ESTRUCTURAS DE CONTROL

ÍNDICE

MOTIVACIÓN 4
PROPÓSITOS 5
PREPARACIÓN PARA LA UNIDAD 6
1. ESTRUCTURAS DE CONTROL 7
1.1. OPERADORES DE COMPARACIÓN 7
1.2. SENTENCIA IF 8
1.3. CONDICIONAL TERNARIO 9
1.4. OPERADORES LÓGICOS 10
1.5. WHILE Y DO-WHILE 10
1.6. FOR 12
1.7. "BREAK" Y "CONTINUE" 13
1.8. SWITCH 14
2. "INCLUDE/REQUIRE/*_ONCE" 15
CONCLUSIONES 23
RECAPITULACIÓN 24
AUTOCOMPROBACIÓN 25
BIBLIOGRAFÍA 28

MOTIVACIÓN

La herramienta más usada en los lenguajes de programación son las estructuras de control, las instrucciones propias de los lenguajes que permiten hacer preguntas, es decir, comparar operadores con un valor y ejecutar una opción dependiendo del valor.

PROPÓSITOS

Al finalizar esta sección serás capaz de:

- Usar las estructuras de control.
- Usar la inclusión de ficheros.

PREPARACIÓN PARA LA UNIDAD

Todos los ejemplos de este apartado pueden verse si se tiene instalado un servidor Apache o similar en el ordenador. Esto es necesario para poder ver cómo usar las estructuras de control y el manejador de errores en PHP.

1. ESTRUCTURAS DE CONTROL

Las estructuras de control nos permiten controlar el flujo del programa, como que tome decisiones, realizar acciones repetitivas, etcétera, dependiendo de unas condiciones que nosotros establezcamos.

Vamos a ver las principales estructuras de control de PHP que nos permiten condicionar la ejecución de un código al resultado de una **comparación** entre **variables o valores.** Estas estructuras de control se denominan sentencias. Y siempre van a devolver verdadero o falso (*true* y *false*).

Primero haremos un repaso a los operadores de comparación vistos anteriormente:

1.1. OPERADORES DE COMPARACIÓN

Permiten comparar dos valores entre sí devolviendo valores *true* y *false.* Son los siguientes:

- *Igual:* devuelve “true” si valor es igual a valor2 (valor == valor2).
- *Idéntico:* devuelve “true” si valor es igual a valor2 y son del mismo tipo (valor === valor2).
- *Distinto:* devuelve “true” si valor no es igual a valor2 (valor != valor2).
- *menor que:* devuelve “true” si valor es menor que valor2 (valor < valor2).
- *mayor que:* devuelve “true” si valor es mayor que valor2 (valor > valor2).
- *menor o igual:* devuelve “true” si valor es menor o igual que valor2 (valor <= valor2).
- *mayor o igual:* devuelve “true” si valor es mayor o igual que valor2 (valor >= valor2).

Si se compara un número con una cadena, cada cadena se convierte a número antes de realizar la comprobación a nivel numérico. Veamos unos ejemplos:

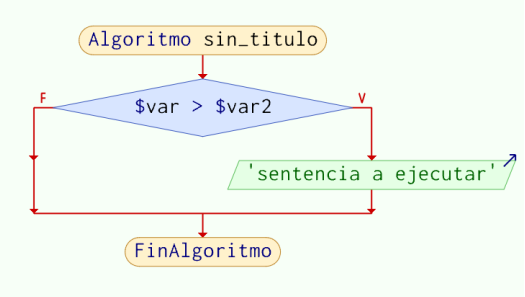
```
<?php
var_dump(0 == "a"); // 0 == 0 -> true
var_dump("1" == "01"); // 1 == 1 -> true
var_dump("10" == "1e1"); // 10 == 10 -> true
var_dump(100 == "1e2"); // 100 == 100 -> true ?>
```

1.2. SENTENCIA IF

La sentencia **if** es la más utilizada. Esta sentencia saca el valor booleano de una expresión dada y, si el valor devuelto es **true**, ejecuta el código que se encuentra dentro. Veamos un ejemplo para ver su estructura:

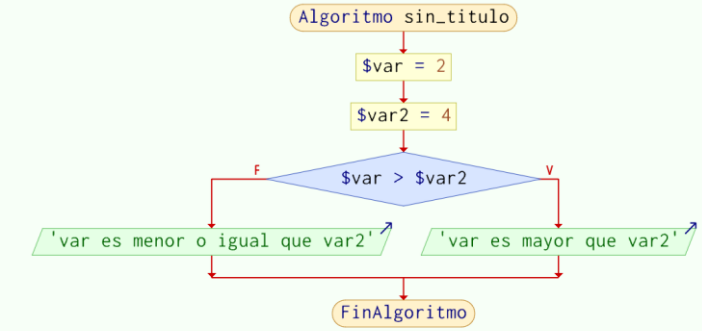
En este ejemplo la sentencia comprueba si el valor de la variable **$var** es mayor del de la variable **$var2.** En caso de ser verdadero, ejecuta el código contenido entre las llaves.

```
if ($var > $var2)
{
  echo “sentencia a ejecutar”;
}
```



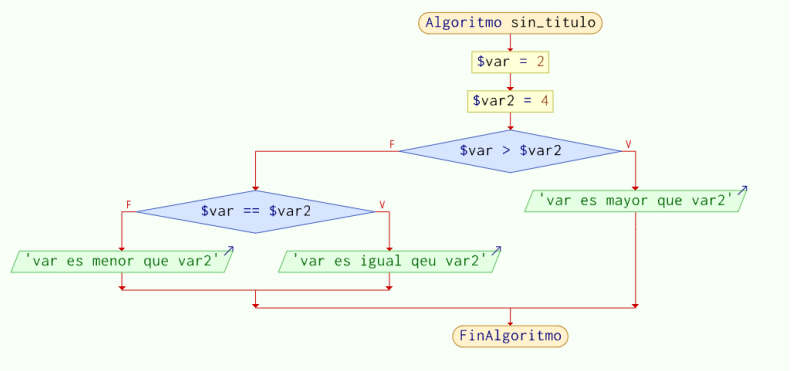
Esta sentencia puede ir seguida de otra denominada **else** y que va siempre después del **if**. Nos permite ejecutar un código diferente si el valor devuelto por la sentencia es **false**. Veamos un ejemplo para entenderlo mejor:

```
$var = 2;
$var2 = 4;
if ($var > $var2){
echo ‘var es mayor que var2”;
}else{
echo ‘var es menor o igual a var2”;
}
```



Por último, existe un método de concatenación de varias sentencias if que se denomina elseif. Nos permite ejecutar una sentencia de código tras un if si el valor devuelto por la sentencia inicial es *false* (siempre que se cumpla la condición de la sentencia del elseif). Veamos un ejemplo para aclararnos:

```
$var = 2;
$var2 = 4;
if ($var > $var2){
echo ‘var es mayor que var2”;
}elseif ($var == $var2){
echo ‘var es igual a var2”;
}else{
echo ‘var es menor o igual que var2”;
}
```



Como puedes ver en el ejemplo: si la condición del if es *false,* comprobamos la del elseif; si esta también es *false*, se ejecutará la del else final. Hay que tener presente que podemos anidar tantos elseif como deseemos.

1.3. CONDICIONAL TERNARIO

Existe una expresión usada en muchos lenguajes de programación que se denomina **condicional ternario.** Se basa en evaluar una primera expresión y, si esta es verdadera, se evalúa una segunda; en caso contrario, se evalúa la tercera. Veamos un ejemplo que lo aclarará:

```
<?php
$primero = 1;
$segundo = 20;
$tercero = 30;
$var = $primero ? $segundo : $tercero;
echo $var; //devolera 20
?>
```

1.4. OPERADORES LÓGICOS

Sirven para preguntar sobre el estado de varias sentencias a la vez. Por ejemplo, para saber si se cumplen dos expresiones o si dos comparaciones se cumplen a la vez. Son las siguientes:

- *Y (and),* devuelve “true” si las dos comparaciones son “true” ($uno and $dos). También podemos ponerlo del a forma ($uno && $dos).
- *O (or),* devuelve “true” si cualquiera de las dos es “true” ($uno or $dos). También podemos ponerlo de la forma: ($uno || $dos).
- *No (not),* devuelve “true” si $uno no es “true” (¡$uno).

Veamos algún ejemplo:

```
$var = (2==2 && 3==4)                    //false
$var = (1 == 3 || 5 == 5)                //true
```

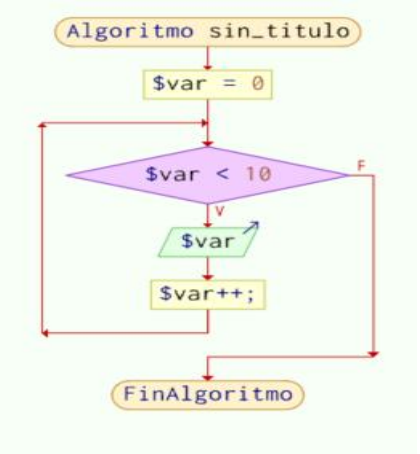
1.5. WHILE Y DO-WHILE

While

Los bucles son otro tipo de sentencias de PHP más complejas. En este caso, los de tipo *while* son los más sencillos, ya que indican a PHP que ejecute un código que se encuentra entre llaves hasta que la condición del *while* devuelva un false. Veamos un ejemplo para ver su sintaxis:

```
$var = 0;
while ($var < 10){

echo $var;
$var ++;
}
```



El ejemplo anterior escribe por pantalla y aumenta en uno una variable **$var** mientras que su valor sea menor que 10.

Do-while

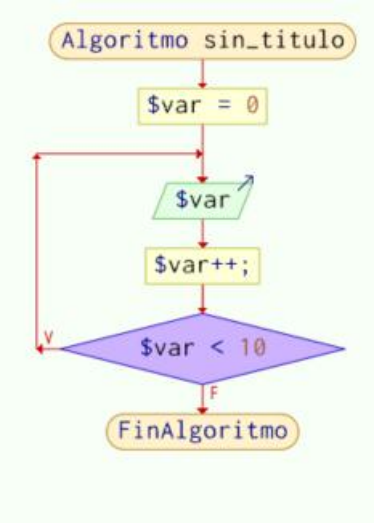
Este bucle es muy parecido al anterior. Lo único que varía es que la sentencia verifica el valor devuelto al final de esta, es decir, el código contenido dentro del bucle se ejecuta al menos una vez antes de que se valide la sentencia y se repita. Un ejemplo:

```
$i = 0;

do {

echo $i;

$i++;
} while ($i > 0);
```



El bucle anterior se ejecutaría solamente una vez, aunque si nos fijamos, el valor de **$i** hace que la sentencia devuelva un “false” desde el principio.

1.6. FOR

For

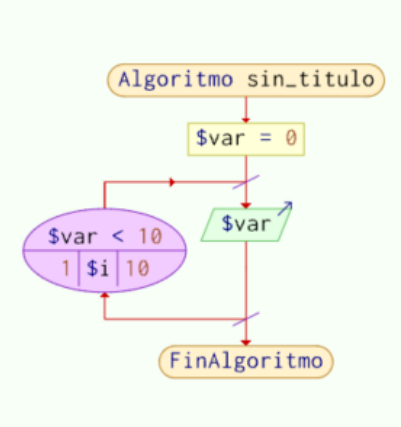
Estos bucles reciben tres parámetros en su creación:

- El primer parámetro se ejecuta una vez al comienzo del bucle y se usa para inicializar a un cierto valor una variable que servirá de contador.
- El segundo parámetro es el que se evalúa en cada vuelta del bucle desde el principio. Si devuelve “true”, se ejecutará el código dentro del bucle; si no, acabará el bucle.
- El tercero se evalúa cada vez que acaba una vuelta del bucle y suele usarse para cambiar el valor de la variable del primer parámetro.

Veamos un ejemplo para aclarar el uso de este bucle:

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

echo $i;
}
```



Con el código anterior escribimos en la página los números del 1 al 10.

1.7. “BREAK” Y “CONTINUE”

Break

Esta sentencia se usa para terminar en cualquier punto del código dentro de una sentencia *for, while* o *switch* la ejecución de la misma. Para ello, basta con llamarla y la ejecución parará y continuará fuera del bucle. Por ejemplo:

```
for ($i = 1;$i < 5;$i++){

  echo $i;
   if ($i == 3){
  break;
  }
}
```

El ejemplo anterior ejecuta un bucle for escribiendo en pantalla los números del uno al tres, y cuando llega al tres sale de la ejecución del for

“Continue”

Esta sentencia se utiliza para, dentro de un bucle, saltarse el resto del código que falte por ejecutar y pasar directamente a la comprobación de la sentencia, no sale del bucle. Veamos un ejemplo:

```
for ($i 1;$i<=5;$i++){

    if ($i == 3){
    continue;
    }
    echo $i;
}
```

El ejercicio anterior muestra los números en pantalla del uno al cinco menos el tres, ya que cuando la variable de la condición vale tres se ejecuta el continue evitando la instrucción echo.

1.8. SWITCH

Esta sentencia es similar a anidar muchos if. Nos sirve para comparar una variable con muchos valores y, según el valor, ejecutar uno u otro código. La comparación podemos realizarla con números o cadenas; cada comparación va precedida de un case y podemos incluir una comparación por defecto denominada default, que se ejecutará en caso de no devolver true en ninguna de las otras. Veamos un ejemplo:

```
$i = 7;

switch ($i) {

case 0: echo "i es igual a 0"; break;
case 1: echo "i es igual a 1"; break;
case 2: echo "i es igual a 2"; break;
default : echo "i no es igual a 0, 1 ni 2";
}
```

En el ejemplo anterior se mostrará en la pantalla el texto “i no es igual a 0 ,1 ni 2”, ya que el resto de comparaciones no se cumplen.

2. “INCLUDE/REQUIRE/*_ONCE”

Include nos permite insertar un archivo completo dentro del código fuente de nuestra página. Este archivo puede contener código de programación web de cualquier tipo: HTML, PHP, JavaScript… Los archivos que se incluyen lo hacen desde la ruta de llamada (si están en una ruta diferente se deberá generar con “..” o “/”) y suelen usarse para cargar trozos de código que se repiten en todas las páginas web como podría ser la cabecera de la página, el menú o un archivo PHP con las funciones comunes.

Cuando se incluye un archivo, el código de su interior se ve afectado con relación a las variables, igual que si estuviera directamente puesto en el archivo padre.

Es extremadamente útil para ahorrarnos trabajo y espacio a la vez que seccionando la página esta se vuelve más clara, puesto que tendrás el código repartido.

- **Include:** puede usarse varias veces para incluir partes de código. Imagínate secciones de una tienda.
- **Include_once:** sirve para incluir una parte del código. Cuando se vuelva a usar será ignorado.
- **Require:** el archivo ha de ser incluido. Si no, el código se PHP se parará, creando un error fatal.
- **Requiere_once:** solo puede ser añadido una vez.

Ejemplo

En este ejemplo verás cómo cargaremos mediante el uso de la sentencia “include” un archivo que contiene la carga de varias variables. Para este ejemplo usaremos dos archivos: **variables.php** (archivo externo).

```
<?php
$mivariable = "oto';
$mivariable2 = 'gato';
$mivariable3 = 'silla';
?>
```

- inicio.php.

```
<html>
<body>
<p>
<?php
$mivariable = 'coche';
$mivariable2 = 'perro';
$mivariable3 = 'mesa';

include 'variables.php';

echo 'resultado: '.$mivariable.'-'.$mivariable2.'-'.$mivariable3;
?>
</p>
</body>
</html>
```

La otra sentencia que nos sirve para cargar archivos es **“require”.** Hace exactamente lo mismo que **“include”,** con la diferencia de que con “include”, en caso de generarse algún error, se producirá una advertencia y el *script* seguirá su ejecución, mientras que usando **“require”** el *script* se parará.

Por último, esta sentencia tiene una análoga llamada **“require_once”,** cuya diferencia con **“require”** reside en que, si el archivo al que apunta ya se ha incluido en algún otro lado de la página, no volverá a incluirse. Include, también tiene su sentencia análoga **“include_once”**.

CONCLUSIONES

Las estructuras de control son una herramienta muy potente del lenguaje PHP. Gracias a ellos podemos controlar la ejecución de código, introducción de datos, generación de contenidos y cualquier cosa que queramos dentro de nuestra página web.

RECAPITULACIÓN

En esta unidad didáctica has aprendido:

- Las estructuras de control de PHP.
- Las sentencias para incluir archivos externos en el PHP.

AUTOCOMPROBACIÓN

1. ¿Qué realiza la sentencia “if”?

a) Saca el valor booleano de una expresión.

b) Ejecuta un código hasta que una condición se cumpla.

c) Ejecuta una condición tantas veces como indiquemos en un contador previo.

d) Todas las respuestas son incorrectas.

2. ¿Cuándo se ejecuta el código contenido dentro de “else”?

a) La expresión dentro de “if” devuelve “true”.

b) La expresión dentro de “if” devuelve “false”.

c) Nunca se ejecuta.

d) Cuando su propia expresión devuelve “true”.

3. Si queremos comprobar una expresión después de un valor “false” de un “if” sin salir de la sentencia, ¿qué usaremos?

a) repeat.

b) elseif.

c) ifelse.

d) forif.

4. **¿Cuál es la diferencia entre “while” y “do while”?**

   a) Que “while” se usa dentro de funciones.

   b) Que “while” ejecuta al menos en una ocasión el código contenido.

   c) Que “while” nunca ejecuta más de cinco veces el código contenido.

   d) Que “do while” ejecuta al menos en una ocasión el código contenido.

5. **El primer parámetro de un “for”:**

   a) Se usa para inicializar el valor de una variable.

   b) Se usa para evaluar cada vuelta del bucle.

   c) Se evalúa en cada vuelta y suele cambiar el valor de la variable que se evalúa para seguir dentro del bucle o salir.

   d) Todas las respuestas son incorrectas.

6. **En caso de querer comparar un valor con una serie de resultados y actuar diferente para cada resultado usando una sola sentencia, ¿qué utilizaremos?**

   a) switch.

   b) elseif.

   c) for.

   d) foreach.

7. **¿Cuál es la diferencia entre “require” y “require_once”?**

   a) Que “require_once” comprueba si ya está cargado el archivo y, de ser así, lo vuelve a cargar.

   b) Que “require_once” carga 11 instancias de un archivo.

   c) Que “require_once” solamente se usa para cargar archivos .php.

   d) Ninguna de las anteriores.

8. **¿Qué términos usa el manejador de excepciones en PHP?**

   a) try …. error.

   b) try …. exception.

   c) try …. catch.

   d) catch …. resume.

9. En una sentencia “for”, si nos encontramos un “continue”, ¿qué nos indica?

a) Que la ejecución sale del “for”.

b) Que la ejecución continúa sin variaciones.

c) Que la ejecución se salta el resto de código dentro del “for” y vuelve a comprobar la expresión del bucle.

d) Ninguna de las anteriores.

BIBLIOGRAFÍA

- CABEZAS, L. M. *PHP 5.* Madrid: Anaya Multimedia, 2010.