Desarrollo Web en entorno servidor

Unidad 2:
Conceptos básicos de PHP

Tabla de contenido

1. PHP y HTML. Código incrustado 2
2. Sintaxis de PHP 2
3. Comentarios en PHP 3
4. Variables y tipos de datos 3
   - 4.1. Declaración de variables 4
   - 4.2. Variables no inicializadas 5
   - 4.3. Asignación por copia y por referencia 5
   - 4.4. Tipos de datos escalares 6
5. Operadores 8
   - 5.1. Operadores de comparación 8
   - 5.2. Operadores aritméticos. 8
   - 5.3. Operadores lógicos 8
   - 5.4. Otros operadores 9
6. Estructuras de control 9
   - 6.1. Estructuras condicionales 9
   - 6.2. Estructuras de repetición 10

1. PHP y HTML. Código incrustado

PHP es el lenguaje de programación para desarrollo web en el lado del servidor. Desde su aparción en 1994 ha tenido gran aceptación y se puede decir que es lenguaje más extendido para el desarrollo en el lado del servidor. Aunque no es la única opción, lo normal es que el intérprete de PHP sea un módulo del servidor web.

El lenguaje PHP es flexible y permite programar pequeños scripts con rapidez. Comparado con lenguajes como java, requiere escribir menos código y, en general, resulta menos engorroso. La sintaxis de los elementos básicos es bastante parecida a la de lenguajes muy extendidos. Como Java y C Por estos motivos, es un lenguaje rápido de aprender para las personas con alguna experiencia en programación.

En el desarrollo web es muy habitual utilizar PHP incrustado dentro ficheros HTML. El código PHP se introduce dentro del HTML utilizando la etiqueta **<?php** para abrir el de PHP y la etiqueta **?>** para cerrarlo.

Existen varias formas incluir contenido en la página web a partir del resultado de la ejecución de código PHP. La forma más sencilla es usando `echo`, que no devuelve nada (`void`), y genera como salida el texto de los parámetros que recibe.

```
void echo (string $arg1, ...);
```
[1]

Otra posibilidad es `print`, que funciona de forma similar. La diferencia más importante entre `print` y `echo`, es que `print` sólo puede recibir un parámetro y devuelve siempre 1.

```
int print (string $arg);
```

Tanto `print` como `echo` no son realmente funciones, por lo que no es obligatorio que pongas paréntesis cuando las utilices.

Ejercicio 1
Crea un fichero HTML que tenga por título "Hola mundo" y muestre, mediante PHP, "Hola mundo"

2. Sintaxis de PHP

El elemento básico en PHP es el bloque. Un bloque PHP está delimitado por las etiquetas respondientes y contiene una serie de sentencias separadas por punto y coma. Por otra parte, cuando un fichero contiene solo PHP se recomienda no cerrar la etiqueta del último bloque. Puede dar problemas si la respuesta del servidor involucra varios ficheros.

```
<?php
      Sentencia 1;
      Sentencia N;
?>
```

[1] La definición de las funciones se encuentra recogida en la documentación de php. En particular, la definición de la función echo es accesible vía la siguiente página web: https://www.php.net/manual/es/function.echo.php

3. Comentarios en PHP

Como en cualquier otro lenguaje de programación en PHP se pueden utilizar comentarios para mejorar la compresión de nuestro código añadiendo desde descripciones básicas de las sentencias utilizadas hasta la generación de documentación externa.

Podemos utilizar dos tipos de comentarios:

Comentarios de bloque: van limitadas entre "/*" y "*/"

Comentarios de línea: comienzan por "//" o por "#"

Veamos un ejemplo con el código HTML que nos generaba un número aleatorio entre 0 y 100

```
1:  <!DOCTYPE html>
2:  <html>
3:    <head>
4:      <title>Número aleatorio</title>
5:    </head>
6:    <body>
7:    <?php
8:      /* Aquí comienzan las sentencias de PHP.
9:     En este ejemplo, mostramos un número aleatorio entre 0 y 100
10:    */
11:       echo rand(0, 100); //echo nos permite mostrar el número aleatorio generado.
12:   // aquí termina nuestro bloque de sentencias
13:    ?>
14:    </body>
15:  </html>
```

4. Variables y tipos de datos

Una de las características principales de PHP es que es un lenguaje **no fuertemente tipado**. Esto quiere decir que no es necesario indicar el tipo de dato al declarar una variable. De hecho, las variables no se declaran, se crean la primera vez que se les asigna un valor. El tipo de dato depende del valor con que se inicialicen.

Esto agiliza la escritura de programas, pero también tiene inconvenientes. Si no se presta atención, puede dar lugar a código de baja de calidad y, a medida que las aplicaciones crecen, pueden darse errores difíciles de depurar.

PHP admite diez tipos primitivos:

a) Cuatro tipos escalares:

- **boolean**: Admite dos valores: True (verdadero o distinto de 0) y False (falso o igual a 0)
- **integer**: Admite cualqueir número entero
- **float**: número de punto flotante, también conocido como double)
- **string**: cadenas de caracteres

b) Cuatro tipos compuestos:

- **array**: conjuntos ordenados de elementos de cualquier tipo. Se denominan vectores o matrices.
- **object**: objeto perteneciente a una clase. Están vinculados a la programación orienta a objetos.
- **callable**: también conocidos como *callbacks* o retrollamadas. Están vinculados con las funciones y los metodos de los objetos.
- **iterable**: Pseudo tipo de variable que admite cualquier array u objeto que se puede recorrer desde su primer elemento al último.

c) Dos tipos especiales:

- **resource**: contiene una referencia a un recurso externo, los cuales son creados y usados por funciones especiales.
- **NULL**: representa una variable sin valor. La variable está declarada pero no inicializada.

4.1. Declaración de variables

En PHP los identificadores de las variables van siempre precedidos por el carácter '**$**'. El identificador de la variable debe comenzar por una letra o un guion bajo y puede estar formado por números, letras y guiones bajos.

Para inicializar una variable solo hay que asignarle un valor:

```
$nombre = valor;
```

PHP las variables en el momento que se encuentra por primera vez `$nombre`, esto es consecuencia de ser un lenguaje **no fuertemente tipado.** En de otros lenguajes de programación en primer lugar se declara la variable asignándole un tipo de dato y, después en una sentencia posterior, se inicializa dicha variable con un valor inicial..

Por ejemplo, esta sentencia declara la variable `$entero`, que será de tipo `integer` porque se inicializa con un entero.

```
<?php
$entero = 4;
```

También es posible cambiar el tipo de dato de una variable simplemente asignándole un valor de otro tipo de dato. La función `gettype()` devuelve el tipo de dato de una variable.

Ejercicio 2

¿Qué salida obtendríamos si ejecutásemos el siguiente código? ¿Por qué ocurre el error en la línea 12?

```
 1:  <?php
 2:  $entero = 4;
 3:  echo gettype($entero);
 4:  echo "<br>";
 5:  $entero = “Hola”;
 6:  echo gettype($entero);
 7:  echo "<br>";
 8:  $a = print "a";
 9:  echo gettype($a);
10:  echo "<br>";
```

```
11:  $b = echo "b";
12:  echo gettype($a);
```

4.2. Variables no inicializadas

Si se intenta utilizar una variable antes de asignarle un valor, se genera un error de tipo E_NOTICE, pero no se interrumpe la ejecución del script. Si una variable no inicializada aparece dentro de una expresión, dicha expresión se calcula tomando el valor por defecto para ese de dato. En el ejemplo `no_init.php`, se puede ver lo que ocurre al utilizar una variable no inicializada dentro de una expresión.

```
1:  <?php
2:      $var1 = 100;
3:      $var3 = 100 + $var2; // $var2 no existe, así que se toma como 0
4:      echo "$var3 <br>";   // muestra 100
5:      $var3 = 100 * $var2; // $var2 no existe, así que se toma como 0
6:      echo "$var3 <br>";   // muestra 0
```

En la línea 3, se intenta sumar una variable no inicializada. Como el valor por defecto para un entero es 0, el resultado de la suma es 100. En la línea 5, se intenta multiplicar por una variable no inicializada y, al multiplicar por 0, el resultado es 0. La salida muestra un mensaje de error por cada intento de utilización de una variable

Warning: Undefined variable $var2 in **C:\xampp\htdocs\DWES\no_init.php** on line **3**
100

Warning: Undefined variable $var2 in **C:\xampp\htdocs\DWES\no_init.php** on line **5**
0

4.3. Asignación por copia y por referencia

La asignación de variables se realiza mediante copia. Es decir, si hacemos:

```
$a = $b
```

Se crea una nueva variable a y se le asigna el valor que tenga b. Las variables a y b representan posiciones diferentes de memoria, aunque tengan el mismo valor después de la asignación. También es posible definir una referencia a una variable utilizando el operador ampersand:

```
$a = &$b
```

En este caso $a no es una nueva variable con el valor de $b. Por el contrario, $a apunta a la misma dirección de memoria que $b, de manera que $a y $b son en realidad dos nombres para el mismo dato. En el ejemplo `copia.php` se muestra que, al cambiar valor de una referencia, se modifica también el de la variable referenciada.

```
1:  <?php
2:      $var1 = 100;
3:      $var2 = &$var1;  // asignación por referencia
4:      $var3 = $var1;   // asignación por copia
```

```
5:      echo "$var2<br>";// muestra 100
6:      $var2 = 300;      // cambia el valor de $var2
7:      echo "$var1<br>";// $var1 también cambia
8:      $var3 = 400;      // este cambio no afecta a $var1
9:      echo $var1;
```

Por tanto, la salida del ejemplo será:

100
300
300

4.4. Tipos de datos escalares

PHP ofrece cuatro tipos de datos escalares: `integer, float, boolean y string`.

A) Integer y float

Para representar números enteros se usa el tipo de dato `integer`. El tamaño y lo valores máximo y mínimo de un entero dependen de la plataforma, y se pueden conocer mediante las sentencias `PHP_INT_SIZE, PHP_INT_MAX y PHP_INT_MIN`, respectivamente.

Para números reales, se utiliza el tipo `float`. El tamaño también depende de la plataforma, para suele ofrecer una precisión de 14 decimales. En cualquier caso, la precisión de los `float` presenta los mismos problemas que en otros lenguajes y los redondeos pueden dar sorpresas.

Recordemos que los números reales se pueden representar mediante notación decimal o mediante notación exponencial. Por ejemplo, el número escrito en notación decimal "-0.03" se escribe como "-3E-2" en notación exponencial. Esta última expresión se lee "-3 por diez elevado a –2". La letra E puede ser mayúscula o minúscula.

La conversión entre `integer` y `float` es automática. Si se recibe un `float` cuando se esperaba un `integer` se trunca. Si al realizar una operación sobre un entero el resultado superase los valores limites o tiene decimales, se convierte a `float`. También se pueden utilizar los operadores de conversión `(int) o (float)`.

El ejemplo `tipos_numericos.php` muestra las diferentes opciones y algunas operaciones entre ellos.

```
1:  <?php
 2:      echo PHP_INT_SIZE.'<br>';
 3:      echo PHP_INT_MIN.'<br>';
 4:      echo PHP_INT_MAX.'<br>';
 5:      $a = 3/2;   // la división entre enteros no da problemas
 6:      echo $a.'<br>'; // 1.5
 7:      $b = 7.5;
 8:      $a = (int) $b; // cambias a entero a int
 9:      echo $a.'<br>'; // 7, se trunca
10:      $b = 7e2; // notación científica
11:      echo $b.'<br>';
12:      $b = 7E2; // no es sensible a mayúsculas y minúsculas
13:      echo $b.'<br>';
14:      $b = 0.1E-2; // numeros real en notación exponencial
```

```
15:     echo $b.'<br>';
16:      $b = 0.001; // el mismo número real en notación decimal
17:     echo $b.'<br>';
18:     $a = 0.99999999999999999;
19:     echo floor($a). '<br>'; // devuelve 1
20:     $a = 0.9999999999999999;
21:     echo floor($a); // devuelve 0
```

B) Cadenas

El tipo de dato `string` permite almacenar cadenas de caracteres. Para delimitar una cadena es posible utilizar comillas simples o dobles, pero hay una diferencia. Si se utilizan **comillas dobles**, las variables que aparezcan dentro de la cadena se sustituirán por su valor. Mientras que las comillas simples mostraran el contenido literal de la cadena.

En el siguiente ejemplo, `comillas.php`, tenemos cada uno de los casos.

```
1:  <HTML>
 2:    <HEAD>
 3:      <TITLE>Variables con Cadenas de caracteres</TITLE>
 4:    </HEAD>
 5:    <BODY>
 6:      <CENTER> <- Cambiar, está obsoleto
 7:      <H2>Trabajando con Cadenas de caracteres</H2>
 8:      <?php
 9:      $lenguaje="PHP";
10:      $ver="v8";
11:      echo "<B>Estamos trabajando con $lenguaje ($ver) </B><BR><BR>";
12:      echo 'La variable $lenguaje contiene: ';
13:      echo $lenguaje;
14:      echo "<BR>";
15:      echo 'La variable $ver contiene: ';
16:      echo $ver;
17:      ?>
18:      </CENTER> <- Cambiar, está obsoleto
19:    </BODY>
20:  </HTML>
```

Es importante tener en cuenta a la hora de formatear la salida que esta va a ser procesada como HTML por un navegador web, es decir, los saltos de línea se ignoran y los espacios en blanco consecutivos colapsan[2], además, los caracteres de escape como `'\n'`, `'\r'` y `'\t'` son ignorados por el navegador. Para introducir un salto de línea la opción más sencilla es incluir la etiqueta de salto de línea de HTML (`"<br>"`) en la salida, como se puede ver en los ejemplos anteriores.

C) Booleanos

Este tipo de dato es para variables booleanas. Solo pueden tomar los valores TRUE y FALSE, verdadero y falso. Este es el tipo de dato que se obtiene, entre otros casos, como resultado de los operadores de comparación y se utiliza en sentencias condicionales y bucles.

[2] Si queremos introducir varios espacios en blanco seguidos en HTML haremos uso del código ` `

Cuando se espera un valor booleano y se recibe otro tipo de dato, se aplican las siguientes reglas de conversión:

a) `integer`
   Si es 0 se toma FALSE, en otro caso como TRUE

b) `float`
   Si es 0.0 se toma FALSE, en otro caso como TRUE

c) `string`
   Si es una cadena vacía o "0", se toma como FALSE, en otro caso como TRUE

d) `Variables no inicializadas o nulas`
   Se toman como FALSE

e) `array`
   Si no tiene elementos se toma FALSE, en otro caso como TRUE

5. Operadores

PHP cuenta con los operadores habituales para operaciones aritméticas, lógicas, de manipulación de cadenas y demás. Podemos encontrar el listado y la definición completa de cada uno de los operadores en la documentación de php, a través del siguiente enlace: PHP: Operadores - Manual

5.1. Operadores de comparación.

A mayores de los **operadores de comparación** habituales (“>”, “<”, “<=”,“>=”, “<>”, “!=”) cabe señalar los operadores `"==="` y `"!=="`, llamados **Idéntico** y **No Idéntico**, que no están presentes en todos los lenguajes.

El operador **Idéntico** se usa para comparar dos expresiones y se evalúa como verdadero cuando las dos expresiones tienen el mismo valor y además el mismo tipo de dato. Se diferencia del operado **Igual**,`"=="`, en que este, cuando las expresiones no tienen el mismo tipo de dato, intenta convertirlas antes de compararlas.

El operador Idéntico es útil para evitar sorpresas inesperadas en la conversión de datos. En el ejemplo `identico.php` se puede comprobar la diferencia.

5.2. Operadores aritméticos.

Los **operadores aritméticos** de las operaciones básicas son lo habituales en los lenguajes de programación: suma “+”, resta “-”, multiplicación “*”, división “/”. A mayores contamos con otros operadores que nos permite realizar operaciones más complejas, como son los operadores módulo “%” y potencia “**”. El operador módulo nos devuelve el resto de la división entera de dos números.

5.3. Operadores lógicos.

Los **operadores lógicos**, también llamados operadores booleanos, se utilizan conjuntamente con expresiones que devuelven valores lógicos. Con estos operadores se pueden combinar varias variables o condiciones y evaluarlas en una sola expresión. La sintaxis es la siguiente:

OPERADOR LÓGICO	EJEMPLO	DEVUELVE **TRUE** CUANDO
&&	$a && $b	$a y $b son ambos TRUE
and	$a and $b	
||	$a || $b	$a o $b son TRUE
or	$a or $b	
!	! $a	$a es FALSE, niega el valor lógico de la variable
xor	$a xor $b	$a es TRUE o $b es TRUE, pero no ambas a la vez

5.4. Otros operadores

A lo largo de los ejemplos anteriores hemos podido ver algunos otros operadores:

- Operador de asignación por valor, "=".
- Operador de asignación por referencia, "=&".
- Operador de concatenación de cadenas, ". " (ejemplo tipos_numericos.php)

En la documentación de PHP, PHP: Operadores - Manual, nos encontramos con operadores bit a bit, de control de errores, de incremento/decremento, de arrays, etc. A medida que avancemos en el curso iremos analizando aquellos otros operadores que fueran necesarios.

6. Estructuras de control

PHP cuenta con las estructuras de control habituales en la programación estructurada para realizar sentencias condicionales y de repetición. La sintaxis es muy parecida a la de Java o C.

6.1. Estructuras condicionales

Las estructuras condicionales de PHP son `if`, `if-else`, `if-elseif` y `switch`. La sentencia condicional más sencilla es la estructura `if`. La sintaxis general es

```
if(condición){
      Sentencia 1;
      Sentencia N;
}
```

La condición es una expresión que se evalúa a verdadero a falso, siguiendo las normas de conversión a `boolean` si es necesario. Si se cumple la condición, es decir, si la condición queda evaluada como TRUE, se ejecutan las sentencias. Si no se cumple, FALSE, no se ejecuta.

Cuando se utiliza un `if` se puede añadir un `else`. Las sentencias dentro del `else` solo se ejecutan cuando la condición del `if` no se cumple. La sintaxis general es

```
if(condición){
      Sentencia 1;
      Sentencia N;
}else {
      Sentencia 1;
```

```
        Sentencia N;
    }
```

Del mismo modo se pueden anidar varias sentencias condicionales mediante `elseif`, que es equivalente a `else if`. La sintaxis general es:

```
    if(condición 1){
        Sentencia 1;
        Sentencia N;
    }elseif(condición 2){
        Sentencia 1;
        Sentencia N;
    } elseif(condición 3){
        Sentencia 1;
        Sentencia N;
    } else {
        Sentencia 1;
        Sentencia N;
    }
```

La primera condición que se cumpla es la que se ejecuta. Si no se cumple ninguna, se ejecuta el `else` final (si lo hay). Podemos ver un ejemplo en `if_elseif.php`

Para agrupar varios `if` puede ser útil la estructura `switch`, también habitual en otros lenguajes. El ejemplo `switch.php` es equivalente al anterior, pero más fácil de leer.

Según el valor de `$var`, se ejecutará un caso u otro. La sección `default` se ejecuta cuando no se da ninguno de los otros casos.

6.2. Estructuras de repetición

Las estructuras de repetición o bucles sirven para repetir un conjunto de instrucción mientras se dé una condición. PHP cuenta con las estructuras habituales: `for, while y do-while`, que tienen la misma sintaxis que en Java o C

Para **el bucle `for`**, la sintaxis es

```
for(instrucción de inicialización, condición, instrucción de iteración) {
        Sentencia 1;
        Sentencia N;
}
```

La `instrucción de inicialización` se realiza una única vez al llegar al bucle.

Las `sentencias` dentro del cuerpo del bucle se repetirán mientras se cumpla la condición. Después de ejecutar la instrucción de inicialización se evalúa la condición. Si se cumple, se ejecutan las sentencias del bucle y la instrucción de iteración, después se vuelve a comprobar la condición.

El proceso se repite hasta que la condición deja de cumplirse y a partir de ahí la ejecución continúa con las instrucciones que estén después del bucle. El ejemplo `bucle_for.php` muestra un bucle `for` básico que se repite cinco veces. La variable `$i` se inicializa a cero y su valor se incrementa en

uno tras cada iteración. El bucle se repite mientras `$i` valga menos que cinco, es decir, de cero a cuatro.

El bucle `while` también sigue la sintaxis habitual:

```
while (condición) {
        Sentencia 1;
        Sentencia N;
}
```

Igual que con el `for`, las instrucciones se ejecutan mientras se cumpla la condición. El `while` no cuenta con instrucciones de inicialización o iteración, pero se pueden añadir antes y al final del bucle, respectivamente. El ejemplo `bucle_while.php` tiene la misma salida que el anterior.

El **bucle `do-while`** es similar, pero la condición se evalúa después de ejecutar las sentencias del bucle.

```
do{
        Sentencia 1;
        Sentencia N;
} while (condición)
```

Tenemos un ejemplo en `bucle_dowhile.php`, el cual es equivalente a los anteriores.

Es **importante** remarcar que el bucle `do-while` se diferencia de los anteriores en que las sentencias dentro se ejecutan por lo menos una vez, como se puede ver en el ejemplo **bucle_diferencia_while_dowhile.php**

Dentro de un bucle es posible usar **`las sentencias break y continue`**. La primera sirve para salir del bucle o switch en el que aparece. En el ejemplo `break.php`, el bucle acaba antes de llegar a cinco porque al llegar a tres se ejecuta el `break`.

En PHP la sentencia **`break`** admite un número, que representa el número de niveles de anidación de los que debe salir. Así se puede salir de un **bucle anidado** utilizando una única secuencia `break` como se puede ver en `break_anidado.php`.

En el primer bucle anidado se utiliza un `break` sin pasarle ningún número, que es lo mismo que pasarle un uno, y por tanto solo sale del bucle interior. El bucle exterior se repetirá tres veces y el interior seis, dos por cada vez que se ejecute el exterior, con los valores de `$j=0` y `$j=1`. En el segundo bucle anidado, al ejecutarse la línea 16 se acaban los dos bucles. Esto ocurre en la primera iteración del bucle externo y la segunda del interno.

Esto también se aplica a la sentencia condicional `switch`. Por ejemplo, si un `switch` está dentro de un bucle, utilizando `break 2` se sale de ambos. Lo podemos ver en el ejemplo `break_switch.php`.

La sentencia **`continue`** fuerza una nueva iteración del bucle. Las instrucciones que estén dentro del bucle, pero después de `continue` no se ejecutan y, si se cumple la condición del bucle, se ejecuta una nueva iteración. Si se trata de un bucle `for`, se ejecutan las instrucciones de autoincremento antes de evaluar la condición.

En el ejemplo `continue.php` se puede observar el funcionamiento de **`continue`**.