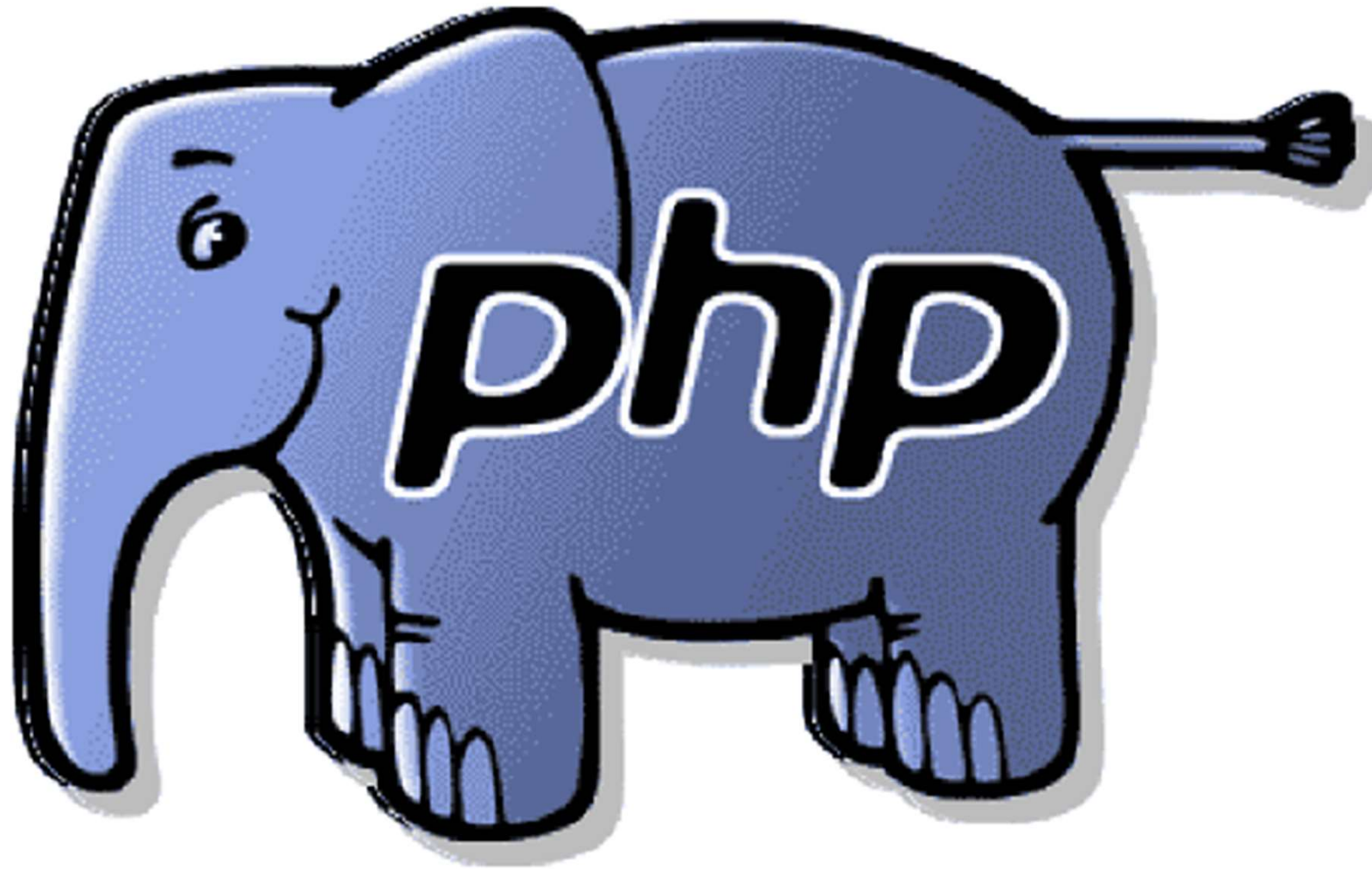


Insertión de código en páginas web



Desarrollo Web en Entorno Servidor

Contenido

1. Lenguajes de script y lenguajes de marcas
2. Sintaxis básica de PHP
3. Tipos de datos
4. Variables
 - Declaración de variables
 - Conversiones de tipos
 - Variables por referencia
 - Ámbito de las variables
5. Constantes
6. Expresiones y operadores
 - Precedencia de operadores

1. Lenguajes de script y lenguajes de marcas

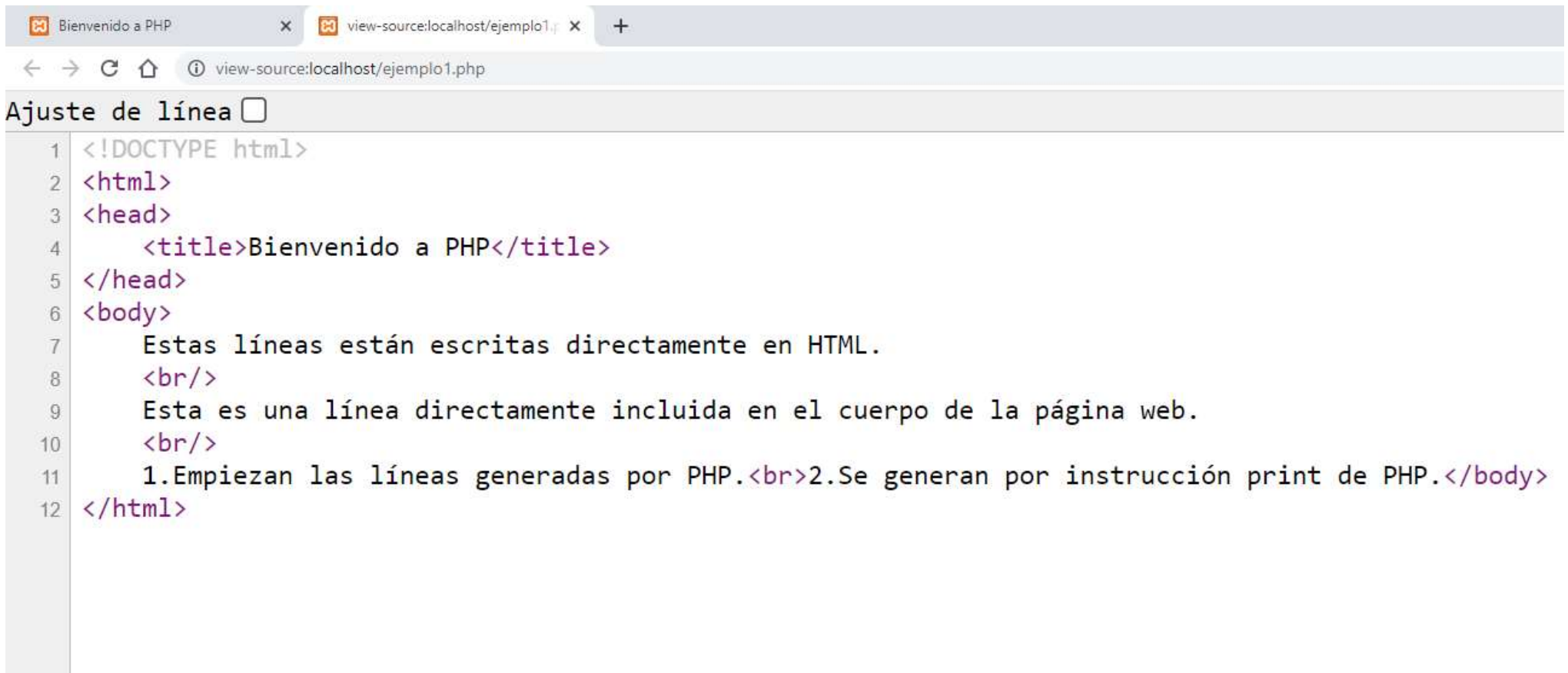
Los lenguajes de script como **PHP** se encontrarán generalmente **integrados con lenguajes de marcas** como HTML o XML. El procesamiento es automático. Si el navegador solicita al servidor un documento .php, el intérprete PHP ejecuta las sentencias transformándolas en el HTML que posteriormente va a interpretar el navegador.

Ejemplo de código PHP integrado con HTML (ejemplo1.php):

```
<!DOCTYPE html>
<html>
<head>
    <title>Bienvenido a PHP</title>
</head>
<body>
    Estas líneas están escritas directamente en HTML.
    <br/>
    Esta es una línea directamente incluida en el cuerpo de la página web.
    <br/>
    <?php
        $expresion="1";
        if ($expresion == "1") {
            print("1.Empiezan las líneas generadas por PHP.<br>");
            print("2.Se generan por instrucción print de PHP.");
        }
    ?>
</body>
</html>
```

1. Lenguajes de script y lenguajes de marcas

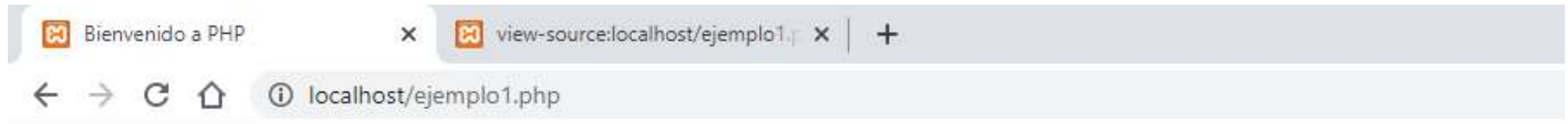
El procesamiento del código anterior en el servidor produce este **código HTML**, que se enviará al cliente:



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Bienvenido a PHP</title>
5 </head>
6 <body>
7   Estas líneas están escritas directamente en HTML.
8   <br/>
9   Esta es una línea directamente incluida en el cuerpo de la página web.
10  <br/>
11  1.Empezan las líneas generadas por PHP.<br>2.Se generan por instrucción print de PHP.</body>
12 </html>
```

1. Lenguajes de script y lenguajes de marcas

En la pantalla del **navegador del cliente** se mostrará lo siguiente:



Estas líneas están escritas directamente en HTML.

Esta es una línea directamente incluida en el cuerpo de la página web.

1. Empiezan las líneas generadas por PHP.
2. Se generan por instrucción print de PHP.

1. Lenguajes de script y lenguajes de marcas

PHP – Hypertext Preprocessor

- Lenguaje de **código abierto**.
- Permite su uso junto con documentos HTML mediante la **inserción de fragmentos** de código PHP acotados por etiquetas especiales.
- El código PHP es **interpretado** por el servidor web donde se aloja el documento y se genera el correspondiente código HTML para mostrar por el navegador.
- Las páginas tienen **extensión *.php**
- Tiene una interfaz de desarrollo basada en la **POO**.
- Admite los principales servidores web y soporte con **diferentes repositorios de bases de datos** (para MySQL posee librerías específicas).

2. Sintaxis básica de PHP

- Hay cuatro tipos de etiquetas para **delimitar bloques de código PHP**:

- Forma general: asegura la portabilidad.

<?php

Instrucciones PHP;

?>

- Forma para editores que no aceptan instrucciones de procesamiento (frontpage):

<script language="php">

Instrucciones PHP;

</script>

- Formato corto: (activar en php.ini la directiva: short_open_tag = on)

<?

Instrucciones PHP;

?>

- Etiquetas de ASP: (activar en php.ini la directiva: asp_tags = on)

<%

Instrucciones PHP;

%>

2. Sintaxis básica de PHP

- PHP es **sensible a las mayúsculas**
- Los espacios en blanco dentro del código embebido no tienen ningún efecto.
- Para indicar el **fin de instrucción** se usa punto y coma “;”.
La marca final ?> implica un ;
- Los scripts PHP pueden situarse en cualquier parte del código.
- El número de scripts es indefinido.
- **Comentarios:**
 - De una línea: # y // (el más habitual).
 - De bloque: /* comentario */
- Para imprimir: **echo** y **print**
 - echo:** muestra una o más cadenas separadas por punto (.) o coma (,):
echo “Hola mundo”;
echo “Hola ” . ”mundo”;
echo “Hola ” , ”mundo”;
 - print:** muestra una o varias cadenas unidas por el operador punto (.):
print “Hola mundo”;
print “Hola “ . ”mundo”;

2. Sintaxis básica de 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

■ 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

3. Tipos de datos

PHP soporta los **tipos de datos primitivos**:

- Números enteros
- Números en coma flotante
- Cadenas de caracteres
- Booleanos
- Objetos
- Recursos
- NULL

El tipo de dato de una variable **no se suele especificar**. Se decide en tiempo de ejecución en función del contexto, y puede variar.

Números enteros: Enteros positivos y enteros negativos.

`$var = 20;` `$var = -20;` `// asignación decimal`

`$var = 024;` `$var = -024;` `// asignación octal`

`$var = 0x14;` `$var = -0x14;` `// asignación hexadecimal`

3. Tipos de datos

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. P.ej.: una conexión a una base de datos.

Funciones de interés:

- **gettype():** devuelve el tipo de dato de una variable.
- **settype():** convierte una variable al tipo indicado por parámetro.
- Funciones para comprobar 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.

3. Tipos de datos

Tipo string:

Las cadenas se encierran entre comillas simples o dobles.

- ‘simples’: admiten los caracteres de escape: \’ (comilla simple) y \\ (barra). Las variables NO se expanden.
- “dobles”: admiten más caracteres de escape: \n, \r, \t, \\, \\$, \”. Las variables SÍ se expanden.

Para acceder a un carácter de la cadena, la forma es: \$letra_inicial = \$nombre{0};

Para que PHP distinga correctamente el texto que forma la cadena del nombre, a veces es necesario encerrarla entre llaves:

```
echo "<p> Módulo: ${modulo} </p>";
```

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 
```

4. Variables

- Las variables en PHP siempre van precedidas de un signo \$
- El nombre es sensible a las mayúsculas.
- Comienzan por letra o guión bajo, seguido de letras, números o guiones bajos.
- Además de las variables definidas por el programador, existen gran cantidad de variables predefinidas que se pueden usar libremente.
- Las **matrices superglobales** centralizan todas las variables predefinidas:
\$GLOBALS, \$_SERVER, \$_GET, \$_POST, \$_COOKIE,
\$_FILES, \$_ENV, \$_REQUEST, \$_SESSION

- **Declaración de variables**

PHP es flexible en lo que se refiere a la declaración de variables.

- No hace falta declarar una variable antes de utilizarla.
- Una variable no se define como perteneciente a un tipo de dato determinado.
- El tipo de dato de una variable puede cambiar según los valores que contenga durante la ejecución del programa.
- El último valor asignado es el que define el tipo de dato de la variable.

4. Variables

■ Ejemplos (ejemplo2.php):

```
<?php
$Cadena = "Tipo de dato de cadena";
$NumeroEntero = 1;           // Un valor entero
$NumeroFlotante= 1.55;       // Un valor numérico con decimales
$Booleano = True;           // Un valor booleano True (1) o False (0)
$Matriz[0] = "A";            // Un valor de matriz con subíndice 0
$Matriz[2] = 3;              // Un valor de matriz con subíndice 2
$NumeroOctal = 012;          // Un valor octal 12 es decimal 10
$NumeroHexadecimal = 0x1C;   // Un valor hexadecimal 1c es 28 en decimal
$NumeroNegativo = -33;       // Los números negativos llevan el signo - delante
$NumeroFlotanteExp = 1.55e3;
echo $Cadena;
echo $NumeroEntero;
echo $NumeroFlotante;
echo $Booleano;
echo $Matriz[0];
echo $Matriz[2];
echo $NumeroOctal;
echo $NumeroHexadecimal;
echo $NumeroNegativo;
echo $NumeroFlotanteExp;
?>
```

4. Variables

- **Conversión de tipos**

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.

Ejemplos:

```
$varN=1;
```

```
$varC='2 flores';
```

```
$varC=$varC+$varN;           // el resultado es 3
```

En una operación aritmética con cadenas intenta obtener el valor numérico de las cadenas.

En operaciones entre enteros y coma flotante resulta 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
```

4. Variables

- **Conversión forzada de tipos**

La conversión automática que realiza PHP no siempre es lo que queremos.

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.

- **Variables por referencia** (ejemplo3.php)

La variable no contiene un valor sino **la dirección de memoria** de otra variable.

```
<?php
```

```
$cadena = "Tipo de dato cadena";
```

```
$ref = &$cadena;
```

```
$cadena = "Nueva asignación";
```

```
echo $ref;                                // salida: Nueva asignación
```

```
?>
```

El signo **&** indica que se está almacenando la dirección de la variable y no su contenido.

4. Variables

- **Ámbito de las variables**

Es la zona de actuación de una variable.

Respecto a su ámbito, podemos clasificar a las variables en:

- **Locales:** Las que se crean por defecto. Cuando se definen dentro de una función solo son accesibles desde la función donde se definen.
- **Globales:** Se definen usando la palabra reservada **global**. Estas variables pueden ser empleadas desde cualquier punto del código dentro del fichero en el que se han definido, pero también puede ser empleada desde otros ficheros.
- **Estática:** Se definen usando la palabra reservada **static**. El ámbito será la función en la que se crea. Las variables locales no son estáticas, las locales pierden el valor cuando el control del programa sale de la función, mientras que las estáticas mantienen su valor aunque el control del flujo del programa haya salido de la función.
- **Superglobal:** Son variables accesibles desde cualquier lugar, desde el fichero donde se definen o desde cualquier otro fichero del sitio web. Solo una serie de variables están definidas así, y son de tipo array definidas en la matriz superglobal **\$GLOBALS**, por lo que podemos dar a sus elementos los valores de las variables superglobales que necesitemos.

4. Variables

- **Ámbito de las variables**

```
<?php
```

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

```
?>
```

5. Constantes

Son identificadores de datos que no cambian de valor durante toda la ejecución del programa.

Definición de constantes: **Define ("PI", 3.1416);**

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 por el sistema. Se pueden utilizar cuando las necesitamos:

- 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_

6. Expresiones y operadores

Una **expresión** es una combinación de operadores, variables, constantes, y funciones que es válida sintácticamente y tiene sentido. Toda expresión produce un valor al ser procesada.

\$a = 6;

\$b+3 == \$a*2;

Un **operador** es un elemento, palabra o símbolo que al aplicarlo sobre otros elementos, los operandos, proporciona un valor.

Operadores aritméticos:

Operación	Sintaxis
Suma	\$a + \$b
Resta	\$a - \$b
Producto	\$a * \$b
Cociente	\$a / \$b
Cociente entero	(int) \$a / \$b
Resto división	\$a % b
Potencia a^b	pow (\$a,\$b)
Raíz Cuadrada	sqrt (\$a)

6. Expresiones y operadores

Operador de asignación:

Operación	Sintaxis	Comentario
Asignación	\$a = 6	Almacena un dato dentro de una variable

Operadores de comparación:

Operación	Sintaxis	Comentario
Igualdad	\$a == \$b	Cierto si \$a es igual que \$b
Identidad	\$a === \$b	Cierto si \$a es igual a \$b y son del mismo tipo
Desigualdad	\$a != \$b	Cierto si \$a no es igual que \$b
Mayor que	\$a > \$b	Cierto si \$a es mayor que \$b
Menor que	\$a < \$b	Cierto si \$a es menor que \$b
Mayor o igual que	\$a >= \$b	Cierto si \$a es mayor o igual que \$b
Menor o igual que	\$a <= \$b	Cierto si \$a es menor o igual que \$b

6. Expresiones y operadores

Operador de incremento o decremento:

Operación	Sintaxis	Comentario
Pre-incremento	<code>++ \$a</code>	Incrementa en 1 \$a y después devuelve \$a
Post-incremento	<code>\$a ++</code>	Devuelve \$a y después incrementa en 1 a \$a
Pre-decremento	<code>--\$a</code>	Decremento en 1 \$a y después devuelve \$a
Post-decremento	<code>\$a--</code>	Devuelve \$a y después decremento en 1 a \$a

Operadores lógicos:

Operación	Sintaxis	Comentario
Y	<code>\$a and \$b</code>	Cierto si \$a y \$b son ciertos
	<code>\$a && \$b</code>	
O	<code>\$a or \$b</code>	Cierto si \$a o \$b son ciertos
	<code>\$a \$b</code>	
O exclusiva	<code>\$a xor \$b</code>	Cierto si \$a o \$b son ciertos, pero no ambos a la vez
Negación	<code>! \$a</code>	Cierto si \$a no es cierto

6. Expresiones y operadores

Operadores de cadena:

Operación	Sintaxis	Comentario
Concatenar	\$a . \$b	Concatena \$a y \$b
Concatenar y asignar	\$a .= \$b	\$a tiene el contenido que tenía inicialmente más lo que tenía \$b

Operadores combinados:

`+=, -=, *=, /=, .=, %=`

Operador condicional ternario:

`exp1 ? exp2 : exp3`

Operador de control de error @: antepuesto a una expresión, evita cualquier mensaje de error que pueda ser generado por la expresión y continúa con la ejecución.

6. Expresiones y operadores

- **Precedencia de operadores**

Especifica que operaciones han de efectuarse primero en caso de existir varias en un segmento de código.

De mayor a menor:

- ++,-- (operadores unarios)
- *, /, %
- +, -
- <, <=, >, >=
- ==, !=
- &&
- ||
- and
- or

Para evitar errores es recomendable establecer la precedencia de las operaciones utilizando **paréntesis ()**.

Manual de PHP: <http://php.net/manual>