Desarrollo web en entorno servidor

Versión 1.0

Ricardo Pérez López

abril de 2018

Índice general

Parte I

Introducción

CAPÍTULO 1

Preparación del entorno de desarrollo

1.1 Instalación automatizada

1.1.1 Acciones previas

Instalar `git`

Crear cuenta en GitHub

Usar https://github.com/ricpelo/conf y seguir las instrucciones del README.md

1.2 Terminal

1.2.1 Zsh

1.2.2 Oh My Zsh

1.3 Navegador

1.3.1 Herramientas de desarrollador

1.4 Editores de texto

1.4.1 Vim y less

1.4.2 Atom

CAPÍTULO 2

Introducción al desarrollo web

2.1 Conceptos básicos

2.1.1 Navegadores y servidores web

2.1.2 Agentes de usuario

2.1.3 Web estática vs. dinámica

2.1.4 Estructura vs. contenido

2.2 Ejemplos de aplicaciones web

2.2.1 Redes sociales: Facebook, Twitter...

2.2.2 Comercio electrónico: Amazon, eBay...

2.2.3 Administración electrónica...

2.2.4 Portales

2.2.5 ERP, CRM

2.3 Tecnologías de desarrollo de aplicaciones web

2.3.1 .NET

2.3.2 Java

2.3.3 Ruby/Rails

2.3.4 Python/Django

Odoo

PrestaShop

Drupal

WordPress

CAPÍTULO 3

Protocolo HTTP y lenguaje HTML

3.1 Arquitectura cliente/servidor

3.2 HTML 5 básico

3.3 Protocolo HTTP

3.3.1 URIs

URL encoding

3.3.2 Peticiones y respuestas

3.3.3 Métodos: GET y POST

3.3.4 Cabeceras HTTP

3.3.5 Códigos de estado

3.3.6 Experimentos

`telnet`

`netcat`

`curl`

`http`

Google Chrome Developer Tools

3.3.7 Envío de datos al servidor

Mediante GET

Mediante POST

Formularios HTML

3.3.8 Cookies

3.4 Apache básico

3.4.1 Instalación

3.4.2 Configuración básica

3.4.3 Sitios virtuales

3.5 Scripts CGI

3.5.1 Configuración de Apache

3.5.2 Ejemplos en Ruby

CAPÍTULO 4

Sistemas de control de versiones

4.1 Git básico

1. config, init, add, commit
2. status, log, diff
3. Alias lg
4. checkout, reset, revert, –amend
5. show
6. rm, mv
7. Atom y Git

4.2 Git avanzado

1. Ramas: branch
2. merge, rebase
3. Resolución de conflictos

4.3 Ramas remotas

1. clone, fetch, push, pull
2. Ramas de seguimiento (tracking branch)

4.4 Repositorios de código (Github.com, GitLab.com, Bitbucket.org...)

Parte II

PHP

CAPÍTULO 5

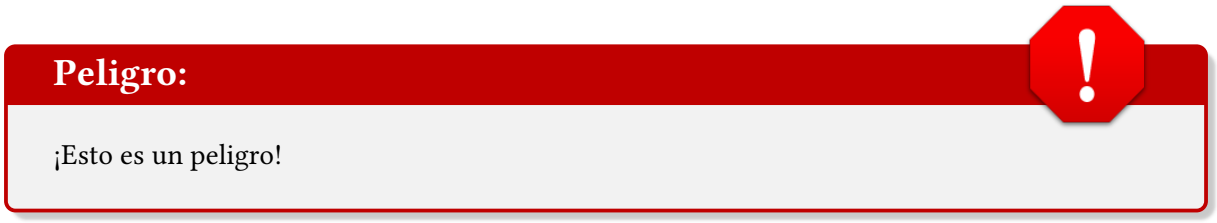
Conceptos básicos de PHP I

ricpelo's note: Programada inicialmente para empezar el 23-10-2017. _____

```
Could not open input file: ../../p.php

$ php ../../p.php
Could not open input file: ../../p.php
```

5.1 Introducción a PHP



5.1.1 Página web de PHP

<http://php.net>

5.1.2 Instalación de PHP

5.1.3 Documentación y búsqueda de información

5.1.4 Configuración básica con `php.ini`

```
error_reporting = E_ALL

display_errors = On
```

```
display_startup_errors = On

date.timezone = 'UTC'
```

5.2 Sintaxis básica

<http://php.net/manual/es/language.basic-syntax.php>

5.2.1 Datos e instrucciones

En todo lenguaje de programación existen dos elementos básicos: los **datos** y las **instrucciones**. Los datos son las porciones de información con las que trabajan los programas, y las instrucciones son las acciones que manipulan esos datos para llevar a cabo la tarea para la que fue concebido el programa. Ambos elementos, datos e instrucciones, constituyen los pilares del lenguaje y de los programas que se escriben con él.

Desde un punto de vista sintáctico, en el código fuente del programa, los datos se codifican en forma de **expresiones**, y las instrucciones toman la forma de **sentencias**.

La diferencia fundamental entre una expresión y una sentencia es la siguiente:

- Una expresión tiene un valor (se dice que *denota* o *representa* un valor), y por eso decimos que una expresión *se puede evaluar*, y al evaluarse, se obtiene el valor de la expresión, que no es más que un dato.
- En cambio, una sentencia no denota ningún valor, por lo que no puede evaluarse. El intérprete *ejecuta* la sentencia y se llevan a cabo las acciones que provoca dicha ejecución.

Las expresiones se evalúan. Las sentencias se ejecutan.

Ejemplos de expresiones

```
5 * (3 + 6)
```

Es una expresión aritmética que involucra números enteros y cuyo valor es `45`.

```
1.6 + 2.3
```

Es una expresión aritmética que involucra números reales y cuyo valor es `3.9`.

```
74
```

Es una constante literal numérica cuyo valor es, precisamente, `74`.

5.2.2 Sentencias y comandos

Las sentencias en PHP pueden ser *simples* o *compuestas*.

- Las **sentencias simples** son las instrucciones más elementales del lenguaje y se escriben siempre acabadas en punto y coma (`;`).
- Las **sentencias compuestas** corresponden a las **estructuras de control** y se estudiarán posteriormente en este capítulo.

Se puede construir una sentencia simple usando sencillamente una expresión y acabándola en punto y coma, como por ejemplo:

```
8 + 3;
```

Pero una sentencia así no tendría mucha utilidad, ya que el intérprete de PHP se limitaría a evaluar la expresión pero no haría nada más con el valor calculado.

Las sentencias realmente útiles son aquellas que provocan **efectos laterales**, es decir, acciones que provocan cambios en el estado interno del programa o que producen resultados que se vuelcan hacia la *salida* (siendo esta cualquier dispositivo de salida, como por ejemplo la pantalla, un archivo del disco o una fila de una tabla de una base de datos relacional).

Otra forma de construir una sentencia simple es usar **comandos**. PHP dispone de varios comandos con los que se pueden escribir sentencias para llevar a cabo instrucciones sencillas. Cada comando consta de una **palabra clave**, que identifica al comando, y de una serie de *argumentos* que completan la sentencia.

El comando `echo`

El ejemplo clásico de comando en PHP es `echo`[1]. El comando `echo` vuelca a la salida el valor de las expresiones que se indican como parámetro en la sentencia. Por ejemplo:

```
echo 25 * 3;
```

Muestra `75` por la salida (normalmente la pantalla). O bien:

```
echo '¡Hola a todos!';
```

Muestra `¡Hola a todos!`.

Puede mostrar varios valores, separando cada uno de ellos entre sí con una coma:

```
echo 'El resultado es: ', 4 * 2;
```

Mostraría `El resultado es: 8`.

5.2.3 Expresiones

El otro tipo de construcción sintáctica que existe en PHP además de las sentencias son las *expresiones*. Una expresión *denota* o *representa* un valor. Una expresión puede ser tan simple

[1] http://php.net/manual/es/function.echo.php

como una constante literal (por ejemplo, el número `25`) o tan compleja que involucre constantes, variables, operadores, funciones, métodos... combinados todos ellos entre sí para formar una única expresión.

Operadores

Un **operador** es un símbolo que representa una operación que se desea realizar sobre sus **operandos**[2]. Los operandos son los valores sobre los que actúa el operador para llevar a cabo la operación deseada. Por ejemplo:

```
4 + 3
```

Aquí, el operador `+` representa la operación *suma* a realizar sobre los números `4` y `3`, que son sus operandos. Como el operador actúa sobre dos operandos, se dice que es un operador *binario*. En cambio:

```
-17
```

Aquí se usa el operador `-` (*signo menos*) para convertir en negativo el valor `17`. Como el operador actúa sobre un único operando, se dice que es un operador *unario*.

En PHP existe un único operador *ternario* que se estudiará posteriormente.

En una misma expresión pueden actuar varios operadores, como en:

```
4 + 3 + 5
```

Que denota el valor `12`, o con varios operadores diferentes:

```
4 + 3 * 5
```

Que evalúa a `19`.

Asociatividad y prioridad

Todas las expresiones anteriores son ejemplos de expresiones *artiméticas*, donde se realizan las operaciones matemáticas usuales (suma, resta, producto y división) sobre números. La evaluación de una expresión (ya sea aritmética o de cualquier otro tipo) depende de las reglas de **asociatividad** y **prioridad** de los operadores que participan en dicha expresión, las cuales tenemos que conocer para entender cómo evaluará el intérprete las expresiones que formen parte de nuestro programa. En el caso de las expresiones aritméticas, las reglas son las habituales que aprendimos en el colegio:

- En una expresión en la que un operando está rodeado a izquierda y derecha por *el mismo operador*, se aplica la regla de la *asociatividad*. Por ejemplo, en la expresión:

[2] El número de operandos de un operador se denomina **aridad**. La aridad puede ser 1, 2 ó 3, según el operador sea *unario*, *binario* o *ternario*, respectivamente.

```
4 + 3 + 5
```

el operando `3` tiene el mismo operador a izquierda y derecha (el `+`), y como dicho operador es *asociativo por la izquierda*, la expresión se evalúa igual que si se hubiera escrito como:

```
(4 + 3) + 5
```

- En una expresión en la que un operando está rodeado a izquierda y derecha por *distintos operadores*, se aplica la regla de la *prioridad*. Por ejemplo, en la expresión:

```
4 + 3 * 5
```

el operando `3` tiene el operador `+` a su izquierda y el `*` a su derecha, pero como el producto tiene más prioridad que la suma, la expresión se evalúa igual que si se hubiera escrito como:

```
4 + (3 * 5)
```

Como se aprecia en los ejemplos anteriores, se pueden usar **paréntesis** para agrupar subexpresiones dentro de una expresión y así aumentar la prioridad de los operadores que vayan entre paréntesis. Por ejemplo, en la expresión:

```
(4 + 3) * 5
```

la suma se hace antes que el producto, aunque este último sea un operador de mayor prioridad. El resultado de dicha expresión es el valor `35`.

Funciones

Las funciones en las expresiones cumplen el mismo papel que en Matemáticas: realizan un cálculo a partir de unos valores de entrada indicados en sus argumentos y *devuelven* el resultado de dicho cálculo. Por ejemplo, la función *coseno* (abreviada como `cos()`[3]) calcula el coseno de un ángulo. En Matemáticas (y en Programación) se representa indicando el nombre de la función y, a continuación, la lista de sus argumentos entre paréntesis y separados por comas. Así, para calcular el coseno de 2.4 radianes, podemos escribir:

```
cos(2.4)
```

Que da como resultado `-0.73739371554125`, y ese sería el valor de dicha expresión.

El coseno es un ejemplo de función con un único argumento, pero hay funciones que admiten o requieren más argumentos. Es el caso de la función `max()`[4], que devuelve el valor máximo de todos los indicados en su lista de argumentos:

```
max(5, 3, 8, 2)
```

[3] http://php.net/manual/es/function.cos.php
[4] http://php.net/manual/es/function.max.php

Devuelve `8`.

Nota:

Cuando usamos una función en una expresión, decimos que estamos *llamando* o *invocando* a la función. La aparición de la función en la expresión es una *llamada* a la función.

En PHP, a diferencia de lo que ocurre en Matemáticas, existen funciones que no devuelven ningún valor, ya que su objetivo es provocar un *efecto lateral*. La más conocida podría ser, sin temor a equivocarnos, la función `var_dump()`[5]. Esta función muestra en la salida información estructurada sobre las expresiones que se le pasan como argumento, incluyendo su valor y su tipo. En cierto sentido, podría considerarse un versión especializada del comando `echo`, pero en forma de función y más orientada a la *depuración* de programas.

Es importante destacar que esa información que muestra se vuelca *en la salida* (normalmente la pantalla). No estamos diciendo que la función *devuelva* dicha información o que esa información sea el valor resultante de *evaluar* la llamada a la función. De hecho, estamos hablamos de llamar a la función como si fuera una sentencia (una sentencia formada únicamente por la llamada a la función y el punto y coma final):

```
var_dump(14 + 3);
```

La sentencia anterior (sí: *sentencia*, porque es una instrucción en sí misma, terminada en punto y coma), como cualquier otra sentencia, no devuelve ningún valor, sino que produce un efecto lateral. En este caso, mostrar a la salida (la pantalla) lo siguiente:

```
int(17)
```

Observamos que `var_dump()` nos informa del valor de la expresión (`17`) y de su tipo (`int`, que significa **número entero**). La importancia de conocer el valor y el tipo de toda expresión que aparezca en nuestros programas se apreciará en breve. Por ahora, veamos dos ejemplos más de utilización de `var_dump()`:

```
var_dump(12.3 - 4);
```

Muestra a la salida:

```
float(8.3)
```

Lo que nos indica que el valor de la expresión `12.3 - 4` es el **número real** `8.3`. Finalmente:

```
var_dump("Saludos");
```

Muestra:

```
string(7) "Saludos"
```

Que nos informa de que `"Saludos"` es una cadena de siete caracteres.

[5] http://php.net/manual/es/function.var-dump.php

Los números (enteros y reales), así como las cadenas, son algunos de los **tipos de datos** que el lenguaje PHP nos proporciona para ayudarnos a manipular la información. Su estudio detallado se hará posteriormente.

5.3 Funcionamiento del intérprete

PHP es un lenguaje *interpretado* y, como tal, requiere de la existencia de un **intérprete**, que es la utilidad encargada de leer el código fuente escrito en el lenguaje y ejecutarlo adecuadamente siguiendo las reglas de dicho lenguaje.

La ejecución de nuestro código se puede llevar a cabo de dos formas:

Por lotes: Nuestro código fuente está almacenado en un archivo de texto (normalmente, con extensión `.php`) y el intérprete lee dicho archivo, lo analiza sintáctica y semánticamente y ejecuta las instrucciones que lo forman. Estos archivos (que contienen el código fuente en PHP) se denominan **scripts**, y el objetivo final de este curso es desarrollar aplicaciones escribiendo los *scripts* necesarios para cumplir con la funcionalidad deseada.

Una manera de ejecutar nuestro *script* es pasárselo al intérprete desde la consola del sistema operativo[6]. Por ejemplo, si tenemos nuestro *script* almacenado en el archivo `prueba.php`, podemos provocar la ejecución del mismo mediante la siguiente orden del sistema operativo:

```
$ php prueba.php
```

Interactiva: El intérprete interactivo solicita al usuario que introduzca una sentencia, normalmente por teclado. Una vez introducida, el intérprete la analiza, la ejecuta y vuelve a solicitar al usuario la introducción de una nueva sentencia. El usuario, por tanto, ve inmediatamente el efecto que produce la ejecución de la sentencia que acaba de introducir en el intérprete interactivo.

<http://php.net/manual/es/language.basic-syntax.phpmode.php>

5.3.1 Intérprete interactivo

La meta principal de este curso es escribir programas y, por tanto, la ejecución por lotes es la más importante y la razón de ser del lenguaje. Pero el intérprete interactivo resulta muy útil para hacerse con el manejo básico del lenguaje y para realizar pruebas rápidas sin necesidad de tener que escribir un programa expresamente para tal fin. Por ello, su utilidad didáctica es innegable, así que empezaremos con él.

Intérprete interactivo integrado (`php -a`)

PHP dispone de un intérprete interactivo integrado que, si bien es algo espartano y no dispone de características adicionales que sí podemos encontrar en otros intérpretes más avanzados,

[6] La otra forma es ejecutar el *script* en el contexto de un **servidor web**, *embebiendo* (o *incrustando*) el código PHP en una página HTML. Hablaremos sobre ello en próximos capítulos.

tiene lo justo y necesario para cumplir su función.

Para empezar a trabajar con dicho intérprete de forma interactiva, usaremos el comando `php` con la opción `-a` desde la consola del sistema operativo:

```
$ php -a
Interactive mode enabled

php >
```

El intérprete nos muestra el *prompt* `php >`, indicándonos que está listo para recibir nuestras sentencias PHP. Probamos:

```
php > echo 75;
75
php > echo 23 * 5;
115
php > echo "Hola a todos";
Hola a todos
php > var_dump(3 + 5);
int(8)
php >
```

Como se ve, el intérprete ejecuta inmediatamente el comando introducido, llevando a cabo las operaciones indicadas en la instrucción (en este caso, evaluar la expresión y mostrar el resultado en pantalla) y, a continuación, solicita un nuevo comando al usuario.

La sentencia introducida debe ser sintácticamente correcta. Si, por ejemplo, nos olvidamos de escribir el punto y coma (`;`), no obtendremos el resultado esperado:

```
php > echo 73
php > echo 25;
PHP Parse error: syntax error, unexpected 'echo' (T_ECHO), expecting ',' or
';' in php shell code on line 2

Parse error: syntax error, unexpected 'echo' (T_ECHO), expecting ',' or ';'
in php shell code on line 2

php >
```

El mensaje de error se debe a que el intérprete ha considerado los dos comandos `echo` como si fueran una única sentencia:

```
echo 73 echo 25;
```

puesto que sólo ha encontrado un `;` al final del todo. Por tanto, se queja de que se ha encontrado la palabra `echo` detrás del `73` cuando se esperaba una `,` o un `;`.

Por otra parte, si introducimos como sentencia una expresión acabada en `;`, no obtendremos ningún resultado en pantalla, ya que la expresión se evaluará sin más pero no se hará nada con dicho valor:

```
php > 3 + 5;
php > 6 * 9;
php >
```

Lo que demuestra su nula utilidad práctica.

Para salir del intérprete interactivo, pulsamos la combinación de teclas `Control-D`:

```
php > ^D
$
```

PsySH

PsySH[7] es una interesantísima aplicación desarrollada por Justin Hileman (y otros) que proporciona un intérprete interactivo para PHP bastante más potente y cómodo que el intérprete interactivo integrado que trae PHP de serie. Entre sus características, incluye:

- Un *depurador* integrado que facilita la introspección de los programas y la localización de errores.
- Autocompletado pulsando `Tab`.
- Uso adecuado de espacios de nombres.
- Histórico de órdenes introducidas.
- Visualización a todo color.
- Admite sentencias y expresiones.

La verdad es que, existiendo una herramienta así, no tiene demasiado sentido usar el intérprete interactivo integrado de PHP. Tú simplemente haz la prueba, comprueba la diferencia y dime si tengo razón o no...

La instalación de PsySH es muy sencilla:

```
$ wget https://git.io/psysh
$ chmod +x psysh
$ sudo mv -f psysh /usr/local/bin
```

Con esto tenemos la herramienta básica. Si además queremos disponer del manual de PHP en línea (cosa altamente recomendable), hacemos también lo siguiente:

```
$ wget https://psysh.org/manual/es/php_manual.sqlite
$ mkdir -p ~/.local/share/psysh
$ mv -f php_manual.sqlite ~/.local/share/psysh
```

5.3.2 Modo dual de operación

ricpelo's note: Se llaman *modo HTML* y *modo PHP*.

[7] http://psysh.org/

5.3.3 Etiquetas `<?php` y `?>`

5.4 Variables

<http://php.net/manual/es/language.variables.php>

5.4.1 Conceptos básicos

<http://php.net/manual/es/language.variables.basics.php>

5.4.2 Destrucción de variables

<http://php.net/manual/es/function.unset.php>

5.4.3 Operadores de asignación por valor y por referencia

<http://php.net/manual/es/language.operators.assignment.php>

ricpelo's note: En `$b =& $a;`, `$b` **NO** está apuntando a `$a` o viceversa. Ambos apuntan al mismo lugar. <http://php.net/manual/es/language.references.whatdo.php>

5.4.4 Variables predefinidas

<http://php.net/manual/es/reserved.variables.php>

ricpelo's note: `$_ENV` no funciona en la instalación actual (ver `variables_order` en `php.ini`. Habría que usar `get_env()`.

5.5 Tipos básicos de datos

<http://php.net/manual/es/language.types.intro.php>

5.5.1 Lógicos (`bool`)

<http://php.net/manual/es/language.types.boolean.php>

ricpelo's note: Se escriben en minúscula: `false` y `true`.
<https://github.com/yiisoft/yii2/blob/master/docs/internals/core-code-style.md#51-types>
ricpelo's note: `boolean` es sinónimo de `bool`, pero debería usarse `bool`.

Operadores lógicos

<http://php.net/manual/es/language.operators.logical.php>

ricpelo's note: *Cuidado*:
- `false and (true && print('hola'))` no imprime nada y devuelve `false`, por lo que **el código va en cortocircuito y se evalúa de izquierda a derecha** incluso aunque el `&&` y los paréntesis tengan más prioridad que el `and`.
- Otra forma de verlo es comprobar que `print('uno') and (1 + print('dos'))` escribe `unodos` (y devuelve `true`), por lo que la evaluación de los operandos del `and` se hace de izquierda a derecha aunque el `+` tenga más prioridad (y encima vaya entre paréntesis).
- En el manual de PHP[8] se dice que: *«La precedencia y asociatividad de los operadores solamente determinan cómo se agrupan las expresiones, no especifican un orden de evaluación. PHP no especifica (en general) el orden en que se evalúa una expresión y se debería evitar el código que se asume un orden específico de evaluación, ya que el comportamiento puede cambiar entre versiones de PHP o dependiendo de código circundante.»*
- Pregunta que hice al respecto en StackOverflow[9].

5.5.2 Numéricos

Enteros (`int`)

<http://php.net/manual/es/language.types.integer.php>

ricpelo's note: `integer` es sinónimo de `int`, pero debería usarse `int`.

Números en coma flotante (`float`)

<http://php.net/manual/es/language.types.float.php>

ricpelo's note: `double` es sinónimo de `float`, pero debería usarse `float`.

Operadores

Operadores aritméticos

<http://php.net/manual/es/language.operators.arithmetic.php>

Operadores de incremento/decremento

<http://php.net/manual/es/language.operators.increment.php>

[8] http://php.net/manual/es/language.operators.precedence.php
[9] https://stackoverflow.com/questions/46861563/false-and-true-printhi

5.5.3 Cadenas (`string`)

<http://php.net/manual/es/language.types.string.php>

ricpelo's note: Se usa `{$var}` y no `${var}` <https://github.com/yiisoft/yii2/blob/master/docs/internals/core-code-style.md#variable-substitution>

Operadores de cadenas

<http://php.net/manual/es/language.operators.string.php>

Concatenación

Acceso y modificación por caracteres

<http://php.net/manual/es/language.types.string.php#language.types.string.substr>

ricpelo's note: - `echo $a[3]`
- `$a[3] = 'x';`

Operadores de incremento/decremento

<http://php.net/manual/es/language.operators.increment.php>

Funciones de manejo de cadenas

<http://php.net/ref.strings>

Extensión *mbstring*

<http://php.net/manual/en/book.mbstring.php>

ricpelo's note: - `$a[3]` equivale a `mb_substr($a, 3, 1)`
- `$a[3] = 'x';` no tiene equivalencia directa. Se podría hacer:

```
$a = mb_substr($a, 2, 1) . 'x' . mb_substr($a, 4);
```

5.5.4 Nulo

<http://php.net/manual/es/language.types.null.php>

ricpelo's note: `is_null()` vs. `=== null` <https://phpbestpractices.org/#checking-for-null>
ricpelo's note: El tipo `null` y el valor `null` se escriben en minúscula.
<https://github.com/yiisoft/yii2/blob/master/docs/internals/core-code-style.md#51-types>

5.5.5 Precedencia de operadores

<http://php.net/manual/es/language.operators.precedence.php>

5.5.6 Operadores de asignación compuesta

ricpelo's note: `$x` <*op*>`= $y`

5.5.7 Comprobaciones

De tipos

`gettype()`

<http://php.net/manual/en/function.gettype.php>

`is_*()`

<http://php.net/manual/es/ref.var.php>

ricpelo's note: Poco útiles en formularios, ya que sólo se reciben `string`s.

De valores

`is_numeric()`

<http://php.net/manual/es/function.is-numeric.php>

`ctype_*()`

<http://php.net/manual/es/book.ctype.php>

5.5.8 Conversiones

<http://php.net/manual/es/language.types.type-juggling.php>

Coerción, moldeado, forzado o *casting*

<http://php.net/manual/es/language.types.type-juggling.php#language.types.typecasting>

ricpelo's note: Conversión de cadena a número

Conversión a `bool`

<http://php.net/manual/es/language.types.boolean.php#language.types.boolean.casting>

Conversión a `int`

<http://php.net/manual/es/language.types.integer.php#language.types.integer.casting>

Conversión a `float`

<http://php.net/manual/es/language.types.float.php#language.types.float.casting>

Conversión de `string` a número

<http://php.net/manual/es/language.types.string.php#language.types.string.conversion>

ricpelo's note: **¡Cuidado!**: La documentación dice que `1 + "pepe"` o `1 + "10 pepe"` funciona, pero en PHP7.1 da un **PHP Warning: A non-numeric value encountered**.

Conversión a `string`

<http://php.net/manual/es/language.types.string.php#language.types.string.casting>

Funciones de obtención de valores

ricpelo's note: Hacen más o menos lo mismo que los *casting* pero con funciones en lugar de con operadores. Puede ser interesante porque las funciones se pueden guardar, usar con *map*, *reduce*, etc.

`intval()`

<http://php.net/manual/es/function.intval.php>

`floatval()`

<http://php.net/manual/es/function.floatval.php>

`strval()`

<http://php.net/manual/es/function.strval.php>

`boolval()`

<http://php.net/manual/es/function.boolval.php>

Funciones de formateado numérico

`number_format()`

<http://php.net/manual/es/function.number-format.php>

`money_format()`

<http://php.net/manual/es/function.money-format.php>

`setlocale()`

<http://php.net/manual/es/function.setlocale.php>

ricpelo's note: `setlocale(LC_ALL, 'es_ES.UTF-8'); // Hay que poner el *locale* completo, con la codificación y todo (.UTF-8)`

5.5.9 Comparaciones

Operadores de comparación

<http://php.net/manual/es/language.operators.comparison.php>

`==` vs. `===`

Ternario (`?:`)

<http://php.net/manual/es/language.operators.comparison.php#language.operators.comparison.ternary>

Fusión de null (`??`)

<https://wiki.php.net/rfc/isset_ternary>

ricpelo's note: Equivalente al `COALESCE()` de SQL.

Reglas de comparación de tipos

<http://php.net/manual/es/types.comparisons.php>

ricpelo's note: `"250" < "27"` devuelve `false`

5.6 Constantes

<http://php.net/manual/es/language.constants.syntax.php>

ricpelo's note: Diferencias entre constantes y variables:
- Las constantes no llevan el signo dólar (`$`) como prefijo.
- Antes de PHP 5.3, las constantes solo podían ser definidas usando la función `define()` y no por simple asignación.
- Las constantes pueden ser definidas y accedidas desde cualquier sitio sin importar las reglas de acceso de variables.
- Las constantes no pueden ser redefinidas o eliminadas una vez se han definido.
- Las constantes podrían evaluarse como valores escalares. A partir de PHP 5.6 es posible definir una constante de array con la palabra reservada `const`, y, a partir de PHP 7, las constantes de array también se pueden definir con `define()`. Se pueden utilizar arrays en expresiones escalares constantes (por ejemplo, `const FOO = array(1,2,3)[0];`), aunque el resultado final debe ser un valor de un tipo permitido.

5.6.1 `define()` y `const`

5.6.2 Constantes predefinidas

<http://php.net/manual/es/language.constants.predefined.php>

5.6.3 `defined()`

<http://php.net/manual/es/function.defined.php>

5.7 Flujo de control

5.7.1 Estructuras de control

<http://php.net/manual/es/language.control-structures.php>

Sintaxis alternativa

<http://php.net/manual/es/control-structures.alternative-syntax.php>

ricpelo's note: El `do { ... } while (...);` **no** tiene sintaxis alternativa.

5.7.2 Inclusión de archivos

`include`, `require`

<http://php.net/manual/es/function.include.php>

ricpelo's note: El nombre del archivo debe aparecer con su extensión. No vale hacer `require 'pepe';`.
ricpelo's note: Cuando un archivo es incluido, el intérprete abandona el modo PHP e ingresa al modo HTML al comienzo del archivo objetivo y se reanuda de nuevo al final.
ricpelo's note: Si el archivo incluido tiene un `return ...;`, el `include` o el `require` que lo incluya devolverá el valor devuelto por el `return`.

`include_once`, `require_once`

<http://php.net/manual/es/function.include-once.php>

5.8 Funciones predefinidas destacadas

5.8.1 `isset()`

<http://php.net/manual/es/function.isset.php>

ricpelo's note: Cuidado si la variable contiene `null`.
ricpelo's note: No da error ni advertencia si la variable no existe.

5.8.2 `empty()`

<http://php.net/manual/es/function.empty.php>

ricpelo's note: Para evitar el problema de `empty("0") === true`:

```
function is_blank($value) {
    return empty($value) && !is_numeric($value);
}
```

ricpelo's note: No da error ni advertencia si la variable no existe.

5.8.3 `var_dump()`

<http://php.net/manual/es/function.var-dump.php>

5.9 Arrays

<http://php.net/manual/es/language.types.array.php>

ricpelo's note: Las claves pueden ser enteros o cadenas.

5.9.1 Operadores para arrays

<http://php.net/manual/es/language.operators.array.php>

ricpelo's note: **Comparaciones**: Un `array` con menos elementos es menor. De otra forma, compara valor por valor.

Acceso, modificación y agregación

<http://php.net/manual/es/language.types.array.php#language.types.array.syntax.modifying>

5.9.2 Funciones de manejo de arrays]

<http://php.net/manual/es/book.array.php> <http://php.net/manual/es/ref.array.php>

Ordenación de arrays

<http://php.net/manual/es/array.sorting.php>

`print_r()`

`'+'` vs. `array_merge()`

`isset()` vs. `array_key_exists()`

<http://php.net/manual/es/function.array-key-exists.php#107786>

5.9.3 `foreach`

<http://php.net/manual/es/control-structures.foreach.php>

5.9.4 Conversión a `array`

<http://php.net/manual/es/language.types.array.php#language.types.array.casting>

5.9.5 *Ejemplo*: `$argv` en CLI

<http://php.net/manual/es/reserved.variables.argv.php>

5.10 Funciones definidas por el usuario

<http://php.net/manual/es/language.functions.php>

5.10.1 Argumentos

<http://php.net/manual/es/functions.arguments.php>

Paso de argumentos por valor y por referencia

<http://php.net/manual/es/functions.arguments.php#functions.arguments.by-reference>

Argumentos por defecto

<http://php.net/manual/es/functions.arguments.php#functions.arguments.default>

ricpelo's note: `php function prueba($opciones = []) { extract($opciones); // ... }`

5.10.2 Ámbito de variables

<http://php.net/language.variables.scope>

Ámbito simple al archivo

Variables locales

Uso de `global`

ricpelo's note: Usar `global $x;` cuando `$x` no existe hace que `$x` empiece a existir y valga `null`.

Variables superglobales

<http://php.net/manual/es/language.variables.superglobals.php>

5.10.3 Declaraciones de tipos

ricpelo's note: **NO** se hacen conversiones implícitas a `array`, ni en argumentos ni en devolución.

Declaraciones de tipo de argumento

<http://php.net/manual/es/functions.arguments.php#functions.arguments.type-declaration>

Declaraciones de tipo de devolución

<http://php.net/manual/es/functions.returning-values.php#functions.returning-values.type-declaration>

Tipos *nullable* (`?`) y `void`

<http://php.net/manual/es/migration71.new-features.php>

Tipificación estricta

<http://php.net/manual/es/functions.arguments.php#functions.arguments.type-declaration.strict>

ricpelo's note: El `declare(strict_types=1);` se pone en el archivo que hace la llamada, no en el que define la función.

Parte III

Yii2

CAPÍTULO 6

Introducción a Yii2

CAPÍTULO 7

Conceptos fundamentales de Yii2

CAPÍTULO 8

Estructura de una aplicación Yii2

CAPÍTULO 9

Gestión de peticiones en Yii2

CAPÍTULO 10

Acceso a bases de datos en Yii2

CAPÍTULO 11

Recogida de datos y validación de formularios

CAPÍTULO 12

Visualización de datos en Yii2

CAPÍTULO 13

Características adicionales de Yii2

CAPÍTULO 14

Seguridad y cacheado en Yii2

CAPÍTULO 15

Pruebas, documentación y mantenimiento

CAPÍTULO 16

Computación en la nube

CAPÍTULO 17

Contenedores

Parte IV

Índices y tablas

Índice