
Desarrollo web en entorno servidor

Versión 1.0

Ricardo Pérez López

noviembre de 2017

Índice general

I	Introducción	1
1.	Preparación del entorno de desarrollo	3
2.	Introducción al desarrollo web	5
3.	Protocolo HTTP y lenguaje HTML	9
4.	Sistemas de control de versiones	11
II	PHP	13
5.	Conceptos básicos de PHP I	15
6.	Persistencia de datos con PHP	33
7.	Programación avanzada en PHP	35
8.	Interoperabilidad	37
III	Yii2	39
9.	Introducción a Yii2	41
10.	Conceptos fundamentales de Yii2	43
11.	Estructura de una aplicación Yii2	45
12.	Gestión de peticiones en Yii2	47
13.	Acceso a bases de datos en Yii2	49
14.	Recogida de datos y validación de formularios	51

15. Visualización de datos en Yii2	53
16. Características adicionales de Yii2	55
17. Seguridad y cacheado en Yii2	57
18. Pruebas, documentación y mantenimiento	59
19. Computación en la nube	61
20. Contenedores	63
 IV Índices y tablas	 65
Índice	67

Parte I

Introducción

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

Odoo

PrestaShop

Drupal

WordPress

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

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

Conceptos básicos de PHP I

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

Hola

```
$ php ../../p.php  
Hola
```

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 la cadena `¡Hola a todos!`.

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

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

5.2.3 Expresiones

El otro tipo de construcción sintáctica que existe en PHP junto con las sentencias son las *expresiones*. Una expresión *denota* o *representa* un valor. Una expresión puede ser tan simple como una constante literal (por ejemplo, el número `25`) o tan compleja que involucre constantes, variables, operadores, funciones y métodos, combinados todos ellos entre sí para formar una única expresión.

¹ <http://php.net/manual/es/function.echo.php>

Operadores

Un **operador** es un símbolo que representa una operación que se desea realizar sobre uno, dos o tres **operandos** (dependiendo de si el operador es *unario*, *binario* o *ternario*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 *aritméticas*, 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:

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:

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`

- 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 sub-expresiones 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 las Matemáticas de toda la vida: realizan un cálculo a partir de unos valores de entrada indicados en sus parámetros y *devuelven* el resultado de dicho cálculo,. Por ejemplo, la función *coseno* (abreviado 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 parámetros 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 parámetro, pero hay funciones que admiten o requieren más parámetros. Es el caso de la función `max()`⁴, que devuelve el valor máximo de todos los indicados en su lista de parámetros:

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

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.

³ <http://php.net/manual/es/function.cos.php>

⁴ <http://php.net/manual/es/function.max.php>

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.

Es importante destacar que esa información 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, 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.

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

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 llamado **script** (normalmente, con extensión `.php`) y el intérprete lee dicho archivo, lo analiza sintáctica y semánticamente y ejecuta las instrucciones que lo forman. El objetivo final de este curso es desarrollar aplicaciones web 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 operativo⁶. 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: Un intérprete interactivo solicita al usuario que introduzca (normalmente por teclado) una sentencia o expresión. Una vez analizada, ejecuta dicha sentencia (o evalúa dicha expresión) y vuelve a solicitar al usuario la introducción de una nueva sentencia o expresión. El usuario, por tanto, ve inmediatamente el efecto (o el resultado) que produce la ejecución (o evaluación) de la sentencia (o expresión) que acaba de introducir en el intérprete interactivo.

```
ssdfsdfsdf.
```

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

5.3.1 Modo dual de operación

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

5.3.2 Etiquetas `<?php y ?>`

5.4 Intérprete interactivo

5.4.1 `php -a`

5.4.2 PsySH

```
<http://psysh.org/>
```

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.

5.5 Variables

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

5.5.1 Conceptos básicos

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

5.5.2 Destrucción de variables

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

5.5.3 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>>

5.5.4 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()`.

5.6 Tipos básicos de datos

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

5.6.1 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](#)⁷ 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](#)⁸.

5.6.2 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>>

⁷ <http://php.net/manual/es/language.operators.precedence.php>

⁸ <https://stackoverflow.com/questions/46861563/false-and-true-printhi>

5.6.3 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);`

5.6.4 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>>

5.6.5 Precedencia de operadores

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

5.6.6 Operadores de asignación compuesta

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

5.6.7 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 `strings`.

De valores

`is_numeric()`

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

`ctype_*()`

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

5.6.8 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)

5.6.9 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

5.7 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 F00 = array(1,2,3)[0];`), aunque el resultado final debe ser un valor de un tipo permitido.

5.7.1 `define()` y `const`

5.7.2 Constantes predefinidas

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

5.7.3 `defined()`

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

5.8 Flujo de control

5.8.1 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.

5.8.2 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>>

5.9 Funciones predefinidas destacadas

5.9.1 `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.

5.9.2 `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.

5.9.3 var_dump()

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

5.10 Arrays

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

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

5.10.1 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>>

5.10.2 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>>

5.10.3 foreach

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

5.10.4 Conversión a array

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

5.10.5 *Ejemplo*: \$argv en CLI

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

5.11 Funciones definidas por el usuario

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

5.11.1 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); // ... }

5.11.2 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>>

5.11.3 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.

CAPÍTULO 6

Persistencia de datos con PHP

CAPÍTULO 7

Programación avanzada en PHP

CAPÍTULO 8

Interoperabilidad

Parte III

Yii2

CAPÍTULO 9

Introducción a Yii2

CAPÍTULO 10

Conceptos fundamentales de Yii2

CAPÍTULO 11

Estructura de una aplicación Yii2

CAPÍTULO 12

Gestión de peticiones en Yii2

CAPÍTULO 13

Acceso a bases de datos en Yii2

CAPÍTULO 14

Recogida de datos y validación de formularios

CAPÍTULO 15

Visualización de datos en Yii2

CAPÍTULO 16

Características adicionales de Yii2

CAPÍTULO 17

Seguridad y cacheado en Yii2

CAPÍTULO 18

Pruebas, documentación y mantenimiento

CAPÍTULO 19

Computación en la nube

CAPÍTULO 20

Contenedores

Parte IV

Índices y tablas

A

asociatividad, 18

C

comandos, 16, 17

cos(), 19

D

datos, 16

E

echo, 17

efectos laterales, 17, 19

enteros, 20

evaluación, *véase* expresiones

expresiones, 16, 17

expresiones, evaluación de, 16

F

funciones, 19

I

instrucciones, 16

N

números, 20

O

operadores, 17

P

palabras clave, 17

prioridad, 18

R

reales, 20

S

scripts, 21

sentencias, 16

T

tipos de datos, 20

V

var_dump(), 19