c1::string}}.
70. `sprintf("Hi %s", $name)` returns a formatted {{c1::string}} without echoing it.

Arrays (Workhorse) ?

71. `[]` creates an {{c1::array}} literal (indexed or associative).
72. Indexed array example: `$a = [10, 20, 30];` uses numeric {{c1::indexes}}.
73. Associative array example: `['name' => 'Ada']` uses string {{c1::keys}}.
74. `$a[] = 99;` appends to the {{c1::end}} of an indexed array.
75. `count($arr)` returns the number of {{c1::elements}}.
76. `in_array($needle, $haystack, true)` uses strict checking when the third argument is {{c1::true}}.
77. `array_key_exists('k', $arr)` checks for the presence of a {{c1::key}} even if its value is {{c2::null}}.
78. `array_map(fn($x) => ..., $arr)` transforms each element and returns a {{c1::new array}}.
79. `array_filter($arr, $fn)` keeps elements where the callback returns {{c1::true}}.
80. `array_reduce($arr, $fn, $initial)` folds an array into a single {{c1::value}}.
81. `sort($arr)` sorts values and {{c1::reindexes}} numeric keys.
82. `asort($arr)` sorts by value while {{c1::preserving keys}}.
83. `ksort($arr)` sorts by {{c1::key}}.

HTTP & Superglobals ?

84. Query string parameters are read from {{c1::$_GET}}.
85. Form body parameters are commonly read from {{c1::$_POST}}.
86. Request metadata (method, headers info, URI) is found in {{c1::$_SERVER}}.
87. Uploaded file info is in {{c1::$_FILES}} (name/type/tmp_name/error/size).
88. Session data uses {{c1::$_SESSION}} after calling {{c2::session_start()}}.
89. A safe read pattern: `$q = $_GET['q'] ?? '';` avoids an {{c1::undefined index}} notice.
90. `header('Location: /path');` triggers an HTTP {{c1::redirect}}.
91. After sending a Location header, you should call {{c1::exit}} to stop further output.
92. `http_response_code(404);` sets the HTTP status code to {{c1::404}}.

Security Defaults ??

93. For HTML output, `htmlspecialchars($s, ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8')` prevents {{c1::XSS}} in HTML/text contexts.
94. `ENT_QUOTES` escapes both {{c1::single}} and {{c2::double}} quotes.
95. Passwords should be stored using {{c1::password_hash}} (not md5/sha1).
96. Verify a password with {{c1::password_verify($pw, $hash)}}.
97. SQL safety best practice: use {{c1::prepared statements}} with bound parameters (not string concatenation).
98. CSRF defense: include a per-request {{c1::token}} and verify it on submission.
99. Never trust `$_GET/$_POST` types: always {{c1::validate}} and/or {{c2::cast}} (e.g., `(int)`).
100. Output escaping is {{c1::context-dependent}} (HTML vs attribute vs URL vs JS).

Composer & Autoloading ??

101. `composer.json` declares dependencies and {{c1::autoload rules}}.
102. `composer.lock` pins the {{c1::exact versions}} installed.
103. Composer’s autoloader entry file is {{c1::vendor/autoload.php}}.
104. In code, `require __DIR__ . '/vendor/autoload.php';` enables {{c1::autoloading}}.
105. PSR-4 maps {{c1::namespaces}} to {{c2::directory paths}}.

PDO (Database) ??

06. In PDO, setting `PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION` makes DB errors throw {{c1::exceptions}}.
07. A prepared statement is created with `$pdo->{{c1::prepare}}(...)`.
08. Parameters are provided via `$stmt->{{c1::execute}}(['email' => $email])` (named placeholders).
09. Fetching one row as an associative array can be done with `$stmt->fetch(PDO::{{c1::FETCH_ASSOC}})`.
10. After an INSERT, `$pdo->{{c1::lastInsertId}}()` gets the last generated ID (driver-dependent).

WordPress Parallels ?

11. WordPress actions are registered with {{c1::add_action}}; filters with {{c2::add_filter}}.
12. A filter callback must {{c1::return}} the modified value; an action callback typically {{c2::does not}}.
13. WordPress escaping helpers: `esc_html`, `esc_attr`, {{c1::esc_url}} for URLs.
14. WordPress sanitizers include `sanitize_text_field` and {{c1::sanitize_email}}.
15. WordPress CSRF protection uses {{c1::nonces}} (e.g., `wp_nonce_field`, `check_admin_referer`).
16. `$wpdb->prepare(...)` is the WordPress pattern for {{c1::safe SQL}}.

“I Forget This” Reminders ??

17. Prefer the full opening tag {{c1::<?php}} (avoid short tags).
18. In pure PHP files, it’s common to omit the closing tag `?>` to avoid accidental {{c1::whitespace output}}.
19. In PHP, the string `'0'` is {{c1::falsy}} (so strict comparisons can matter).
20. HTTP headers must be sent before any {{c1::output}} (even whitespace), otherwise you get “headers already sent.”

Extra High-Value Additions (Fits the Topic) ?

---

21. `error_reporting(E_ALL);` and `ini_set('display_errors', '1');` are useful in {{c1::development}} (but not in production).
22. Prefer `filter_input(INPUT_GET, 'q', FILTER_SANITIZE_SPECIAL_CHARS)` for simple input handling, but still {{c1::validate}} properly.
23. `json_encode($data, JSON_UNESCAPED_UNICODE)` produces {{c1::JSON}} output; set header `Content-Type: application/json`.
24. To read JSON request bodies: `$raw = file_get_contents('php://input'); $data = json_decode($raw, true);` — `true` yields an {{c1::associative array}}.
25. `isset($x)` is false if `$x` is {{c1::not set}} or {{c2::null}}.
26. `empty($x)` treats values like `0`, `'0'`, `[]`, and `null` as {{c1::empty}} (be careful).
27. Use `===` when checking `strpos(...)` results because position {{c1::0}} is a valid match but is {{c2::falsy}}.
28. The directory constant `__DIR__` gives the current file's {{c1::directory path}} (safer than relative paths).
29. `require` vs `require_once`: `require_once` adds overhead; prefer {{c1::autoloading}} for classes instead of many `*_once`.
30. Use `password_hash` with `PASSWORD_DEFAULT` so the algorithm can {{c1::upgrade over time}}.
31. In prepared statements, never interpolate variables directly into SQL; bind them as {{c1::parameters}}.
32. In PHP 8+, `match` has no fall-through and will throw `UnhandledMatchError` if no case matches and there's no {{c1::default}}.
33. For arrays, `===` compares both order and types; for "same key/value pairs regardless of order," use {{c1::ksort}} first.
34. In OOP, prefer constructor injection: `__construct(private Logger $logger)` is promoted property syntax (PHP {{c1::8.0+}}).
35. For safe redirects, validate destination URLs to prevent {{c1::open redirect}} vulnerabilities.

If you tell me whether you're using **PHP 7.4, 8.0–8.4**, and whether you're focusing on **WordPress plugin/theme dev** or **general backend**, I can generate a second deck with *scenario-based* clozes (debugging, forms, auth, PDO pitfalls) □□□

Revision #4
Created 2026-04-10 03:28:38 UTC by art10m
Updated 2026-04-10 03:54:35 UTC by art10m