Tema 2.- Inserción de código en páginas web

Indice

Tema 2.- Inserción de código en páginas web

1.- Sintaxis y tipos de datos en PHP

1.1.- Sintaxis PHP

Todo código PHP dentro de nuestros documentos HTML, debe ir encerrado entre las etiquetas **<?php** y **?>**.

```
1	<?php
2	echo "Hola Mundo";
3	?>
```

El código dentro de estas etiquetas **<?php y ?>** será interpretado como código PHP por el servidor.

Sin embargo, podemos realizar comentarios dentro de un código PHP que no serán tenidos en cuenta por el servidor, y que serán de gran utilidad para el mantenimiento de la aplicación, ya que seguramente, dentro de unos meses, no recordemos para qué servía ese código que habíamos escrito.

Existen varias formas de realizar un comentario en PHP:

// para un comentario que ocupe una única línea.
/ */ para un comentario que ocupe varias líneas.*

```
1	<?php
2	//Esto es un comentario de una línea
3
4	/* Esto
5	es un comentario
6	de varias líneas */
7	?>
```

Todas las instrucciones en PHP, finalizan con un punto y coma (;)

Si dejamos una instrucción sin cerrar, probablemente se lance un mensaje de error a la hora de visualizar el resultado en el navegador.

```
<?php
Instrucción 1;
Instrucción 2;
Instrucción 3;
?>
```

Todos los documentos que incluyan código PHP, deberán guardarse con la extensión **.php** para que estos sean tratados correctamente por el servidor (index.php, clientes.php, contacta.php…).

1.2- Tipos de datos en PHP

Los diferentes tipos de datos que soporta PHP son:

- Integer (entero): números enteros (1, 2, 3, 4, 5, 6, 7, 8, 9…).
- Double (reales): números en coma flotante (con decimales) (1,45 3,89875 8,7724 etc).
- String: cadenas de caracteres.
- Boolean: valor lógico que solo admite true o false (verdadero o falso).
- Array: conjunto de valores.
- Object: tipo especial de dato complejo.
- Resourse: identificador que hace referencia a un recurso externo.
- Null: valor que indica ausencia de valor.

2.- Variables y constantes en PHP

Una **variable** es un espacio de memoria reservado para almacenar un valor, con el objeto de poder visualizarlo, modificarlo u operar con él cuando se lo necesite durante la ejecución de un programa.

Esto significa que en cualquier momento, una variable puede cambiar de valor (de ahí su nombre).

La **constante** es muy similar, con la diferencia de que su valor no cambiar en ningún momento de la ejecución del programa.

2.1- Variables

En PHP, las variables son declaradas anteponiendo el símbolo del dólar ($) al nombre de esta. Los nombres de las variables deberán comenzar por una letra o por un guión bajo (_). No podrán comenzar por un número o un carácter especial.

```
1    <?php
2    $nombre = “Sergio”;
3    ?>
```

En este ejemplo hemos declarado una variable llamada “nombre” y le hemos asignado como valor el texto “Sergio”.

Los nombres de variable en PHP son sensibles a mayúsculas y minúsculas, por lo que:

$nombre, no es igual que **$Nombre**. Ambas son consideradas variables distintas.

Todos los valores que sean cadenas de caracteres, deben especificarse entre comillas (“ “).

Sin embargo, si el valor se trata de un número con el que vamos a realizar operaciones aritméticas, se especificará sin comillas.

$numero = 2;

Si el número se especifica entre comillas, ($numero = “2”), PHP no generará ningún error, simplemente tratará al número como si fuese otro carácter normal.

Ejemplos:

```
1    <?php
2    $num1 = 4;
3    $num2 = 6;
4    echo $num1 + $num2;
5    ?>
```

Hemos declarado una variable llamada $num1 a la que se le ha asignado el valor 4, y otra variable llamada $num2 a la que se le ha asignado el valor 6. Hemos pedido que muestre en pantalla el resultado de sumar ambas variables.

Este ejemplo mostrará en pantalla: 10

La instrucción **“echo”, se usa para imprimir información en la pantalla** dentro de un código PHP.

Con él podemos visualizar el contenido de variables, redactar cadenas de texto, e incluso incrustar código HTML.

Dentro de “echo” incluso podemos mezclar cadenas de texto con variables.

Veamos un ejemplo:

```
1    <?php
2    $edad = 30;
3    echo “Me llamo Jorge y tengo “ .  $edad  . “ años”;
4    ?>
```

El resultado en pantalla sería: Me llamo Jorge y tengo 30 años.

Analizando el código, hemos declarado una variable llamada $edad asignándole el valor 30.

Mediante la instrucción “echo”, pedimos que se visualice la cadena de texto “Me llamo Jorge y tengo “, a continuación hemos concatenado una variable usando el símbolo del punto (.), especificamos el nombre de la variable ($edad), y volvemos a concatenar con el punto (.) la cadena de texto “ años”;

Los espacios que dejamos al final o principio de las comillas, sirven para que el valor de la variable no se muestre pegada al texto, para ello especificamos los espacios dentro de las comillas.

Otro ejemplo más completo:

```
1    <?php
2    $nombre = “Jorge”;
3    $edad = 30;
4    echo “Me llamo “ . $nombre . “ y tengo “ .  $edad  . “ años”;
5    ?>
```

2.2- Modificar el valor de una variable

En este ejemplo vamos a modificar el valor de una variable:

```
1    <?php
2    $edad = 30;
3    echo "Mi edad es " . $edad;
4    $edad++;
5    echo "Mi edad es " . $edad;
6    ?>
```

En la primera instrucción se le asigna a la variable $edad el valor 30 y muestra en pantalla el texto «Mi edad es 30 años»;

La siguiente instrucción aumenta el valor de $edad en una unidad ($edad++), gracias al operador de incremento (++), que solo se puede utilizar en variables de tipo numérico.

Por lo que al volver a usarlos en una frase, ahora el texto cambia a «Mi edad es 31».

2.3- Contantes

Existen casos en los que necesitamos almacenar un valor en concreto, pero sabemos que ese valor no se va a modificar en ningún momento de la ejecución del programa. Para ello usamos una constante.

La forma de declararla es la siguiente:

```
1    <?php
2    define (“constante”, “valor”);
3    echo (constante);
4    ?>
5
6    <?php
```

```
7    define (“PI”, “3,14159265358979323846”);
8    echo (PI);
9    ?>
```

Un ejemplo práctico de uso de constantes puede ser declarar un nombre de usuario y una contraseña para poder acceder a un área privada dentro de un sitio web.

Como sabemos que ese usuario y contraseña van a ser siempre el mismo, la mejor forma es declararlos mediante el uso de constantes.

```
1    <?php
2    define (“USUARIO”, “administrador”);
3    define (“PASSWORD”, “12345678”);
4    ?>
```

En este capítulo hemos aprendido como declarar variables, asignarles un valor, modificarlo y mostrarlo en pantalla junto a cadenas de texto y como declarar constantes para el almacenamiento de datos que no van a cambiar en ningún momento del programa.

3.- Operadores en PHP

Los operadores son utilizados en programación para realizar cálculos aritméticos, comparaciones y condiciones.

Veamos cómo utilizarlos en PHP:

Operadores aritméticos

Realizan operaciones matemáticas.

Dentro de ellas encontramos:

- Suma: $a + $b
- Resta: $a – $b
- Multiplicación: $a * $b
- División: $a / $b
- Resto de una división: $a % $b
- Incremento: $a++
- Decremento: $b–

PHP establece prioridades entre los operadores aritméticos. Una de las prioridades es anteponer la multiplicación a la suma y a la resta.

Por ejemplo, si tenemos el siguiente código:

```
1    <?php
2    $num1 = 5;
3    $num2 = 7;
4    $num3 = 2;
5    echo $num1 + $num2 * $num3;
6    ?>
```

Podemos pensar que primero sumará 5 + 7 y luego multiplicará por 2. Sin embargo lo que hará es multiplicar 7 x 2 y después sumar 5.

La operación se interpretaría como: 5 + (7 x 2) = 19

Si lo que queremos es que primero sume y después multiplique, deberemos encerrar la suma entre paréntesis, para que de prioridad a la suma frente a la multiplicación.

```
1    <?php
2    $num1 = 5;
3    $num2 = 7;
4    $num3 = 2;
5    echo ($num1 + $num2) * $num3;
6    ?>
```

Ahora el resultado será (5 + 7) x 2 = 24

Incremento / Decremento:

```
1    <?php
2    $num = 5;
3    $num++;
4    echo $num;
5    ?>
```

En este ejemplo, la variable $num, valdría 6 a la finalización del programa.

Decremento (–) realiza la misma operación pero restando una unidad al valor que ya tuviera la variable.

Estos operadores son muy utilizados a la hora de programar contadores con la ayuda de bucles. Veremos ejemplos prácticos más adelante.

Operadores de comparación

- Menor que: $a < $b
- Mayor que: $a > $b
- Menor o igual que: $a <= $b
- Mayor o igual que: $a >= $b
- Igual que: $a == $b

- Distinto que $a != $b

Estos operadores son muy utilizados a la hora de realizar condiciones, que veremos en el apartado de **Estructuras condicionales (if).**

Uso del símbolo igual (=)

PHP utiliza igual simple (=) o igual doble (==) dependiendo de si la acción es "asignar" o "comparar".

Asignación de valores a una variable:

$numero = 1 //Le estamos asignando el valor 5 a la variable $numero.

Comparación de dos variables:

$num1 == $num2 //Estamos comparando si los valores de ambas variables coinciden.

Operadores de cadenas

$a . $b = concatena cadenas de caracteres formando una única cadena.

```
1   <?php
2   $cadena1 = "Hoy es";
3   $cadena2 = " lunes";
4   echo $cadena1 . $cadena2;
5   ?>
```

Operadores lógicos

- AND o && = Verdadero si ambos son verdadero.
- OR o || = Verdadero si alguno de los dos es verdadero.
- XOR = Verdadero si sólo uno de los dos es verdadero.
- ! = Negación

4.- Conversiones de datos en PHP

4.1- Conversión automática de tipos

PHP es muy flexible en el manejo de los tipos de datos, tan flexible es que nos permite sumar una cadena de caracteres a un número. Por ejemplo:

```
1$varN = 1;
2$varC = '2 flores';
3$varC = $varC + $varN; // el resultado será 3
```

Si la variable de caracteres no comenzase con números sumaría cero, por ejemplo:

```
1$varN = 1;
2$varC = 'flores';
3$varC = $varC + $varN; // el resultado será 1
```

PHP evalúa la operación que se utiliza (por ejemplo, suma, resta, concatenación, multiplicación, división) y el tipo de datos de los operandos. Según el resultado de la evaluación adaptará los operandos para poder realizar la operación lo mejor posible. Así es como convierte internamente '2 flores' en el entero 2 para llevar a cabo la operación.

En una operación aritmética con cadenas intentará obtener el valor numérico de las cadenas. En una operación entre variables enteros y coma flotante dará como resultado una variable de coma flotante.

```
1$varN = 1;
2$varC = 1.35;
3$varN = $varC + $varN; // el resultado será 2.35
```

Esta conversión automática también sucede cuando una variable numérica se quiere utilizar como cadena de caracteres. Una operación de concatenación de cadenas (.) necesita operandos de cadena de caracteres, por lo cual, si queremos concatenar una cadena de caracteres con una o más variables numéricas lo que sucederá es que PHP convertirá a estas últimas a cadenas de caracteres.

```
1$varN = 1;
2$varC = "4 flores";
3$varN = $varN . $varC; // el resultado será 14 flores
```

4.2- Conversión forzada de tipos

La conversión automática que realiza PHP no siempre es la que más nos conviene. Pero PHP también nos da una herramienta para convertir el tipo de una variable a otro tipo concreto. Las conversiones posibles son las siguientes:

- (int), (integer): Fuerza la conversión a entero
- (real), (double), (float): Fuerza la conversión a coma flotante
- (string): Fuerza la conversión a cadena de caracteres
- (array): Fuerza la conversión a matriz
- (object): Fuerza la conversión a un objeto

```
1 $var = (integer) 250.75;
2 // $var queda con el valor 250 -trunca sin redondeo
3 $var = (real) "250.75 es una cadena";
4 // $var queda con el valor 250.75
5
5 $a = array('a', 'b', 'c');
6 echo $a[0] . "<br>"; // devuelve 'a'
```

```
7 echo $a[1] . "<br>"; // devuelve 'b'
8
9 $país = array('nombre' => 'UK', 'capital' => 'Londres');
10 echo $país['capital']; // devuelve 'Londres'
11
```

5.- include()/require()

Es muy común separar el código de un programa PHP en diferentes archivos y luego ir llamando a unos u otros según sea necesario para una determinada solicitud. Para ello se pueden utilizar las siguientes funciones:

- require(«ruta/archivo.php»)
- include(«ruta/archivo.php»)

Todas importan código desde dicho archivo PHP, pero, **¿qué diferencia hay entre utilizar include() o require()?**

- require() vs include()

Ambas funciones importan o insertan el código contenido en el archivo.php dentro de otro. La diferencia puede deducirse de su nombre:

require() establece que el código del archivo invocado es requerido, es decir, obligatorio para el funcionamiento del programa. Por ello, si el archivo especificado en la función require() no se encuentra saltará un error «*PHP Fatal error*» y el programa PHP se detendrá.

include(), por el contrario, si no se encuentra dicho código, saltará un error tipo «*Warning*» y el programa seguirá ejecutándose (aunque como consecuencia de no incluirse el código puede que no funcione correctamente, o sí, depende de la situación).

Por ejemplo, si tenemos un script PHP para mostrar artículos sería mejor utilizar require(), para cargar el código que realiza la consulta a la base de datos y recibe los datos del artículo a mostrar, mientras que podemos utilizar include() para invocar el archivo que contiene código html como puede ser el pie de la página web u otra parte de la plantilla.