# TEMA 3: PROGRAMACIÓN WEB UTILIZANDO PHP

## 1. INTEGRACIÓN CON LOS LENGUAJES DE MARCAS. SINTAXIS

Todo código PHP siempre va "encerrado" o delimitado por los caracteres de inicio de código PHP **<?php** y por los caracteres de fin de código PHP **?>**. Podemos utilizar PHP en solitario o junto con código HTML. Ejemplo de código PHP en solitario:

```
<?php
echo "Hola, Soy un script PHP!";
?>
```

NOTA: La instrucción echo de PHP muestra por pantalla lo que viene a continuación, en este capítulo veremos más posibilidades de funcionamiento de esta instrucción

Ejemplo: Código PHP junto a código HTML

```
<html>
<head>
<title>EJEMPLO</title>
</head>
<body>
<?php
echo "Hola, Soy un script PHP!";
?>
<P>Una parte en HTML
<?
echo "esta es una segunda parte en PHP";
?>
</body>
</html>
```

## 2. COMENTARIOS.

PHP Soporta comentarios 'C', 'C++' y Unix estilo de consola.

Los comentarios de estilo "una sola línea" solo comentan hasta el final de la línea o del bloque actual de código PHP, lo primero que suceda. Se utilizan los caracteres // ó # para indicar un comentario de este tipo

Los comentarios del estilo 'C' comienzan con /* finalizan al primer */ que se encuentre. No es conveniente utilizar muchos comentarios del estilo 'C'. Es muy fácil cometer errores cuando se trata de comentar un gran bloque de código.

```
<?php
echo 'Esto es una prueba';
// Esto es un comentario estilo c++ de una sola línea

/* Esto es un comentario multilínea
si! otra línea de comentarios */

echo 'Esto es otro test';
echo 'Un test final';
```

```
# Esto es un comentario estilo consola de una sola línea
?>
```

## 3. ELEMENTOS DEL LENGUAJE ESTRUCTURADO: TIPOS DE DATOS, VARIABLES, OPERADORES, ESTRUCTURAS DE CONTROL, SUBPROGRAMAS…

### a. Tipos de datos

PHP soporta los siguientes tipos: Entero, números en coma flotante, cadenas, arrays y objetos

El tipo de una variable normalmente no lo indica el programador; en su lugar, lo decide PHP en tiempo de ejecución dependiendo del contexto en el que se utilice esa variable.

Si se quisiese obligar a que una variable se convierta a un tipo concreto, se podría forzar la variable o usar la función settype() para ello.

Nótese que una variable se puede comportar de formas diferentes en ciertas situaciones, dependiendo de qué tipo sea en ese momento.

### b. Enteros

Los enteros se pueden especificar usando una de las siguientes sintaxis:

```
$a = 1234; # número decimal
$a = -123; # un número negativo
$a = 0123; # número octal (equivalente al 83 decimal)
$a = 0x12; # número hexadecimal (equivalente al 18 decimal)
```

### c. Números en punto flotante

Los números en punto flotante ("double") se pueden especificar utilizando cualquiera de las siguientes sintaxis:

```
$a = 1.234;
$a = 1.2e3;
```

### d. Cadenas

Las cadenas de caracteres se pueden especificar usando uno de dos tipos de delimitadores.

Si la cadena está encerrada entre dobles comillas ("), las variables que estén dentro de la cadena serán expandidas (sujetas a ciertas limitaciones de interpretación). Como en C y en Perl, el carácter de barra invertida ("\") se puede usar para especificar caracteres especiales:

| Secuencia | Significado |
|---|---|
| \n | Nueva línea |
| \r | Retorno de carro |
| \t | Tabulación |
| \\ | Barra invertida |

| | |
|---|---|
| \s | Signo del dólar |
| \” | Comillas dobles |

La segunda forma de delimitar una cadena de caracteres usa el carácter de comilla simple ("'"). Cuando una cadena va encerrada entre comillas simples, los únicos caracteres de escape que serán comprendidos son "\\" y "\'". Esto es por convenio, así que se pueden tener comillas simples y barras invertidas en una cadena entre comillas simples. Las variables no se expandirán dentro de una cadena entre comillas simples.

Las cadenas se pueden concatenar usando el operador '.' (punto). Nótese que el operador '+' (suma) no sirve para esto.

Se puede acceder a los caracteres dentro de una cadena tratándola como un array de caracteres indexado numéricamente

### e. Conversión de cadenas

Cuando una cadena se evalúa como un valor numérico, el valor resultante y el tipo se determinan como sigue.

La cadena se evaluará como un doble si contiene cualquiera de los caracteres '.', 'e', o 'E'. En caso contrario, se evaluará como un entero.

El valor viene dado por la porción inicial de la cadena. Si la cadena comienza con datos de valor numérico, este será el valor usado. En caso contrario, el valor será 0 (cero). Los datos numéricos válidos son un signo opcional, seguido por uno o más dígitos (que opcionalmente contengan un punto decimal), seguidos por un exponente opcional. El exponente es una 'e' o una 'E' seguidos por uno o más dígitos.

Cuando la primera expresión es una cadena, el tipo de la variable dependerá de la segunda expresión

### f. Arrays

Los arrays actualmente actúan tanto como tablas hash (arrays asociativos) como arrays indexados (vectores).

PHP soporta tanto arrays escalares como asociativos. De hecho, no hay diferencias entre los dos. Se puede crear una array usando las funciones list() o array(), o se puede asignar el valor de cada elemento del array de manera explícita.

```
$a[0] = "abc";
$a[1] = "def";
$b["foo"] = 13;
```

También se puede crear un array simplemente añadiendo valores al array. Cuando se asigna un valor a una variable array usando corchetes vacíos, el valor se añadirá al final del array.

```
$a[] = "hola"; // $a[2] == "hola"
$a[] = "mundo"; // $a[3] == "mundo“
```

Los arrays se pueden ordenar usando las funciones asort(), arsort(), ksort(), rsort(), sort(), uasort(), usort(), y uksort() dependiendo del tipo de ordenación que se desee.

Se puede contar el número de elementos de un array usando la función count().

Se puede recorrer un array usando las funciones next() y prev(). Otra forma habitual de recorrer un array es usando la función each().

Los arrays multidimensionales son bastante simples. Para cada dimensión del array, se puede añadir otro valor [clave] al final:

```
$a[1] = $f; # ejemplos de una sola dimensión
$a["foo"] = $f;
$a[1][0] = $f; # bidimensional
$a["foo"][2] = $f; # (se pueden mezclar índices numéricos y asociativos)
$a[3]["bar"] = $f; # (se pueden mezclar índices numéricos y asociativos)
$a["foo"][4]["bar"][0] = $f; # tetradimensional!
```

### g. Variables

En PHP las variables se representan como un signo de dólar seguido por el nombre de la variable. El nombre de la variable es sensible a minúsculas y mayúsculas.

```
$var = "Bob";
$Var = "Joe";
echo "$var, $Var"; // produce la salida "Bob, Joe"
```

PHP ofrece otra forma de asignar valores a las variables: asignar por referencia. Esto significa que la nueva variable simplemente referencia (en otras palabras, "se convierte en un alias de" o "apunta a") la variable original. Los cambios a la nueva variable afectan a la original, y viceversa. Esto también significa que no se produce una copia de valores; por tanto, la asignación ocurre más rápidamente. De cualquier forma, cualquier incremento de velocidad se notará sólo en los bucles críticos cuando se asignen grandes arrays u objetos. Para asignar por referencia, simplemente se antepone un ampersand (&) al comienzo de la variable cuyo valor se está asignando (la variable fuente). Por ejemplo, el siguiente trozo de código produce la salida 'Mi nombre es Bob' dos veces:

Algo importante a tener en cuenta es que sólo las variables con nombre pueden ser asignadas por referencia.

### h. Constantes

Una constante es un identificador para expresar un valor simple. Como el nombre sugiere, este valor no puede variar durante la ejecución del script. Una constante es sensible a mayúsculas por defecto.

Por convención, los identificadores de constantes suelen declararse en mayúsculas El nombre de una constante sigue las mismas reglas que cualquier etiqueta en PHP.

Un nombre de constante válido empieza con una letra o un caracter de subrayado, seguido por cualquier número de letras, números, o subrayados. Se puede definir una constante usando la función define(). Una vez definida, no puede ser modificada ni eliminada.

Solo se puede definir como constantes valores escalares boolean, integer, float y string

Para obtener el valor de una constante solo es necesario especificar su nombre. A diferencia de las variables, no se tiene que especificar el prefijo $. También se puede utilizar la función constant(), para obtener el valor de una constante, en el caso de que queramos expresarla de forma dinámica Usa la función get_defined_constants() parar obtener una lista de todas las constantes definidas.

Estas son las diferencias entre constantes y variables:

- Las constantes no son precedidas por un símbolo de dolar ($)
- Las constantes solo pueden ser definidas usando la función() define , nunca por simple asignación
- Las constantes pueden ser definidas y accedidas sin tener en cuenta las reglas de alcance del ámbito.
- Las constantes no pueden ser redefinidas o eliminadas después de establecerse
- Las constantes solo puede albergar valores escalares

### i. Constantes predefinidas

PHP ofrece un largo número de constantes predefinidas a cualquier script en ejecución. Muchas de estas constantes, sin embargo, son creadas por diferentes extensiones, y solo estarán presentes si dichas extensiones están disponibles, bien por carga dinámica o porque has sido compiladas.

### j. Expresiones

Las expresiones son la piedra angular de PHP. En PHP, casi cualquier cosa que escribes es una expresión. La forma más simple y ajustada de definir una expresión es "cualquier cosa que tiene un valor".

Las formas más básicas de expresiones son las constantes y las variables. Cuando escribes "$a = 5", estás asignando ’5’ a $a. ’5’, obviamente, tiene el valor 5 o, en otras palabras ’5’ es una expresión con el valor 5 (en este caso, ’5’ es una constante entera).

Después de esta asignación, esperarás que el valor de $a sea 5 también, de manera que si escribes $b =$a, esperas que se comporte igual que si escribieses $b = 5. En otras palabras, $a es una expresión también con el valor 5. Si todo va bien, eso es exactamente lo que pasará.

## 4. OPERADORES ARITMÉTICOS

| Operación | Sintaxis | $a | $b | Resultado |
|---|---|---|---|---|
| Suma | $a+$b | 12 | 7,3 | 19,3 |
| Diferencia | $a-$b | 12 | 7,3 | 4,7 |
| Producto | $a*$b | 12 | 7,3 | 87,6 |

| | | | | |
|---|---|---|---|---|
| Cociente | $a/$b | 12 | 7,3 | 1,64383561644 |
| Cociente entero | (int)$a/$b | 12 | 7,3 | 1 |
| Resto división | $a%$b | 12 | 5 | 2 |
| Potencia $a^b$ | pow($a,$b) | 2 | 3 | 8 |
| Raiz cuadrada | sqrt($a) | 9 | | 3 |

```
<?
# definimos dos variables numéricas asignandoles valores

$a=23;
$b=34;

/* Hacemos la suma  y escribimos directamente los resultados utilizando las
instrucciones print y echo  */

print ("La suma de $a + $b es: " . $a . "+" . $b . "=" . ($a+$b)."<BR>");
echo "La suma de $a + $b es: " . $a . "+" . $b . "=" . ($a+$b) ."<BR>";

# guardamos ahora el resultado de esa operación en una nueva variable

$c=$a+$b;

/* ahora presentamos el resultado utilizando esa nueva variable  */

print "La suma de $a + $b es: " . $a . "+" . $b . "=" . $c ."<BR>";
echo "La suma de $a + $b es: " , $a , "+" , $b , "=" , $c ,"<BR>";

/* modificamos ahora los valores de $a y $b comprobando que el cambio no
modifica lo contenido en la variable $c */

$a=513; $b=648;
print ("<br> C sigue valiendo: " . $c ."<br>");

/* Experimentamos con los paréntesis en un supuesto de operaciones combinada.
Sumamos la variable $a con la variable $b y multiplicamos el resultado por $c.
*/
print "<br>No he puesto paréntesis y el resultado es: ".($a+$b*$c);
print "<br>He puesto paréntesis y el resultado es: ".(($a+$b)*$c);
?>
```

## 5. OPERADORES DE ASIGNACIÓN

El operador básico de asignación es "=". A primera vista podrías pensar que es el operador de comparación "igual que". Pero no. Realmente significa que el operando de la izquierda toma el valor de la expresión a la derecha, (esto es, "toma el valor de").

El valor de una expresión de asignación es el propio valor asignado. Esto es, el valor de "$a = 3" es 3. Esto permite hacer cosas curiosas como

```
$a = ($b = 4) + 5; // ahora $a es igual a 9, y $b vale 4.
```

Además del operador básico de asignación, existen los "operadores combinados" para todas las operaciones aritméticas y de cadenas que sean binarias. Este operador combinado te permite, de una sola vez, usar una variable en una expresión y luego establecer el valor de esa variable al resultado de la expresión. Por ejemplo:

Fíjate en que la asignación realiza una nueva copia de la variable original (asignación por valor), por lo que cambios a la variable original no afectan a la copia. Esto puede tener interés si necesitas copiar algo como un array con muchos elementos dentro de un bucle que se repita muchas veces (cada vez se realizará una nueva copia del array).

## 6. OPERADORES DE COMPARACIÓN

| Nombre | Ejemplo | Resultado |
|---|---|---|
| Igualdad | $a = = $b | Cierto si $a es igual a $b |
| Identidad | $a = = = $b | Cierta si $a es igual a $b y son del mismo tipo (solo PHP4) |
| Desigualdad | $a != $b | Cierto si $a no es igual a $b |
| Mayor que | $a > $b | Cierto si $a es estrictamente mayor que $b |
| Menor que | $a < $b | Cierto si $a es estrictamente 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 |

## 7. OPERADORES DE INCREMENTO/DECREMENTO

| Nombre | Ejemplo | Resultado |
|---|---|---|
| Preincremento | ++$a | Incrementa en 1 $a y después devuelve $a |
| Postincremento | $a++ | Devuelve $a y después incrementa $a en 1 |
| Predecremento | --$a | Decrementa en 1 $a y después devuelve $a |
| postdecremento | $a-- | Devuelve $a y después decrementa $a en 1 |

## 8. OPERADORES LÓGICOS

| Nombre | Ejemplo | Resultado |
|---|---|---|
| Y | $a and $b<br>$a && $b | Cierto si $a y $b son ciertos |
| O | $a or $b<br>$a \|\| $b | Cierto si $a o $b son ciertos |
| O exclusiva | $a xor $b | Cierto si $a o $b son ciertos, pero no ambos a la vez |
| Negación | !$a | Cierto si $a no es cierto |

## 9. OPERADORES DE CADENAS

| Nombre | Ejemplo | Resultado |
|---|---|---|
| Concatenación | $a . $b | Concatena $a y $b |
| Concatenación y asignación | $a .= $b | $a tiene el contenido de lo que tenía anteriormente más lo que tiene $b |

## 10. NÚMEROS ALEATORIOS

PHP dispone de dos funciones capaces de generar números aleatorios. Se trata de la función rand() y de la función mejorada mt_rand().La función de PHP mt_rand() genera aleatorios con un algoritmo que es de promedio 4 veces más rápido que el algoritmo que utiliza rand().

La forma más simple -y más desaconsejable- de generación de un número aleatorio es esta:

```
echo rand(5,10);  # genera un numero aleatorio entre 5 y 10
echo mt_rand(1,100);  /* genera un numero aleatorio entre 1 y 100 */
```

Para mejorar la aleatoriedad de rand podemos utilizar la siguiente función antes de generar los números aleatorios:

```
srand(time());
mt_srand(time());
```

Caso práctico 1: Vamos a hacer un generador de resultados para la lotería primitiva. El juego consiste en elegir entre los números 1 al 49, 6 de ellos, para acertar la Combinación Ganadora en el sorteo correspondiente, formada por 6 bolas de las 49 que se extraen del bombo.

Para la realización de este caso práctico nos vamos a servir de las funciones vistas en este apartado tales como mt_srand() y mt_rand()

```
<?
# generador de números para el sorteo de la primitiva

mt_srand(time()); # mejoramos la aleatoriedad de los números
# generamos y presentamos por pantalla los 6 números de la combinación
echo mt_rand(1,49);
echo ",";
echo mt_rand(1,49);
echo ",";
echo mt_rand(1,49);
echo ",";
echo mt_rand(1,49);
echo ",";
echo mt_rand(1,49);
echo ",";
echo mt_rand(1,49);
echo "<br>";
?>
```

## 11. ESTRUCTURAS DE CONTROL

Todo archivo de comandos PHP se compone de una serie de sentencias. Una sentencia puede ser una asignación, una llamada a función, un bucle, una sentencia condicional e incluso una sentencia que no haga nada (una sentencia vacía). Las sentencias normalmente acaban con punto y coma. Además, las sentencias se pueden agrupar en grupos de sentencias encapsulando un grupo de sentencias con llaves. Un grupo de sentencias es también una sentencia. En este apartado se describen los diferentes tipos de sentencias.

### a. if

La construcción if es una de las más importantes características de muchos lenguajes, incluido PHP. Permite la ejecución condicional de fragmentos de código. PHP caracteriza una estructura if que es similar a la de C:

```
if (expr)
        sentencia
```

Como se describe en la sección sobre expresiones, expr se evalúa a su valor condicional. Si expr se evalúa como TRUE, PHP ejecutará la sentencia, y si se evalúa como FALSE - la ignorará.

El siguiente ejemplo mostraría a es mayor que b si $a fuera mayor que $b:

```
if ($a > $b)
print "a es mayor que b";
```

A menudo, se desea tener más de una sentencia ejecutada de forma condicional. Por supuesto, no hay necesidad de encerrar cada sentencia con una cláusula if. En vez de eso, se pueden agrupar varias sentencias en un grupo de sentencias. Por ejemplo, este código mostraría a es mayor que b si $a fuera mayor que $b, y entonces asignaría el valor de $a a $b:

```
if ($a > $b) {
print "a es mayor que b";
$b = $a;
}
```

Las sentencias if se pueden anidar indefinidamente dentro de otras sentencias if, lo cual proporciona una flexibilidad completa para ejecuciones condicionales en las diferentes partes de tu programa.

### b. else

A menudo queremos ejecutar una sentencia si se cumple una cierta condición, y una sentencia distinta si la condición no se cumple. Esto es para lo que sirve else. else extiende una sentencia if para ejecutar una sentencia en caso de que la expresión en la sentencia if se evalúe como FALSE. Por ejemplo, el siguiente código mostraría a es mayor que b si $a fuera mayor que $b, y a NO es mayor que b en cualquier otro caso:

```
if ($a > $b) {
print "a es mayor que b";
}
else {
print "a NO es mayor que b";
}
```

La sentencia else se ejecuta solamente si la expresión if se evalúa como FALSE, y si hubiera alguna expresión elseif - sólo si se evaluaron también a FALSE (Ver elseif).

### c. elseif

elseif, como su nombre sugiere, es una combinación de if y else. Como else, extiende una sentencia if para ejecutar una sentencia diferente en caso de que la

expresión if original se evalúa como FALSE. No obstante, a diferencia de else, ejecutará esa expresión alternativa solamente si la expresión condicional elseif se evalúa como TRUE. Por ejemplo, el siguiente código mostraría a es mayor que b, a es igual a b o a es menor que b:

```
if ($a > $b) {
print "a es mayor que b";
}
elseif ($a == $b) {
print "a es igual que b";
}
else {
print "a es mayor que b";
}
```

Puede haber varios elseifs dentro de la misma sentencia if. La primera expresión elseif (si hay alguna) que se evalúe como TRUE se ejecutaría. En PHP, también se puede escribir 'else if' (con dos palabras) y el comportamiento sería idéntico al de un 'elseif' (una sola palabra). El significado sintáctico es ligeramente distinto (si estas familiarizado con C, es el mismo comportamiento) pero la línea básica es que ambos resultarían tener exactamente el mismo comportamiento.

La sentencia elseif se ejecuta sólo si la expresión if precedente y cualquier expresión elseif precedente se evalúan como FALSE, y la expresión elseif actual se evalúa como TRUE.

### d. while

Los bucles while son los tipos de bucle más simples en PHP. Se comportan como su contrapartida en C. La forma básica de una sentencia while es:

```
        while (expr) sentencia
```

El significado de una sentencia while es simple. Le dice a PHP que ejecute la(s) sentencia(s) anidada(s) repetidamente, mientras la expresión while se evalúe como TRUE. El valor de la expresión es comprobado cada vez al principio del bucle, así que incluso si este valor cambia durante la ejecución de la(s) sentencia(s) anidada(s), la ejecución no parará hasta el fin de la iteración (cada vez que PHP ejecuta las sentencias en el bucle es una iteración). A veces, si la expresión while se evalúa como FALSE desde el principio de todo, la(s) sentencia(s) anidada(s) no se ejecutarán ni siquiera una vez.

En el siguiente ejemplo mostrará por pantalla los números del 0 al 10 (incluidos ambos)

```
$i=0;
while($i<=10) {
        echo $i;
        echo "<br>"; // hacemos un "intro" entre número y número
        $i++;
}
```

### e. do while

Los bucles do..while son muy similares a los bucles while, excepto que las condiciones se comprueban al final de cada iteración en vez de al principio. La principal diferencia frente a los bucles regulares while es que se garantiza la ejecución de la primera iteración de un bucle do..while (la condición se comprueba sólo al final de la iteración), mientras que puede no ser necesariamente ejecutada con un bucle while regular (la condición se comprueba al principio de cada iteración, si esta se evalúa como FALSE desde el principio la ejecución del bucle finalizará inmediatamente). Hay una sola sintaxis para los bucles do..while:

```
$i = 0;
do {
print $i;
}
while ($i>0);
```

El bucle de arriba se ejecutaría exactamente una sola vez, después de la primera iteración, cuando la condición se comprueba, se evalúa como FALSE ($i no es más grande que 0) y la ejecución del bucle finaliza.

Los usuarios avanzados de C pueden estar familiarizados con un uso distinto del bucle do..while, para permitir parar la ejecución en medio de los bloques de código, encapsulandolos con do..while(0), y usando la sentencia break. El siguiente fragmento de código demuestra esto:

```
do {
        if ($i<5) {
                print “i no es suficientemente grande”;
                break;
        }
$i *= $factor;
if ($i <$minimum_limit) {
break;
}
print “i es correcto”;
.......... procesa i …….
}
while (0);
```

### f. for

Los bucles for son los bucles más complejos en PHP. Se comportan como su contrapartida en C. La sintaxis de un bucle for es:

```
for (expr1; expr2; expr3) sentencia
```

La primera expresión (expr1) se evalúa (ejecuta) incondicionalmente una vez al principio del bucle. Al comienzo de cada iteración, se evalúa expr2 . Si se evalúa como TRUE, el bucle continúa y las sentencias anidadas se ejecutan. Si se evalúa como FALSE, la ejecución del bucle finaliza.

Al final de cada iteración, se evalúa (ejecuta) expr3.

Cada una de las expresiones puede estar vacía. Que expr2 esté vacía significa que el bucle debería correr indefinidamente (PHP implicitamente lo considera como TRUE, al igual que C). Esto puede que no sea tan inútil como se podría pensar, puesto que a menudo se quiere salir de un bucle usando una sentencia break condicional en vez de usar la condición de for.

Considera los siguientes ejemplos. Todos ellos muestran números del 1 al 10:

```
/* ejemplo 1 */

for ($i =1; $i<=10;$i++) {
        print $i;
}
```

```
/* ejemplo 2 */

for ($i=1;;$i++)
{
        if ($i>10) {
                break;
        }
print $i;
$i++;
}
```

```
/* ejemplo 3 */

$i=1;
for (;;) {
        if ($i>10)
{
        break;
}
print $i;
$i++;
}
```

```
/* ejemplo 4 */

for ($i=1;$i<=10;print$i, $i++);
```

### g. Foreach

Simplemente da un modo fácil de iterar sobre arrays. Hay dos sintaxis; la segunda es una extensión menor, pero útil de la primera:

```
foreach(expresion_array as $value) sentencia
foreach(expresion_array as $key => $value) sentencia
```

La primera forma recorre el array dado por expresion_array. En cada iteración, el valor del elemento actual se asigna a $value y el puntero interno del array se avanza en una unidad (así en el siguiente paso, se estará mirando el elemento siguiente).

La segunda manera hace lo mismo, salvo que la clave del elemento actual será asignada a la variable $key en cada iteración.

```
/* ejemplo 1*/

$arr=array(5,4,3,2,1);
foreach ($arr as $value) {
echo “valor: $value<br>”;
}
```

```
/* ejemplo 2 */

$arr=array(5,4,3,2,1);
foreach ($arr as $key => $value) {
echo “key: $key; valor:
$valor<br>”;
}
```

```
/* ejemplo 3*/

$a = array(
“uno” => 1,
“dos” => 2,
“tres” => 3,
“diecisiete” => 17
);
foreach ($a as $k => $v) {
  print “\$a[$k] => $v.\n”;
}
```

```
/* ejemplo 4*/

<?
$a = array(
"nombre1" => "implantación",
"nombre2" => "de",
"nombre3" => "aplicaciones",
"nombre4" => "Web"
);
foreach ($a as $k => $v) {
  print " $v";
}
?>
```

### h. Break

break escapa de la estructuras de control iterante (bucle) actuales for, while, o switch. Break acepta un parámetro opcional, el cual determina cuantas estructuras de control hay que escapar.

En el siguiente ejemplo cuando el recorrido del bucle llega a “parar” se interrumpe la ejecución del foreach y se abandona el bucle

```
<?
$a = array(
"nombre1" => "implantación",
"nombre2" => "parar",
"nombre3" => "aplicaciones",
"nombre4" => "Web"
);
foreach ($a as $k => $v) {
  if ($v==”para”) {
                break;
}
print " $v";
}
?>
```

### i. Switch

La sentencia switch es similar a una serie de sentencias IF en la misma expresión. En muchas ocasiones, se quiere comparar la misma variable (o expresión) con nuchos valores diferentes, y ejecutar una parte de código distinta dependiendo de a qué valor es igual. Para ello sirve la sentencia switch.

Los siguientes dos ejemplos son dos modos distintos de escribir la misma cosa, uno usa una serie de sentencias if, y el otro usa la sentencia switch:

| If | Swich |
|---|---|
| If ($i == 0) {<br>Print “i es igual a 0”;<br>}<br>if ($i == 1) {<br>Print “i es igual a 1”;<br>}<br>if ($i == 2) { | switch($i) {<br>case 0:<br>print (“i es igual a 0”;<br>break;<br>case 1:<br>print (“i es igual a 1”;<br>break;<br><br>case 2: |

```
        Print “i es igual a
2”;
}
```

```
                print (“i es igual a 2”;
                break;
}
```

Es importante entender cómo se ejecuta la sentencia switch para evitar errores. La sentencia switch ejecuta línea por línea (realmente, sentencia a sentencia). Al comienzo, no se ejecuta código. Sólo cuando se encuentra una sentencia case con un valor que coincide con el valor de la expresión switch PHP comienza a ejecutar las sentencias. PHP continúa ejecutando las sentencias hasta el final del bloque switch, o la primera vez que vea una sentencia break. Si no se escribe una sentencia break al final de una lista de sentencias case, PHP seguirá ejecutando las sentencias del siguiente case. Por ejemplo:

```
switch ($i) {
        case 0:
                print “1 es igual a 0”;
        case 1:
                print “1 es igual a 1”;
        case 2:
                print “1 es igual a 2”;
}
```

Aquí, si $i es igual a 0, ¡PHP ejecutaría todas las sentecias print! Si $i es igual a 1, PHP ejecutaría las últimas dos sentencias print y sólo si $i es igual a 2, se obtendría la conducta ’esperada’ y solamente se mostraría ’i es igual a 2’. Así, es importante no olvidar las sentencias break (incluso aunque pueda querer evitar escribirlas intencionadamente en ciertas circunstancias).

En una sentencia switch, la condición se evalúa sólo una vez y el resultado se compara a cada sentencia case. En una sentencia elseif, la condición se evalúa otra vez. Si tu condición es más complicada que una comparación simple y/o está en un bucle estrecho, un switch puede ser más rápido.

La lista de sentencias de un case puede también estar vacía, lo cual simplemente pasa el control a la lista de sentencias del siguiente case.

```
/* ejemplo 1*/

switch ($i) {
 case 0:
 case 1:
 case 2:
  print “i es menor que 3, pero no
negativo”;
  break;
 case 3:
    print “i es 3”;
}
```

```
/*ejemplo 2*/

switch ($i) {
        case 0:
                print “i es 0”;
                break;
        case 1:
                print “i es 1”;
                break;
        case 2:
                print “i es 2”;
                break;
        default:
                print “i no 0,
1 o 2”;
}
```

La expresión case puede ser cualquier expresión que se evalúe a un tipo simple, es decir, números enteros o de punto flotante y cadenas de texto. No se pueden usar aquí ni arrays ni objetos a menos que se conviertan a un tipo simple.