



PHP | Sintaxis básica

Después de esta lección podrás:

1. Manejarte con los conceptos básicos de PHP.
2. Crear tu primer script en el backend.
3. Empezar a ver las diferencias entre PHP y JavaScript.

Etiquetas PHP

Cuando PHP analiza un fichero, busca las etiquetas de apertura y cierre, que son **<?php** y **?>**, y que indican a PHP dónde empezar y finalizar la interpretación del código. Este mecanismo permite embeber a PHP en todo tipo de documentos, ya que todo lo que esté fuera de las etiquetas de apertura y cierre de PHP será ignorado por el analizador.

```
<?php echo "HOLA!";?>
```

Si un fichero contiene solamente código de PHP, es preferible omitir la etiqueta de cierre de PHP la final del mismo.

Cualquier cosa fuera de un par de etiquetas de apertura y cierre es ignorado por el intérprete de PHP, lo que permite que los ficheros de PHP tengan contenido mixto. Esto hace que PHP pueda ser embebido en documentos HTML para, por ejemplo, crear plantillas.

```
<p>Esto va a ser ignorado por PHP y mostrado por el navegador.</p>
<?php echo 'Mientras que esto va a ser interpretado.'; ?>
<p>Esto también será ignorado por PHP y mostrado por el navegador.</p>
```

Este ejemplo funciona como estaba previsto, porque cuando PHP intercepta las etiquetas de cierre `?>`, simplemente comienza a imprimir cualquier cosa que encuentre

Separación de instrucciones

PHP requiere que las instrucciones terminen en punto y coma al final de cada sentencia. La etiqueta de cierre de un bloque de código de PHP automáticamente implica un punto y coma; no es necesario usar un punto y coma para cerrar la última línea de un bloque de PHP. La etiqueta de cierre del bloque incluirá la nueva línea final inmediata si está presente.

```
<?php echo "Esto es una sentencia";?>
```

Es equivalente a lo siguiente:

```
<?php echo "Esto es una sentencia" ?>
```

Comentarios

Los comentarios son fragmentos de texto que no son ejecutados por el interprete de PHP pero pueden ser útiles para el desarrollador.

Pueden ser en una sola línea (`//`):

```
<?php echo "HOLA!"; // aqui viene el comentario
```

O para comentar más de una línea (`/* */`):

```
<?php
/*
    este es un
    comentario de 2 lineas
*/
echo "HOLA";
```

Tipos de variables

Aunque PHP es un lenguaje débilmente tipado, es decir, no controlan los tipos de las variables que declara, , de este modo, es posible usar variables de cualquier tipo en un mismo escenario.

Básicamente los tipos son los diferentes ‘tipos’ (valga la redundancia) que pueden almacenar las variables: pueden ser números, o cadenas de caracteres, o estructuras más complejas. Una misma variable, en PHP puede contener un valor de tipo número, por ejemplo 342, y eso no impide que más adelante esa misma variable pueda contener un el string «HOLA».

NOTA: en el bootcamp, veremos como el párrafo anterior tiene sus matices... 😊

Veremos ahora divididos en 3 grandes grupos los diferentes tipos que admite PHP.

Tipos escalares

Boolean

Este es el tipo más simple. Un boolean expresa un valor que indica verdad. Puede ser **TRUE**(verdadero) o **FALSE** (falso).

Por ejemplo para indicar si una persona es mayor de edad, o si cumple cierta validación, etc...

Sintaxis

Para añadirle a una variable un valor de tipo boolean se emplean las constantes **TRUE** o **FALSE**. Ambas no son susceptibles a mayúsculas y minúsculas.

```
$isValid = True; // asigna el valor TRUE a $isValid
```

Integer

Un número entero (o integer) es un número del conjunto $\mathbb{Z} = \{..., -2, -1, 0, 1, 2, ...\}$.

Por ejemplo lo usarías, para guardar la edad de una persona, el número de hijos, número de veces a repetir algo, etc...

Sintaxis

En este curso de nivelación, solamente veremos cómo indicar los números en base decimal (base 10)

```
$edad = 18; // número decimal $saldo = -123; // un número negativo
```

Float

Los números de punto flotante (también conocidos como «de coma flotante» son usado para representar números reales \mathbb{R} , ya sabéis aquellos con mantisa y exponente que se daba en el cole.

Se puede utilizar por ejemplo para representar temperaturas en grados Celsius, y en general cualquier cosa con decimales.

NOTA: Los números de punto flotante tienen una precisión limitada. Ya veremos más en el bootcamp, pero ojo con ellos.

Sintaxis

Veremos la manera habitual de representarlos.

`$grados = 24.5 // grados Celsius`

String o cadena

Llegamos a uno de los tipos más usado en cualquier lenguaje, las cadenas de texto.

Un string, o cadena, es una serie de caracteres donde cada carácter es lo mismo que un byte.

Se utiliza para guardar por ejemplo nombres de usuarios, contraseñas, textos, códigos alfanuméricos, en general cualquier cosa que se represente con caracteres (incluso números).

NOTA: Un string puede llegar a alcanzar hasta 2 GB de tamaño (2147483647 bytes máximo), eso es muuucho texto, será bastante difícil que llegues a almacenar toda esa información en una variable, pero solo para que lo sepas.

Sintaxis

Un literal de tipo string se puede especificar de cuatro formas diferentes:

- entrecomillado simple
- entrecomillado doble
- sintaxis heredoc y nowdoc (no lo veremos en el curso de nivelación)

Entrecomillado simple

La manera más sencilla de especificar un string es delimitarlo con comillas simples (el carácter `'`).

¿Cómo hacer para poder añadir el carácter comilla simple (`'`) a una cadena que ya está delimitada por comillas simples? se ha de escapar con una barra invertida (`\`). Para especificar una barra invertida literal, se duplica (`\\`).

echo 'Es una cadena sencilla';**echo** 'El título es \' Star Wars\'';**echo** 'Ruta: c:*. *';

Entrecomillado doble

Si un string está delimitado con comillas dobles (`«`), PHP interpretará algunas secuencias de escape como caracteres especiales, como por ejemplo `\n` como un salto de línea, `\t` como un tabulador.

La característica más importante del entrecomillado doble de un string es el hecho de que se expanden los nombres de las variables.

```
$color = "azul";// Válido.echo "El cielo es de color $color".PHP_EOL; // Inválido.  
"es" es un carácter válido para un nombre de variable, pero la variable es  
$color.echo "Hay muchos tonos de $colores.";// Válido. Explícitamente especifica  
el final del nombre de la variable encerrándolo entre llaves:echo "Hay muchos  
tonos de ${color}es."
```

El resultado anterior sería:

El cielo es de color azul.Hay muchos tonos de .Hay muchos tonos de azules.

Array o matriz

Y llegamos al tipo de dato más versátil en PHP, estate muy atento porque dominando este tipo podrás hacer cosas fantásticas con PHP, guardar listas de personas, acciones, arrays multidimensionales y demás valores compuestos con solo una variable!

Es una variable que puede almacenar una lista de valores, como por ejemplo nombres de personas, o un array multidimensional es decir, que los valores del array pueden ser otros arrays, ej: una lista de personas, y cada una de ellas tener un listado de amigos.

Sintaxis

```
$nombres = array("kiko", "ricky", "moi", "jorge");//también se puede con notación  
corta $nombres = ["kiko", "ricky", "moi", "jorge"];
```

La clave puede ser un integer o un string. El valor puede ser de cualquier tipo:

```
$usuario = [ 'nombre' ⇒ 'kiko', 'edad' ⇒ 36];
```

Ya veremos en el bootcamp en la manipulación de tipos como “castear” ciertos tipos a otros de manera implícita.

Esto de 'castear' que puede parecer muy lioso, viene a decir que en PHP una variable que contiene el número 25 se convierte a la cadena '25' bajo ciertas circunstancias sin que tu hagas nada, y se trata como una cadena.

La clave es opcional. Si no se especifica, PHP usará el incremento de la clave de tipo integer mayor utilizada anteriormente:

```
$nombres = ['kiko', 'moi']; // internamente lo que hace el intérprete es $nombres = [0 => 'kiko', 1 => 'moi'];
```

Null

El valor especial NULL representa una variable sin valor. NULL es el único valor posible del tipo null.

Una variable es considerada null si:

- se le ha asignado la constante **NULL**.
- no se le ha asignado un valor todavía.
- se ha destruido con unset().

Sintaxis

No hay más que un valor de tipo null, y es la constante **NULL** insensible a mayúsculas/minúsculas.

```
$var = NULL; // es lo mismo poner $var = null
```

Variables y ámbito

Las variables son aquellas estructuras que se utilizan para almacenar valores. Como ya lo has podido ver en lecciones anteriores, en PHP las variables se representan con un signo de dólar seguido por el nombre de la variable. El nombre de la variable es sensible a minúsculas y mayúsculas.

Los nombres de variables siguen las mismas reglas que otras etiquetas en PHP. Un nombre de variable válido tiene que empezar con una letra o un carácter de subrayado (underscore), seguido de cualquier número de letras, números y caracteres de subrayado.

```
$var = 'Roberto';  
$Var = 'Juan';  
echo "$var, $Var";      // imprime "Roberto, Juan"  
$4site = 'aun no';     // inválido; comienza con un número  
$_4site = 'aun no';    // válido; comienza con un carácter de subrayado
```

Ámbito de las variables

Aunque algunas cosas que te contaremos en este apartado todavía no las entiendas (inclusión de ficheros, funciones, etc...), centrémonos solamente en el concepto de ámbito.

El ámbito de una variable es el contexto dentro del que la variable está definida. La mayor parte de las variables PHP sólo tienen un ámbito simple. Este ámbito simple también abarca los ficheros incluídos y los requeridos

```
$a = 1;  
include 'util.php';
```

Aquí, la variable \$a estará disponible al interior del script incluido util.php. Sin embargo, al interior de las funciones definidas por el usuario se introduce un ámbito local a la función. Cualquier variable usada dentro de una función está, por omisión, limitada al ámbito local de la función.


```
$a = 1; /* ámbito global */

function test()
{
    echo $a; /* referencia a una variable del ámbito local */
}

test()
```

Este script no producirá salida, ya que la sentencia echo utiliza una versión local de la variable \$a, a la que no se ha asignado ningún valor en su ámbito.

Hay maneras de declarar y utilizar variables globales dentro de funciones, por ejemplo con la palabra reservada **global**, o utilizando **closures**, pero eso ya lo veremos en el bootcamp.

Operadores

Un operador es algo que toma uno más valores (o expresiones, en la jerga de programación) y produce otro valor (de modo que la construcción en si misma se convierte en una expresión). ¡No asustarse con esta definición! es más sencillo de lo que parece.

Vamos a mostrar aqui algunos operadores básicos. Dejaremos sin comentar los de tipo array y objeto.

Es importante tener en cuenta algo llamado **precedencia de operadores**.




Precedencia de operadores

La precedencia de un operador indica qué tan «estrechamente» se unen dos expresiones juntas. Por ejemplo, en la expresión $1 + 5 * 3$, la respuesta es 16 y no 18 porque el operador de multiplicación («*») tiene una precedencia mayor que el operador de adición («+»). Los paréntesis pueden ser usados para forzar la precedencia, si es necesario. Por ejemplo: $(1 + 5) * 3$ se evalúa como 18.

Para un listado con la precedencia de operadores visitar la [documentación](#) oficial de PHP.

Operadores aritméticos

Son como en la escuela 😊

 Ejemplo	 Nombre	 Resultado
<u>+\$a</u>	Identidad	Conversión de \$a a int o float según el caso
<u>-\$a</u>	Negación	Opuesto de \$a
<u>\$a + \$b</u>	Suma	Suma de \$a y \$b
<u>\$a - \$b</u>	Resta	\$a menos \$b
<u>\$a * \$b</u>	Multiplicación	Producto de \$a y \$b
<u>\$a / \$b</u>	División	Cociente de \$a y \$b
<u>\$a % \$b</u>	Módulo	Resto de la división de \$a y \$b
<u>\$a ** \$b</u>	Exponenciación	Resultado de elevar \$a a la potencia \$bésima




Operador de asignación

El operador básico de asignación es «=». Se podría inclinar a pensar primero que es como un «igual a». No lo es. Realmente significa que el operando de la izquierda se establece con el valor de la expresión de la derecha (es decir, «se define como»).

Operadores de comparación

Los operadores de comparación, como su nombre lo indica, permiten comparar dos valores.

En este punto es vital entender como PHP realiza la comparación de tipos.

 Ejemplo	 Nombre	 Resultado
<u>\$a == \$b</u>	Igual	TRUE si el valor de \$a es igual al valor de \$b

Aa Ejemplo	≡ Nombre	≡ Resultado
<u><code>\$a === \$b</code></u>	Idéntico	TRUE si \$a y \$b son iguales en valor y tipo
<u><code>\$a != \$b</code></u>	Diferente	TRUE si el valor de \$a es distinto al valor de \$b
<u><code>\$a <> \$b</code></u>	Diferente	TRUE si el valor de \$a es distinto al valor de \$b
<u><code>\$a !== \$b</code></u>	No idéntico	TRUE si \$a y \$b son distintos en valor o tipo
<u><code>\$a < \$b</code></u>	Menor que	TRUE si \$a es menor que \$b
<u><code>\$a > \$b</code></u>	Mayor que	TRUE si \$a es mayor que \$b
<u><code>\$a <= \$b</code></u>	Menor o igual que	TRUE si \$a es menor o igual que \$b
<u><code>\$a >= \$b</code></u>	Mayor o igual que	TRUE si \$a es mayor o igual que \$b
<u><code>\$a <=> \$b</code></u>	Nave espacial	-1 cuando \$a es menor que \$b 0 cuando \$a es igual que \$b 1 cuando \$a es mayor que \$b
<u><code>\$a ?? \$b ?? \$c</code></u>	Fusión de null	Devuelve el primer operando que encuentre que sea distinto de NULL

Si se compara un número con un string o la comparación implica strings numéricos, entonces cada string es convertido en un número y la comparación realizada numéricamente.

Operador ternario




Otro operador condicional es el operador «?:» (o ternario).

La expresión *(expr1) ? (expr2) : (expr3)* evalúa a *expr2* si *expr1* se evalúa como TRUE y a *expr3* si *expr1* se evalúa como FALSE.

```
$foo = [];
// Ejemplo de uso para: Operador Ternario
$action = (empty($foo)) ? 'default' : 'especial';
// Lo anterior es idéntico a esta sentencia if/else
if (empty($foo)) {
    $action = 'default';
} else {
```

```
$action = 'especial';
}
```




Operadores de incremento/decremento

 Ejemplo	 Nombre	 Efecto
<code>++\$a</code>	Pre-incremento	Incrementa en uno \$a, y luego devuelve \$a
<code>\$a++</code>	Post-incremento	Devuelve \$a y luego incrementa su valor en uno
<code>--\$a</code>	Pre-decremento	Decrementa \$a en uno, y luego devuelve \$a
<code>\$a--</code>	Post-decremento	Devuelve \$a y luego decrementa su valor en uno
<u>Untitled</u>		

Ej.

```
$a = 5;
echo $a++; // imprime 5
$a = 5;
echo ++$a; // imprime 6
```

Operadores lógicos

 Ejemplo	 Nombre	 Resultado
<code>\$a and \$b</code>	And (y)	TRUE si tanto \$a como \$b son TRUE
<code>\$a or \$b</code>	Or ("o" inclusivo)	TRUE si cualquiera de \$a o \$b son TRUE
<code>\$a xor \$b</code>	Xor ("o" exclusivo)	TRUE si \$a o \$b son TRUE pero no ambos a la vez
<code>!\$a</code>	Not (no)	TRUE si \$a es FALSE
<code>\$a && \$b</code>	And (y)	TRUE si tanto \$a como \$b son TRUE
<code>\$a \$b</code>	Or ("o" inclusivo)	TRUE si cualquiera de \$a o \$b son TRUE
<u>Untitled</u>		

Operadores para string

Existen dos operadores para datos tipo string. El primero es el operador de concatenación ('.'), el cual devuelve el resultado de concatenar sus argumentos derecho e izquierdo. El segundo es el operador de asignación sobre concatenación ('.='), el cual añade el argumento del lado derecho al argumento en el lado izquierdo.

```
$a = "Hello ";  
$b = $a . "World!"; // ahora $b contiene "Hello World!"  
$a = "Hello ";  
$a .= "World!";      // ahora $a contiene "Hello World!"
```