PHP tutorial

Un lenguaje importante y que hemos dejado de lado por años es el ya muy famoso lenguaje **PHP**. En desagravio publicamos esta entrada y le queremos dar características especiales: *no insisteremos en ser detallistas al extremo, se trata de una introducción, un tutorial que lleve paso por paso a cualquier persona que desee aprender dicho lenguaje de programación.* Por ello no publicaremos su historia, configuración del servidor, etcétera. Os recordamos que para aprender PHP sería bueno tener conocimientos mínimos de HTML, CSS y JavaScript.

PHP de manera "sucia y rápida".

No nos asustéis con la traducción al castellano del popular refrán en idioma inglés "quick and dirty", que siempre se recuerda cuando se necesita desarrollar algún software que goce de pragmatismo al extremo. Lo que queremos decir es que cuando uno necesita resultados rápidos pues conviene saltarse los "paso a paso" que siempre usamos para aprender algo nuevo.

Elementos básicos en PHP.

El lenguaje PHP bien puede utilizarse en comandos por una venta terminal o bien lo podemos usar un proceso por lotes almacenados en un archivo de texto al cual le colocaremos una extensión ".php" para identificar de manera rápida su contenido. Siendo entonces un fichero en texto plano procedemos a escribir dentro de él lo siguiente:

Estas tres simples líneas le indican al servidor dónde comienza y termina las instrucciones que va a ejecutar en lenguaje PHP. Esto es así porque dentro de un archivo "**.html**" podremos incluir código especial que muy probablemente emita código en lenguaje HTML de manera dinámica (por ejemplo, una consulta en una base de datos para mostrar los valores seleccionados por el usuario). Las dos barras inclinadas "**//**" indica que todo lo que está a la derecha y hasta el final de la línea son comentarios y que no deben ni serán ejecutados. Tambien sirve "**#**" para comentar una línea, pero si necesitamos comentar varias líneas escribimos "**/*** comentario(s) ***/**".

Con excepción de las líneas de comentarios, inicio y cierre, toda las demás líneas deben terminar con un punto y coma "**;**".

Variables en PHP.

Toda variable comienza con "$" y un caracter A-Z (mayúsculas y/o minúsculas) y/o el guión bajo "_", luego se pueden usar números y/o caracteres *sin dejar espacios.* Se pueden clasificar en:

- Número entero (positivos o negativos).
- Número flotante (número decimal positivos o negativos).
- Booleano (verdadero o falso).
- Cadena de caracteres (las almacenamos con comillas simples).

Hay cuatro tipos más de variables avanzadas que veremos en su debida oportunidad. Una vez hallamos guardado un valor en una variable se pueden mostrar por pantalla por medio de las ordenes **echo** o **print(),** las cuales tiene muchas opciones que veremos más adelante pero detallaremos que no es lo mismo usar comillas simples que dobles con dichas funciones, mirad este ejemplo:

Y devolverá lo siguiente (código HTML recibido por el navegador, en Mozilla Firefox pulsad las teclas CTRL+U):

```
El valor es $miValor.

El valor es 5.
```

Así vemos que el comando **echo** bien puede "pasarle" al navegar el texto (en este caso texto + lenguaje HTML) o bien puede mediante instrucciones especiales "pasar" el valor almacenado en una variable. *Notad que aunque los **comandos** en PHP es indistinto si los escribimos en mayúsculas y/o minúsculas **las variables son sensibles:** no es lo mismo $miValor que $MiValor, mucho cuidado con eso que causa bastantes dolores de cabeza a nosotros los programadores.*

Otro uso en el campor de las variables, aunque poco utilizado, es declarar una segunda variable por referencia con otra variable. Para esto usamos el caracter "**&**" justo delante de la variable que queremos pasar. Ejemplo:

```
$a = "texto"; $b = &$a; echo $b;
```

Suma de variables en PHP.

Para sumar dos variables, ya sea para presentarla por pantalla o para almacenarla en otra variable, utilizamos los operadores matemáticos de costumbre, mirad:

Concatenar variables en PHP.

Decimos concatenar en el caso de variables de texto, en PHP con un simple punto unimos la variables:

Declaración y alcance de las variables en PHP.

Como estudiamos hace poco, no es necesario decirle a PHP qué tipo de variable vamos a trabajar, simplemente ponemos los valores y el lenguaje se encarga de almacenarlo en el lugar adecuado. Cuando vayamos a realizar operaciones con dichas variables éstas se convertirán de a un valor correcto, por ejemplo:

El resultado por pantalla será "5", es decir, el valor de **$a** es una cadena de texto cuyo valor numérico es cero y así lo convierte para sumarlo a **$b**. *Por el contrario, en la siguiente línea **$a.$b** estamos concatenando cadenas de texto* y el valor de **$b** se convierte en el caracter "5" para mostrar lo siguiente:

```
5  palabra5
```

En PHP podemos "forzar" que una variable sea de un tipo específico que necesitemos, para ello usar el comando **settype()** con dos argumentos entre paréntesis separado por una coma: nombre de la variable y luego el tipo de variable ("integer", "double", "string", "booelan"), ejemplo: «**setttype($a , "integer")**».

Otra manera de forzar a un tipo de variable es de la siguiente manera: «**$variable = (int) $variable;**» en este caso **(int)** o **(integer)** indica que lo cambie a tipo entero pero podemos usar también **(real)**, **(double)**, **(float)**, **(boolean)** o **(bool)**.

Otra cosa es el alcance de las variables: si están en el módulo principal las variables serán globales y si están dentro de una función serán locales y solamente podrán ser accesibles dentro de la función donde fueron "creadas". Si necesitamos que una variable dentro de una función esté disponible para otras funciones (y en el módulo principal) usaremos la palabra **global** y a

continuación le asignaremos el valor; **de facto estamos declarando la variable al asignarle un valor.**

Cuando declaramos una variable como **global** ésta se adiciona a la matriz **$GLOBALS['nombre_de_variable']** y podemos acceder a ellas de esta manera nemotécnica, especialmente es útil para usarla dentro de funciones; ejemplo: «echo $GLOBAL['cadena']». Aunque aún falta mucho para ver las funciones, podemos adelantar el siguiente código que ilustra el alcance de las variables locales y globales:

Si necesitamos que una variable dentro de una función conserve su valor entre llamadas, debemos declararla con el comando **static** dentro de la función a la que pertenece y asignarle un valor, por supuesto.

Incrementos y decrementos en variable numéricas.

Como es popular en lenguaje C++ los incrementos se logran colocándole "**++**" (o "--" si es un decremento) luego del nombre de la variable. En este ejemplo ambas líneas son equivalentes pero en una escribimos más que en la otra:

Pero fijaos que al mostrarlo en pantalla, cuando el incremento (o decremento) lo colocamos ANTES o DESPUÉS de la variable se comporta de manera diferente, cambia el orden de como se ejecutan las instrucciones:

Operaciones artiméticas en PHP.

Aparte de las cuatro operaciones básicas (suma "+", resta "-", multiplicación "*" y división "/") debemos señalar también al módulo "%" (devuelve el residuo de la división, si es cero la división es exacta) y la exponenciación "**".

Otros operadores importantes son la suma y asignación "+=", resta y asignación "-=", multiplicación y asignación "*=", división y asignación "/=", módulo y asignación "%=" y concatenación y asignación ".=". Esencialmente lo que hace es aplicar el operador aritmético al valor ya almacenado en la variable con el otro valor a la derecha del signo de igualdad. El

siguiente código ilustrará mejor:

Operadores de comparación.

Necesitamos oepradores que permitan comparar dos variables, pero en PHP hay ciertas condiciones a evaluar. Para empezar ya sabemos que las variables puede almacenar valor numérico y en la siguiente línea le podemos almacenar una cadena de texto, es por ello que necesitamos comparar, incluso, si son el mismo tipo de variable. Por ejemplo, para nosotros 5 es igual que 5,00 paro para PHP el primer valor es un número entero positivo y el segundo es un valor decimal positivo (nosotros en este sitio web nuestro separador decimal es la coma pero a nivel de lenguajes de computación el separador decimal es el punto). Veamos los que disponemos:

- **"==":** Comprueba si son iguales.
- **"!=":** Comprueba si son distintos.
- **"===":** Comprueba si son iguales y de exactamente el mismo tipo.
- **"!==":** Comprueba si son distintos o de distinto tipo.
- **""**: Diferente (igual que !=).
- **""**: Mayor que.
- **"="**: Mayor o igual que.
- **""**: Comparador de orden (PHP 7).
- **"??"**: uno o el otro (PHP 7).

Trabajando con variables de texto.

PHP contiene comandos (funciones no declaradas por nosotros, funciones implícitas) que nos facilitan grandemente nuestro trabajo con cadenas de texto, ya vimos **echo** y **print()** veamos otras más.

Contando el número de caracteres.

Usaremos el comando **strlen()** pasando como argumento nuestra variable (osea colocamos nuestra variable dentro de los paréntesis del comando). Por ejemplo, el siguiente código devolverá el valor de treinta y cinco letras:

Pero el problema aquí es que no son 35 caracteres sino 34: la letra u acentuado es contada como

2 caracteres porque se necesitan 2 bytes para representarla. ***En realidad*** esta función ***devuelve el número de bytes, no el número de caracteres.*** Ya vemos que nuestro hermoso idioma castellano, normado por nuestro brillante Don Andrés Bello, nos da y seguirá dando trabajo a nosotros los programadores.

Contando el número de palabras en una cadena de texto.

Teniendo en cuenta que una palabra esta delimitada antes y después por uno o más espacios *y esté constituida por caracteres A~Z y/o a~z*, el comando que usaremos es **str_word_count()**. Siguiendo con el ejemplo anterior:

Y el resultado será simplemente cinco ¿Qué sucedió si sólo hay cuatro palabras? Pues lo mismo de siempre: los lenguajes de programación están escritos por gente de habla inglesa para los cuales los acentos, la letra eñe -y demás caracteres *no pertenecientes* al idioma inglés **simplemente no existen.** Pero esperen, aún hay más: los números *tampoco* cuentan como caracteres así que por lo menos no nos podemos quejar de que ellos allá en el departamente de programación de PHP *no son estrictos con ellos mismos.* Veremos de forma ampliada que este comando acepta un segundo parámetro llamado **format** que permite el resutlado de manera distinta: si es **cero** es el valor por defecto, **uno** devuelve una matriz conteniendo cada una de las palabras y **dos** devuelve la posición numérica de cada una de las palabras, veamos:

Lo cual devuelve lo siguiente:

```
Palabras encontradas:5    Valor FORMAT por defecto cero:  5    Matriz con
 cada una de las palabras:  Array ( [0] => Rep [1] => blica [2] => Boliva
riana [3] => de [4] => Venezuela )     Matriz con la ubicación de cada un
a de las palabras:  Array ( [0] => Rep [5] => blica [16] => Bolivariana [
35] => de [42] => Venezuela )
```

Fijaos que la cadena de texto hemos introducido más espacios adicionales, la hemos escrito en dos líneas *pero PHP la interpreta como una sola ya que al final de la segunda línea es que se encuentra el punto y coma que indica el final.* También ya sabéis que la letra u acentuada cuenta como dos caracteres (ya que esa característica del juego de caracteres UTF-8 permite "ampliar" el número de bytes para representar los más de 65 mil caracteres que componen el UNICODE) y es

por ello que "blica" comienza en 5 y no en 4.

Pensando ellos allá en lo anterior, a partir de la versión PHP 5.1.0 se le agregó un tercer argumento que permite especificar una excepción a los caracteres que nosotros consideramos **no delimita una palabra.** Esto conlleva definir **un valor constante** a lo largo de todo nuestro programa por medio del comando **define()** que toma dos argumentos separados por una coma: primero el nombre de la variable en sí y segundo el conjunto de caracteres que consideramos NO delimita una palabra y encerrados entre comillas, mirad el siguiente ejemplo:

Importante hacer notar que los valores constantes NO necesitan el signo "$" delante: ya PHP sabe que es un valor "especial" a tomar en cuenta. La función **include()** acepta un tercer parámetro: podemos especificar que el nombre de la variable NO sea sensible a mayúsculas y minúsculas. *Esto quiere decir que si colocamos **true** en el tercer argumento, podremos llamara a nuestra constante a lo largo del programa como "Castellano" o "CASTELLANO" o "cASTELLano", etc. Toda constante declarada es de tipo GLOBAL. A partir de PHP 7 acepta valores de matrices, solo como nota informativa.*

Cadenas reversadas.

Si queremos encontrar palíndromos esta función está adecuada para ello: **strrev()** devuelve una cadena de texto al revés, un ejemplo en idioma latín:

Encontrar una subcadena en una cadena de texto.

Función bastante usada: necesitamos saber si en una cadena de texto existe una subcadena: **strpos()** devolverá un valor numérico, cero o mayor si consigue la subcadena, de lo contrario devolverá un valor **falso**. Debemos tener en cuenta que falso=cero y cualquier otro valor es verdadero, como esta función puede devolver cualquier valor entonces debemos evaluar con el triple signo de igualdad para estar seguros que no se consiguió la subcadena. Ejemplo:

Sustituir una subcadena de texto.

Dada una cadena de texto necesitamos sustituir, si se consigue dentro de ésta, por otra cadena de

texto: **str_replace()** hará el trabajo si le colocamos como primer argumento la subcadena a buscar, el segundoa rgumento la cadena que sustituirá -de ser encontrada- y la cadena de texto en donde se buscará. Ejemplo:

Extraer una subcadena de texto.

Muchas veces necesitamos extraer un subcadena de texto a partir de su posición numérica hasta el fin de la cadena de texto: **substr()** con tres argumentos: cadena de texto, comienzo y longitud. Si se omite la longitud pues devolverá hasta el final de la cadena de texto. Si el número de comienzo es negativo devolverá la cadena contando como primer caracter el primero de la derecha, osea el final. Si no consigue la cadena devolverá un valor falso o una cadena de texto vacía.

Instrucciones condicionales.

Si condicional.

Para evaluar una variable y un valor o dos variables (o dos valores, algo menos común) usaremos el **si condicional**. En PHP tiene la siguiente estructura:

```
if (condición a evaluar) {instrucciones a ejecutar}
```

```
if (condición a evaluar) {    instrucciones a ejecutar;  }
```

De la segunda manera resulta más ordenada y estructurada pues permite multilíneas comodamente. Si necesitamos ejecutar instrucciones diferentes según la condición verdadera o falsa:

```
if (condición a evaluar)
{    instrucciones a ejecutar;  } else {    instrucciones a ejecutar;  }
```

Lo coloreado en verde se ejecuta si la condición es verdadera y lo coloreada en morado de ser falso. Cuando necesitamos evaluar varias condiciones, "filtro de múltiples tamices" utilizaremos esta estructura:

```
if (condición a evaluar)
{    instrucciones a ejecutar;  } elseif
(condición a evaluar) {
 instrucciones a ejecutar;  } else {   instrucciones a ejecutar;  }
```

Con el siguiente ejemplo ilustraremos mejor la estructura: con la función **date()** obtenemos la hora y fecha del equipo pero si le pasamos la máscara "H" nos devuelve solamente la parte de la hora y en formato militar (hora de 24 horas); se pasa por un tamiz dicho resultado y según la hora saluda de manera adecuada:

Como vemos con tres condiciones revista ya cierta complejidad por lo que es necesario un operador más poderoso: el comando **switch().**

Interruptores condicionales.

Acá debemos abrir un poco la mente: muchas veces necesitamos comparar una variable contra diferentes valores para ejecutar una o más instrucciones *e incluso necesitamos que dicha variable sea evaluada más de una vez.* Para ello existe el comando **switch()** que evalua exactamente cada valor y si coincide exactamente ejecuta la(s) instrucción(es) y sale del ciclo con un comando **break()**:

El ejemplo se establece el valor en cero y solo se ejecuta la primera instrucción y sale del ciclo. *Cualquier valor que NO sea 0, 1 ó 2 imprimirá "que i es mayor a 2"* ***incluso si $i tiene un valor negativ0.*** *Es por ello que este comando debe ser ejecutado con precaución.* Otro detalle a tomar en cuenta es que si queremos ejecutar varia líneas de instrucciones debemos encerrarlos entre corchetes.

Comandos de ciclo.

Ciclo mientras "while".

Con el ciclo **while()** si se cumple la condición evaluada se ejecutarán una y otra vez cada una de las instrucciones contenidas y se espera que dentro del ciclo hayan eventos que cambien el valor y poder salir del bucle, sino decimos que el ordenador "está colgado", *así que mucho cuidado con esto último:*

```
while (condición evaluada) {       instrucciones a ser ejecutados;   }
```

Como ejemplo utilizaremos la variable *k* con un valor inicial de 1 y lo incrementaremos de dos en dos hasta que supere a diez:

Ciclo hacer mientras "do ... while".

El ciclo mientras "while" evalúa la condición y de ser cierta ejecuta el bloque de instrucciones deseado *en cambio el ciclo hacer-mientras entra al ciclo, ejecuta y luego evalúa la condición: si se cumple vuelve a ejecutar de lo contrario sale del ciclo y continúa el flujo del programa.* **Para ilustrar bien este punto colocamos el siguiente ejemplo muy a propósito para demostrarlo:**

Este código mostrará "El valor de *k* es: 10" que sabemos que es mayor que 5 *sin embargo el bloque de instrucciones se ejecuta al menos en una oportunidad.*

Ciclo para "for".

El anterior **hacer-mientras** se hace más fácil de ejecutar con el ciclo **para** el cual tiene tres parámetros: el que inicializa la variable, la condición que evalua la variable y el incremento en cada ciclo que le vamos a aplicar a la variable:

for (*inicia_contador; evalua-contador; incrementa_contador*) {
  código_a_ejecutar*;*
}

Con valores numéricos puede ser más "fácil" de comprender:

Ciclo para cada "for each".

Este comando se utiliza con matrices, se estudiará más adelante cuando veamos este tipo de objetos.

Funciones personalizadas.

Ya tocamos el tema de bastantes "comandos" o funciones implícitas en PHP: no es necesario declararlas, ya están implementadas y aunque podemos hacer bastante con el trabajo hecho de manera previa *esto no es suficiente para desarrollar nuestros programas.* Por esta razón PHP (y la mayoría de los lenguajes de programación) permiten hacer nuestras propias funciones, aquellas que esperamos usar de manera repetitiva en nuestros programas.

```
function nombre_de_funcion(argumento) {  }
```

Las funciones NO son sensibles a minúsculas y/o mayúsculas: no importa como la escribamos ellas son llamadas y ejecutadas. Con valores:

Acá comprobamos que no importa como llamemos la función, ésta es ejecutada. Por ahora a esta función no le pasamos ningún argumento pero a continuación la modificamos y dentro de ella con el valor que le pasamos podemos trabajar y procesarlo para mostrarlo o devolver algún resultado en memoria:

También podemos pasar un argumento con un valor por defecto, dado el caso no le pasemos un valor:

Por último con la orden **return *variable*** podremos beneficiarnos del uso más poderoso de las funciones: devolver valores:

Fuentes consultadas.

En idioma inglés:

- "PHP 5 introduction" at w3schools.com
- "Concatenar variables con PHP" por SapoClay.