DevSheets

PHP

Everyday syntax, arrays, and functions.

01 Run a PHP program

```php
<?php
declare(strict_types=1);
$name = "Maya";
echo "Hello, $name!\n";
var_dump(PHP_VERSION);
```

Save as hello.php; run php hello.php. Check syntax with php -l hello.php. PHP 8+ examples stand alone: add the opening tag and strict_types declaration to later snippets. Omit the closing tag in PHP-only files.

02 Variables & values

```php
$count = 3;
$price = 19.95;
$ready = true;
$name = "Maya";
const LIMIT = 10;
var_dump($count, $price, $ready, LIMIT);
```

Variables start with $ and are case-sensitive. Common types: int, float, bool, string, array, object, and null. Assignments do not require a type declaration. Constants omit $.

03 Operators & comparisons

```php
$total = 7 / 2;       // 3.5
$whole = intdiv(7, 2); // 3
$rest = 7 % 2;        // 1
$same = 3 === "3";    // false
$ok = $whole > 0 && !$same;
var_dump($total, $whole, $rest, $ok);
```

Use === and !== to compare value and type; == allows conversions. &&, ||, and ! combine booleans. + - * / % ** do arithmetic; . concatenates strings.

04 Conditions & match

```php
$score = 82;
if ($score >= 60) { echo "Pass\n"; }
$role = "editor";
$access = match ($role) {
    "admin", "editor" => "write",
    default => "read",
};
```

elseif/else add branches. match returns a value and compares strictly, without fallthrough. Cover all cases or use default.

05 Loops & foreach

```php
for ($i = 0; $i < 3; $i++) echo $i;
$names = ["Maya", "Leo"];
foreach ($names as $index => $name) {
    echo "$index: $name\n";
}
```

while repeats while true; do...while runs at least once. foreach visits values or key/value pairs. break exits; continue skips ahead.

06 Strings

```php
$name = trim(" Maya ");
$greeting = "Hello, $name";
$literal = 'Hello, $name';
echo $greeting . "!\n";
var_dump(strlen($name));
var_dump(str_contains($name, "ay"));
```

Double quotes interpolate; single quotes mostly keep literal text. strlen counts bytes. explode splits; implode joins. Multibyte text helpers need mbstring.

07 Indexed arrays

```php
$nums = [10, 20, 30];
$nums[] = 40;
$last = array_pop($nums);
$part = array_slice($nums, 1, 2);
var_dump(count($nums), $last, $part);
var_dump(in_array(20, $nums, true));
```

PHP arrays are ordered maps. [] appends a value; indexes commonly start at zero. array_slice returns a portion. The third in_array argument enables strict comparisons.

08 Associative arrays

```php
$user = ["name" => "Maya", "age" => 28];
$user["age"] = 29;
$user["role"] = "editor";
unset($user["role"]);
var_dump($user["name"]);
var_dump(array_key_exists("age", $user));
```

Use string keys for named values. array_key_exists checks presence even if the value is null; isset is false for missing or null values. Reading a missing key raises a warning.

09 Functions & types

```php
function add(int $a, int $b = 0): int {
    return $a + $b;
}
echo add(2, 3), "\n";
echo add(b: 4, a: 1), "\n";
```

Declare parameter and return types; void means no return value. strict_types prevents scalar coercion for calls made from that file (int may satisfy float). Named arguments use parameter names.

PHP

Types, errors, and practical web APIs.

10 Null & optional values

```php
$user = ["name" => null];
$name = $user["name"] ?? "Guest";
$user["role"] ??= "reader";
var_dump($name, $user);
$profile = null;
var_dump($profile?->name);
```

?? defaults missing/null values; ??= assigns a default. ?-> stops on null. ?string permits null; int|string is a union. ?: also replaces other falsy values.

11 Closures & array helpers

```php
$nums = [1, 2, 3]; $factor = 2;
$double = fn(int $n): int => $n * $factor;
$out = array_map($double, $nums);
$yes = array_filter($out, fn($n) => $n > 2);
var_dump(array_values($yes));
```

Arrow functions capture by value. Closures use function (...) use ($x) { ... }. array_filter preserves keys; array_values reindexes them.

12 Classes & methods

```php
class Counter {
    public int $value = 0;
    public function add(): void {
        $this->value++;
    }
}
$c = new Counter(); $c->add();
```

-> accesses members; $this is the instance. public/protected/private set visibility. __construct initializes; extends inherits; implements supplies an interface.

13 Exceptions & cleanup

```php
try {
    throw new RuntimeException("Oops");
} catch (RuntimeException $e) {
    echo "Handled: ", $e->getMessage();
} finally { echo "\nFinished\n"; }
```

catch handles matching exceptions; finally runs after try/catch. Throwable covers Error and Exception. Warnings are not automatically exceptions.

14 JSON & checked decoding

```php
$flags = JSON_THROW_ON_ERROR;
try {
    $user = json_decode(
        '{"id":7}', true, 512, $flags);
} catch (JsonException $e) {
    echo "Invalid JSON data";
}
```

true decodes objects as arrays. JSON_THROW_ON_ERROR makes decode/encode failures throw. Validate the result shape; valid JSON can be a scalar.

15 Read & write files

```php
$text = file_get_contents("notes.txt");
if ($text === false) exit("Read failed");
$n = file_put_contents("copy.txt", $text);
if ($n === false || $n !== strlen($text)) {
    throw new RuntimeException("Write");
}
```

Reads return string/false; writes return byte count/false. Failure can emit warnings. Helpers close files; writing replaces the destination.

16 Validate request input

```php
$raw = $_GET["age"] ?? null;
$age = is_string($raw) ? filter_var(
    $raw, FILTER_VALIDATE_INT) : false;
if ($age === false || $age < 0) {
    http_response_code(400);
    exit("Invalid age");
}
```

Web context: $_GET/$_POST may contain arrays. Check type and domain rules. Strictly compare filter failure: zero is a valid integer.

17 Escape HTML output

```php
$raw = $_GET["name"] ?? "Guest";
$name = is_string($raw) ? $raw : "Guest";
$safe = htmlspecialchars(
    $name, ENT_QUOTES | ENT_SUBSTITUTE,
    "UTF-8");
echo "<p>Hello, " . $safe . "</p>";
```

Escape HTML text or quoted attribute values at output. JavaScript, CSS, and URLs need different rules. Validation does not replace escaping.

18 Reuse a PHP file

```php
// math.php
function twice(int $n): int { return $n*2; }
// app.php
require_once __DIR__ . "/math.php";
echo twice(3);
```

Split into two PHP files; add opening tags and strict_types. __DIR__ anchors the path. require_once loads once. Use application-controlled paths.