PHP - Conceptos Generales

El código PHP se escribe en un documento de texto al que se debe indicar la extensión **.php**. En realidad se trata de una página web HTML (o XHTML) normal a la que se le añaden etiquetas especiales.

1. Etiqueta .php

Cuando en un documento web queremos añadir código **php** se indica por esta etiqueta:

```
<?php
            ……código PHP
?>
```

El código PHP se coloca en la zona de la página web donde más nos interese hacerlo.

2. Bases de la escritura

El código PHP se incrusta dentro del código HTML.

```
<?php
   echo "Mi nombre es <b>Marta</b>";
?>
```

Las normas básicas para escribir lenguaje son:

- Todas las líneas de código deben de finalizar con un punto y coma.
- Se puede agrupar el código en bloques que se escriben entre llaves.
- Una línea de código se puede partir o sangrar (añadir espacios al inicio) a voluntad con el fin de que sea más legible, siempre y cuando no partamos una palabra o un valor.
- PHP obliga a ser estricto con las mayúsculas y las minúsculas en algunos casos como el nombre de las variables; sin embargo con las palabras reservadas del lenguaje no es estricto. Es decir PHP entiende que ***WHILE***, ***while*** e incluso ***wHiLe*** es lo mismo al ser una palabra reservada. Sin embargo ***$var*** y ***$VAR*** no son iguales al ser el nombre de una variable.
- Aunque hay muchas funciones de escritura (para escribir en lo que será la página final) las fundamentales son **echo** y **print**.

```
<?php
   echo "Mi nombre es <b>Marta</b>";

   echo 'Mi nombre es <b>Antonio</b>';

   print "Mi nombre es <b>Jorge</b>";
?>
```

3. Comentarios

Dentro del código PHP se pueden hacer tres tipos de comentario:

Comentarios de varias líneas. Comienzan por **/*** y terminan por ***/.**

```
<?php
   /*
   Soy un comentario
   De varias líneas
   */
   echo 'Mi nombre es <b>Antonio</b>';
?>
```

Comentarios de una línea. Se ponen tras la barra doble //.

```
<?php
   echo 'Mi nombre es <b>Antonio</b>'; //esto es un comentario
?>
```

Comentarios de una línea. Se ponen tras la almohadilla #:

```
<?php
   echo 'Mi nombre es <b>Antonio</b>'; #esto es un comentario
?>
```

4. Variables

En todos los lenguajes de programación (y PHP no es una excepción) las **variables** son contenedores que sirven para almacenar los datos que utiliza un programa. Dicho más sencillamente, son nombres que asociamos a determinados datos.

La realidad es que cada variable ocupa un espacio en la memoria RAM del servidor que ejecute el código para almacenar el dato al que se refiere. Es decir cuando utilizamos el nombre de la variable realmente estamos haciendo referencia a un dato que está en memoria.

Las variables tienen un nombre (un **identificador**) que tiene que cumplir estas reglas:

- Tiene que empezar con el símbolo **$**. Ese símbolo es el que permite distinguir a una variable de otro elemento del lenguaje PHP.
- El segundo carácter puede ser el guion bajo (_) o bien una letra.
- A partir del tercer carácter se pueden incluir números, además de letras y el guión bajo.
- No hay límite de tamaño en el nombre.

- Por supuesto el nombre de la variable no puede tener espacios en blanco (de ahí la posibilidad de utilizar el guión bajo)

Es conveniente que los nombres de las variables indiquen de la mejor forma posible su función. Es decir: ***$saldo*** es un buen nombre, pero ***$x123*** no lo es aunque sea válido.

También es conveniente poner a nuestras variables un nombre en minúsculas. Si consta de varias palabras el nombre podemos separar las palabras con un guión bajo en vez del espacio o empezar cada nueva palabra con una mayúscula. Por ejemplo: ***$saldo_final*** o ***$saldoFinal***.

4.1. Declaración

La primera sorpresa para los programadores de lenguajes estructurados es que en PHP no es necesario declarar una variable. Simplemente se utiliza y ya está.

```
$edad;
$_edad;
$e123;
```

Y no será necesario indicar de qué tipo es esa variable. Esto es tremendamente cómodo pero también nos complica tremendamente la tarea de depurar nuestros programas al no ser nada rígido el lenguaje y permitir casi todo.

4.2. Predefinidas

El servidor web que aloje las páginas PHP pone a disposición del programador variables de sistema ya definidas para su uso en el programa.

La mayoría son simplemente informativas.

Todas suelen llevar el símbolo de subrayado en el segundo carácter además de escribirse en mayúsculas y la mayoría son arrays.

```
echo $_SERVER["SERVER_PORT"];
```

Escribirá el número de puerto por el que se comunica el servidor web. Para mostrar todas las variables predefinidas en el servidor podemos escribir este código dentro de una página PHP.

```
<?php
   echo '<pre>';
   print_r ($_SERVER);
   echo '</pre>';
?>
```

Las etiquetas ***pre***, permite que se muestra la información de una forma más atractiva (porque respeta los cambios de línea y tabulaciones devueltos por ***print_r***). Por su parte ***print_r*** es una función para mostrar el contenido de los arrays.

4.3. Asignación de valores

Esta operación consiste en dar valor a una variable y se realiza con el símbolo =.

Como se comentó anteriormente en PHP no es necesario declarar una variable antes de su uso.

En el caso de asignar números se escriben tal cual, los decimales se separan con el punto decimal. Los textos se encierran entre comillas (simples o dobles, aunque se aconseja usar las dobles salgo cuando nos venga mejor las simples).

```
$x=12;
$saldo=0.6;
$nombre="Carmen";
```

Se pueden asignar resultados de aplicar fórmulas mediante operadores, como:

```
$beneficios=($gastos-$ingresos)*0.6;
$saldo=0.6;
$nombre="Carmen";
```

Otra cosa sorprendente es que una variable puede cambiar el tipo de datos que almacena, por ejemplo es válido:

```
$x=18; //asignamos un número
$x="Hola"; //asignamos un texto
```

4.4. Variables sin asignación de valores

Un problema surge cuando queremos utilizar variables a las que no se les ha asignado ningún valor. Como:

```
<?php
   echo $x; //es la primera vez que se usa $x. ERROR
   $y=$x+2; //error, aunque y valdrá 2
   echo $y;
?>
```

Ocurrirá un error al hacer ese uso de la variable. Aunque en realidad la directiva del archivo **php.ini, error_reporting** permite modificar los errores de aviso que se lanzan. Si bajamos su nivel de aviso, no detectará estos fallos. También podemos usar esa función al inicio de nuestro código para indicar fallos. Sería:

```
<?php
   error_reporting(E_ALL); //avisa de todos los errores
?>
```

Si deseamos que no nos avise de estos fallos. Habría que pasar a error_reporting otra constante; por ejemplo **E_USER_ERROR** avisa sólo si hay errores de nivel usuario (o más graves) en el código

A las variables sin declarar se les da valores por defecto, que dependerán del contexto en que se usen. Hay una función llamado **isset** a la que se le pasa un variable e indica si la variable tenía asignado un valor; en cualquier otro caso devuelve falso. Ejemplo:

```
<?php
   echo isset($x); //escribe falso, o sea no escribe nada
   $x=12;
   echo isset($x); //devuelve verdadero, escribe 1
?>
```

4.5. Tipos de datos

Enteros

A las variables se les puede asignar valores enteros. Los números enteros se usan tal cual. Pueden ser positivos o negativos:

```
$n1=17;
$n2=-175;
```

Se puede usar sistema octal si se coloca un cero antes de la cifra entra:

```
$octal=071;
echo $octal; //escribe 56
```

Además pueden estar en sistema hexadecimal si a la cifra se la antecede de un cero y una equis:

```
$hexa=0xA2BC;
echo $hexa; //escribe 41660
```

Coma flotante

Los números decimales en PHP son de tipo coma flotante. Este es un formato decimal para máquinas digitales que se manejan muy rápido por parte de un ordenador, ocupan poco en memoria, pero desgraciadamente no son exactos. Eso provoca que pueda haber algunos decimales que se pierdan.

Para asignar decimales hay que tener en cuenta en PHP que el formato es el inglés, por lo que las cifras decimales se separan usando un **punto** como separador decimal. Además es posible usar notación científica.

```
$n1=234.12;
$n2=12.3e-4; //eso es 12,3·10-4, es decir 0,00123
```

Cadenas

Se denomina así a los textos, que en programación se les denomina cadenas de caracteres o **Strings**. Se asignan a las variables entrecomillando (en simples o dobles) el texto a asignar.

```
$nombre="Jorge Sánchez";
```

Si el propio texto lleva comillas, se puede utilizar combinaciones de las comillas para evitar el problema.

```
$frase = 'Antonio dijo "Hola" al llegar';
```

Como queremos almacenar en ***$frase*** el texto con ***Hola*** entre comillas, entonces englobamos todo el texto con comillas simples.

Otra opción es usar caracteres especiales, concretamente:

secuencia de escape	**significado**
\t	Tabulador
\n	Nueva línea
\f	Alimentación de página
\r	Retorno de carro
\"	Dobles comillas
\'	Comillas simples
****	Barra inclinada (***backslash***)
\$	Símbolo dólar

Y así podremos:

```
$frase = "Antonio dijo \"Hola\" al llegar";
```

Las secuencias de escape sólo funcionan si están encerradas entre comillas dobles. Es decir, no funciona:

```
$frase = 'Antonio dijo \"Hola\" al llegar';
```

Las barras saldrán por pantalla también. Un hecho de PHP muy interesante es que en un texto se puede incluir el valor de una variable.

```
$días=15;
$texto="Faltan $días días para el verano";
echo $texto;    //escribe: Faltan 15 días para el verano
```

Por lo que el símbolo **$** sólo se puede asignar a un texto si se usa su secuencia de escape **\$**.
Los textos se pueden **concatenar** con ayuda del operador punto (**.**).

```
$nombre="Jorge";
$apellidos="Sánchez Asenjo";
echo "Nombre completo: ".$nombre." ".$apellidos;
//escribe Nombre completo: Jorge Sánchez Asenjo
```

Booleanos

Sólo pueden tomar como valores **TRUE** (verdadero) o **FALSE** (falso);

```
$verdadero=True;
echo $verdadero; //escribe 1
```

Sorprende que **echo** devuelva el valor uno; la explicación es que True está asociado a valores positivos, mientras que False se asocia al cero. En concreto hay una relación todos los tipos de datos:

- Enteros: cero=False, resto=True
- Coma flotante: 0.0=False, resto=True
- Cadenas: False si están vacías
- Arrays: False si no posee ningún elemento
- Recurso: False si el recurso no es válido.

Conversiones

En PHP como las variables pueden cambiar de tipo cuando se nos antoje, resulta que tiene que intentar que todas las expresiones tengan sentido y así, este código:

```
$v1=18;
$v2="3 de Diciembre";
echo $v1+$v2;
```

Escribe 21 (suma el 18 y el tres). Pero sin embargo:

```
$v1=18;
$v2="3 de Diciembre";
echo $v1.$v2;
```

Escribe ***183 de Diciembre.***

No obstante se pueden convertir de forma forzosa los valores al tipo deseado; de esta forma elegiremos nosotros cómo realizar las conversiones. Se trata del habitual ***operador de casting***.

```
$x=2.5;
$y=4;
$z=(int)$x * $y;
//$z vale 8 al convertir $x en un entero.
```

Posibilidades:

- **(int)** o **(integer)**. Convierte a entero
- **(real), (double)** o **(float)**. Convierte a coma flotante
- **(string)**. Convierte a forma de texto
- **(array)**. Convierte a forma de array.
- **(object)**. Convierte a un objeto.

4.6. Referencias &

Es una de las claves de la programación en cualquier lenguaje. Se trata de variables que sirven para modificar otras variables existentes. Es decir, son variables que en lugar de almacenar valores, almacenan la dirección de otra variable. No es una copia de la otra variable, más bien es un sinónimo de la otra variable. Su uso principal aparece cuando se utilizan funciones.

```
$nombre="Antonio";
$ref=&$nombre; //ref es una referencia a la variable $nombre
echo $ref,"<br />"; //escribe Antonio
```

```
$ref="Marisa";
echo $nombre,"<br />"
//escribe Marisa, a través de la referencia se ha cambiado el nombre
```

4.7. Constantes

Las constantes almacenan valores que no cambian en el tiempo. La forma de definir constantes es gracias a la función **define**. Que funciona indicando el nombre que tendrá la constante, entre comillas, y el valor que se le otorga.

```
define("PI",3.141592);
```

Las constantes no utilizan el signo **$** de las variables, simplemente utilizan el nombre. Es decir escribir el valor de la constante PI tras haberla declarado con la instrucción anterior, sería:

```
echo PI;
```

Desde la versión 5.3 de PHP es posible definir una constante de una forma más estructurada. Se trata de utilizar la palabra clave **const** habitual en la mayoría de lenguajes de programación.

```
const PI=3.141592;
```

Aunque no es obligatorio, es conveniente usar mayúsculas para las constantes para diferenciarlas de las variables (aunque en PHP el signo dólar $ ya hace esa diferenciación) ya que se considera una norma de buen estilo de escritura en cualquier lenguaje de programación.

5. Operadores

Lo habitual al programar en PHP es utilizar expresiones que permiten realizar comprobaciones o cálculos. Las expresiones dan un resultado que puede ser de cualquiera de los tipos de datos comentados anteriormente (enteros, decimales, booleanos, strings,…)

Aritméticos

operador	significado
+	Suma
-	Resta
*	Producto
/	División
%	Módulo (resto)

```
$x=15.5;
$y=2;
echo $x+$y,"<br />"; //escribe 17.5
echo $x-$y,"<br />"; // escribe 13.5
echo $x*$y,"<br />"; // escribe 31
echo $x/$y,"<br />"; //escribe 7.75
echo $x%$y,"<br />"; //escribe 1, sólo coge la parte entera
```

Operadores condicionales

Sirven para comparar valores. Siempre devuelven valores booleanos.

operador	significado
<	Menor
>	Mayor
>=	Mayor o igual
<=	Menor o igual
==	Igual, devuelve verdadero si las dos expresiones que compara son iguales
===	Equivalente, devuelve verdadero si las dos expresiones que compara son iguales y además del mismo tipo
!=	Distinto
!	No lógico (NOT)
&&	"Y" lógico
AND	"Y" lógico
||	"O" lógico
OR	"O" lógico
XOR	"OR" exclusivo

Los operadores lógicos (**AND**, **OR** y **NOT**), sirven para evaluar condiciones complejas. NOT sirve para negar una condición.

```
$edad = 21;
$mayorDeEdad = $edad >= 18;     //mayorDeEdad será true
$menorDeEdad = !$mayorDeEdad;   //menorDeEdad será false
echo $mayorDeEdad."\t".$menorDeEdad;
```

El operador **&&** (**AND**) sirve para evaluar dos expresiones de modo que si ambas son ciertas, el resultado será **true** sino el resultado será **false**.

```
$carnetConducir=TRUE;
$edad=20;
$puedeConducir= ($edad>=18) && $carnetConducir;
//puedeConducir es TRUE, porque edad mayor de 18  y carnet es TRUE
```

El operador || (**OR**) sirve también para evaluar dos expresiones. El resultado será **true** si al menos uno de las expresiones es **true**.

```
$nieva =TRUE;
$llueve=FALSE;
$graniza=FALSE;
$malTiempo= $nieva || $llueve || $graniza;
//malTiempo es TRUE porque nieva
```

La diferencia entre la igualdad y la equivalencia se puede explicar con este ejemplo:

```
$resultado= (18 == 18.0); //resultado es verdadero
$resultado= (18 === 18.0); //resultado es falso
```

En el ejemplo ***18===18.0*** devuelve falso porque aunque es el mismo valor, no es del mismo tipo.

Operadores de asignación

Ya se ha comentado el operador de asignación que sirve para dar valor a una variable.

```
$x=$y+9.8;
```

Sin embargo existen operadores que permiten mezclar cálculos con la asignación.

```
$x += 3;
```

En el ejemplo anterior lo que se hace es sumar ***3*** a la ***x*** (es lo mismo ***$x+=3***, que ***$x=$x+3***). Eso se puede hacer también con todos estos operadores:

+=	-=	*=	/=
&=	|=	^=	%=
>>=	<<=	.=	

Otros operadores de asignación son "++" (incremento) y "--" (decremento).

```
$x++; //esto es $x=$x+1;
$x--; //esto es $x=$x-1;
```

Pero hay dos formas de utilizar el incremento y el decremento. Se puede usar por ejemplo ***x++*** o ***++x*** La diferencia estriba en el modo en el que se comporta la asignación. Ejemplo:

```
$x=5;
$y=5;
$z=$x++; //z vale 5, x vale 6
$z=++$y; //z vale 6, y vale 6
```

Operadores de BIT

Manejan números, a los cuales se les pasa a formato binario y se opera con ellos BIT a BIT. Son:

operador	significado
&	AND
|	OR
~	NOT
^	XOR
>>	Desplazamiento a la derecha
<<	Desplazamiento a la izquierda

Concatenación

El **punto (.)** es un operador que permite unir textos. Su uso es muy sencillo.

```
$x="Hola ";
$y="a todo el mundo";
$z=$x.$y; //$z vale "Hola a todo el mundo"
```

Hay operador de asignación y concatenación de texto, se trata del operador **.=**.

```
$x="Hola ";
$x.="a todo el mundo";
echo $x; //escribe "Hola a todo el mundo
```

También es válido el uso de la concatenación en la asignación.

```
$texto = "Hola ";
$texto.= "amigos y ";
$texto.="amigas"; //$texto es Hola amigos y amigas
```

6. Estructuras de Control

Hasta ahora las instrucciones que hemos visto, son instrucciones que se ejecutan secuencialmente; es decir, podemos saber lo que hace el programa leyendo las líneas de izquierda a derecha y de arriba abajo.

Las instrucciones de control de flujo permiten alterar esta forma de ejecución. A partir de ahora habrá líneas en el código que se ejecutarán o no dependiendo de una **condición**.

Esa condición se construye utilizando lo que se conoce como **expresión lógica**. Una expresión lógica es cualquier tipo de expresión que dé un resultado verdadero o falso.

Las expresiones lógicas se construyen a través de los operadores relacionales **(==,** >, <,…) y lógicos **(&&,**||, **!,…**) vistos anteriormente.

6.1. Sentencia if

Sentencia condicional simple

Se trata de una sentencia que, tras evaluar una expresión lógica, ejecuta una serie de instrucciones en caso de que la expresión lógica sea verdadera. Si la expresión tiene un resultado falso, no se ejecutará ninguna expresión. Su sintaxis es:

```
if (expresión lógica){
   instrucciones
   …
}
```

Las llaves se requieren sólo si va a haber varias instrucciones. En otro caso se puede crear el **if** sin llaves:

```
if (expresión lógica)sentencia;
```

```
if($nota>=5){
    echo "Aprobado";
    $aprobados++;
}
```

Sólo se escribe ***Aprobado*** si la variable ***$nota*** es mayor o igual a cinco.

Hay otra posibilidad de sintaxis:

```
if (expresión lógica):
   instrucciones
   …
endif;
```

En lugar de las llaves, cierra el bloque **if** con la palabra **endif**. Los dos puntos tras el paréntesis son obligatorios. Se trata de una forma simplificada (que, ciertamente, hay que usar con cuidado).

Sentencia condicional compuesta

Es igual que la anterior, sólo que se añade un apartado **else** que contiene instrucciones que se ejecutarán si la expresión evaluada por el **if** es falsa. Sintaxis:

```
if (expresión lógica){
   instrucciones a ejecutar si la expresión es verdadera
   …
else{
   instrucciones a ejecutar si la expresión es falsa
   …
}
```

Como en el caso anterior, las llaves son necesarias sólo si se ejecuta más de una sentencia en el bloque.

Ejemplo de sentencia **if-else**:

```
if($nota>=5){
   echo "Aprobado";
   $aprobados++;
}
else {
   echo "Suspenso";
   $suspensos++;
}
```

La forma simplificada de esta sentencia sería:

```
if($nota>=5):
   echo "Aprobado";
   $aprobados++;
else :
   echo "Suspenso";
   $suspensos++;
endif;
```

No se usan las llaves, en su lugar se usan los dos puntos y el cierre con **endif**.

Sentencia condicional múltiple

Permite unir varios **if** en un solo bloque.

```php
if($nota<5){
   echo "suspenso";
   $suspensos++;
}
elseif($nota<7){
   echo "aprobado";
   $aprobados++;
}
elseif($nota<9){
   echo "notable";
   $aprobados++;
}
else{
   echo "sobresaliente";
   $aprobados++;
}
```

Cada sentencia **elseif** permite añadir una condición que se examina si la anterior no es verdadera. En cuanto se cumpla una de las expresiones, se ejecuta el código del **if** o **elseif** correspondiente. El bloque **else** se ejecuta si no se cumple ninguna de las condiciones anteriores.

La versión simplificada sería:

```php
if($nota<5):
   echo "suspenso";
   $suspensos++;
elseif($nota<7):
   echo "aprobado";
   $aprobados++;
elseif($nota<9):
   echo "notable";
   $aprobados++;
else:
   echo "sobresaliente";
   $aprobados++;
}
```

6.2. Sentencia switch

```
switch (expresión lógica){
   case valor1:
      instrucciones del valor 1
      [break]
   [case valor2:
      instrucciones del valor 2
      [break]
   [   .
       .
       .]
   [default:
      instrucciones que se ejecutan si la expresión
      no toma ninguno de los valores anteriores
```

```
        .]
}
```

Los corchetes indican que su contenido es opcional. Es decir, los **break** son opcionales y la cláusula default también.

Esta instrucción se usa cuando tenemos instrucciones que se ejecutan de forma diferente según evaluemos el conjunto de valores posible de una expresión. Cada **case** contiene un valor de la expresión; si efectivamente la expresión equivale a ese valor, se ejecutan las instrucciones de ese **case** y de los siguientes.

La instrucción **break** se utiliza para salir del **switch**. De tal modo que si queremos que para un determinado valor se ejecuten las instrucciones de un apartado **case** y sólo las de ese apartado, entonces habrá que finalizar ese **case** con un **break.**

El bloque **default** sirve para ejecutar instrucciones para los casos en los que la expresión no cumpla ninguno de los valores anteriores.

```
srand(time());
$diaSemana=rand(1,7);
switch ($diaSemana) {
    case 1:
       $dia="Lunes";
       break;
    case 2:
       $dia="Martes";
       break;
    case 3:
       $dia="Miércoles";
       break;
    case 4:
       $dia="Jueves";
       break;
    case 5:
       $dia="Viernes";
       break;
    case 6:
       $dia="Sábado";
       break;
    case 7:
       $dia="Domingo";
       break;
    default:
       $dia="?";
}
```

No usar break para salir del switch puede servir para cosas como:

```
srand(time());
$diaSemana=rand(1,7);
switch ($diaSemana) {
    case 1:
```

```
    case 2:
    case 3:
    case 4:
    case 5:
       echo "Laborable";
       break;
    case 6:
    case 7:
       echo "Festivo";
       break;
    default:
       echo "?";
}
```

Hay una versión simplificada de switch que permite usar los dos puntos al final de la línea del switch y sustituir la llave final del switch por el texto **endswitch**.

6.3. Bucles

Un bucle es un conjunto de sentencias que se repiten mientras se cumpla una determinada condición.

Los bucles agrupan instrucciones las cuales se ejecutan continuamente hasta que una determinada condición que se evalúa sea falsa.

bucle while

```
while (expresión lógica){
   sentencias que se ejecutan si la condición es true
}
```

El programa se ejecuta siguiendo estos pasos:

1. Se evalúa la expresión lógica
2. Si la expresión es verdadera ejecuta las sentencias, sino el programa abandona la sentencia **while**
3. Tras ejecutar las sentencias, volvemos al paso 1

Ejemplo (escribir números del 1 al 100):

```
$i=1;
while ($i<=100){
   echo $i."<br />";
   $i++;
}
```

while tiene también una versión simplificada. El ejemplo anterior quedaría:

```
$i=1;
while ($i<=100):
   echo $i."<br />";
```

```
   $i++;
endwhile;
```

Este tipo de bucles son los fundamentales. Todas las demás instrucciones de bucle se basan en el bucle **while**.

Por ello se comentan con él los dos principales tipos de bucle: de centinela y de contador.

Otra sintaxis posible es:

```
while (expresión lógica):
   sentencias que se ejecutan si la condición es true
endwhile;
```

bucles de contador

Se llaman así a los bucles que se repiten una serie determinada de veces. Dichos bucles están controlados por un **contador** (o incluso más de uno). El contador es una variable que va variando su valor (de uno en uno, de dos en dos,… o como queramos) en cada vuelta del bucle. Cuando el contador alcanza un límite determinado, entonces el bucle termina.

En todos los bucles de contador necesitamos saber:

(1) Lo que vale la variable contadora al principio. Antes de entrar en el bucle.
(2) Lo que varía (lo que se incrementa o decrementa) el contador en cada vuelta del bucle.
(3) Las acciones a realizar en cada vuelta del bucle.
(4) El valor final del contador. En cuanto se rebase el bucle termina. Dicho valor se pone como condición del bucle, pero a la inversa; es decir, la condición mide el valor que tiene que tener el contador para que el bucle se repita y no para que termine.

```
$i=10; //Valor inicial del contador, empieza valiendo 10
while ($i<=200){
/* condición del bucle, mientras i sea menor de 200, el bucle se
repetirá, cuando i rebase este valor, el bucle termina */
   echo $i."<br />"; /*acciones que ocurren en cada vuelta del
bucle en este caso simplemente escribe el valor del contador */
   $i+=10; /* Variación del contador, en este caso cuenta de 10 en
10*/
}

/* Al final el bucle escribe:
10
20
30
…
y así hasta 200
*/
```

bucles de centinela

En este bucle se utiliza la **condición lógica** llamada **centinela**, que puede ser desde una simple variable booleana hasta una expresión lógica más compleja, sirve para decidir si el bucle se repite o no. De modo que cuando la condición lógica se incumpla, el bucle termina.

Esa expresión lógica a cumplir es lo que se conoce como centinela y normalmente la suele realizar una variable booleana.

```
$salir=FALSE; /* En este caso el centinela es una variable
booleana que inicialmente vale falso */
while($salir==FALSE){
/* Condición de repetición: que salir siga siendo falso. Ese es el
centinela.   También   se   podía   haber   escrito   simplemente:
while(!salir) */
   $n=rand(1,500); // Lo que se repite en el
   echo($n); /* bucle: calcular un número aletorio de 1 a 500 y
escribirlo */
   $salir=($n%7==0); /* El centinela vale verdadero si el número
es múltiplo de 7 */
}
```

Comparando los bucles de centinela con los de contador, podemos señalar estos puntos:

- Los bucles de contador se repiten un número concreto de veces, los bucles de centinela no
- Un bucle de contador podemos considerar que es seguro que finalice, el de centinela puede no finalizar si el centinela jamás varía su valor (aunque, si está bien programado, alguna vez lo alcanzará)
- Un bucle de contador está relacionado con la programación de algoritmos basados en series.

Un bucle podría ser incluso mixto: de centinela y de contador. Por ejemplo imaginar un programa que escriba números de uno a 500 y se repita hasta que llegue un múltiplo de 7, pero que como mucho se repite ocho veces.
Sería:

```
$salir = FALSE; //centinela
$i=1; //contador
while ($salir == false && $i<=8) {
   $n = rand(1,500);
   echo $n."<br />";
   $i++;
   $salir = ($n % 7 == 0);
}
```

Bucle do while

La única diferencia respecto a la anterior está en que la expresión lógica se evalúa después de haber ejecutado las sentencias. Es decir el bucle al menos se ejecuta una vez. Es decir los pasos son:

(1) Ejecutar sentencias
(2) Evaluar expresión lógica
(3) Si la expresión es verdadera volver al paso 1, sino continuar fuera del while

```
do{
   instrucciones
}while(expresión lógica);
```

Ejemplo (contar de uno a 1000):

```
$i=0;
do {
    $i++;
    echo $i."<br />";
} while ($i<1000);
```

Se utiliza cuando sabemos al menos que las sentencias del bucle se van a repetir una vez (en un bucle **while** puede que incluso no se ejecuten las sentencias que hay dentro del bucle si la condición fuera falsa, ya desde un inicio).

De hecho cualquier sentencia **do..while** se puede convertir en while. El ejemplo anterior se puede escribir usando la instrucción while, así:

```
int i=0;
$i++;
echo $i."<br />";
while (i<1000) {
    $i++;
    echo $i."<br />";
}
```

Bucle For

Es un bucle más complejo especialmente pensado para rellenar arrays o para ejecutar instrucciones controladas por un contador. Una vez más se ejecutan una serie de instrucciones en el caso de que se cumpla una determinada condición.

```
for(inicialización;condición;incremento){
   sentencias
}
```

Segunda versión:

```
for(inicialización;condición;incremento):
   sentencias
endfor;
```

Las sentencias se ejecutan mientras la condición sea verdadera. Además antes de entrar en el bucle se ejecuta la instrucción de inicialización y en cada vuelta se ejecuta el incremento. Es decir el funcionamiento es:

(1) Se ejecuta la instrucción de inicialización
(2) Se comprueba la condición

(3) Si la condición es cierta, entonces se ejecutan las sentencias. Si la condición es falsa, abandonamos el bloque ***for***
(4) Tras ejecutar las sentencias, se ejecuta la instrucción de incremento y se vuelve al paso 2

Ejemplo (contar números del 1 al 1000):

```
for($i=1;$i<=1000;$i++){
    echo $i."<br />";
}
```

La ventaja que tiene es que el código se reduce. La desventaja es que el código es menos comprensible. El bucle anterior es equivalente al siguiente bucle **while:**

```
$i=1;   //sentencia de inicialización
while($i<=1000) {   //condición
    echo $i."<br />";
    $i++;   //incremento
}
```