21-09-28 - PHP - Variables y tipos de datos

comandos para escribir sentencias (?)

Variables

En PHP, todas la variables empiezan por dólar (`$`) seguido del nombre que le asignemos.

Las variables no presentan ningun comportamiento especial comparado con otros lenguajes de programación:

> Las variables se componen de identificadores, que son ligados a las variables, las cuáles contienen una referencia hacia el espacio de memoria usado para guardar un dato.

Definir variables

Para definir una variable y asignar un valor usaremos el operador de asignación (`=`).
`>>> $x = 2;`

> El resultado es un identificador `$x`, ligado ó que "apunta" a la variable, la cual contendrá una referencia al espacio de memoria donde se encuentra almacenado el valor 2.

De forma predeterminada, las variables siempre se asignan por valor. Ésto implica que, cuando se asigna una expresión a una variable, el valor completo de la expresión original se copiará en la variable de destino.

Por ejemplo:

```
>>> 5   // '5' es una expresión canónica del valor 5.
==> 5
>>> $a = 5;     // Ahora $a también es una expresión del valor 5.
>>> $b = $a     // Se comportará como '$b = 5'.
```

En PHP no existe forma de declarar una variable sin asignar un valor o una referencia.

Asignación por referencia

Podemos crear referencias a variables existentes por medio del operador `&` justo delante del identificador de la variable. Su uso produce un "alias", una variable que "apunta a" la variable original.

```
>>> $a = 2;     // Asignamos el valor 2 al identificador $a.
>>> $b = &$a;   // Guardamos la referencia de $a en $b.
>>> $b += 2;    // Modificamos el valor de almacenado.
```

```
>>> echo $a;    // La mod. se refleja en todas sus ligaduras.
==> 4
```

Destruir variables

Es posible destruir una o varias variables mediante la construcción del lenguaje unset (`unset($variable, [$variable2]+);`). Ésta produce el efecto lateral de destruir las variables pasadas como argumentos.

> ¿Asignarle `null` a una variable destruye la destruye?

Es importante tener en cuenta que esta construcción del lenguaje (*no es una función*) tiene ciertas particularidades:

- Destruir variables referencia:
  Por ejemplo, Si realizamos una asignación por referencia y borramos cualquiera de sus referencias, mientras quede un identificador que apunte a la dicha referencia, la variable seguirá existiendo:

```
>>> $a = 2;     // Asignamos el valor 2 al identificador $a.
>>> $b = &$a;   // Guardamos la referencia de $a en $b.
>>> unset($a);  // Borramos la primera asignación al valor 2.
>>> echo $b;    // El identificador $b continúa apuntando al 2.
==> 2           // Por lo que la variable sigue existiendo.
```

- Destruir variables globales:
  Si tratamos de destruir una variable global dentro del cuerpo de una función, sólo la variable local al entorno de la función será destruida, manteniéndose intacta la variable fuera de su entorno:

```
>>> function destruir_nombre()
    {
        global $nombre;
        unset($nombre);
    }

>>> $nombre = 'Paco';
>>> destruir_nombre();
>>> echo $nombre;       // Teoricamente debería devolver null.
==> Paco        // ... pero no!
```

Éstas y otras particularidades están recogidas en la documentación de esta construcción.

¿Cómo saber si una variable existe?

Para comprobar si una variable existe, tenemos la construcción del lenguaje isset (`isset($variable)`). Devolverá true or false dependiendo de si la variable existe, y su valor no es null. Fue planteada para usarse en formularios.

> Se trata de una construcción algo especial, ya que recibiendo como argumento una variable que no existe, en lugar de producir un error devolverá un resultado (false).

Variables predefinidas

Existe una serie de variables predefinidas que, tan pronto arranques un script, estarán disponibles y serán muy útiles de cara a la web. Éstas dependerán del SAPI en el que nos encontremos.
Por ejemplo, las siguientes variables sólo existen en el SAPI de linea de comandos *(cli)*.

- $argc — El número de argumentos pasados a un script.
- $argv — Array de argumentos pasados a un script.

Tipos de datos

Al ser un lenguaje de tipado debil, no nos preocupara demasiado el tipo de dato,.
Los tipos de datos en PHP se asemejan muchos a los que encontramos en lenguajes como Java.
Dentro de estos tipos disponemos de:

- Tipos primitivos. En concreto disponibles de 10 tipos primitivos predefinidos en el lenguaje:
- Tipos escalares
  - boolean (bool)
  - integer
  - float
  - double
- Tipos compuestos
  - array
  - object
  - callable
  - iterable
- Tipos especiales
  - resource
  - null

boolean

Se puede referir como "boolean", aunque al final se está acotumbrando más el uso de "bool". Este tipo de dato sólo dispone de dos estados:

- Verdadero (true)
- False (false)

Hay una gran cantidad de formas de referirnos a ellos:

- Para verdadeo: true, TRUE, tRUE, 1, "1", "true", etc...
- Para falso: false, FALSE, FaLsE, 0, "0", "false", etc...

Esto es posible gracias al tipado débil, que permite varias condiciones en las que se tratará de realizar una conversión implicita a `true` o `false`, respectivamente, pero **por regla de estilo**, la manera más adecuada es en **minúsculas**.

> PSR: PHP Standard Recommendation.

Particularidades de los booleanos en PHP

Dicha conversión implícita también se produce en una operación aritmética, si encuentra un booleano, se convertira a entero:
- 1 para true
- 0 para false.

Integer

Que pasa com los números? Nada que no suceda en otros lenguajes de programación: Que podemos sufrir **desbordamientos**. Existen ciertas constantes que nos derterminan los límites que tiene el tipo entero en PHP.

- PHP_INT_SIZE: El tamaño de un integer (8 bytes).
- PHP_INT_MAX: el valor máximo (9223372036854775807).
- PHP_INT_MIN: el valor mínimo (-9223372036854775807).

> Estos valores límites están incluidos en el rango.

Si el resultado de una operación devuelve un valor fuera de los límites del tipo Integer.

var_dump(variable): Es el volcado de variable. Devuelve el valor y el tipo de lo que le pases como argumento y lo vuelca a la salida.

float

> Por motivos historicos tambien funciona double.

El redondeo aquí siempre se produce por debajo (truncado).
echo (int) ((0.1+0.7) * 10); // Devuelve 7 por error de redondeo. El valor 0.1 no es posible de almacenar, su representación binaria tiende a infinito. Por lo tanto resuelven este conflicto interpretando como 0.1 a un número lo más cercano posible (0.0999999...).

Por lo tanto, en el ejemplo anterior:

1. Sumamos 0.09999... a 0.7, devuelve 0.799999....
2. Multiplicamos su resultado por 10, obteniendo 7,999999...
3. Al solicitar conversión explicita a entero, se trunca todo valor tras la coma, devolviendo 7.

Podemos refererirnos a este tipo de dato, si usamos la expresión "4e5" ($4 * 10^5$)

string

El problema de php es que tiene un sorporte UNICODE muy pobre, por motivos historicos.
Una cadena es una secuecnia de bytes, no una secuencia de caracteres.
El problema reside en UTF-8, donde un caracter puede ocupar más de un byte

¿Como escribir literales de tipo cadena?

4 formas:

- comillas dobles
- comillas simples
- heredoc
- nowdoc

1. comillas dobles: Suponen que lo que hay dentro que el interprete de php lo interpreta. Se lleva a cabo una interpretacion, por lo que habra una serie de caracteres que se sustituiran Un ejemplo son las secuencias de escape, que comienzan con backslash `\`. `\n` por ejemplo sólo funcionará en comillas dobles.

Interpolación (expansion de variables)

Otra cosa es la interpolación (expansión de variables). Consiste en meter una variable dentro de una cadena, haciendo que esta se sustituya por su valor

La regla de estilo dice que, siempre que no haga falta lo conrtario, las cadenas deben ir siempre en comillas simples. Usar solo comillas dobles cuando se necesite interpretar/interpolación.

es posible capar la secuencia de escapa, poniendo otro backslash. Tambien funcoina con variable, usando backslash delante del identificador

Podemos indicar una variable encerrándola entre llaves:

```
>>> $h = "pepe"
=> "pepe"
>>> echo "hola {$h}ola"
hola pepeola⏎
```

Indexación
en PHP no existe el tipo caracter, por lo que al usar indexacion ($cadena[0]), el indice se mueve por bytes. Dado que algunos caracteres ocupan mas de un byte, a la hora de querer acceder a la indexación de un caracter especial, (la letra 'ó' de la palabra 'cami´on') no

con la indexacion tambien podemos alterar el caracter de una cadena, indicando su posición y asignandole una cadena que contenga un solo caracter (solo se espera una cadeana con una longitud de un byte).

Esta operación practicamente no la vamos a utilizar.

Debido al problema de que las cadenas solo se miden por secuencias de bytes, tenemos el modulo mbstring que nos proporciona una serie de funciones compatibles con caracteres multibyte.

strlen(string $string): la longitud de la cadena
trim: elimina espacios de una cadena
rtrim: elimina espacios por la deerecha
ltrim,: elimina espacios por la izquierda

- strpos: devuelve la posicion de la primera ocurrencia de un caracter en una cadena, false si no esncuentra ninguna. tiene un pparametro adiciona, offset, indicando la posicion desde la cual quieres comenzar a buscar. Por defecto su valor es 0. No se pueden usar valores negativos en el offset

mb-substr(cadena, posicion inicio, longitud)

Si yo quiero de "camión" la letra ó:
mb_substr("camión", 4, 1);

2. comilla simples

heredoc

heredoc permite escribir una cadena que ocupe varias lineas
Todo lo que haya dentro, a partir de los caracteres `<<<`, indicando con un marcador donde quieres que termine el texto. Aqui se respetaran los saltos de linea, y si dentro pones una variable se interpola, si tuviesemos secuancias de escape, se interpretará

```
>>> $nombre ='Pepe';
>>> $bloque = <<<EOT
Yo
me
llamo
$nombre
\n
EOT;
```

nowdoc

nowdoc es a las comillas simples, lo que heredoc es a las comillas dobles. No se realizaraña ningun tipo de interpretación o interpolación.
Para usarlo, el marcador que usabamos en nowdoc

Peculiaridad del tipado débil con números y cadenas

En una operación aritmética de cadenas que contengan números, partiendo de la condición de que dicha cadena comience por un número, si después de ese número se encuentran otros caracteres, la operación funcionará pero produciendo un aviso (*warning*). A partir de detectar un caracter distinto de un número, detendrá la lectura de la cadena y sólo convertirá las cifras encontradas. Significa que, si después de un caracter distinto de un número aparecieran más números, estos serán ignorados:

```
>>> echo "4 patos y 1 más" + "5 patos"; // Se esperan 10 patos?.
<warning>PHP Warning:  A non-numeric value encountered in phar://eval()'d code on line 1</warning>
<warning>PHP Warning:  A non-numeric value encountered in phar://eval()'d code on line 1</warning>
==> 9   // El 1 tras la palabra 'patos y' fue ignorado.
```

4 tipos compuestos

- array
- object
- callable
- iterable

Finalmente dis tipos especiales:

- resource
- null

null

Tipo de dato, cuyo unico valor es null

si uso var_dump, me volcara a la salida el valor y el tipo de una expresión

is_null($x);

tambien podemos usar `$x == null`
pero esto nos provocara efectos indeseados, por ejemplo

```
>>> $x = null;
=> null
>>> is_null($x);
=> true
>>> $x == null;
=> true
>>> false == null;
=> true
>>> "" == null;
=> true
```

null siempre es convertido a string vacio.

Arrays

El tipo de datos mñas importante con diferencia. Mucho mas flexives que los arrays de java. Son muy parecidos a los dicionarios de python. Una especie de fusión entre ambos

> Los arrays en PHP son llamados *arrays asociativos*

Es una estrucutra de datos que contiene elementos, a los que se puede acceder mediante una clave. Cadad elemento de un array es una pareja clave-valor. Por lo que se trata de una colección de datos. Las claves pueden ser numeros enteros, o cadenas, pero siempre serán **únicas**

Con los numeros, lo que tendriamos es un array al estilo java, con la ventaja de que los arrays en php son dinamicos: puede crecer y encojerse en tiempo de ejecucion; no necesitamos determinar su tamaño de anteano. Ademas los tipos de los valores no hay que determinarlos de antemano, por lo que cada pareja clave-valor podra contener valores de distintos tipos.

Crear array

- forma antigua: La función array. recibe los elementos como parametros, sdeparados por comas.

`array('a', 'b', 'c')`

> Llamarlo posición sería incorrecto. La palabra adecuada para referirnos a estos *indice* es **clave**

- Forma moderna: Poner entre corchetes los elementos tal como queremos que vayan ordenados:

```
>>> $array =  ['a', 'b', 'c'];
=> [
     "a",
     "b",
     "c",
   ]
```

```
>>> $array[1];
=> "b"
>>> $array[1] = 'z';
=> "z"
>>> $array;
=> [
     "a",
     "z",
     "c",
   ]
>>>
```

Tambien podemos indicar maualmente cada clave y asignarla con `=>` al valor que deseemos:
`[0=>'a', 1=>'b', 2=>'c']`
También:
`[0=>'a', 'b', 'c']`

Todo elemento nuevo agregado se ubicará al final del array, y a php no le importa el orden de los elementos, ni si sus claves son correlativas.

Por eso podemos hacer esto:
`$a[] = 'm'`
de este modo, le decimos a php que agregue un nuevo elemtno, revisando el ultimo indice mas alto que se utilizo en el array, para usarlo sumandole 1. Usa una especie de variable que recuerda el entero más alto utilizado en dicho array.

tambien podemos usar:
`$a[-1] = 'y'`
Recordemos que esto no son indices como tal, son simplemente **claves**.

en definitiva, son diccionarios con la capacidad de compatibilizar con el concepto primitivo de arrays en C

eliminar elementos de un array

Usaremos la orden ya mencionada `unset`:
`unset($var[indice]);`

Comprobar si un elemento existe en un array

- con la construcción `isset`
  `isset($array[clave])`

El gran problema de isset es que devuelve true si la variable existe, y su valor es distinto de nulo. Para los casos en los que tenemos valores null en nuestro array podemos usar la funcion array_key_exists

- Con la función array_key_exists
  independientemente del valor que contenga la clave que solicitamos, esta solo se centrará en comprobar si existe la clave:
  `array_key_exists(clave, array);`

Buscar valores en un array

array_search devuelve la primera clave correspondiente en caso de exito, o false si no existe.

`array_search(valor, array)`

Buscar elementos a partir de su clave.

Esto lo haremos a traves de la indexacion

array_unique: elimina valores duplicados de un array

Entre todas las funciones disponibles para arrays en este lenguaje, encontramos algunas que devuelven una copia del array recibido, sienod asi funciones puras, y otras probocaran el estado lateral de modificar el array existente.

asort ordena un array, por valores devolviendo true si ha modificando el array original con exito, false en caso contrario.

ksort ordena un array, por claves devolviendo true si ha modificando el array original con exito, false en caso contrario.

array_combine realizará una combinación entre dos arrays:
`array_combine(arrays_keys($array1), arrays_values($array2))`

contar elementos de un array

`count($array)`

podemos comprobar si un array está vacío si hacemos:

empty($array) para comprobar si un array está vacío
empty funciona con otros tipos de datos, porque por motivos de conversión explicita podra devolver resultados inesperados:

`empty("0")` devolverá true, ya que por conversión de tipos, la cadena "0" se convertirá a boolean, y false se considera que no ocupa espacio, dando como resultado true.

Operadores

`.` es el operador de concatenación.

Hay un perador de inversión de bits, `~`.

Operador ternario

`<expr> ? <expr2> : <expr3>`

Operadores lógicos

AND: `&&` y `and`.

OR `||` y `or`.

XOR (eXclusive Or): `xor`.

NOT: `!`.

Observamos varias formas de referirnos a operador AND y al OR inclusivo. La diferencia entre ellos es su precedencia, siendo `&&` y `||`más prioritarios que `and` y `or`.

Teniendo claras las precedencia, podemos usarlas a nuestro favor. Ejemplo: comprobar si una expresion es verdadera o false, y si esa expresión es falsa hacer algo.

función matar: `die()`.

07/10/2021 17:42

Tipos Nulables

Se definen con '?tipo-de-dato'