Índice

Estructuras de Control

Un programa PHP se ejecuta en principio de forma secuencial, desde la primera instrucción hasta la última y de una en una. Las estructuras de control permiten modificar este flujo, eligiendo entre instrucciones alternativas o repitiendo instrucciones.

Bloques de instrucciones

Para indicar a PHP que varias instrucciones forman un bloque de instrucciones que se ejecutarán secuencialmente (aunque un bloque pueda a su vez contener estructuras de control), se utilizan las llaves **{** y **}**

```
<?php
{
echo "<p>Hola</p>";
echo "<p>Adiós</p>";
}
?>
```

```
<p>Hola</p>
<p>Adiós</p>
```

```
<?php
  { echo "<p>Hola</p>"; echo "<p>Adiós</p>"; }
?>
```

```
<p>Hola</p>
<p>Adiós</p>
```

Se recomienda sangrar las instrucciones que forman el bloque y escribir las llaves en líneas distintas de las instrucciones del bloque.

```
<?php
{
  echo "<p>Hola</p>";
  echo "<p>Adiós</p>";
}
?>
```

```
<p>Hola</p>
<p>Adiós</p>
```

No es necesario escribir un punto y coma (;) después de las llaves y, en general, se recomienda no hacerlo.

```
<?php
{
  echo "<p>Hola</p>";
  echo "<p>¿Qué tal?</p>";
};
echo "<p>Adiós</p>";
?>
```

```
<p>Hola</p>
<p>¿Qué tal?</p>
<p>Adiós</p>
```

```
<?php
{
  echo "<p>Hola</p>";
  echo "<p>¿Qué tal?</p>";
}
echo "<p>Adiós</p>";
?>
```

```
<p>Hola</p>
<p>¿Qué tal?</p>
<p>Adiós</p>
```

No se suelen utilizar bloques cuando no hay estructuras de control. Si se quiere indicar visualmente al programador que unas instrucciones están relacionadas entre sí, basta con dejar una línea en blanco en el código …

```
<?php
  echo "<p>Hola</p>";
  echo "<p>¿Qué tal?</p>";

  echo "<p>Recuerdos a la familia</p>";
  echo "<p>Adiós</p>";
?>
```

```
<p>Hola</p>
<p>¿Qué tal?</p>
<p>Recuerdos a la familia</p>
<p>Adiós</p>
```

... o añadir líneas de comentarios:

```
<?php
  // Saludo
  echo "<p>Hola</p>";
  echo "<p>¿Qué tal?</p>";

  // Despedida
  echo "<p>Recuerdos a la familia</p>";
  echo "<p>Adiós</p>";
?>
```

```
<p>Hola</p>
<p>¿Qué tal?</p>
<p>Recuerdos a la familia</p>
<p>Adiós</p>
```

Sentencia condicional if ... elseif ... else ...

Las construcciones if, else y elseif permiten condicionar la ejecución de un bloque de sentencias al cumplimiento de una condición.

Sentencia condicional if ...

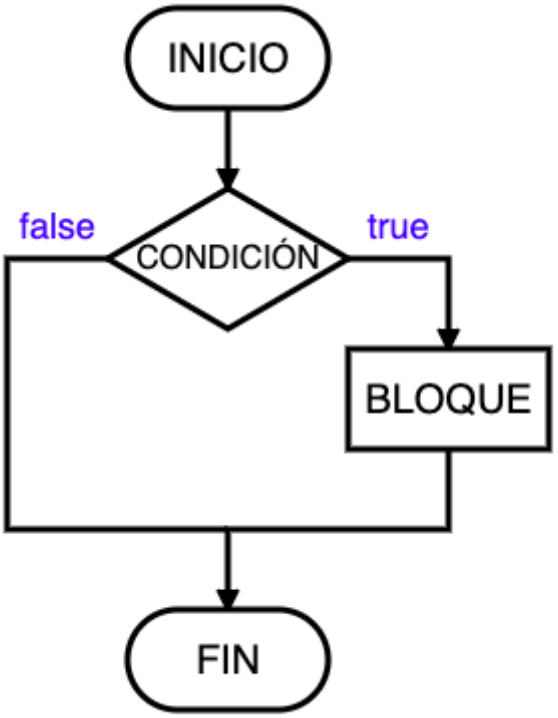
La sintaxis de la construcción **if** más sencilla es la siguiente:

```
if (condición) {
   bloque_de_sentencias_1
}
```

La ejecución de esta construcción es la siguiente:

- La condición se evalúa siempre.
    - Si el resultado es **true** se ejecuta el bloque de sentencias
    - Si el resultado es **false** no se ejecuta el bloque de sentencias.

Diagrama de flujo de la sentencia condicional condicional if ...



Sentencia condicional if ... else ...

La construcción **if** se puede ampliar añadiendo la instrucción **else**:

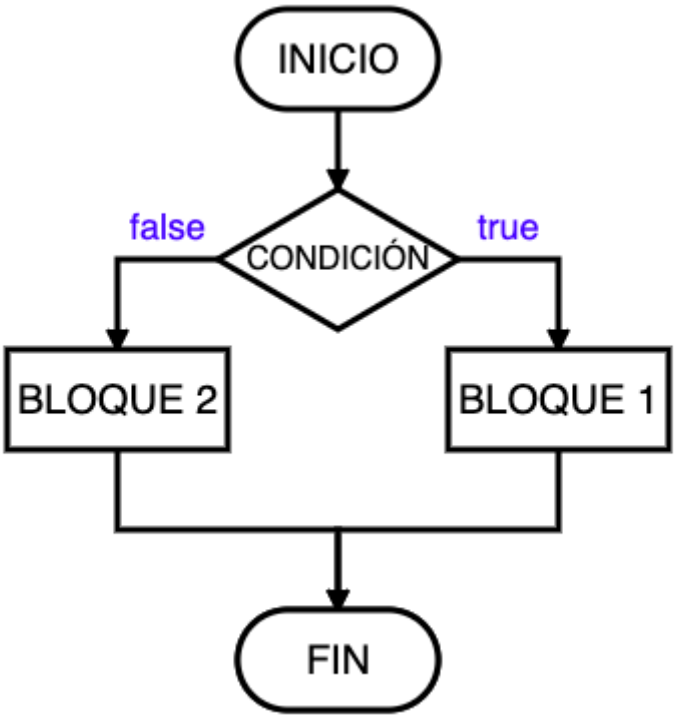
```
if (condición) {
   bloque_de_sentencias_1
} else {
   bloque_de_sentencias_2
}
```

La ejecución de esta construcción es la siguiente:

- La condición se evalúa siempre.
    - Si el resultado es **true** se ejecuta solamente el bloque de sentencias 1
    - Si el resultado es **false** se ejecuta solamente el bloque de sentencias 2.

En cualquier caso, solamente se ejecuta uno de los dos bloques de sentencias.

Diagrama de flujo de la sentencia condicional if ... else ...



Notación abreviada .. ? ... : ...

Si una sentencia condicional **if** ... **else** ... sigue el patrón siguiente:

```
if (condicion_1) {
   $variable = expresion_1;
} else
   $variable = expresion_2;
}
```

... entonces se puede sustituir por la construcción más compacta:

```
$variable = (condicion_1) ? expresion_1 : expresion_2;
```

que se puede escribir en varias líneas para facilitar la legibilidad:

```
$variable = (condicion_1)
    ? expresion_1
    : expresion_2;
```

Sentencia condicional if ... elseif ... else ...

La construcción **if** ... **else** ... se puede extender añadiendo la instrucción **elseif**:

```
if (condicion_1) {
   bloque_de_sentencias_1
} elseif (condicion_2) {
   bloque_de_sentencias_2
} else {
   bloque_de_sentencias_3
}
```

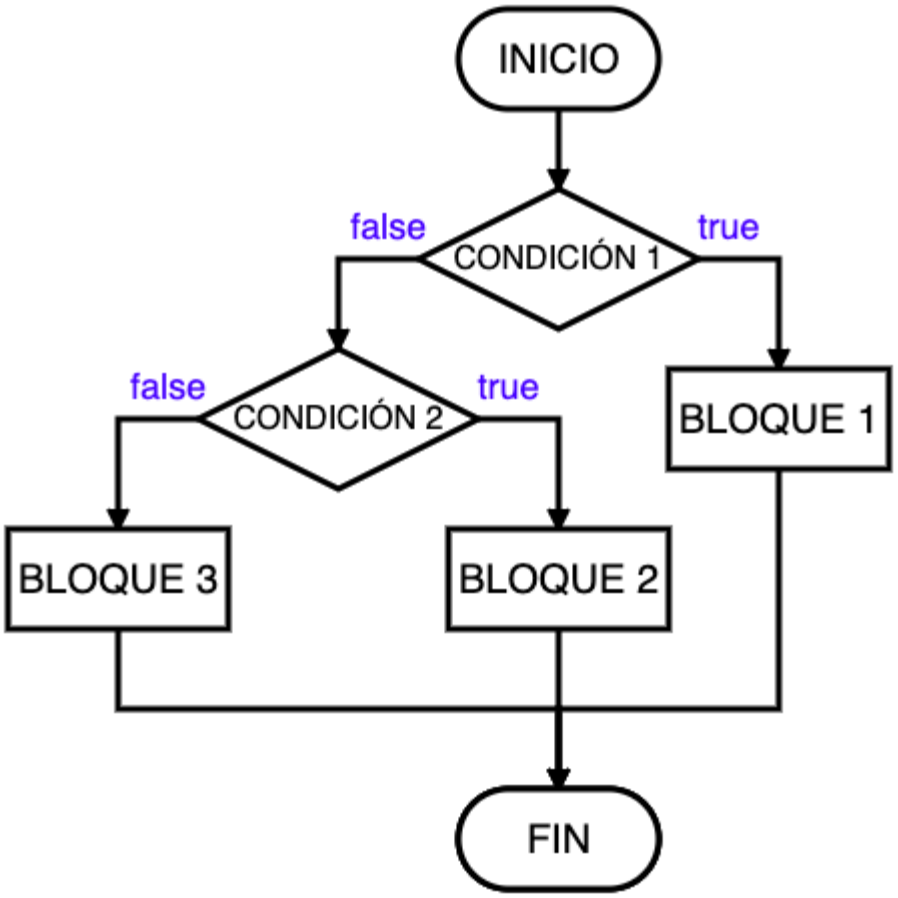
La ejecución de esta construcción es la siguiente:

- La condición 1 se evalúa siempre.
    - Si el resultado es **true** se ejecuta solamente el bloque_de_sentencias_1
    - Si el resultado es **false** se evalúa la condición 2.
        - Si el resultado es **true** se ejecuta solamente el bloque_de_sentencias_2
        - Si el resultado es **false** se ejecuta solamente el bloque_de_sentencias_3

En cualquier caso, solamente se ejecuta uno de los tres bloques de sentencias.
Se pueden añadir tantas instrucciones **elseif** como se desee, teniendo en cuenta que en cualquier caso solamente se ejecuta uno de los bloques de sentencias.

```
if (condicion_1) {
   bloque_de_sentencias_1
} elseif (condicion_2) {
   bloque_de_sentencias_2
} elseif (condicion_3) {
   bloque_de_sentencias_3
...
} elseif (condicion_n) {
   bloque_de_sentencias_n
} else {
   bloque_de_sentencias_n+1
}
```

Diagrama de flujo de la sentencia condicional if ... elseif ... else ...



Sintaxis alternativa sentencia condicional if ... elseif ... else ...

Para facilitar la integración con HTML dispones de una sintaxis alternativa un poco más agradable a la vista.

```
<html>
    <body>
    <?php if (condición): ?>
    // Código que se ejecuta cuando la condición es cierta
    <?php else: ?>
    // Código que se ejecuta cuando la condición no es cierta
    <?php endif; ?>
    </body>
</html>
```

o con un elseif.

```
<html>
```

```
    <body>
    <?php if (condición1): ?>
    // Código que se ejecuta cuando la condición 1 es cierta
    <?php elseif (condición2): ?>
    // Código que se ejecuta cuando la condición 2 es cierta
    <?php else: ?>
      // // Código que se ejecuta cuando ninguna condición es
cierta
    <?php endif; ?>
    </body>
</html>
```

Nota: No se admite la mezcla de sintaxis en el mismo bloque de control.

Sentencia condicional switch

La sentencia switch es equivalente a una construcción **if ... elseif ...** en las que las expresiones son comparaciones de igualdad de la misma condición con valores distintos.
La sintaxis de la sentencia switch es la siguiente:

```
switch (expresion_1) {
  case valor_1:
      bloque_de_sentencias_1
      break;
  case valor_2:
      bloque_de_sentencias_2
      break;
  ...
  case valor_n:
      bloque_de_sentencias_n
      break;
}
```

que es equivalente a :

```
if (expresion_1 == valor_1) {
   bloque_de_sentencias_1
} elseif (expresion_1 == valor_2) {
   bloque_de_sentencias_2
} elseif (expresion_1 == valor_3) {
   bloque_de_sentencias_3
...
} elseif (expresion_1 == valor_n) {
   bloque_de_sentencias_n
```

```
}
```

La sentencia **switch** ejecuta línea por línea (en realidad, sentencia por sentencia). Al principio, ningún código es ejecutado. Solo cuando se encuentra una sentencia **case** cuya expresión se evalúa a un valor que coincida 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 hasta la primera vez que vea una sentencia **break**. Si no se escribe una sentencia **break** al final de la lista de sentencias de un **case**, PHP seguirá ejecutando las sentencias. Por ejemplo:

```
<?php
switch ($i) {
    case 0:
       echo "<p>i es igual a 0</p>";
    case 1:
       echo "<p>i es igual a 1</p>";
    case 2:
       echo "<p>i es igual a 2</p>";
}
?>
```

Aquí, si **$i** es igual a 0, PHP ejecutaría todas las sentencias **echo**. Si **$i** es igual a 1, PHP ejecutaría las últimas dos sentencias **echo**. Se obtendría el comportamiento esperado sólo si **$i** es igual a 2. Por lo tanto, es importante no olvidar las sentencias **break** (aunque es posible que se desee evitar proporcionarla a propósito bajo determinadas circunstancias).
La lista de sentencias para un **case** también puede estar vacía, lo cual simplemente pasa el control a la lista de sentencias para el siguiente **case**.

```
<?php
switch ($i) {
    case 0:
    case 1:
    case 2:
       echo "<p>i es menor que 3 pero no negativo</p>";
       break;
    case 3:
       echo "<p>i es 3</p>";
}
?>
```

Un caso especial es el **default**. Este caso coincide con cualquier cosa que no se haya correspondido por los otros casos. Por ejemplo:

```
<?php
switch ($i) {
```

```
    case 0:
      echo "<p>i es igual a 0</p>";
      break;
    case 1:
      echo  "<p>i es igual a 1</p>";
      break;
    case 2:
       echo  "<p>i es igual a 2</p>";
       break;
    default:
       echo  "<p>i no es igual a 0,1 ni 2</p>";
}
?>
```

Sintaxis alternativa sentencia condicional switch

Advertencia: Cualquier salida (incluyendo espacios en blanco) entre una sentencia switch y el primer case resultará en un error de sintaxis. Por ejemplo, esto no es válido:

```
<?php
  $i = 0;
?>
<?php switch ($i): ?>
  <?php case 0: ?>
       <p>i es igual a 0</p>
  <?php break ?>
  <?php case 1: ?>
        <p>i es igual a 1</p>
  <?php break ?>
  <?php case 2: ?>
        <p>i es igual a 2</p>
  <?php break ?>
  <?php default: ?>
  <?php case 2: ?>
        <p>i no es igual a 0,1 ni 2</p>
<?php endswitch ?>
```

```
Parse error: syntax error, unexpected T_INLINE_HTML " ",
expecting "endswitch" or "case" or "default" in ejemplo.php on
line 4
```

Mientras que lo siguiente es válido, ya que la nueva línea al final después de la sentencia switch es considerada como parte del ?> de cierre, no generando nada entre el switch y el case:

```
<?php
  $i = 0;
?>
<?php switch ($i): ?>
<?php case 0: ?>
       <p>i es igual a 0</p>
<?php break ?>
<?php case 1: ?>
        <p>i es igual a 1</p>
<?php break ?>
<?php case 2: ?>
        <p>i es igual a 2</p>
<?php break ?>
<?php default: ?>
<?php case 2: ?>
        <p>i no es igual a 0,1 ni 2</p>
<?php endswitch ?>
```

Bucle for

La sintaxis del **bucle for** es la siguiente:

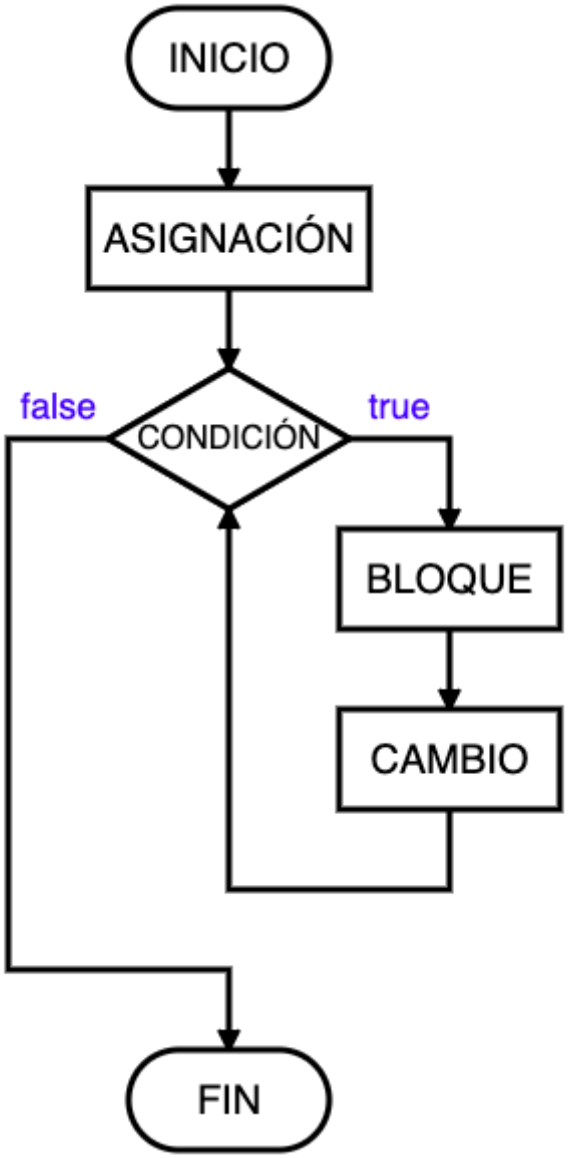
```
for (asignacion_inicial; condicion_continuacion; cambio_variable)
{
   bloque_de_sentencias
}
```

La ejecución de esta estructura de control es la siguiente:

- Se establece el valor inicial de la variable de control definida en la asignación inicial.
- Evalúa la condición de continuación:
    - si el resultado es **true** se ejecuta el bloque de sentencias, se efectúa el cambio de la variable de control y se evalúa nuevamente la condición de continuación;
    - si el resultado es **false** el bucle se termina.

Cuando se programa un bucle for hay que tener cuidado en que la condición de continuación vaya a dejar de cumplirse en algún momento, porque si no es así, el bucle no terminaría nunca.

Diagrama de flujo del bucle for



Ejemplos de bucles

```
<?php
  echo "<p>Comienzo</p>";
  for ($i = 0; $i < 3; $i++) {
    echo "<p>$i</p>";
  }
  echo "<p>Final</p>";
?>
```

```
<p>Comienzo</p>
<p>0</p>
<p>1</p>
```

```
<p>2</p>
<p>Final</p>
```

```
<?php
  for ($contador = 0; $contador < 5; $contador++) {
   echo "<p>$contador</p>";
  }
?>
```

```
<p>0</p>
<p>1</p>
<p>2</p>
<p>3</p>
<p>4</p>
```

Los bucles no tienen por qué ir contando de uno en uno, la expresión de cambio de variable puede ser cualquiera:

```
<?php
  echo "<p>Comienzo</p>";
  for ($i = 1; $i < 6; $i += 2) {
    echo "<p>$i</p>";
  }
  echo "<p>Final</p>";
?>
```

```
<p>Comienzo</p>
<p>1</p>
<p>3</p>
<p>5</p>
<p>Final</p>
```

La variable también puede tomar valores descendientes:

```
<?php
  echo "<p>Comienzo</p>";
  for ($i = 10; $i > 5; $i -= 3) {
    echo "<p>$i</p>";
  }
  echo "<p>Final</p>";
?>
```

```
<p>Comienzo</p>
<p>10</p>
```

```
<p>7</p>
<p>Final</p>
```

Nada impide que la variable de control del bucle se modifique en el cuerpo del bucle, pero eso puede afectar al número de veces que se ejecuta el bucle:

```
<?php
  print "<p>Comienzo</p>";
  for ($i = 1; $i < 6; $i += 2) {
    echo "<p>$i</p>";
    $i = $i + 3;
  }
  echo "<p>Final</p>";
?>
```

```
<p>Comienzo</p>
<p>1</p>
<p>Final</p>
```

Si no se va a utilizar el valor de variable auxiliar, los valores concretos no son importantes, lo único que importa es el número de veces que se va a ejecutar el bucle. En estos casos se pueden escribir distintas asignaciones y condiciones. Por ejemplo, estos tres programas escribirían cinco veces "Hola":

- En este ejemplo, la variable auxiliar tomaría los valores 0, 1, 2, 3 y 4:

```
<?php
  print "<p>Comienzo</p>";
  for ($i = 0; $i < 5; $i++) {
    echo "<p>Hola</p>";
  }
?>
```

```
<p>Hola</p>
<p>Hola</p>
<p>Hola</p>
<p>Hola</p>
<p>Hola</p>
```

- En este ejemplo, la variable auxiliar tomaría los valores 1, 2, 3, 4 y 5:

```
<?php
  print "<p>Comienzo</p>";
  for ($i = 1; $i <= 5; $i++) {
    echo "<p>Hola</p>";
  }
```

```
?>
```

```
<p>Hola</p>
<p>Hola</p>
<p>Hola</p>
<p>Hola</p>
<p>Hola</p>
```

- En este ejemplo, la variable auxiliar tomaría los valores 1, 2, 3, 4 y 5:

```
<?php
  print "<p>Comienzo</p>";
  for ($i = 1; $i < 6; $i++) {
    echo "<p>Hola</p>";
  }
?>
```

```
<p>Hola</p>
<p>Hola</p>
<p>Hola</p>
<p>Hola</p>
<p>Hola</p>
```

La primera forma suele ser la más utilizada en programación y se aconseja acostumbrarse a utilizarla. Sus ventajas son que empieza a contar en 0, como empiezan de forma predeterminada los arrays de índices numéricos, y que el número de iteraciones aparece en la condición de continuación.

Ejemplos de bucles incorrectos

El problema más importante que podemos tener con los bucles es crear un bucle infinito, es decir, un bucle que no se termine nunca. Esto ocurre cuando la condición de continuación se cumple siempre. En estos casos, el servidor no enviaría nunca la página al navegador ni sería capaz de responder a nuevas peticiones. Si se ejecuta un programa que contenga un bucle infinito, a veces se puede recuperar la situación deteniendo la página en el navegador, otras veces se puede detener el servidor Apache desde el panel de control de **xampp**, pero normalmente no queda más remedio que finalizar el proceso (en Windows, habría que hacerlo con el Administrador de Tareas, finalizando el proceso httpd.exe) y reiniciar el servidor.
El bucle infinito se puede provocar de varias maneras:

- si aunque cambie la variable de control, la condición de continuación se cumple siempre, el bucle no terminará nunca.

```
<?php
  echo "<p>Comienzo</p>";
```

```
  for ($i = 10; $i > 3; $i++) {
    echo "<p>$i</p>";
  }
  echo "<p>Final</p>";
?>
```

```
<p>Comienzo</p>
<p>10</p>
<p>11</p>
<p>12</p>
<p>...</p>
```

Nota: Dado que la variable de control es de tipo entero, en realidad el bucle sí que terminaría, ya que cuando la variable alcanzara el valor más alto que puede guardar en una variable de tipo entero, al aumentar pasaría a tener un valor negativo y la condición de continuación dejaría de cumplirse. Pero se trata de un valor tan grande, que a efectos prácticos es como si el bucle no fuera a terminar nunca.

- si la expresión de paso no modifica la variable de control, la condición de continuación no dejaría de cumplirse nunca:

```
<?php
  print "<p>Comienzo</p>";
  for ($i = 1, ; $i < 3; $j++) {
    echo "<p>$i</p>";
  }
  print "<p>Final</p>";
?>
```

```
<p>Comienzo</p>
<p>1</p>
Notice: Undefined variable: j in ejemplo.php on line 3
<p>1</p>
<p>1</p>
<p>...</p>
```

- si la variable de control se modifica dentro del cuerpo del bucle, la condición de continuación podría no dejar de cumplirse nunca, como en el caso siguiente:

```
<?php
  print "<p>Comienzo</p>";
  for ($i = 1; $i < 3; $i++) {
    echo "<p>$i</p>";
    $i--;
  }
```

```
  echo "<p>Final</p>";
?>
```

```
<p>Comienzo</p>
<p>1</p>
<p>1</p>
<p>1</p>
<p>...</p>
```

- Es conveniente que la expresión de continuación se defina mediante desigualdades en vez de por igualdades, para evitar caer en bucles infinitos.

```
<?php
  echo "<p>Comienzo</p>";
  for ($i = 1; $i != 4; $i += 2) {
    echo "<p>$i</p>";
  }
  echo "<p>Final</p>";
?>
```

```
<p>Comienzo</p>
<p>1</p>
<p>3</p>
<p>5</p>
<p>...</p>
```

Otro problema con los bucles puede ser que la expresión de continuación no se cumpla nunca, lo que provoca que el bucle no se ejecute. Este problema es menos grave que el anterior porque el programa no entraría en un bucle infinito, pero seguramente el programa no hará lo que se espera de él.

```
<?php
  echo "<p>Comienzo</p>";
  for ($i = 1; $i > 3; $i++) {
    echo "<p>$i</p>";
  }
  echo "<p>Final</p>";
?>
```

```
<p>Comienzo</p>
<p>Final</p>
```

Sintaxis alternativa bucle for

```
for (expr1; expr2; expr3):
    sentencia
    ...
endfor;
```

Contadores y acumuladores

Entre las aplicaciones de los bucles se encuentran los contadores y acumuladores.

Contador

Se entiende por contador una variable que lleva la cuenta del número de veces que se ha cumplido una condición. El ejemplo siguiente es un ejemplo de programa con contador (en este caso, la variable que hace de contador es la variable cuenta):

```
<?php
  print "<p>Comienzo</p>";
  $cuenta = 0;
  for ($i = 1; $i < 6; $i++) {
    if ($i % 2 == 0) {
      $cuenta += 1;
    }
  }
  echo "<p>Desde 1 hasta 5 hay $cuenta múltiplos de 2.</p>";
  echo "<p>Final</p>";
?>
```

```
<p>Comienzo</p>
<p>Desde 1 hasta 5 hay 2 múltiplos de 2.</p>
<p>Final</p>
```

Detalles importantes:

- En cada iteración, el programa comprueba si la variable de control **$i** es múltiplo de 2.
- El contador se modifica sólo si la variable de control **$**i es múltiplo de 2.
- El contador va aumentando de uno en uno.
- Antes del bucle se debe dar un valor inicial al contador (en este caso, 0)

Acumulador

Se entiende por acumulador una variable que acumula el resultado de una operación. El ejemplo siguiente es un ejemplo de programa con acumulador (en este caso, la variable que hace de acumulador es la variable suma):

```php
<?php
  print "<p>Comienzo</p>";
  $suma = 0;
  for ($i = 1; $i < 6; $i++) {
    $suma += $i;
  }
  echo "<p>La suma de los números de 1 a 5 es $suma.</p>";
  echo "<p>Final</p>";
?>
```

```
<p>Comienzo</p>
<p>La suma de los números de 1 a 5 es 15.</p>
<p>Final</p>
```

Detalles importantes:

- El acumulador se modifica en cada iteración del bucle (en este caso, el valor de la variable de control **$i** se añade al acumulador $suma).
- Antes del bucle se debe dar un valor inicial al acumulador (en este caso, 0)

Bucles anidados

Un bucle anidado es un bucle que se encuentra incluido en el bloque de sentencias de otro bloque. Los bucles pueden tener cualquier nivel de anidamiento (un bucle dentro de otro bucle dentro de un tercero, etc.).
Al bucle que se encuentra dentro del otro se le puede denominar bucle interior o bucle interno. El otro bucle sería el bucle exterior o bucle externo.
En los bucles anidados es importante utilizar variables de control distintas, para no obtener resultados inesperados.

Bucles anidados con variables independientes

Los bucles anidados con variables independientes son los bucles en los que ninguna de las variables de uno de los bucles interviene ni en la condición de continuación ni en el cambio de variable (expresión de paso) de los otros bucles.
Un ejemplo de bucles anidados con variables independientes sería el siguiente:

```php
<?php
  print "<p>Comienzo</p>";
  $suma = 0;
  for ($i = 1; $i < 3; $i++) {  // Bucle exterior
    for ($j = 10; $j < 12; $j++) { // Bucle interior
      echo "<p>i: $i -- j: $j</p>.";
    }
  }
  echo "<p>Final</p>";
?>
```

```
<p>Comienzo</p>
<p>i: 1 -- j: 10</p>
<p>i: 1 -- j: 11</p>
<p>i: 2 -- j: 10</p>
<p>i: 2 -- j: 11</p>
<p>Final</p>
```

En estos casos, si el bucle exterior se ejecuta M veces y el bucle interior se ejecuta N veces cada vez que se ejecuta el bucle exterior, en total el bloque de instrucciones se ejecutará M x N veces.

Bucles anidados con variables dependientes

Los bucles anidados con variables dependientes son los bucles en los que la variable de uno de los bucles interviene en la condición de continuación o en la expresión de paso de los otros bucles.
Un ejemplo de bucles anidados con variables dependientes sería el siguiente:

```
<?php
  echo "<p>Comienzo</p>";
  $suma = 0;
  for ($i = 1; $i < 3; $i++ ) {       // Bucle exterior
    for ($j = 0; $j < $i; $j++ ) { // Bucle interior en el que
      echo "<p>i: $i -- j: $j</p>."; // aparece $i en la
    }                              // condición de continuación
  }
  echo "<p>Final</p>";
?>
```

```
<p>Comienzo</p>
<p>i: 1 -- j: 0</p>
<p>i: 2 -- j: 0</p>
<p>i: 2 -- j: 1</p>
<p>Final</p>
```

En estos casos, es difícil decir a priori cuántas veces se ejecutará el bloque de instrucciones.

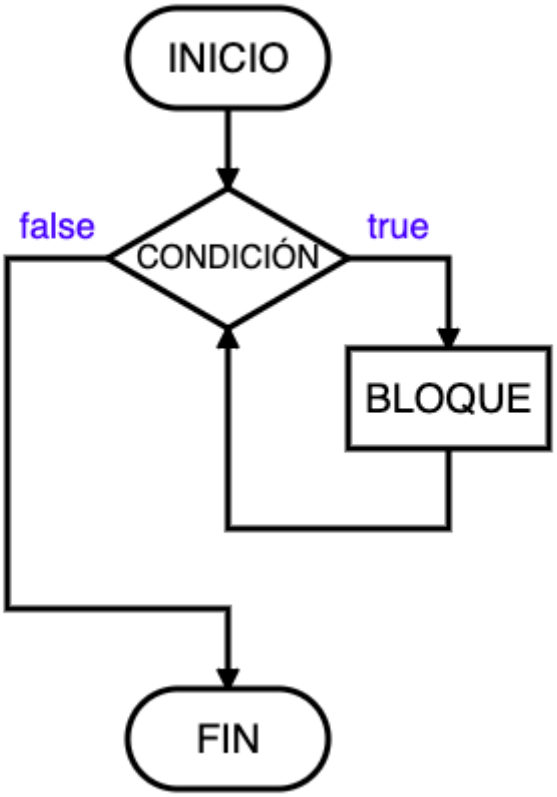
Bucle while

La sintaxis del bucle while es la siguiente:

```
while (condicion) {
   bloque_de_sentencias
}
```

La condición se evalúa al principio de cada iteración: si el resultado es **true** se ejecuta el bloque de sentencias; si el resultado es **false** el bucle se termina.

Diagrama de flujo de while



Cuando se programa un bucle **while** hay que tener cuidado en que la condición deje de cumplirse en algún momento, porque si no es así, el bucle no terminaría nunca.
En los bucles while más sencillos, antes del bucle se inicializa una variable que se evalúa en la condición y dentro del bucle se modifica la variable, como muestra el ejemplo siguiente:

```
<?php
  $i = 0;
  while ($i < 5) {
    echo "<p>$i</p>";
    $i++;
  }
?>
zz
```

```
<p>0</p>
<p>1</p>
<p>2</p>
```

```
<p>3</p>
<p>4</p>
```

Sintaxis alternativa bucle while

```
while (expr):
    sentencias
    ...
endwhile;
```

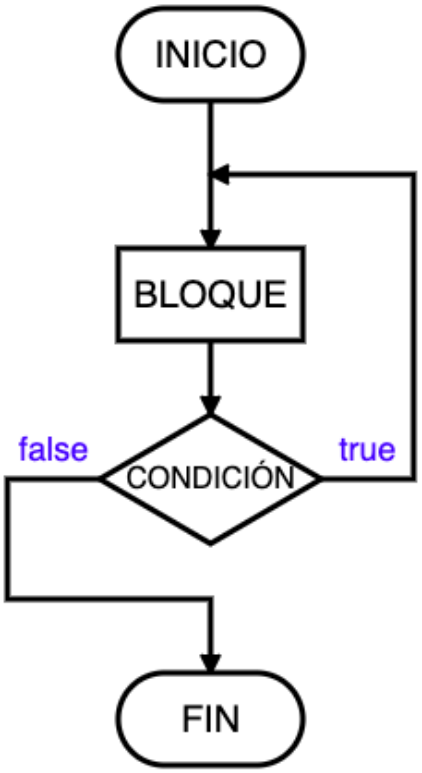
Bucle do ... while

La sintaxis del bucle **do ... while** es la siguiente:

```
do {
   bloque_de_sentencias
} while (condicion)
```

La condición se evalúa al final de cada iteración: si el resultado es **true** se ejecuta el bloque de sentencias; si el resultado es **false** el bucle se termina.

Diagrama de flujo de do … while



El bucle **do ... while** es muy similar al bucle **while**, la principal diferencia es que en el bucle **do ... while** el bloque de sentencias se ejecuta por lo menos una vez mientras que en el bucle **while** depende de si la condición es cierta o no la primera vez que se evalúa.
Cuando se programa un bucle **do ... while** hay que tener cuidado en que la condición deje de cumplirse en algún momento, porque si no es así, el bucle no terminaría nunca.
En los bucles do ... while más sencillos, antes del bucle se inicializa una variable que se evalúa en la condición y dentro del bucle se modifica la variable, como muestra el ejemplo siguiente:

```
<?php
  $i = 0;
  do {
    echo "<p>$i</p>";
    $i++;
  } while ($i < 5)
?>
```

```
<p>0</p>
<p>1</p>
<p>2</p>
<p>3</p>
<p>4</p>
```

NOTA: PHP no proporciona una sintaxis alternativa la estructura do ... while

Bucle foreach

El bucle foreach es un tipo especial de bucle que permite recorrer estructuras que contienen varios elementos (como matrices, recursos u objetos) sin necesidad de preocuparse por el número de elementos.

Problemas al recorrer matrices con el bucle for

En otros lenguajes de programación en los que los índices de las matrices son números naturales correlativos, las matrices se pueden recorrer fácilmente con bucles for. Pero como las matrices de PHP son matrices asociativas y los índices de las matrices no tienen por qué ser valores numéricos correlativos, nos podemos encontrar con problemas. Por ejemplo, podemos hacer involuntariamente referencia a un término inexistente o algunos términos pueden resultar inaccesibles.

- En el ejemplo siguiente, el bucle llega más allá de los límites de la matriz, generando mensajes de error:

```
<?php
  $array = ["a", "bb"];
  echo "<pre>";
```

```
  print_r($array);
  echo "</pre>";

  for ($i = 1; $i <= 3; $i++ ) {
    echo "<p>$array[$i]</p>";
  }
  echo "<p>Final</p>";
?>
```

```
Array
(
    [0] => a
    [1] => bb
)

<p>bb</p>
Notice: Undefined offset: 2 in ejemplo.php on line 8
Notice: Undefined offset: 3 in ejemplo.php on line 8
<p>Final</p>
```

Nota: Este problema se podría evitar fácilmente, puesto que los índices son correlativos. La solución sería en este caso expresar la condición de continuación en función del tamaño del array, utilizando $i < count($array) en vez de $i <= 3.
En el ejemplo siguiente, a la matriz le faltan valores intermedios, por lo que también se generarían mensajes de error al recorrerla:

```
<?php
  $array = [0 => "a", 2 => "bb"];
  echo "<pre>";
  print_r($array);
  echo "</pre>";

  for ($i = 1; $i <= 2; $i++ ) {
    echo "<p>$array[$i]</p>";;
  }
  print "<p>Final</p>";
?>
```

```
Array
(
    [0] => a
    [2] => bb
)

Notice: Undefined offset: 1 in ejemplo.php on line 8
```

```
<p>bb</p>
<p>Final</p>
```

Nota: los avisos de error de este ejemplo se podrían evitar incluyendo una comprobación de existencia del elemento antes de imprimirlo (if isset($array[$i]) ...), aunque tendríamos el problema de no saber a priori hasta qué valor debemos ejecutar el bucle para recorrer todos los valores.
Finalmente, en el ejemplo siguiente, el array no tiene índices numéricos, por lo que sólo se generarían mensajes de error al recorrerla:
En el ejemplo siguiente, a la matriz le faltan valores intermedios, por lo que también se generarían mensajes de error al recorrerla:

```
<?php
  $array = ["uno" => "a", "dos" => "bb"];
  echo "<pre>";
  print_r($array);
  echo "</pre>";

  for ($i = 1; $i <= 1; $i++ ) {
    print "<p>$array[$i]</p>";;
  }
  echo "<p>Final</p>";
?>
```

```
Array
(
    [uno] => a
    [dos] => bb
)

Notice: Undefined offset: 1 in ejemplo.php on line 8
<p>Final</p>
```

Nota: Esta situación no se podría evitar pues los índices del array no son valores numéricos.
La sintaxis del bucle foreach más simple es la siguiente:

```
foreach ($matriz as $valor) {
   bloque_de_sentencias
}
```

```
foreach (matriz as $indice => $valor) {
   bloque_de_sentencias
}
```

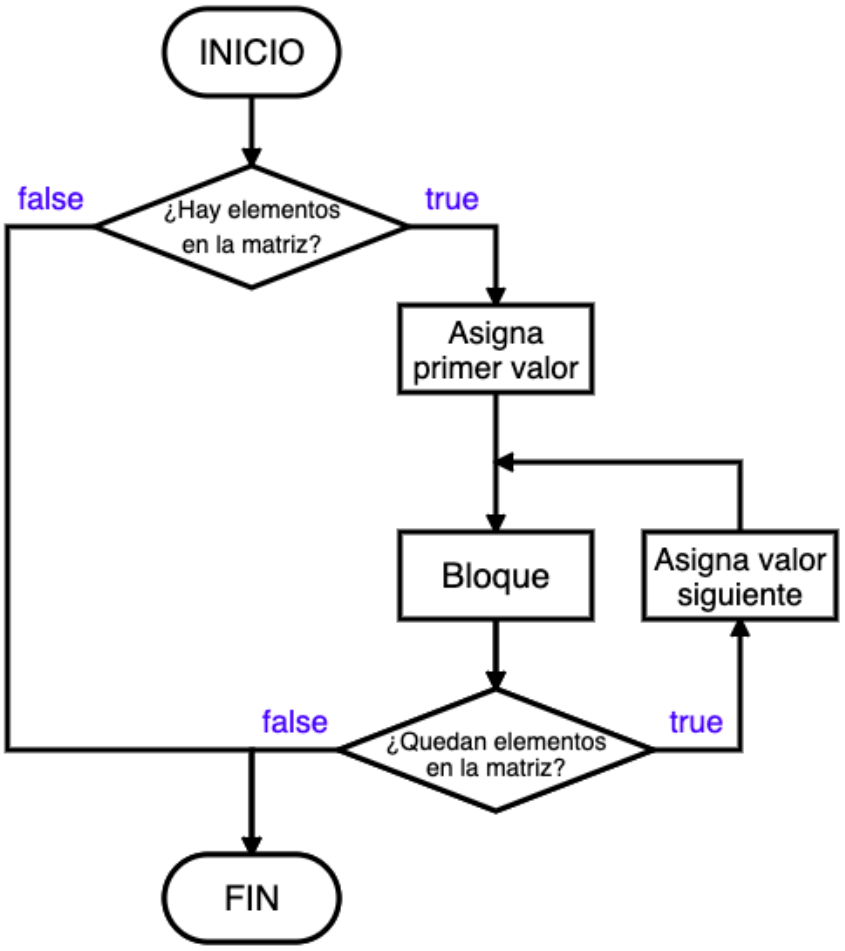
La ejecución de esta estructura de control es la siguiente:

- Si el array no contiene elementos, el bucle no se ejecuta.
- Si el array contiene elementos:
    - Se asigna el primer valor del array a la variable auxiliar (y en su caso, el primer índice a la otra variable auxiliar)
    - Se ejecuta el bloque de sentencias
    - Si el array no contiene más elementos, el bucle deja de ejecutarse.
    - Si el array todavía contiene más elementos:
        - Se asigna el siguiente valor del array a la variable auxiliar (y en su caso, el siguiente índice a la otra variable auxiliar)
        - Se ejecuta de nuevo el bloque de sentencias.

El bucle se ejecuta tantas veces como elementos tiene lel array. En cada iteración, las variables **$indice** y **$valor** van tomando los valores de los índices y del array para ese índice.

Si sólo se necesitan los valores almacenados en el array se puede utilizar tanto la primera como la segunda forma. Si se necesitan tanto los índices como los valores se debe utilizar la segunda forma. Si sólo se necesitan los índices también se debe utilizar la segunda forma.

Diagrama de flujo de foreach



Sintaxis alternativa bucle foreach

```
foreach(expr):
    sentencias
    ...
endforeach;
```

Ejemplo de sintaxis alternativa

```
<?php
  foreach (range(1, 5) as $num):
?>
<p><?php echo $num; ?></p>
<?php endforeach; ?>
```

```
<p>1</p>
<p>2</p>
<p>3</p>
<p>4</p>
<p>5</p>
```

Ejemplos

En el ejemplo siguiente, el bucle recorre un array imprimiendo sus valores

```
<?php
  $array = ["a", "bb", "ccc"];
  echo "<pre>";
  print_r($array);
  echo "</pre>";

  foreach ($array as $valor) {
    echo "<p>$valor</p>";;
  }
  echo "<p>Final</p>";
?>
```

```
Array
(
    [0] => a
    [1] => bb
    [2] => ccc
)
<p>a</p>
<p>bb</p>
```

```
<p>ccc</p>
<p>Final</p>
```

En el ejemplo siguiente, el bucle recorre un array imprimiendo sus índices y valores

```
<?php
  $array = ["uno" => "a", 2 => "bb", "tres" => "ccc"];
  echo "<pre>";
  print_r($array);
  echo "</pre>";

  foreach ($array as $indice => $valor) {
    echo "<p>$indice - $valor</p>";;
  }
  echo "<p>Final</p>";
?>
```

```
Array
(
    [uno] => a
    [2] => bb
    [tres] => ccc
)
<p>uno - a</p>
<p>2 - bb</p>
<p>tres - ccc</p>
<p>Final</p>
```

Si el array es una array vacía, el bucle simplemente no se ejecuta, como muestra el siguiente ejemplo:

```
<?php
  $array = [];
  echo "<pre>";
  print_r($array);
  echo "</pre>";
  foreach ($array as $indice => $valor) {
    echo "<p>$indice - $valor</p>";;
  }
  echo "<p>Final</p>";
?>
```

```
Array
(
)
```

```
<p>Final</p>
```

Recorrer arrays multidimensionales con foreach

Para poder leer un array con más de una dimensión, tendremos que realizar bucles anidados.

Vamos a partir de un array de dos dimensiones.

```
<?php
  $productos = [
    123 => [
      'nombre' => 'Camisa a cuadros',
      'precio' => 29.95,
      'sexo' => 'Hombre'
    ],
    234 => [
      'nombre' => 'Falda manga',
      'precio' => 19.95,
      'sexo' => 'Mujer'
    ],
    345 => [
      'nombre' => 'Bolso minúsculo',
      'precio' => 50,
      'sexo' => 'Mujer'
    ]
  ];
?>
```

Si quisiera mostrar toda la información de los productos.

```
<?php
  foreach ($productos as $producto) {
    echo "<pre>";
    var_dump($producto);
    echo "</pre>";
  }
?>
```

```
Array(3) {

  ["nombre"]=>

  string(16) "Camisa a cuadros"

  ["precio"]=>
```

```
  float(29.95)
  ["sexo"]=>
  string(6) "Hombre"
}
Array(3) {
  ["nombre"]=>
  string(11) "Falda manga"
  ["precio"]=>
  float(19.95)
  ["sexo"]=>
  string(5) "Mujer"
}
Array(3) {
  ["nombre"]=>
  string(16) "Bolso minúsculo"
  ["precio"]=>
  int(50)
  ["sexo"]=>
  string(5) "Mujer"
}
```

Ha iterado 3 veces, y en cada ocasión me devuelve un array. Es lo que hay contenido dentro del primer array, otros arrays. Entonces debo recorrer cada uno de los otros arrays con otro foreach.

```
<?php
  foreach ($productos as $producto) {
    foreach ($producto as $elemento) {
        echo "<p>$elemento</p>";
    }
}
?>
```

```
<p>Camisa a cuadros</p>
<p>29.95></p>
<p>Hombre</p>
<p>Falda manga</p>
<p>19.95></p>
<p>Mujer</p>
<p>Bolso minúsculo</p>
<p>50></p>
<p>Mujer</p>
```