

【PHP】基礎

目次

1. 環境編
2. 基礎文法編
3. オブジェクト編
4. 便利編
5. クラス編
6. モジュール編
7. ウェブ開発編
8. データベース編

環境編

■環境構築

- ▶ バージョンを確認

基礎文法編

■初歩的注意

- ▶ ※ PHPファイルを用意し、`<?php` と `?>` の間にPHPのコードを書く。それらのタグで囲まれている部分にはHTMLを書くことができる。（本編、オブジェクト編、データベース編で紹介するコードではこのタグは省略する）
- ▶ ※ HTMLを書かないPHPファイルの場合、すなわちPHPのコードしか書かない場合、閉じタグは書くべきではないという理解がある。
- ▶ ※ 大文字と小文字を区別する言語である。
- ▶ ※ 文字列は"か""どちらで囲んでもいい。
- ▶ ※ ビルトイン関数について詳細を知りたいならGoogle検索。

■データ型の種類

- string 'hello' "world"
- int 5 -20

【PHP】基礎

目次

1. 環境編
2. 基礎文法編
3. オブジェクト編
4. 便利編
5. クラス編
6. モジュール編
7. ウェブ開発編
8. データベース編

環境編

■環境構築

- ▶ バージョンを確認 `$ php -v`

基礎文法編

■初歩的注意

- ▶ ※ PHPファイルを用意し、`<?php` と `?>` の間にPHPのコードを書く。それらのタグで囲まれている部分にはHTMLを書くことができる。（本編、オブジェクト編、データベース編で紹介するコードではこのタグは省略する）
- ▶ ※ HTMLを書かないPHPファイルの場合、すなわちPHPのコードしか書かない場合、閉じタグは書くべきではないという理解がある。
- ▶ ※ 大文字と小文字を区別する言語である。
- ▶ ※ 文字列は"か""どちらで囲んでもいい。
- ▶ ※ ビルトイン関数について詳細を知りたいならGoogle検索。

■データ型の種類

- string 'hello' "world"
- int 5 -20

- float 3.2 -1.8
- null null
- bool true false
- array [5, 3, 8]
- object new Object()

■基礎	
▶ コメントのしかた	
▶ 変数を定義	
▶ 定数を定義	
▶ データ型の変換	
▶ 値がfalseに等しいか	
▶ 値が設定済みかnullか	
▶ 値がnullなら別の値に	

■標準出力	
▶ 出力	
▶ 配列を出力	
▶ 型や要素数も出力	

■条件分岐	
▶ 条件分岐	
▶ 比較演算子	
▶ 論理演算子	
▶ 2 股分岐の略記	
▶ switch文	
▶ bool以外も評価します	

■繰り返し処理	
▶ n回処理を繰り返す	
▶ while文	
▶ do-while文	
▶ foreach	
▶ foreach (indexかkey付)	

- float 3.2 -1.8
- null null
- bool true false
- array [5, 3, 8]
- object new Object()

■基礎	
▶ コメントのしかた	// や # で行末まで。 /* */ で囲めば改行可能。
▶ 変数を定義	\$hoge = 値; ※大文字小文字は区別される
▶ 定数を定義	define('HOGE', 値); または const HOGE = 値;
▶ データ型の変換	\$hoge = (型の名前)値;
▶ 値がfalseに等しいか	empty(変数) ※bool以外も評価 ※nullも含むということ
▶ 値が設定済みかnullか	変数 != null ⇔ isset(変数)
▶ 値がnullなら別の値に	設定済みか怪しいもの ?? nullの場合の値 ※null合体演算子

■標準出力	
▶ 出力	echo 式; か printf("%sは%10dです", 変数1, 変数2);
▶ 配列を出力	print_r(式); ※厳密には配列だけではない。
▶ 型や要素数も出力	var_dump(式); ※独特な形式で出力

■条件分岐	
▶ 条件分岐	if elseif else ※ elseif は else if でもいい。
▶ 比較演算子	< <= > >= == === != !==
▶ 論理演算子	&& and or ! !()
▶ 2 股分岐の略記	条件式 ? 真での値 : 偽での値
▶ switch文	switch (式) { case 値: 処理; break; default: 処理; }
▶ bool以外も評価します	false ±0 ±0.0 '0' '' null []

■繰り返し処理	
▶ n回処理を繰り返す	for (\$i = 1; \$i <= \$n; \$i++) { 処理 }
▶ while文	while (条件式) { 処理; 条件に関する処理; }
▶ do-while文	do { 処理; 条件の処理; } while (条件式); ※一度は必ず実行
▶ foreach	foreach (配列 as 好きな変数) { 処理 }
▶ foreach (indexかkey付)	foreach (配列 as indexかkey用の変数 => 好きな変数) { 処理 }

▶ 中断し次へ・脱出	
------------	--

■関数

▶ 関数を定義	
▶ 名付けずに関数定義	
▶ 値返すだけの即席関数	
▶ 引数や返り値の型指定	
▶ デフォルト値を設定	
▶ 返り値がない場合	
▶ 厳格な型付けに	
▶ 可変長引数	
▶ 関数外の変数を使う	

▶ 関数の呼び出し	
▶ 引数名を指定して渡す	

- ▶ ※ 関数内で定義された変数はその関数内ではしか使えない。
- ▶ ※ もちろん配列も返せる。

■例外処理

▶ 強制終了	
▶ わざと例外を投げる	
▶ 例外を受け取って処理	
▶ 例外が発生しても処理	

オブジェクト編

■文字列

▶ 特殊な文字を表現	
▶ 改行コード	
▶ 文字列の結合	
▶ 変数展開	

▶ 中断し次へ・脱出	continue; • break;
------------	--------------------------

■関数

▶ 関数を定義	function helloWorld(\$p1, ...){ … return 値; }
▶ 名付けずに関数定義	\$変数 = function(\$p1, ...){ … return 値; };
▶ 値返すだけの即席関数	fn(\$p1, ...) => 引数を使った式
▶ 引数や返り値の型指定	function hoge(型1 \$p1, ...): 返り値の型名
▶ デフォルト値を設定	function hoge(\$p1 = 80, ...):
▶ 返り値がない場合	function hoge(\$p1, ...): void
▶ 厳格な型付けに	declare(strict_types = 1);
▶ 可変長引数	hoge(\$p1, \$p2, ...\$p3)
▶ 関数外の変数を使う	global \$変数; (必須) ※しかし非推奨

▶ 関数の呼び出し	hoge(<i>arg1</i> , ...) ※ 末尾に <code>;</code> が必要なことも当然ある
▶ 引数名を指定して渡す	hoge(\$p1: <i>arg1</i> , ...)

- ▶ ※ 関数内で定義された変数はその関数内ではしか使えない。
- ▶ ※ もちろん配列も返せる。

■例外処理

▶ 強制終了	exit; exit('何らかのメッセージ');
▶ わざと例外を投げる	try { … throw new 例外クラス名(引数あるかも); … } catch (例外クラス名 \$e) { 処理※; exit; } ※ <code>\$e->メソ</code> を使う
▶ 例外を受け取って処理	finally { 処理 } ※ catch のなかの <code>exit;</code> は消しておく！！

オブジェクト編

■文字列

▶ 特殊な文字を表現	\' \" \t
▶ 改行コード	PHP_EOL (※End of Line の略) か "\n" (※必ず"")
▶ 文字列の結合	. ※代入演算子 (<code>.=</code>) が使えます！！
▶ 変数展開	"Hello, \$name" か "Hello, \${name}" ※必ず"" か sprintf("%sは%.3fです", 変数1, 変数2)

▶ 長い文字列	
▶ 長い文字列で変数展開	
▶ 数値への変換	
▶ 文字数	
▶ （上記の日本語版）	
▶ 前後の空白除去	
▶ <i>str1</i> 中の <i>str2</i> の位置	
▶ （上記の日本語版）	
▶ 正規表現の初一致箇所	
▶ 正規表現の全一致箇所	
▶ 置換	
▶ 正規表現で置換	
▶ MID	
▶ REPLACE	
▶ TEXTJOIN	
▶ Split	

■数値	
▶ 代数演算子	
▶ 算術代入演算子	
▶ 複合代入演算子	
▶ 小数第何位まで丸める	
▶ 上の桁を0で埋める	
▶ コンマ付ける	
▶ 端数処理	
▶ $min \leq \text{乱数} < max \in \mathbb{N}$	
▶ 最大値・最小値	
▶ 円周率	
▶ 2 の平方根	

■配列	
▶ 配列を定義	

▶ 長い文字列	<code>\$text = <<<'EOT' と EOT; の間に書く</code>
▶ 長い文字列で変数展開	<code>\$text = <<<"EOT" と EOT; の間に書く ※""がなくてもいい</code>
▶ 数値への変換	自然と変換
▶ 文字数	<code>strlen(<i>str</i>)</code>
▶ （上記の日本語版）	<code>mb_strlen(<i>str</i>)</code>
▶ 前後の空白除去	<code>trim(<i>str</i>)</code>
▶ <i>str1</i> 中の <i>str2</i> の位置	<code>strpos(<i>str1</i>, <i>str2</i>)</code> ※初位置を返す(0, 1, ...)。無ければ""。
▶ （上記の日本語版）	<code>mb_strpos(<i>str1</i>, <i>str2</i>)</code>
▶ 正規表現の初一致箇所	<code>preg_match('/正規表現/', 文字列, 好きな変数);</code>
▶ 正規表現の全一致箇所	<code>preg_match_all('/正規表現/', 文字列, 好きな変数);</code>
▶ 置換	<code>str_replace(<i>what</i>, <i>replacement</i>, <i>str</i>)</code>
▶ 正規表現で置換	<code>preg_replace('/正規表現/', 置換後の文字列, 文字列)</code>
▶ MID	<code>substr(<i>str</i>, <i>start</i>, <i>length</i>)</code>
▶ REPLACE	<code>substr_replace(<i>str</i>, <i>replacement</i>, <i>start</i>, <i>length</i>)</code>
▶ TEXTJOIN	<code>implode(区切り文字, 配列)</code>
▶ Split	<code>explode(区切り文字, 文字列)</code>

■数値	
▶ 代数演算子	<code>+ - * / % **</code>
▶ 算術代入演算子	<code>= += -= *= /= %= **=</code>
▶ 複合代入演算子	<code>\$x++ ++\$x \$x-- --\$x</code>
▶ 小数第何位まで丸める	<code>sprintf("これが%10.2fです", 変数1)</code> など ※第2位までなら
▶ 上の桁を0で埋める	<code>sprintf("これが%010.2fです", 変数1)</code> など
▶ コンマ付ける	<code>number_format(数値)</code>
▶ 端数処理	<code>floor(数) ceil(数) round(数) round(数, 小数点以下桁数)</code>
▶ $min \leq \text{乱数} < max \in \mathbb{N}$	<code>mt_rand(<i>min</i>, <i>max</i>)</code>
▶ 最大値・最小値	<code>max(数1, 数2, ...)</code> <code>min(数1, 数2, ...)</code>
▶ 円周率	<code>M_PI</code> （か`pi()`）
▶ 2 の平方根	<code>M_SQRT2</code>

■配列	
▶ 配列を定義	<code>\$scores = [90, 40, 100];</code>

▶ 連想配列を定義	
▶ 要素を参照	
▶ 要素を別配列内で展開	
▶ 各要素を楽に格納	
▶ 値を交換	
▶ 空かどうか	
▶ 要素数	
▶ 頭尾に要素を追加	
▶ 頭尾の 1 要素を削除	
▶ 途中要素を追加・削除	
▶ スライス処理	
▶ 昇・降順に並び替え	
▶ キーで昇・降順に	
▶ 2次元連想配列のソート	
▶ ランダムに 1 つ取得	
▶ シャッフル	
▶ すべて同一値の配列	
▶ Pythonのrange()	
▶ 合計	
▶ 最大値・最小値	
▶ 2 つの配列を連結	
▶ 2 つの配列の差をとる	
▶ 2 つの配列の共通要素	
▶ UNIQUE	
▶ 全要素に関数を適用	
▶ FILTER	
▶ キーor値を配列に	
▶ キーor値の存在確認	
▶ 値に対応するキー	
▶ 2次元連想配列の一行	

▶ 連想配列を定義	<code>\$scores = ['abc' => 2, 'def' => true];</code>
▶ 要素を参照	配列[インデックス] 配列['キー']
▶ 要素を別配列内で展開	...配列
▶ 各要素を楽に格納	<code>list(変数1, 変数2, ...) = 配列;</code> か <code>[変数1, 変数2, ...] = 配列;</code>
▶ 値を交換	<code>[\$a, \$b] = [\$b, \$a];</code>
▶ 空かどうか	<code>empty(配列)</code> ※: 1か"
▶ 要素数	<code>count(配列)</code>
▶ 頭尾に要素を追加	<code>array_unshift(配列, 要素1, ...);</code> <code>array_push(配列, 要素1, ...);</code>
▶ 頭尾の 1 要素を削除	<code>array_shift(配列);</code> <code>array_pop(配列);</code>
▶ 途中要素を追加・削除	<code>array_splice(配列, 位置, 個数, [要素1, ...]※);</code> ※ 1 要素なら数値で
▶ スライス処理	<code>array_slice(配列, 開始位置※, 個数)</code> ※負の数OK
▶ 昇・降順に並び替え	<code>sort(\$arr);</code> ・ <code>rsort(\$arr);</code> キーも一緒に: <code>asort(\$arr);</code> ・ <code>arsort(\$arr);</code>
▶ キーで昇・降順に	<code>ksort(\$arr);</code> ・ <code>krsort(\$arr);</code>
▶ 2次元連想配列のソート	<code>usort(\$arr, 独特な関数);</code> 細かく: <code>array_multisort(列配列1, ..., \$arr);</code>
▶ ランダムに 1 つ取得	<code>array_rand(配列, 個数)</code> ※ キーを返す
▶ シャッフル	<code>shuffle(配列);</code>
▶ すべて同一値の配列	<code>array_fill(start_index, count, value)</code>
▶ Pythonのrange()	<code>range(start, end)</code> (<code>range(start, end, step)</code>)
▶ 合計	<code>array_sum(配列)</code>
▶ 最大値・最小値	<code>max(配列)</code> <code>min(配列)</code>
▶ 2 つの配列を連結	<code>array_merge(配列1, 配列2)</code> ※ [...配列1, ...配列2] と同義
▶ 2 つの配列の差をとる	<code>array_diff(配列1, 配列2)</code> ※配列1 - 配列2 (~差集合)
▶ 2 つの配列の共通要素	<code>array_intersect(配列1, 配列2)</code> ※配列1 配列2 (~積集合)
▶ UNIQUE	<code>array_unique(配列)</code>
▶ 全要素に関数を適用	<code>array_map(関数, 配列)</code>
▶ FILTER	<code>array_filter(配列, 真偽値を返す関数)</code>
▶ キーor値を配列に	<code>array_keys(配列)</code> <code>array_values(配列)</code>
▶ キーor値の存在確認	<code>array_key_exists(キー, 配列)</code> <code>in_array(値, 配列)</code> ※: 1か"
▶ 値に対応するキー	初: <code>array_search(キー, 配列)</code> 全: <code>array_keys(配列, キー)</code>
▶ 2次元連想配列の一行	<code>array_column(配列, 抜き出したい列のキー)</code>

便利編

■ファイル操作

▶ 開く	
▶ 書き込む	
▶ あるサイズ`まで読み込む	
▶ 一行ずつ読み込む	
▶ 閉じる	
▶ ファイルサイズ	
▶ 開かずに書き込む	
▶ 開かずに読み込む	
▶ 各行が要素の配列に	

■ディレクトリ操作

▶ 開く	
▶ 1 ファずつ読み込む	
▶ ファを検索→配列	
▶ 存在確認	
▶ 書き込み可能か	
▶ 読み込み可能か	
▶ ベースネーム	

■日時

▶ UNIXタイムスタンプ	
---------------	--

クラス編

■用語

▶ オブジェクト	
▶ クラス	

便利編

■ファイル操作

▶ 開く	<code>fopen(file, 'r'や'w'や'a'など)</code> ※: FPLリリース
▶ 書き込む	<code>fwrite(FPLリリース, 文字列);</code>
▶ あるサイズ`まで読み込む	<code>fread(FPLリリース, 最大何バイトまで読み込むか)</code>
▶ 一行ずつ読み込む	<code>fgets(FPLリリース)</code> ※読み込めなくなったらfalseを返す
▶ 閉じる	<code>fclose(FPLリリース);</code>
▶ ファイルサイズ	<code>fsize(file)</code>
▶ 開かずに書き込む	<code>file_put_contents(file, 文字列);</code>
▶ 開かずに読み込む	<code>file_get_contents(file, 文字列)</code>
▶ 各行が要素の配列に	<code>file(file, FILE_IGNORE_NEW_LINES)</code>

■ディレクトリ操作

▶ 開く	<code>opendir(dir)</code> ※: DPLリリース
▶ 1 ファずつ読み込む	<code>readdir(DPLリリース)</code> ※最後falseを返す ※.や..も含む
▶ ファを検索→配列	<code>glob('ワイルドカード付きパス')</code>
▶ 存在確認	<code>file_exists(path)</code> ※ディモ。 ※: 1か"
▶ 書き込み可能か	<code>is_writeable(path)</code> ※ディモ。 ※: 1か"
▶ 読み込み可能か	<code>is_readable(path)</code> ※ディモ。 ※: 1か"
▶ ベースネーム	<code>basename(path)</code> ※パスから純粋なファイル名のみにする

■日時

▶ UNIXタイムスタンプ	<code>time()</code> ※1970年1月1日0:00 からの経過秒秒
---------------	--

クラス編

■用語

▶ オブジェクト	クラスで定義し生成する
▶ クラス	オブジェクトを定義したもの

モジュール編

■他のPHPファイルとの連携

▶ 外部ファ読み	
▶ 既読なら読込まず	
▶ クラスを自動読み	
▶ 名前空間	
▶ 外部で例外を投げ 基ファで例外処理	

ウェブ開発編

■開発環境導入

- ▶ ☆ 開発環境の設定のイメージ
- ▶ ☆ 設定の手順
- ▶ ☆ 「localhost で接続が拒否されました」と出る場合
- ▶ ※ 「An error occurred」と出る場合 → Hyper-V を自動にしなければならない

■PHPを埋め込む

- ▶ ☆ PHPを埋め込む方法
- ▶ <?php echo ○○ ?> の略記
- ▶ 記号がコードとして解釈されぬよう
- ▶ ※ { … } を使った構文は : … end○○; に書き換えられる（do-whileを除く）。
- ▶ ※ マークアップ部分とPHP部分の関係は、小説でのセリフと地の文の関係に似ている

■フォームから値を受け取る

- ▶ GET形式で送られたら
- ▶ 空白なら～～する
- ▶ 改行も反映
- ▶ 複数の値を受け取る

モジュール編

■他のPHPファイルとの連携

▶ 外部ファ読み	require(PHPファイルのパス); ※読み失敗なら止まる include(PHPファイルのパス); ※読み失敗しても止まらない
▶ 既読なら読込まず	require_once(PHPファイルのパス); include_once(PHPファイルのパス);
▶ クラスを自動読み	spl_autoload_register(function (\$class) { require(\$class . '.php'); });
▶ 名前空間	namespace 好きな名前※; ※ベンダー名\プロジェクト名 など use その名前 as 短名; 短名\クラス名
▶ 外部で例外を投げ 基ファで例外処理	throw new Exception('例外メッセージ'); try { … } catch (Exception \$e) { … \$e->メソ; … }

ウェブ開発編

■開発環境導入

- ▶ ☆ 開発環境の設定のイメージ
- ▶ ☆ 設定の手順
- ▶ ☆ 「localhost で接続が拒否されました」と出る場合
- ▶ ※ 「An error occurred」と出る場合 → Hyper-V を自動にしなければならない

■PHPを埋め込む

- ▶ ☆ PHPを埋め込む方法
- ▶ <?php echo ○○ ?> の略記 <?= ○○ ?>
- ▶ 記号がコードとして解釈されぬよう htmlspecialchars(str, ENT_QUOTES, 'UTF-8')
- ▶ ※ { … } を使った構文は : … end○○; に書き換えられる（do-whileを除く）。
- ▶ ※ マークアップ部分とPHP部分の関係は、小説でのセリフと地の文の関係に似ている

■フォームから値を受け取る

- ▶ GET形式で送られたら filter_input(INPUT_GET, 'name属性の値')
- ▶ 空白なら～～する trim() を利用
- ▶ 改行も反映 nl2br() で囲む
- ▶ 複数の値を受け取る name="names[]"
filter_input(INPUT_GET, 'names',

▶ ラジオボタンから受け取る	
■Cookieを使う	
▶ Cookieをセット	
▶ Cookieの値を読み出す	
▶ Cookieを削除	
▶ ※ CookieはChromeのデベロッパーツールで管理できる	
■Sessionを使う	
▶ ☆ Cookie と Session の違い	
▶ Sessionを使い始める	
▶ Sessionをセット	
▶ Sessionの値を読みだす	
▶ Sessionを削除	
■Webページ	
▶ リダイレクト	
■POST形式	
▶ ☆ GETとPOSTの違い	
▶ POSTで値を受け取る	
▶ POSTでアクセスされたかどうか	
▶ 二重の送信を防ぐ	
▶ 外部からの送信（CSRF）を防ぐ	

データベース編

■開発環境導入	
▶ ☆ 開発環境の設定のイメージ	※ Web開発編でのそれに追加がされている
▶ ☆ db コンテナにログイン	
■PDO（PHP Database Objects）	
▶ ※ PDOとは、PHPからDBへのアクセスを抽象化するためのオブジェクトのこと。PDO を使って処理を実装しておけば、データベースの種類がこのように変わったとしても、こちらで	

	FILTER_DEFAULT, FILTER_REQUIRE_ARRAY)
▶ ラジオボタンから受け取る	未選択ならnullが渡されることに注意
■Cookieを使う	
▶ Cookieをセット	setcookie(<i>Name</i> , <i>Value</i>); ※ 手前で出力禁止！
▶ Cookieの値を読み出す	filter_input(INPUT_COOKIE, <i>Name</i>)
▶ Cookieを削除	setcookie(<i>Name</i> , '');
▶ ※ CookieはChromeのデベロッパーツールで管理できる	
■Sessionを使う	
▶ ☆ Cookie と Session の違い	
▶ Sessionを使い始める	session_start();
▶ Sessionをセット	\$_SESSION[<i>Name</i>] = <i>Value</i> ;
▶ Sessionの値を読みだす	\$_SESSION[<i>Name</i>]
▶ Sessionを削除	unset(\$_SESSION[<i>Name</i>]);
■Webページ	
▶ リダイレクト	header('Location: <i>URL</i> '); ※ 手前で出力禁止！
■POST形式	
▶ ☆ GETとPOSTの違い	
▶ POSTで値を受け取る	filter_input(INPUT_POST, 'name属性の値')
▶ POSTでアクセスされたかどうか	\$_SERVER['REQUEST_METHOD'] === 'POST'
▶ 二重の送信を防ぐ	→ フォームの送信先を自身にする
▶ 外部からの送信（CSRF）を防ぐ	トークンを使う

データベース編

■開発環境導入	
▶ ☆ 開発環境の設定のイメージ	※ Web開発編でのそれに追加がされている
▶ ☆ db コンテナにログイン	
■PDO（PHP Database Objects）	
▶ ※ PDOとは、PHPからDBへのアクセスを抽象化するためのオブジェクトのこと。PDO を使って処理を実装しておけば、データベースの種類がこのように変わったとしても、こちらで	

やりとりしているコードを変える必要がないような仕組みになっている。

- ▶ ※ セットアップされているマシン (host) : `db` データベース : `myapp`
そこにアクセスするためのユーザー名 : `dbuser` パスワード : `dbpass` とする。

- ▶ ※ ここでは MySQL (MariaDB) を利用している場合の書き方を紹介する。

- ▶ PDOを生成
- ▶ PDOの設定を変更
- ▶ クエリを投げる
- ▶ 結果を受け取る
- ▶ 結果を出力

- ▶ ☆ クラスのインスタンスとして結果を受け取る

■バインド (bind)

- ▶ ※ バインドとは、クエリに値を埋め込むこと。プレースホルダという記法により埋め込みたい箇所を明示したうえで、幾通りかの方法で最終的に値を埋め込む。

- ▶ バインド箇所の明示

- ▶ 実際にバインドする

- ▶ ☆ `query()` メソッドで直接 値を埋め込むのではなく `prepare()` メソッドを使う理由

- ▶ ※ プレホルのあたりをシングルクォートが取り囲んでいると埋め込んでくれない。

■直前に処理した録の情報

- ▶ 直前に削除挿入更新
クエリに遭った録の数
- ▶ 直前の挿入の録のID

■トランザクション

- ▶ 開始と終了
- ▶ 開始前の状態を回復

やりとりしているコードを変える必要がないような仕組みになっている。

- ▶ ※ セットアップされているマシン (host) : `db` データベース : `myapp`
そこにアクセスするためのユーザー名 : `dbuser` パスワード : `dbpass` とする。

- ▶ ※ ここでは MySQL (MariaDB) を利用している場合の書き方を紹介する。

- ▶ PDOを生成 `$pdo = new PDO('dataSourceName', 'userName', 'password');`
- ▶ PDOの設定を変更 `$pdo = new PDO(" , " , " , [設定項目1 => モード1, ...])`
- ▶ クエリを投げる `$pdo->query("クエリ");` や `$stmt = $pdo->query("クエリ");`
- ▶ 結果を受け取る
結果が 1 つの録だけなら `$result = $stmt->fetch();`
結果が複数なら `$results = $stmt->fetchAll();`
- ▶ 結果を出力
1 録 `var_dump($result);`
複数 `foreach ($results as $result) { echo $result['列1'] . PHP_EOL; }`

- ▶ ☆ クラスのインスタンスとして結果を受け取る

■バインド (bind)

- ▶ ※ バインドとは、クエリに値を埋め込むこと。プレースホルダという記法により埋め込みたい箇所を明示したうえで、幾通りかの方法で最終的に値を埋め込む。

- ▶ バインド箇所の明示 `$stmt = $pdo->prepare("クエリ※");`
※埋めたい箇所の「全てを `?`」か「各々を `:好きな名前`」に

- ▶ 実際にバインドする `execute()` や `bindValue()` (や `bindParam()`) を用いる

- ▶ ☆ `query()` メソッドで直接 値を埋め込むのではなく `prepare()` メソッドを使う理由

- ▶ ※ プレホルのあたりをシングルクォートが取り囲んでいると埋め込んでくれない。

■直前に処理した録の情報

- ▶ 直前に削除挿入更新 `$stmt = $pdo->prepare("削挿更クエリ");` `$pdo->execute(適宜引数);`
クエリに遭った録の数 `$stmt->rowCount()` ※必ず実行 `execute()` が必要1
- ▶ 直前の挿入の録のID `$pdo->lastInsertId()` ※ `$stmt` ではなく **`$pdo`** であることに注意

■トランザクション

- ▶ 開始と終了 `try { $pdo->beginTransaction(); 一連処理 $pdo->commit(); }`
- ▶ 開始前の状態を回復 `catch (例外クラス名 $e) { $pdo->rollback(); .. exit; }`