Ciclo de Grado Superior: Desarrollo de Aplicaciones WEB

MÓDULO 2ºDAW:
Desarrollo Web en entorno Servidor.

Autor: Jorge López.

Tema 01: Sintaxis básica en PHP - Parte 1ª

01 Introducción a PHP.

- **Razones del éxito de PHP**

En este primer punto, comentaremos brevemente la historia de PHP. Es muy **importante**, de cara a poseer un mapa conceptual claro, que si un programador va a empezar a programar en **PHP**, sepa un poco la historia de este popular **lenguaje** de **programación** en entorno **servidor**.

Una de las preguntas más interesantes que podemos hacer respecto a PHP es ¿**qué hace que PHP tenga tanto éxito**?, distintas personas nos han dado respuesta a esta pregunta.

Algunos sostienen que es su excelente conexión con bases de datos, otros que se debe al código abierto; por el contrario otros mantienen que es su funcionamiento. Otros opinan que permite que cualquier usuario obtenga rápidamente resultados de forma sencilla, incluso aquellos que no posean experiencia en programación. Al mismo tiempo PHP permite el desarrollo de proyectos tan complejos como se desee. **Todas estas razones son las que hacen que PHP sea tan especial**.

Lograr el **equilibrio** entre estas dos líneas (**la potencia** y **la sencillez**) no ha sido fácil, pero sí imprescindible para el éxito de PHP. **PHP5**-**PHP7** son los productos de este importante equilibrio.

Aunque dotado de muchas funciones nuevas, esta versión es tan accesible como lo fueron **PHP3** y **PHP4**, con lo que permite un rápido aprendizaje para aquellos que se inicien.

La nueva generación PHP también supone una buena noticia para desabolladores experimentados. **En primer lugar**, por simplificar y mejorar el uso y manejo de un gran número de funciones y, **en segundo lugar**, por poner a su disposición un gran número de nuevas funciones.

Las nuevas opciones orientadas a objetos son un regalo para el desarrollo de aplicaciones de media y gran escala.

- **Historia de PHP**

Rasmus Lerdorf, miembro del equipo de desarrollo de Apache, creó **PHP 1.0** (Personal Home Page) en **1994** (después de una encuesta entre desarrolladores y usuarios de PHP se decidió cambiar el significado de PHP a **PHP Hypertext Preprocessor**.). Su única intención fue la de crear un pequeño sistema de control para verificar el número de personas que leían su curriculum vitae en la Web.

En los meses siguientes a su creación, PHP se desarrolló en torno a un grupo de programadores que comprobaban el código y sus revisiones. Para dar más potencia al sistema, Rasmus creó funciones en lenguaje C para permitir conexión a bases de datos. **Este fue el comienzo de la potencia real del leguaje**.

En **1995**, apareció un conjunto de herramientas sobre PHP. Esta biblioteca se llamó "**Herramientas para páginas personales**" y contenían un analizador de código muy sencillo, un libro de visitas, un contador y algunas macros que facilitaban el trabajo de los diseñadores.

PHP 2.0 vio la luz a mediados de **1995**, (**PHP/FI 2.0**) esta nueva versión contaba con un analizador sintáctico reescrito desde 0, además de unas herramientas escritas para el tratamiento de datos desde un formulario (de ahí el nombre de FI, Form ínterpreter) y conectividad con mSQL (Gestor de bases de datos).

Hacia **1997**, PHP/FI 2.0 se estaba usando en más de 50.000 páginas en todo el mundo. En este período de tiempo, Zeev Suraski y Andi Gutmans decidieron crear una nueva versión de PHP/FI 2.0 para solventar unos problemas con una aplicación de comercio electrónico que estaban desarrollando.

PHP 3.0 vio la luz en junio de **1998**, nació con suculentas innovaciones como la conectividad con varios gestores de bases de datos, protocolos y una API ampliada. En esta versión ya se contemplaba la programación orientada a objetos.

En **1999** se realizó la primera revisión del motor Zend (Zend Engine), que aportaba modularidad, claridad y herramientas de optimización para páginas de gran escala. Zend viene de la unión de Zeev y Andi.

PHP 4.0 vio la luz en mayo de **2000**, dividida en 3 partes: El motor Zend, la API de servidor y los módulos de funciones. El motor Zend es el responsable de analizar el código PHP, definir la sintaxis y del lenguaje de programación. La API permite la comunicación con el servidor. Con esta API es posible utilizar PHP desde varios servidores. Los módulos contienen funciones para el manejo de cadenas, archivos XML o tratamiento de imágenes.

La orientación a objetos no está muy lograda en PHP 4.0. Los objetos tienen un tratamiento muy pobre e ilógico. La definición de las variables miembro (propiedades) y los métodos son siempre públicos, por lo que la encapsulación es nula. Todos los objetos se pasan por valor por defecto cuando deberían pasarse por referencia.

Todas estas propuestas realizadas por el equipo de desarrollo de PHP han desembocado en la creación del motor Zend 2.0. y su consecuencia PHP 5.0.

PHP 5.0 vio la luz en julio de **2004**, incorpora una verdadera orientación a objetos. Añadiendo las palabras reservadas public, protected y prívate a la definición de las propiedades y métodos de los objetos, se permite una verdadera encapsulación. Además del considerable avance con respecto a los objetos, PHP 5 incorpora un control de errores muy mejorado, al estilo de los lenguajes de programación más avanzados.

PHP 6.0 el desarrollo de **PHP 6** fue suspendido porque los desarrolladores decidieron que el enfoque para tratar cadenas Unicode no era correcto. Las mejoras planeadas para **PHP 6** fueron añadidas en su lugar en **PHP 5.3.0** (Soporte para espacios de nombre, enlace estático en tiempo de ejecución, funciones lambda, clausuras, goto) y en **PHP 5.4.0** (traits, revinculación de clausura).

PHP 7.0 vio la luz en noviembre de **2015**, proporciona un rendimiento mayor y ofrece múltiples novedades.

PHP 8.3 es la última versión de este popular lenguaje de programación en entorno servidor.

https://www.php.net/

02 Páginas PHP.

Las **páginas PHP** pueden ser páginas Web normales a las que se cambia la extensión, poniendo **.php** en vez de **.htm** ó **.html**.

En una página cuyo nombre tenga por extensión **.php** se pueden insertar instrucciones escritas en lenguaje PHP, anteponiendo los símbolos **<?php** a la primera instrucción y escribiendo después de la última instrucción los símbolos **?>**.

A cada uno de estos "bloques de instrucciones" le llamaremos un **script**.

La primera instrucción PHP que veremos es esta: **echo** "esto es un texto de ejemplo";

La instrucción **echo** seguida de un texto entrecomillado hará que el PHP escriba en la página Web resultante el contenido de esa cadena de texto. Al final de cada instrucción debemos insertar siempre un punto y coma (;). El (;) indicará a PHP que lo que viene a continuación es una nueva instrucción.

Para facilitar la depuración de los scripts no suelen escribirse dos instrucciones en una misma línea.

Las páginas escritas en PHP, además, pueden ser páginas en HTML que contienen, además de las etiquetas normales de una página HTML, el programa que queremos ejecutar en PHP.

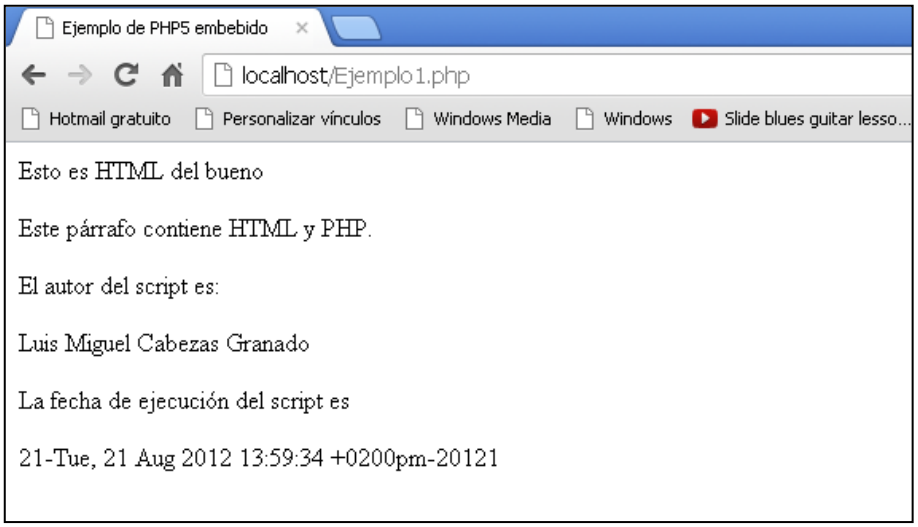
EJEMPLO 01: (php y html mezclados)

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <TITLE>Ejemplo 01</TITLE>
</HEAD>
<BODY>
    <p>Ejemplo de código PHP dentro de HTML</p>
    <p>Esto es HTML del bueno</p>
    <!-- Esto es un comentario en HTML -->
    <!-- Con este símbolo comienza el código PHP -->
    <?php
    // lo siguiente es una simple asignación de variables y su salida por pantalla
    $nombre="Luis Miguel";
    $apellidos="Cabezas Granado";
    $fecha_hoy=date('d-ra-Y1');
    // con este símbolo termina el código PHP
    ?>
    <p>Este párrafo contiene HTML y PHP.</p>
    <p>El autor del script es:</p>
    <?php echo ("$nombre $apellidos") ?>
    <p>La fecha de ejecución del script es:</p>
    <?php echo("$fecha_hoy") ?>
</BODY>
</HTML>
```

No existe límite en cuanto al número de scripts distintos que pueden insertarse dentro de una página.

Hemos guardado este ejercicio con el nombre de **ejemplo1.html**, si visualizamos ambos ejemplos veremos que el resultado es distinto:

En el caso del **ejemplo1.php** este sería el resultado, se **ejecuta** todo el código **php**:



En el caso del **ejemplo1.html** este sería el resultado, el **servidor** **ignora** todo el código **php**:

03 Comentarios.

¿Por qué usar líneas de comentario?

A primera vista pueden parecer inútiles. ¿**Para qué recargar las páginas con contenidos que no se van a ver ni ejecutar**?

Las líneas de comentario sirven para poder recordar en un futuro qué es lo que hemos hecho al escribir un script y por qué razón lo hemos hecho así.

A medida que vayamos avanzando verás que en muchos casos tendremos que aplicar estrategias individuales para resolver cada problema concreto.

Cuando necesites hacer una corrección o una modificación al cabo de un tiempo verás que confiar en la memoria no es una buena opción. **Es mucho mejor utilizar una línea de comentario que confiar en la memoria**. (**¡Palabra! de un programador**).

Además piensa que pueden darse distintas situaciones, el código que tu programes hoy puede ser modificado el día de mañana por otra persona, o que tu trabajo sea el tener que modificar el código que haya programado otra persona, así que parece lógico que **cuanto más documentado esté un programa con líneas de comentario, más fácil será la labor de modificarlo**.

- Para insertar comentarios en los scripts de PHP podemos optar entre varios métodos y varias posibilidades:

  - **Una sola línea**:

    Basta colocar los símbolos **//** al comienzo de la línea o detrás del punto y coma que señala el final de una instrucción. También se puede usar el símbolo **#** en cualquiera de las dos posiciones.

  - **Varias líneas**:

    Si un comentario va a ocupar más de una línea podremos escribir **/*** al comienzo de la primera de ellas y ***/** al final de la última. Las líneas intermedias no requieren de ningún tipo de marca.

Conclusión

A pesar de lo que hemos comentado anteriormente, **el script real que se sube al servidor es un script sin comentarios**, los comentarios es texto sin funcionalidad, un script con muchos comentarios será más lento en su ejecución (ya que los comentarios forman parte de la página web y también se cargan) y por tanto **se eliminarán todos los comentarios**.

Existirá una versión de ese mismo script con comentarios para una futura actualización.

Veamos un **ejemplo** de todo esto:

EJEMPLO 02: (php y html mezclados)

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <TITLE>Ejemplo 02</TITLE>
</HEAD>
<BODY>
    <?php
    // Este comentario no se verá en la página

    /* ponemos <br> al final del texto para que cuando se ejecute cada una
      de las instrucciones echo se escriba, además del texto, un
      salto de línea HTML. De este modo, el resultado de cada ECHO
      aparecerá en una línea diferente */

    echo "Este texto se leerá <br> "; // Esto no se leerá

    /* Este es un comentario de múltiples líneas no terminará
      hasta que no lo cerremos con el siguiente símbolo */

    echo "Este es el segundo texto que se leerá<br>";

    # Este es un comentario tipo shell que tampoco se leerá
    # Este, tampoco

    echo ("Aquí el tercer texto visible<br><br>"); #comentario invisible
    ?>
<!--Esto es un comentario en HTML fuera del script de PHP-->
</BODY>
</HTML>
```

04 Constantes.

¿Qué es una constante?. Definición

Una **constante** es un valor (un número o una cadena) que no va a ser modificado a lo largo del proceso de ejecución de los scripts que contiene un documento.

Para mayor comodidad, a cada uno de esos valores se le asigna un nombre, de modo que cuando vaya a ser utilizado baste con escribir su nombre.

Cuando ponemos nombre a una constante se dice que definimos esa constante. En PHP las constantes se definen mediante la siguiente instrucción:

```
define("Nombre","Valor")
```

Los valores asignados a las constantes se mantienen en todo el documento, incluso cuando son invocadas desde una función.

No es necesario escribir entre comillas los valores de las constantes cuando se trata de constantes numéricas.

Si se realizan operaciones aritméticas con constantes tipo cadena, y su valor comienza por una letra, PHP les asigna valor cero.

Si una cadena empieza por uno o varios caracteres numéricos, al tratar de operarla aritméticamente PHP considerará únicamente el valor de los dígitos anteriores a la primera letra o carácter no numérico.

El punto entre caracteres numéricos es considerado como separador de parte decimal.

Tal como puedes ver en el código fuente del **ejemplo 03**, es posible definir constantes a las que se asigne como valor el resultado de una operación aritmética.

Mediante una **sola instrucción echo** se pueden presentar (en la ventana del navegador del cliente) de forma simultánea varias cadenas de caracteres y/o constantes y variables. Basta utilizar una **coma** o **punto** como separador entre cada una de ellas.

La forma anterior no es la única (ni la más habitual) de enlazar elementos mediante la instrucción echo. Si en vez de utilizar la coma usáramos un punto (el concatenador de cadenas) conseguiríamos el mismo resultado.

Cuando enlacemos elementos distintos (**cadenas**, **constantes** y/o **números**) hemos de tener muy en cuenta lo siguiente:

– Cada una de las sucesivas cadenas debe ir encerrada entre sus propias comillas.

– Los nombres de constantes nunca van entre comillas.

Constantes predefinidas

PHP dispone de algunas constantes predefinidas que no requieren la instrucción: define("Nombre","Valor")

Algunas de ellas son estas: (hay algunas más)

__FILE__ : recoge el nombre del fichero que se está ejecutando y la ruta completa de su ubicación en el servidor.

__LINE__ : recoge el número de línea (incluidas líneas en blanco) del fichero PHP cuyos scripts está interpretando. Puede resultar muy útil para depurar programas escritos en PHP.

PHP_OS: recoge información sobre el Sistema Operativo que utiliza el servidor en el que se está interpretando el fichero.

PHP_VERSION: recoge la versión de PHP que está siendo utilizada por el servidor.

Veamos un **ejemplo** de todo esto:

EJEMPLO 03: (php y html mezclados)

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <TITLE>Ejemplo 03</TITLE>
</HEAD>
<BODY>
    <?php
    /* Definiremos la constante C1 y le asignaremos el valor 100 */
    define("C1",100);
    /* Definiremos la constante C2 y le asignaremos el valor 200 */
    define("C2",200);

    // Visualizamos los valores:
    // Ojo!!
    // Observa las distintas formas en las que utilizamos echo

    echo "Valor de C1=".C1. " y Valor de C2=".C2."<br>";
    echo "Valor de C1*C2=".C1*C2."<br><br>";

    echo "Valor de C1=",C1," y Valor de C2=",C2,"<br>";
    echo "Valor de C1*C2=",C1*C2,"<br><br>";

    echo "La ruta completa de este fichero es: ".__FILE__."<br><br>";

    echo "Esta es la línea: ".__LINE__." del fichero <br><br>";

    echo "Estamos utilizando la versión: ".PHP_VERSION." de PHP <br>";
    ?>
</BODY>
</HTML>
```

05 Variables.

¿Qué es una variable?

Podríamos decir que una **variable** es un espacio de la memoria RAM del ordenador que se reserva a lo largo del tiempo de ejecución de un script, para almacenar un determinado tipo de datos cuyos valores son susceptibles de ser modificados por medio de las instrucciones contenidas en el propio programa.

Nombres de variables

En PHP todos los nombres de variable tienen que empezar por el símbolo **$**.

Los nombres de las variables han de llevar una letra inmediatamente después del símbolo **$**:
$pepe1 es un nombre válido, pero **$1pepe** no es un nombre válido.

Para PHP las letras mayúsculas y las minúsculas son distintas:
la variable **$pepe** es distinta de la variable **$Pepe**.

Definición y Tipos de variables

PHP **no requiere una definición previa de las variables**. Se definen en el momento en que son necesarias y para ello basta que se les asigne un valor.

Ya que en PHP no es necesario definir el tipo de variable, por lo tanto, una misma variable puede contener una cadena de caracteres en un momento del proceso y, posteriormente, un valor numérico, susceptible de ser operado matemáticamente.

La sintaxis para declarar una variable es esta:

```
$variable=valor;
```

Existen varios tipos de variables:

- **Entero (integer)**: almacena números sin decimales. Por defecto usa la notación decimal, se puede utilizar también la notación Octal (0) o Hexadecimal (0x).
- **Coma Flotante (double)**: números con decimales.
- **Carácter (string)**: texto o información numérica escrita entre comillas dobles ("") o simples ('').
- **Boolean**: sólo tiene dos posibles valores: Verdadero o Falso (1/0).
- **Nulo {NULL)**: es un tipo especial que solo contiene un valor: NULL.
- **Vectores (array)**: colecciones de datos.
- **Objetos (object)**: Conjunto de datos y funciones independientes.

Los 5 primeros tipos son **simples** y los 2 siguientes (array y object) son **compuestos**.
Trataremos los tipos compuestos en capítulos sucesivos.

Veamos un **ejemplo** de todo esto:

EJEMPLO 04: (php y html mezclados)

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <TITLE>Ejemplo 04</TITLE>
</HEAD>
<BODY>
    <?php
    // Asignación de números enteros, de coma flotante y cadenas de caracteres
    $numero_entero = 12343;
    $numero_flotante = 12343.123;
    $cadena_caracter = "esto es una cadena";

    // Asignación de los tipos especiales boolean y NULL
    $verdadero = TRUE;
    $vacio = NULL;

    // Visualizamos por pantalla el valor de las variables
    // ojo!!
    // Observa las distintas formas en las que utilizamos echo

    echo "El valor de la variable numero_entero es: ",$numero_entero,"<br>";
    echo "El valor de la variable numero_flotante es: ".$numero_flotante."<br>";
    echo "El valor de la variable cadena_caracter es: $cadena_caracter"."<br>";
    echo "El valor de la variable verdadero es: ".$verdadero."<br>";
    echo "El valor de la variable vacio es: ".$vacio."<br>";
    ?>
</BODY>
</HTML>
```

Ámbito de las variables

Los valores de una variable que ha sido definida en cualquier parte de un script (siempre que no sea dentro de una función) pueden ser utilizados desde cualquier otra parte de ese script, **excepto** desde dentro de las funciones que contuviera el propio script o desde las que pudieran estar contenidas en un fichero externo.

Si una variable es definida dentro de una función sólo podrá ser utilizada dentro esa función.

Si en una función aludimos a una variable externa a ella, PHP considerará esa llamada como si la variable tuviera valor cero (en caso de ser tratada como número) o una cadena vacía "" (en caso de ser tratada como una cadena). Igual ocurriría si desde fuera de una función hiciéramos alusión a una variable definida en esa función.

Si definimos dos variables con el mismo nombre (**nunca hagas esto!!**), una dentro de una función y otra fuera, PHP las considerará distintas. La función cuando sea ejecutada la utilizará y asignará sus propios valores sin que sus resultados modifiquen la variable externa.

Variables globales

Lo comentado anteriormente, admite algunas excepciones. Las funciones pueden utilizar valores de variables externas a ellas pero ello **requiere incluir dentro de la propia función la siguiente instrucción**:

global nombre de la variable;

Veamos un **ejemplo** de esto:

EJEMPLO 05: (php y html mezclados)

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <TITLE>Ejemplo 05</TITLE>
</HEAD>
<BODY>
    <?php
    // variable definida fuera de la función
    // la función en principio (sin usar 'global') no puede acceder al valor de esta variable
    $variable_prueba = "esto es el valor";

    function ejemplo()
    {
      // con global la función puede acceder a la variable $variable_prueba, declarada fuera de la función
      // *****OJO!!***** prueba este script descomentando la siguiente línea
      global $variable_prueba;
      // con la línea de global comentada la siguiente línea no imprimirá el valor
      echo "El valor de la variable en la función antes del cambio es: ".$variable_prueba."<br><br>";
      // con la línea de global comentada esta variable sería distinta a la que se ha declarado fuera
      $variable_prueba="cambio el valor";
      echo "El valor de la variable en la función después del cambio es: ".$variable_prueba."<br><br>";
    }
    // visualizo el valor de la variable ANTES de llamar a la función ejemplo
    echo "El valor de la variable ANTES DE es: ".$variable_prueba."<br><br>";
    ejemplo();
    // Visualizo el valor de la variable DESPUÉS de llamar a la función ejemplo
    echo "El valor de la variable DESPUÉS DE es: ".$variable_prueba."<br><br>";
    ?>
</BODY>
</HTML>
```

Como podemos ver en este ejemplo, dentro de una función estamos utilizando una variable (**$variable_prueba**) que ha sido declarada fuera de dicha función, con la instrucción global tenemos acceso al contenido de esa variable, y como podemos apreciar también podemos modificar el valor de dicha variable.

Valores de las variables

Una variable que ha sido definida dentro de una función, **pierde su valor en el momento en el que abandonemos el ámbito de esa función**, es decir, cuando finaliza su ejecución.

Variables estáticas:

Para poder conservar el último valor de una variable definida dentro de una función basta con definirla como estática. La instrucción que permite establecer una variable como estática es la siguiente:

```
static nombre = valor;
```

La variable conservará el último de los valores que pudo habérsele asignado durante la ejecución de la función que la contiene. No retomará el valor inicial hasta que se actualice la página.

Veamos un **ejemplo** de esto: (página siguiente)

EJEMPLO 06: (php y html mezclados)

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <TITLE>Ejemplo 05</TITLE>
</HEAD>
<BODY>
    <?php
    function sinEstaticas()
    {
        $a=0; $b=0;

        echo "ejecución de sinEstaticas <br>";

        // Imprimamos estos valores iniciales
        echo "Valor inicial de a: ",$a,"<br>";
        echo "Valor inicial de b: ",$b,"<br>";

        /* Modifiquemos esos valores */
        $a +=5; $b -=7;

        # Visualicemos los nuevos valores de las variables
        echo "Nuevo valor de a: ",$a,"<br>";
        echo "Nuevo valor de b: ",$b,"<br>";
    }

    function conEstaticas()
    {
        # Definimos $a y $b como variables estáticas
        static $a=0;static $b=0;

        echo "ejecución de conEstaticas <br>";

        # Imprimamos estos valores iniciales
        echo "Valor inicial de a: ",$a,"<br>";
        echo "Valor inicial de b: ",$b,"<br>";

        /* Modifiquemos esos valores */
        $a +=5; $b -=7;

        echo "Nuevo valor de a: ",$a,"<br>";
        echo "Nuevo valor de b: ",$b,"<br>";
    }

    sinEstaticas();
    sinEstaticas();
    sinEstaticas();
    echo "<br><br><br>";
    conEstaticas();
    conEstaticas();
    conEstaticas();
    ?>
</BODY>
</HTML>
```

EJEMPLO 07: (paso de **parámetros** por **valor** y por **referencia** en **PHP**)

```php
<?php
// declaramos una variable pública
$numero=100;

// valor
// a la función se le pasa una copia de la variable
// a la variable original no le afectan los cambios
function por_valor($variable)
{
   $variable=$variable+100;
}

// referencia
// a la función se le pasa la variable $numero y NO una copia de la variable
// a la variable original SI le afectan los cambios
function por_referencia(&$variable)
{
   $variable=$variable+100;
}

por_valor($numero);
echo "llamamos a la función por_valor() <br>";
echo "la variable después de llamar a la función tiene un valor de: ".$numero."<br><br>";

por_referencia($numero);
echo "llamamos a la función por_referencia() <br>";
echo "la variable después de llamar a la función tiene un valor de: ".$numero."<br><br>";

?>
```

Determinación del tipo de variable utilizada

Dado que PHP gestiona las variables de forma automática y modifica los tipos de acuerdo con los valores que va tomando durante la ejecución del script, se pude recurrir a la función siguiente para **determinar el tipo de una determinada variable en el momento actual**:

gettype(nombre de la variable)

Esta función devolverá una cadena de caracteres indicando el tipo de la variable que contiene. La cadena devuelta por esta función puede ser: Integer, Double o String.

En la tabla siguiente tienes algunos ejemplos de aplicación de esa función:

Ejemplos de determinación del tipo de una variable		
Variable	Sintaxis	Devuelve
$a1=347	**echo gettype($a1)**	**integer**
$a2=2147483647	**echo gettype($a2)**	**integer**
$a3=-2147483647	**echo gettype($a3)**	**integer**
$a4=23.7678	**echo gettype($a4)**	**double**
$a5=3.1416	**echo gettype($a5)**	**double**
$a6="347"	**echo gettype($a6)**	**string**
$a7="3.1416"	**echo gettype($a7)**	**string**
$a8="Solo literal"	**echo gettype($a8)**	**string**
$a9="12.3 Literal con número"	**echo gettype($a9)**	**string**
$a10=""	**echo gettype($a10)**	**string**

Forzado de tipos

PHP permite forzar los tipos de las variables. Eso quiere decir que se puede obligar a PHP a asignar un tipo determinado a una variable determinada, siempre que los valores que contenga estén dentro del rango del nuevo tipo de variable.

Los tipos se pueden forzar tanto en el momento de definir la variable como después de haber sido definida.

Al asignar un valor a una variable, se puede forzar su tipo de la siguiente forma. Si deseamos que la variable pase a ser tipo **double** basta con anteponer a su valor la palabra double entre paréntesis, tal como se indica una de las expresiones: (double), (integer) o (string).

Por **ejemplo**:

```
$a=(double) 45;

$a=(string) 45;

$a=(integer) 45;
```

¡Cuidado! Al modificar los tipos de variables pueden modificarse sus valores. Si forzamos a entera una variable que contenga un número decimal se perdería la parte decimal y la variable modificada solo contendría el valor de la parte entera. Si tratamos de convertir a numérica una variable alfanumérica el nuevo valor sería cero.

Aquí tienes algunos ejemplos relacionados con la advertencia anterior:

Nuevos valores de la variable		
Valor inicial	Sintaxis	Nuevo valor
$a1=347	**echo ((double)$a1)**	**347**
$a2=2147483647	**echo ((double)$a2)**	**2147483647**
$a3=23.7678	**echo ((integer)$a3)**	**23**
$a4="3.1416"	**echo ((double)$a4)**	**3.1416**
$a5="347"	**echo ((int)$a5)**	**347**
$a6="3.1416"	**echo ((string)$a6)**	**3.1416**
$a7="Solo literal"	**echo ((int)$a7)**	**0**
$a8="12.3 Literal con número"	**echo ((double)$a8)**	**12.3**
$a9=""	**echo ((int)$a9)**	**0**
$a10="esto es una prueba"	**echo ((int)$a10)**	**0**

La forma más aconsejable de forzado de tipos en variables que ya estuvieran definidas previamente, es el uso de la siguiente función:

settype(nombrevariable,tipo)

donde **nombrevariable** es el nombre de la variable cuyo tipo pretendemos modificar y **tipo** una expresión que puede contener (entre comillas) uno de estos valores: '**double**', '**integer**', o '**string**' según se trate de forzar a: **real**, **entero**, o **cadena**.

Un **ejemplo** podría ser este:
settype($a,'integer') que convertiría a tipo entero la variable $a.

La ejecución de la función **settype** devuelve (da como resultado) un valor que puede ser: true o false (1 ó 0) según la conversión se haya realizado con éxito o no haya podido realizarse.

En la tabla siguiente tienes algunos ejemplos de aplicación de esa función:

Forzado de tipos con settype()		
Variable	Sintaxis	Devuelve
$a1=347	**echo (settype($a1,'double')**	**1**
	echo gettype($a1)	**double**
	echo $a1	**347**

$a2=2147483647	echo (settype($a2,'double')	1
	echo gettype($a2)	double
	echo $a2	2147483647
$a3=-2147483647	echo settype($a3,'double')	1
	echo gettype($a3)	double
	echo $a3	-2147483647
$a4=23.7678	echo settype($a4,'integer')	1
	echo gettype($a4)	integer
	echo $a4	23
$a5=3.1416	echo settype($a5,'integer')	1
	echo gettype($a5)	integer
	echo $a5	3
$a6="347"	echo settype($a6,'double')	1
	echo gettype($a6)	double
	echo $a6	347
$a7="3.1416"	echo settype($a7,'integer')	1
	echo gettype($a7)	integer
	echo $a1	3
$a8="Solo literal"	echo settype($a8,'double')	1
	echo gettype($a8)	double
	echo $a8	0
$a9="12.3 Literal con número"	echo settype($a9,'integer')	1
	echo gettype($a9)	integer
	echo $a9	12

Algunas funciones útiles

- **isset()**

Con esta función podemos averiguar si una **variable** o **función** existe dentro de nuestro programa. Si existe devuelve true y si no existe false.

Crea un fichero llamado **prueba.php** y prueba este código:

```
<?php
$DNI = "8868543-Z";
if (isset($DNI))
    {
    echo ("La variable DNI existe!!!");
    }
$DNI =null;
if (isset($DNI))
    {
    echo ("La variable DNI existe!!!");
    }
?>
```

○ **unset()**

Libera la memoria ocupada por una variable, destruyendo su nombre y su contenido, después de usar unset ().

Crea un fichero llamado **prueba.php** y prueba este código:

```
<?php
$Nombre = "Pepe López";
if (isset($Nombre))
    {
    echo ("El nombre existe!!! <BR>");
    }
//Podemos comprobar qué pasa si liberamos la variable $Nombre
unset($Nombre);
if (isset($Nombre))
    {
    echo ("El nombre existe!!!");
    }
else
    {
    echo ("El nombre ya no existe!!!");
    }
?>
```

El resultado es el siguiente:

El nombre existe !!!
El nombre ya no existe !!!

○ **is_integer(), is_double(), is_string()**

Estas funciones devuelven true si la variable pasada coincide con el tipo que indica la función.

Crea un fichero llamado **prueba.php** y prueba este código:

```
<?php
$numero_entero = 0;
if (is_integer($numero_entero))
    {
    echo ("numero_entero es del tipo integer");
    }
?>
```

Si la variable **$numero_entero** se evalúa con la función **is _ integer()** devolverá **true**.
Por tanto el resultado es el siguiente: numero_entero es del tipo integer

○ **intval(), doubleval(), strval()**

Convierte el valor de una determinada variable. Esta función no permite la conversión a tipos object o array.

Crea un fichero llamado **prueba.php** y prueba este código:

```
<?php
//Conversión de un tipo string a un integer

$cadena = "232pepe";
echo "El tipo de la variable cadena es ".gettype($cadena)."<br>";
$numero = intval($cadena) ;
echo ("el numero es $numero"."<br>");
echo "El tipo de la variable $numero es ".gettype($numero)."<br>";
echo ("el valor de la variable cadena es $cadena"."<br>");

?>
```

El resultado es el siguiente:

El tipo de la variable $cadena es **string**
el valor de la variable $numero es **232**
El tipo de la variable $numero es **integer**
el valor de la variable $cadena es **232pepe**

Como podemos apreciar estas funciones no cambian de tipo de datos a la variable sino solo el contenido.

Variables SuperGlobales

A partir de la **versión 4.1.0 de PHP** se ha creado un nuevo tipo de variables capaces de comportarse como globales sin necesidad de que se definan como tales.

Estas variables que no pueden ser creadas por usuario, recogen de forma automática información muy específica (nombre, rutas, nombres de páginas, IP del servidor, etc.) y tienen nombres preasignados que no pueden modificarse:

$_SERVER, **$_POST**, **$_FILE, $_SESSION, $_GET** o **$_ENV** son de las más importantes.

Los distintos tipos:

Veamos los diferentes tipos de variables predefinidas que existen en PHP. Por ahora, **no te preocupes demasiado sobre la forma de utilizarlas. Las incluimos aquí como una simple enumeración y con una breve descripción de su utilidad**.

En temas posteriores haremos referencia a ellas. Por el momento nos bastará con conocer su existencia. Vamos a ver los distintos tipos de estas variables:

○ **Variables de sesión**:

Las identificaremos por los nombres **$_SESSION**.

Este tipo de variables las utilizaremos cuando hagamos mención al uso de sesiones.

La utilización de sesiones (ya abundaremos en ello) es una forma de recoger, de forma temporal en un documento del mismo carácter, información específica generada a través de los accesos de cada uno de los usuarios.

Por ejemplo, cuando accedes a un portal donde debes introducir clave y contraseña se crea un documento temporal en el servidor con un número único y exclusivo para ese acceso (identificador de sesión) que te permite acceder a diferentes apartados de ese espacio sin necesidad de que reescribas, en cada una de las páginas, esos mismos valores.

El carácter efímero de las sesiones seguramente lo has comprobado más de una vez cuando al actualizar una página te ha aparecido un mensaje diciendo que la sesión ha caducado (por ejemplo una sesión caduca si tras un corto periodo de tiempo no haces nada con el ordenador).

○ **Variables del método POST**:

Las identificaremos por los nombres **$_POST**.

Este tipo de variables (que utilizaremos con frecuencia) recogen la información que se envía desde el cliente para ser utilizada por el servidor.

Recuerda el carácter dinámico de PHP y que ese dinamismo (interacción cliente – servidor) requiere que el servidor guarde los datos remitidos por el cliente.

○ **Variables del método GET**:

Las identificaremos por los nombres **$_GET**.

Son muy similares a las anteriores. La existencia de los dos tipos se justifica porque también existen dos tipos de métodos (maneras) de enviar datos desde el cliente hasta el servidor.

Cuando el método de envío es el llamado GET los datos se recogen en variables de este tipo, y, por el contrario, si ese método envío fuera POST se recogerían en aquellas.

○ **Variables de transferencia de ficheros**:

Las identificaremos por el nombre **$_FILES**.

Para el caso de transferencia de ficheros desde el cliente al servidor (subir ficheros) es necesario un procedimiento distinto de los anteriores.

Será en este caso cuando se utilicen variables de este tipo.

○ **Variables de tipo GLOBALS**:

A diferencia de las anteriores tipos de variables, las variables de este tipo disponen de una sintaxis única **$GLOBALS**, sin que quepa ninguna otra opción.

Su finalidad es recoger en una tabla los nombres de **todas la variables establecidas como globales** (en cada momento) así como sus valores.

Conocida la existencia de los diferentes tipos de variables predefinidas **no será preciso que profundicemos más en el asunto**.

Lo trataremos en el momento en el que tengamos que hacer uso de cada una de ellas.