



## **UD2. Introducción a PHP**

### **Desarrollo Web en Entorno Servidor**

**Profesora: Silvia Vilar Pérez**

**curso 2024-2025**

# Contenidos

- Características PHP.
- Etiquetas para inserción de código.
- Uso de directivas. PHP.ini
- Ámbito de las variables.
- Constantes.
- Variables Superglobales
- Variables variables
- Tipos de datos. Conversiones entre tipos de datos.
- Sintaxis básica de PHP

# Características PHP

- El código PHP está embebido en documentos HTML, para introducir dinamismo fácilmente a un sitio web.
- El intérprete PHP ignora el texto del fichero HTML hasta que encuentra una etiqueta de inicio del bloque de código PHP embebido.
- Como PHP se ejecuta del lado del servidor sólo puede tener acceso a los datos del propio servidor.
  - ✓ No puede acceder a los recursos del cliente
  - ✓ No puede saber qué hora es en el cliente
  - ✓ No puede acceder a los archivos del cliente  
(Con la excepción de las Cookies)

# Etiquetas para inserción de código

Hay varias formas para delimitar los bloques de código PHP:

1. La opción que asegura portabilidad (**recomendada**):

**<?php** *codigo php* **?>**

2. Usando las short tags o etiquetas de formato corto:

**Requiere activar directiva short\_open\_tag=on**

**<? código php ?>**

3. Podemos asociar el lenguaje a etiquetas de scripting (**Eliminada** desde la versión 7.0):

**<script language="PHP">** *codigo php* **</script>**

4. Usando las etiquetas de ASP (**Eliminada** desde la versión 7.0): **Requiere activar directiva asp\_tags=on**

**<% código php %>**

**NOTA:** el delimitador **<?=>** es abreviatura de **<?php echo**

# Etiquetas para inserción de código

Dentro de dichas etiquetas aplicaremos la sintaxis de PHP. Las sentencias de código finalizan con “**;**”

- Los espacios, tabulaciones, intros, etc en el código embebido no tienen otro efecto que **mejorar legibilidad**
- Los scripts se pueden incrustar en cualquier sección del HTML y puede haber un número indefinido en el fichero.
- Además, debemos guardarlo con la extensión **.php** para que el servidor use el intérprete de PHP. Éste sustituye el script por el resultado de su ejecución, incluyendo las etiquetas de inicio y fin.

# Uso de directivas – php.ini

Para configurar el entorno del Servidor para PHP, se modifica el fichero **php.ini** (intérprete de PHP)

- El fichero **php.ini** indica una serie de valores que determinan el comportamiento del intérprete PHP
- Se encuentra ubicado en el directorio raíz bajo los nombres **php.ini-development** y **php.ini-production**
- En Ubuntu lo encontramos en /etc/php/8.3/cli/php.ini (8.3 es la versión instalada, puede variar)
- Las instrucciones del fichero se denominan **Directivas**

Listado de directivas de php.ini

<https://www.php.net/manual/es/ini.list.php>



# Uso de directivas – php.ini

- Las directivas se forman por una pareja de clave-valor.
- Las directivas que comienzan por **;** están comentadas y son ignoradas por el motor del intérprete.
- Para indicar las rutas dentro del fichero se utilizan los formatos:
  - C:\directorio\directorio**
  - \directorio\directorio**
  - /directorio/directorio**
- El fichero php.ini se lee cada vez que se arranca el servidor web.
- El servidor busca el fichero php.ini por este orden:
  - ✓ En el propio directorio de php.
  - ✓ En la ruta definida como variable de entorno.
  - ✓ En el directorio del sistema (Ej: C:\Windows) que es la opción recomendada.

# Ámbito de las variables

Es el contexto en el que se puede acceder a una variable.

- En PHP existen variables **locales** y **globales**.
- Las variables se definen como globales precediéndolas de la palabra **global**.  
**global** \$var=5;
- También las podemos definir como globales asignándolas a la matriz superglobal **\$GLOBALS**.  
**\$GLOBALS**[\$var]=5;



# Ámbito de las variables

Si queremos mantener el valor de una variable local en las sucesivas llamadas a la función hay que definirla como **static**

Salida del código:

01

12

23

```
<?php
function test(){
    static $a = 0;
    echo $a;
    $a++;
    echo $a;
}
test();
echo "\n";
test();
echo "\n";
test();
?>
```

# Ámbito de las variables

```
<?php
function PruebaSinGlobal(){
    $var++;
    echo "Prueba sin global. \$var: ".$var."\n";
}
function PruebaConGlobal(){
    global $var;
    $var++;
    echo "Prueba con global. \$var: ".$var."\n";
}
function PruebaConGlobals(){
    $GLOBALS["var"]++;
    echo "Prueba con GLOBALS. \$var: ".$GLOBALS["var"]."\n";
}
//variable global
$var=20;
PruebaSinGlobal();
PruebaConGlobal();
PruebaConGlobals();
?>
```

## Resultado:

```
Prueba sin global. $var: 1
Prueba con global. $var: 21
Prueba con GLOBALS. $var: 22
```

# Constantes

- Una constante es un identificador de un dato que no cambia de valor durante toda la ejecución de un programa.
- Las constantes no se asignan con el operador **=**, sino con la función **define**:

```
define(nombre_constante_entre_comillas, dato_constante);  
define ("PI", 3.1416);  
print PI;
```

- No llevan \$ delante
- La función `defined("PI")` devuelve TRUE si existe la constante.
- Son siempre globales por defecto.
- Sólo se pueden definir constantes de los tipos escalares (boolean, integer, double, string)

# Constantes Predefinidas

Dependen de las extensiones que se hayan cargado en el servidor, aunque hay constantes predefinidas que siempre están presentes:

- `PHP_VERSION`: Indica la versión de PHP que se está utilizando.
- `PHP_OS`: Nombre del sistema operativo que ejecuta PHP.
- `TRUE`
- `FALSE`
- `E_ERROR`: Indica los errores de interpretación que no se pueden recuperar.
- `E_PARSE`: Indica errores de sintaxis que no se pueden recuperar.
- `E_ALL`: Representa a todas las constantes que empiezan por `E_`.

# Utilización de las variables

- Restricciones sobre las variables:
  - ✓ Deben comenzar por **\$**
  - ✓ Seguidamente debe haber una letra (may/min) o guión bajo (`_`)
  - ✓ El resto de caracteres pueden ser números, letras guiones bajos
- PHP es case sensitive
- Si una variable se compone de varias palabras, se aconseja escribirla en minúsculas excepto el inicio de la siguiente palabra (camelCase)
- Las **variables predefinidas** siguen el patrón **`$_VARIABLE`** por lo que se desaconseja usar este mismo patrón
- Las variables o cadenas se pueden concatenar usando el punto (`.`)

# Variables

Además de las variables definidas por el programador, existen gran cantidad de *variables predefinidas* que se pueden usar libremente:

- ✓ ***Variables de entorno***: Variables que el servidor pone a disposición de PHP e indirectamente del programador
- ✓ ***Variables de PHP***: Variables predefinidas que pertenecen al intérprete PHP y que éste pone a disposición del programador.
- A partir de PHP 4 se incluyen matrices **superglobales** que centralizan todas las variables predefinidas.  
\$GLOBALS, \$\_SERVER, \$\_GET, \$\_POST, \$\_COOKIE, \$\_FILES, \$\_ENV, \$\_REQUEST, \$\_SESSION



# Variables Superglobales

- **\$\_GET**: lleva los datos de forma "visible" al cliente (navegador web). El **medio de envío es la URL**. Para recoger los datos que llegan en la url se usa **\$\_GET**.

Ejemplo: `www.midominio.com/action.php?nombre=silvia&apellidos1=vilar`

- **\$\_POST**: consiste en datos "ocultos" (porque el cliente no los ve) **enviados por un formulario** cuyo método de envío es POST. Ideal para formularios. Para recoger los datos que llegan por este método se usa **\$\_POST**.
- **\$\_REQUEST**: Con la variable **\$\_REQUEST** **recuperaremos los datos** de los formularios enviados tanto por **GET como por POST**.

# Variables Superglobales \$\_GET

Cadena de caracteres añadida a la URL:

`http://example.com/?nombre=Silvia&apellido=Vilar`

- Código PHP:

```
<?php
echo '¡Hola ' . $_GET["nombre"] . ' ' . $_GET["apellido"]!';
echo '¡Hola ' . $_REQUEST["nombre"] . ' ' .
$_REQUEST["apellido"]!';
?>
```

- Resultado:  
¡Hola Silvia Vilar!

# Variables Superglobales \$\_POST y \$\_REQUEST

A través de un formulario:

- Formulario HTML:

```
<form action="ejemplo.php" method="POST">  
  Nombre usuario: <input type="text" name="username" /><br />  
  Email: <input type="text" name="email" /><br />  
  <input type="submit" name="btnEnviar" value="Enviar" />  
</form>
```

- Código PHP:

```
<?php  
echo "¡Hola " . $_POST['username'] . "!";  
echo "¡Hola " . $_REQUEST['username'] . "!";  
?>
```

- Resultado:

```
¡Hola Silvia!  
¡Hola Silvia!
```

# Variables Superglobales \$\_COOKIE

**\$\_COOKIE**: Recoge las variables pasadas al script actual mediante Cookies

- Ejemplo:

//creación de la cookie

```
<?php
    setcookie("nombre", 'Silvia', time()+3600);
?>
```

//obtención de la cookie

```
<?php
echo '¡Hola ' .
htmlspecialchars($_COOKIE["nombre"]) . '!';
?>
```

- Resultado

¡Hola Silvia!

# Variables Superglobales \$\_SERVER

**\$\_SERVER**: contiene información de cabeceras, rutas, ubicaciones, etc. del script

- Ejemplo:

```
<?php  
echo $_SERVER['SERVER_NAME'];  
?>
```

- Resultado

www.example.com

# Variables de Variables

Se pueden crear nombres de variables dinámicamente anteponiendo **\$\$** a una variable.

- La variable ***variable*** toma su nombre del valor de otra variable previamente declarada.
- Ejemplo:

```
<?php
$var = "uno";
$$var = "dos";
print ($var);    // produce el texto: "uno"
print ($uno);    // produce el texto: "dos"
print ($$var);   // produce el texto: "dos"
print (${ $var});
```

- A diferencia de las variables por referencia, se están creando dos variables distintas que ocupan direcciones de memoria distintas.



# Variables de Variables

Se pueden crear nombres de variables dinámicamente anteponiendo \$\$ a una variable.

- Otro Ejemplo:

```
<?PHP
    $mensaje_es="Hola";
    $mensaje_en="Hello";
    $idioma = "en";
    $mensaje = "mensaje_" . $idioma;
    print $$mensaje;
?>
```

Devolvería Hello

# Tipos de datos. Conversiones entre tipos

- PHP soporta los tipos de datos primitivos:
  - ✓ Números enteros
  - ✓ Números en coma flotante
  - ✓ Cadenas de caracteres
  - ✓ Booleanos
  - ✓ Objetos
  - ✓ Recursos
  - ✓ NULL
- El tipo de una variable no se suele especificar. Se decide en tiempo de ejecución en función del contexto y puede variar.

# Tipos de datos. Conversiones entre tipos

- Números enteros: Enteros positivos y negativos  
\$var = 20; \$var = -20; // asignación decimal  
\$var = 024; \$var = -024; // asignación octal  
\$var = 0x14; \$var = -0x14; // asignación hexadecimal
- Números en coma flotante: Permiten almacenar una parte fraccionaria.  
\$var = 260.78;  
\$var = 26078e-2;
- Booleanos: Pueden almacenar los valores True (1) y False (0).
- Recursos: Son valores especiales que hacen referencia a una información de estado o memoria de origen externo a PHP. Ejemplo: una conexión a BD

# Tipos de datos. Conversiones entre tipos

Funciones de interés:

- La función **gettype()** devuelve el tipo de una variable
- Las funciones **is\_type** comprueban si una variable es de un tipo dado:  
is\_array(), is\_bool(), is\_float(), is\_integer(),  
is\_null(), is\_numeric(), is\_object(), is\_resource(),  
is\_scalar(), is\_string()
- La función **var\_dump()** muestra el tipo y el valor de una variable. Es especialmente interesante con los arrays.

# Tipos de datos. Conversiones entre tipos

Tipo string:

- Comillas simples o dobles (en este caso interpreta secuencias de escape como caracteres especiales)
- Otra forma de inicializar cadenas es utilizar la sintaxis heredoc (desde PHP 4) y nowdoc (desde PHP 5.3) que utilizan el símbolo de documento incrustado (“<<<”) y un identificador para marcar el final del documento.

Ver:

<https://www.php.net/manual/es/language.types.string.php#language.types.string.syntax.heredoc>

<https://www.php.net/manual/es/language.types.string.php#language.types.string.syntax.nowdoc>

**NOTA:** Es importante no escribir ningún carácter, salvo \n, antes y después del identificador de cierre de la cadena.

- Acceso a un carácter de la cadena: \$inicial=\$nombre{0};

# Tipos de datos. Conversiones entre tipos

- Ejemplos de inicialización de cadenas:

```
$a = 9;
```

```
print 'a vale $a\n'; // muestra a vale $a\n
```

```
print "a vale $a\n"; // muestra a vale 9 y avanza una línea
```

```
print "<IMG SRC='logo.gif'>"; // muestra <IMG  
SRC='logo.gif'>
```

```
print "<IMG SRC=\"logo.gif\">"; //muestra <IMG  
SRC="logo.gif">
```

```
$nombre="Silvia";
```

```
$var = <<<xxx // Sintaxis heredoc con delimitador xxx
```

```
Esta es una cadena que termina al encontrarse xxx. $nombre  
xxx;
```

```
/* Resultado a mostrar: Esta es una cadena que termina al  
encontrarse xxx Silvia*/
```



# Conversiones automáticas de tipos de datos.

- PHP es muy flexible en el manejo de los tipos de datos.
- PHP evalúa la operación a realizar y el tipo de los operandos, y adapta los operandos para poder realizar la operación lo más correctamente posible.
- En operaciones entre enteros y coma flotante el resultado es un número en coma flotante
- Una concatenación de cadenas con una variable numérica hace que ésta última sea convertida a cadena.

```
$varN=1;
```

```
$varC='4 flores';
```

```
$varC=$varN.$varC; // el resultado es 14 flores
```

Silvia Vilar Pérez

27

Sin embargo, `$varN=$varC+$varN` el resultado sería 5

# Conversiones automáticas de tipos de datos.

Reglas automáticas de conversión de tipos:

- En operaciones lógicas, los datos NULL, 0, '0' y ' ' se consideran FALSE. Cualquier otro dato se considera TRUE (incluida la cadena 'FALSE').
- En operaciones aritméticas no unitarias las cadenas se intentan leer como números y, si no se puede, se convierten en 0, TRUE se convierte en 1, y FALSE se convierte en 0.
- En operaciones de comparación, si un operando es un número, el otro también se convertirá en un número. Sólo si ambos operandos son cadenas se compararán como cadena.
- En operaciones de cadenas de caracteres, NULL y FALSE se convierten en ' ', y TRUE se convierte en '1'.

# Conversión forzada de tipos de datos.

- La conversión automática que realiza PHP no siempre es lo que queremos obtener.
- PHP permite otras conversiones implícitas de tipos :
  - (int) : 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.

# Sintaxis básica PHP

- Comentarios (como en C):
  - ✓ `/* ... */` varias líneas
  - ✓ `//` una línea
  - ✓ `#` Comentario estilo shell para una línea
- Para imprimir: `echo` y `print`
  - ✓ **echo**: muestra una o más cadenas separadas por comas
    - `echo "Hola mundo";`
    - `echo "Hola ", "mundo";` //elementos separados
    - `echo "Hola " . "mundo";` //elementos concatenados
  - ✓ **print**: muestra una cadena o varias unidas por el operador punto ( `.` )
    - `print "Hola " . "mundo";`
    - `print "Hola mundo";`

# Sintaxis básica PHP

- Uso de `\n` para generar código HTML legible
  - ✓ Sin el carácter `\n`

Código PHP

```
print("<P>Párrafo 1</P>");  
print("<P>Párrafo 2</P>");
```

Código HTML

```
<P>Párrafo 1</P><P>Párrafo 2</P>
```

Salida

Párrafo 1

Párrafo 2

# Sintaxis básica PHP

- Uso de \n para generar código HTML legible
  - ✓ Con el carácter \n

Código PHP

```
print ("<P>Párrafo 1</P>\n");  
print ("<P>Párrafo 2</P>\n");
```

Código HTML

```
<P>Párrafo 1</P>  
<P>Párrafo 2</P>
```

Salida

```
Párrafo 1  
  
Párrafo 2
```



# Expresiones y Operadores

- Operadores aritméticos: +, -, \*, /, %, ++, --
- Operador de asignación: =
- Operadores combinados: -=, +=, \*=, /=, .=", %=
- Ejemplos:  

```
$a = 3; $a += 5; // a vale 8  
$b = "hola ";  
$b .= "mundo"; // b vale "hola mundo"
```
- Operadores de comparación: ==, !=, <, >, <=, >= y otros

# Expresiones y Operadores

- Operador de identidad **===** compara también el tipo de las variables.

- Operador de control de error: **@**

Antepuesto a una expresión, evita cualquier mensaje de error que pueda ser generado por la expresión y continua la ejecución

```
<?php
$var1=3; $var2=0;
$huboerror="no se produce resultado por error";
$nohuboerror="variable con valor";
@$resultado = $var1/$var2;
echo (empty($resultado))? huboerror : nohuboerror;
?>
```

- Operadores lógicos: && (and), || (or), ! , xor
- Operadores de cadena:
  - ✓ concatenación: **.** (punto)
  - ✓ asignación con concatenación: **.=**

# Inclusión de Ficheros externos en PHP

- La inclusión de ficheros externos se consigue con:
  - ✓ **include()**
  - ✓ **require()**
- Ambos incluyen y evalúan el fichero especificado
- Diferencia: en caso de error **include()** produce un warning y **require()** un error fatal
- Se usará **require()** si al producirse un error debe interrumpirse la carga de la página
- Si usamos **include\_once()** o **required\_once()**, sólo lo realizará la primera vez

# Inclusión de Ficheros externos en PHP

```
<HTML>
  <HEAD>
    <TITLE>Título</TITLE>
    <?PHP
      // Incluir bibliotecas de funciones
      require ("conecta.php");
      require ("fecha.php");
      require ("cadena.php");
      require ("globals.php");
    ?>
  </HEAD>
  <BODY>
    <?PHP
      include ("cabecera.html");
    ?>
    // Código HTML + PHP
    <?PHP
      include ("pie.html");
    ?>
  </BODY>
</HTML>
```