UNIDAD II – ESTRUCTURAS DE CONTROL

Sentencias repetitivas y matrices

Objetivos

- Explicar cómo funcionan los ciclos o lazos de PHP.
- Reconocer las diferencias entre las distintas estructuras repetitivas de PHP.
- Distinguir las particularidades de las distintas sentencias repetitivas.
- Determinar la sentencias repetitivas más adecuada para la solución de un problema particular.

Contenidos

1. Estructuras repetitivas.
2. Tipos de sentencias repetitivas.
3. Sentencia *while*
    i. Sintaxis de la sentencia *while*
    ii. Consideraciones sobre el *while*
    iii. Sintaxis alternativa de la sentencia *while*
4. Sentencia *do-while*
    i. Sintaxis de la sentencia do-while
5. Sentencia *for*
    i. Sintaxis de la sentencia *for*
    ii. Consideraciones para utilizar el *for*
    iii. Sintaxis alternativa de la sentencia *for*
6. Sentencias de control de ciclos (*break* y *continue*).

SENTENCIAS REPETITIVAS

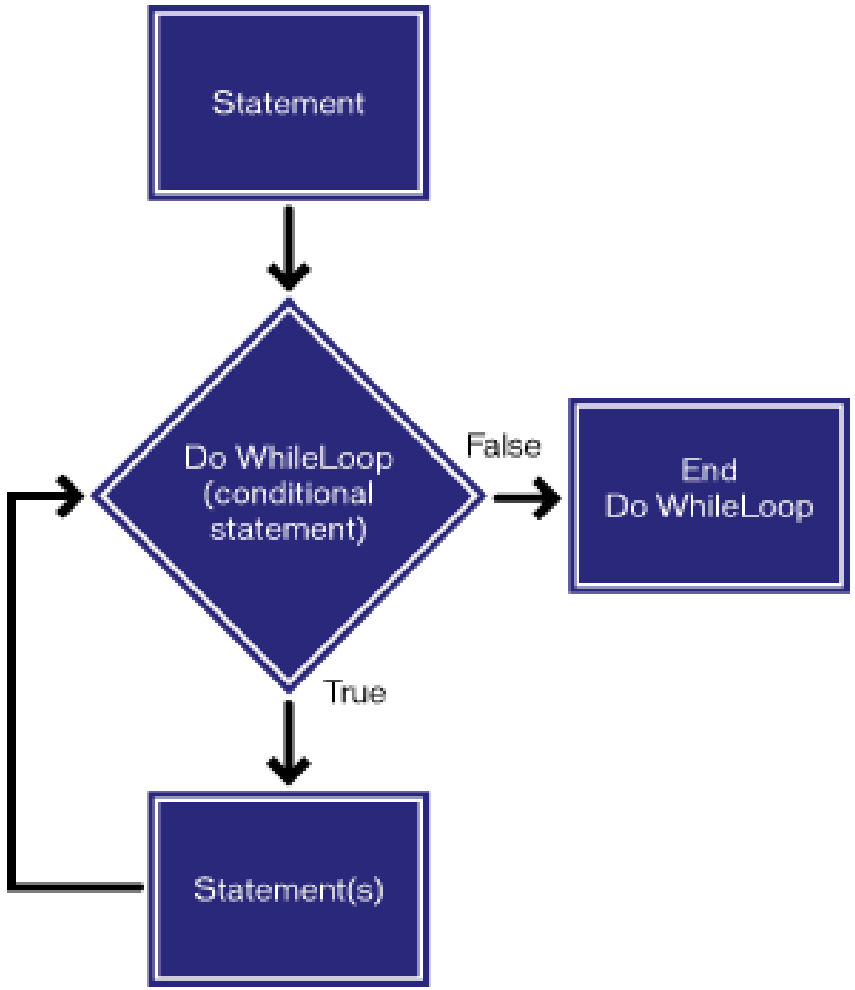
While, Do-While, For, For-Each

Estructuras repetitivas

- Las sentencias repetitivas se utilizan para realizar la iteración repetida de un bloque de instrucciones.
- La idea tras una sentencia repetitiva es ejecutar el mismo bloque de código más de una vez con un propósito definido.
- Este propósito puede ser realizar conteos, acumular resultados, asignar valores o recorrer las propiedades de una matriz, entre otros.

Tipos de sentencias repetitivas

- Al igual que otros lenguajes PHP dispone de varias sentencias que operan como estructuras de control repetitivas. Estas son:
  - while
  - do-while
  - for
  - foreach

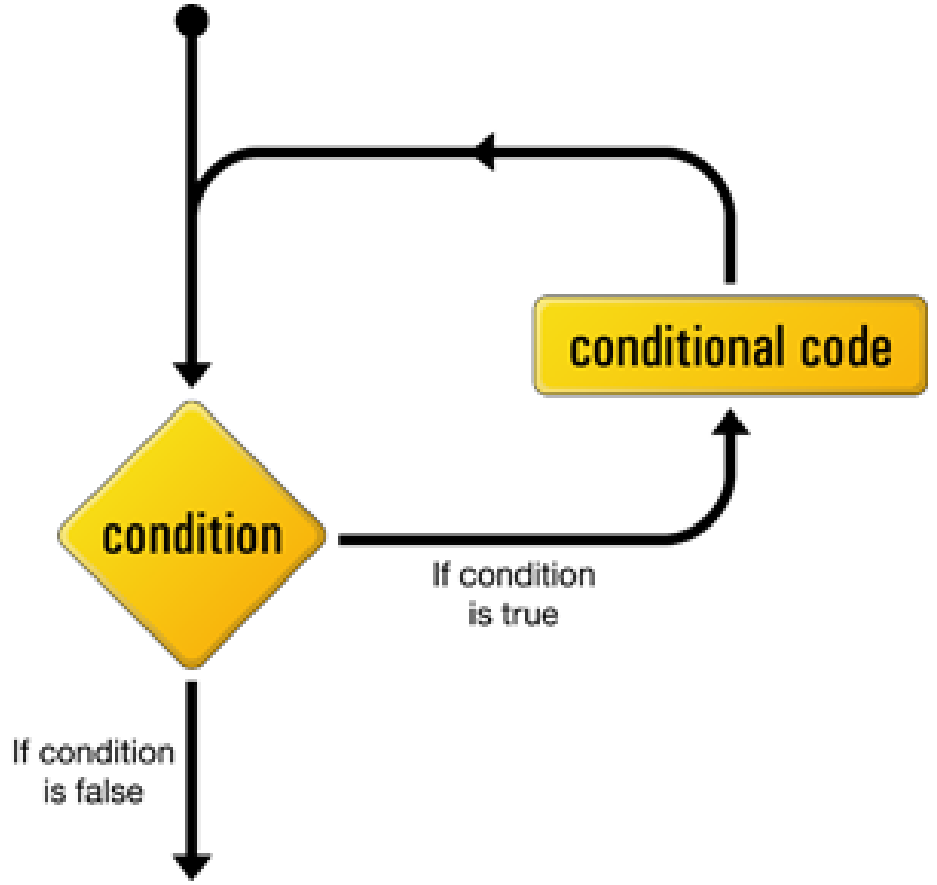


Sentencia *while*

- Esta sentencia es la más simple de las estructuras repetitivas de PHP y de casi cualquier otro lenguaje de programación.
- En esta instrucción se **evalúa una condición** y **si resulta verdadera se ejecuta el bloque de instrucciones** una vez, de lo contrario no. Luego, se vuelve a evaluar la condición y de nuevo si resulta verdadero se vuelve a ejecutar y si resulta falso se termina la instrucción. La instrucción termina hasta que la condición resulte falso.
- La **condición se evalúa antes de cada iteración**. Por lo tanto, **si la comparación resulta falsa desde la primera vez**, el **bloque de instrucciones no se ejecutará ninguna vez.**

Sentencia *while* (II)

- El ciclo ***while*** tiene por propósito repetir un bloque de instrucciones un número indeterminado de veces, lo cual podría ser una vez, varias veces, o incluso, ninguna vez.
- El **bloque de instrucciones se ejecutará en tanto la condición sea** evaluada como **verdadera.**
- En el momento que resulte falsa se termina la ejecución del bloque de instrucciones.



Sintaxis de la sentencia *while*

```
while(condicion)[{]
      //bloque de sentencias;
[}]
```

Donde:

`condicion` es una expresión condicional que es evaluada con el objeto de obtener un valor lógico (verdadero o falso) que determinará si se ejecuta o no el bloque de instrucciones. Si valor devuelto es verdadero se sigue ejecutando, una vez más, el bloque de instrucciones. En caso contrario se termina el ciclo o lazo sin ejecutar dicho bloque.

Consideraciones sobre el *while*

- Lo más importante que hay que recordar acerca de la sentencia ***while*** es que debe proporcionar un mecanismo para que se finalice la ejecución del bloque de instrucciones.
- Esto se puede realizar con un contador que se inicialice afuera del bloque ***while***.
- Dentro de este bloque tiene que incrementarse/decrementarse o simplemente cambiar el valor de este contador para que en determinado momento se vuelva falsa la expresión condicional que debe contener dicho contador.

Sintaxis alternativa de la instrucción *while*

```
while(condicion):
    //bloque de sentencias;
endwhile;
```

Note que en lugar de llaves se utilizan dos puntos (:) para iniciar el bloque. Para finalizar dicho bloque se utiliza la palabra clave ***endwhile***.

Sentencia *do-while*

- Esta instrucción funciona exactamente igual que la sentencia ***while***.
- La única diferencia es que en un ***do-while*** se ejecuta primero el bloque de instrucciones y después se evalúa la condición para determinar si se ejecuta o no el bloque de instrucciones una vez más.
- Esto implica que un ciclo ***do-while*** siempre ejecutará el bloque de instrucciones, al menos una vez.

Sintaxis de la sentencia ***do-while***

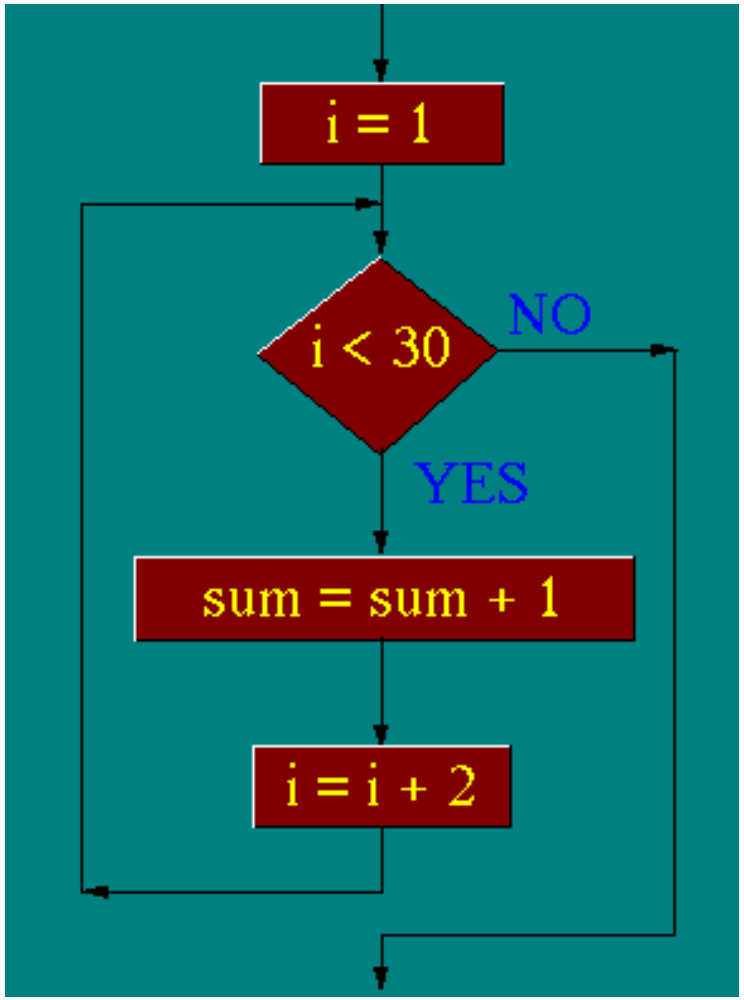
```
do[{]
    //bloque de instrucciones;
[}]while(condicion);
```

Donde:

`condicion` es una expresión condicional que se evaluará para determinar si se realiza el bloque de instrucciones una vez más. De modo que si es verdadera se sigue ejecutando el bloque de instrucciones. En caso contrario no.

Sentencia *for*

- Esta estructura de control es la más compleja de las sentencias repetitivas de PHP.
- En ella se utiliza una variable contador cuyo valor actual es incrementado y verificado en cada iteración para determinar si se ejecuta el bloque de código otra vez.



Sintaxis de la sentencia *for*

```
for([inicializacion];[condicion];[incremento])[{]
    //bloque de instrucciones;
[}]
```

Donde:

Incializacion es una expresión que puede incluir una asignación de valor inicial y/o una declaración de una variable contador. Esta parte del ***for*** se ejecuta una sola vez al inicio del ciclo o lazo. Si se incluye más de una variable debe separarlas con coma.

Ejemplos:

$i=0;

$i=10;

Sintaxis de la sentencia ***for*** (II)

`condicion` es una expresión condicional que se evalúa para determinar si se ejecuta el bloque de instrucciones o no. Si resulta verdadera se ejecuta el bloque de instrucciones, de lo contrario no, haciendo que se finalice el ciclo o lazo. Esta expresión se evalúa antes de cada iteración. Por tanto, es posible que el ciclo o lazo no se ejecute ni una vez. Para formar una expresión condicional debe utilizar operadores de comparación y, posiblemente, lógicos.

Sintaxis de la sentencia *for* (III)

`incremento` se utiliza para incrementar/decrementar el valor de la(s) variable(s) contador utilizadas en el lazo. Pueden ser incrementos/decrementos de uno en uno o de más de uno.

Ejemplos:

```
$i++, $i--, $i +=5, $i-=3, etc
```

Consideraciones para utilizar el *for*

- Una sentencia *for* resulta ser más compleja que una sentencia *while* o *do-while*, principalmente porque en ella se utiliza en su encabezado, como mínimo, una inicialización, una condición y un incremento.
- Sin embargo, en ciertas situaciones, es mucho más cómodo plantear un ciclo repetitivo mediante una sentencia *for*.
- Típicamente, se utiliza una sentencia *for* cuando se sabe el número de veces que se ejecutará el ciclo o lazo.
- También es frecuente utilizar una sentencia *for* cuando se sabe a partir de cuánto comenzar el conteo y en qué valor terminarlo.

Sintaxis alternativa de la sentencia *for*

```
for(inicializacion; condicion; incremento/decremento):
    //bloque de sentencias;
endfor;
```

La diferencia fundamental de la sintaxis alternativa con respecto a la tradicional es la sustitución de la llave de apertura ({) por los dos puntos (:) y de la llave de cierre (}) por la palabra reservada ***endfor***.

Sentencias para control de ciclos o lazos

- PHP soporta el uso de dos sentencias especiales con las que se puede modificar el flujo en la ejecución del bloque de instrucciones dentro de un ciclo, lazo o bucle.
- Estas instrucciones son ***break*** y ***continue***.

Sentencia *break*

- La sentencia ***break*** ya se había mencionado cuando se explicó la sentencia ***switch***.
- En un ciclo o lazo se utiliza de la misma forma; es decir, permite terminar prematuramente la ejecución del bloque de instrucciones dentro de cualquiera de las sentencias repetitivas: ***while***, ***do-while***, ***for*** o ***foreach***.

```
273     public function getParameters()
274     {
275         foreach(array(1,2,3) as $num) {
276             $num1 = $num;
277             break;
278         }
279         return $num1;
280     }
```

Sentencia ***break*** (II)

- Normalmente una sentencia ***break*** se incluirá con una condición que de cumplirse debe producir la terminación temprana del bloque de instrucciones, aun y cuando la expresión condicional, que determina si el lazo se continuará ejecutando, siga siendo evaluada con un valor verdadero.
- Realmente cuando se ejecuta la sentencia ***break***, el control del flujo de programa pasa directamente a la siguiente instrucción después del ciclo repetitivo. La expresión condicional no se evalúa.

Sentencia *continue*

- Una sentencia ***continue*** permite que en la ejecución de uno o varios ciclos en particular no se completen todas las instrucciones del bloque, sino que en determinada situación, que debe evaluarse mediante una condición dentro del ciclo repetitivo, el flujo del programa regrese a la expresión condicional de la estructura repetitiva a evaluar si se ejecuta o no, una nueva iteración.

```
for ($i=1;$i<=10;$i++) {
   if (!($i%2)) {
      continue;
   }
   echo "\$i = $i <br>";
}
```

Demostraciones con ejemplos

MATRICES

Array()

Objetivos

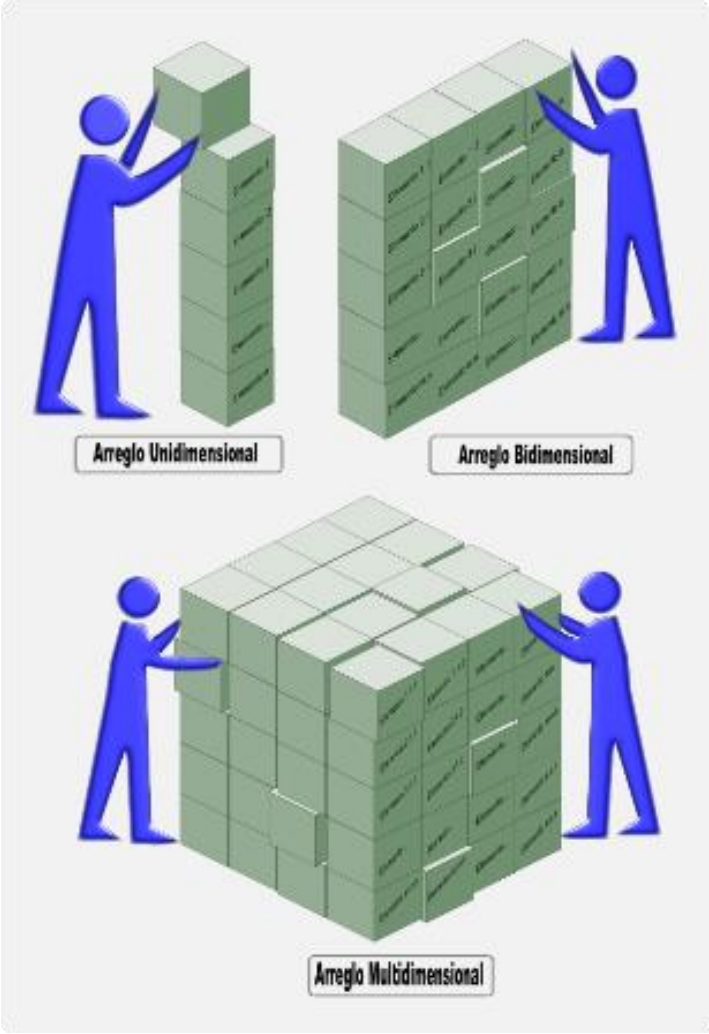
- Lograr un amplio dominio del tipo de dato matriz.
- Desarrollar habilidades para el uso sintácticamente correcto de las matrices.
- Utilizar la sentencia repetitiva foreach adecuadamente con las matrices asociativas.
- Adquirir dominio de las principales funciones para trabajo con matrices de PHP.

Contenido

- Concepto de matriz.
- Creación de una matriz en PHP.
  - Asignación directa.
  - Usando el constructor *array()*.
  - Usando la función *range()*.
- Tipos de matrices en PHP.
- Sentencia repetitiva *foreach*.
- Acceso a los elementos de una matriz con *for* y *foreach*.
- Manipulación de matrices utilizando funciones de PHP.

Concepto de matriz

- Una matriz en PHP y en otros lenguajes de script o de programación, es una estructura de datos compleja que se utiliza para almacenar una lista de valores para manejarlos en una única variable utilizando índices.
- Las matrices en PHP tienen la particularidad de poder almacenar diversos tipos de valores, como enteros, flotantes, cadenas, booleanos e incluso otras matrices en una única variable, a la cual se le asigna un identificador que junto a un índice (numérico o asociativo) servirá para acceder a un elemento de la matriz.



Concepto de matriz

- Cuando examinamos los tipos de datos que maneja PHP, hablamos de variables que almacenan un único valor. Ese tipo de dato que puede almacenar una variable es llamado escalar. En tanto que los tipos de datos que representan múltiples valores sólo pueden ser almacenados en tipos de datos compuestos como las matrices y los objetos.
- Una matriz o arreglo (*array*) es una variable que permite almacenar una colección de datos escalares que pueden ser de distinto tipo.
- Cuando se desea almacenar un único número o una sola cadena, es lógico pensar que con una variable escalar sería suficiente, pero si se necesita almacenar las notas de los alumnos de un curso de varios o muchos alumnos, definir una variables escalar, resultaría poco práctico. En ese caso, conviene utilizar una matriz, que facilitaría hacer ciertos cálculos en combinación con ciclos o lazos.

Concepto de matriz

- En su forma más simple, una matriz contiene una lista de valores que se encuentran relacionados por una misma denominación.
- Cada uno de estos valores constituye un elemento de la matriz y para acceder a estos se asocia un índice de forma individual a cada elemento almacenado.
- En PHP, los elementos de una matriz pueden ser de tipos heterogéneos de datos; es decir, puede existir un arreglo con un elemento entero, uno flotante, una cadena y un valor lógico.
- En tanto que los índices pueden ser **escalares**, en caso de ser números enteros, o **asociativos**, en caso de ser cadenas.

	$elementos[0]	$elementos [1]	$elementos [2]	$elementos [3]
Índice	0	1	2	3
Valor	"Pedro"	"José"	5.4	false

Creación de una matriz en PHP

- En PHP existen diferentes formas de crear e inicializar matrices. Las formas más usuales son:
  - Asignación directa de valores a los elementos.

```
$matriz[0] = valor1;
$matriz[1] = valor2;
```

  - Utilizando la función *array()* de PHP.

```
$matriz = array(val1, val2, …, valN);
```

  - Utilizando la función *range()*.

```
$matriz = range(val_ini, val_fin);
```

Asignación directa

- Realizando asignación directa a los elementos de la matriz:

```
$paises[0] = "El Salvador";
$paises[1] = "Guatemala";
$paises[]  = "Honduras";
$paises[]  = "Costa Rica";
```

- El índice que se le asignará al elemento con valor "Honduras" será 2. Y sucesivamente, al elemento "Costa Rica" se le asignará el índice 3.
- Lo que se concluye es que, al no especificar el índice, PHP automáticamente busca el siguiente valor de índice disponible. Este valor siempre será mayor o igual que cero.

Usando el constructor *array()*

- Si se utiliza el constructor *array()* para crear la matriz:

```
$matriz = array(); //crea una matriz vacía
$matriz = array(5.2, 'Internet', true, 17);
```

- Lo anterior es equivalente a:

```
$matriz[0] = 5.2;          $matriz[1] = 'Internet';
$matriz[2] = true;         $matriz[3] = 17;
```

- También se puede indicar el índice de cada elemento utilizando el constructor *array()*:

```
$precios = array(
                'Llantas' => 39,
                'Aceite' => 3.50,
                'Silicon' => 2
 );
```

Usando la función *range()*

- Utilizando la función *range()*:

```
$valini = 1;
$valfin = 10;
$numeros = range($valini, $valfin[, $step=1]);
```

- La función anterior creará un arreglo con los número del 1 al 10.

```
$alfabeto = range('a', 'z');
```

- La instrucción anterior crearía un arreglo con las letras del alfabeto inglés de la 'a' a la 'z'.

Tipos de matrices

- Las matrices se pueden clasificar de dos formas. De acuerdo a:
  1. El número de índices para acceder a los elementos. Pueden ser unidimensional, bidimensional o, en general, multidimensional.
  2. El tipo de índice. Este puede ser escalar o asociativo.
- Una matriz es **unidimensional** si para acceder a sus elementos se requiere únicamente de un índice.

Tipos de matrices

- La matriz es **multidimensional** si se requiere de más de un índice para acceder a sus elementos. Particularmente, si se necesitan dos índices, se trata de una matriz **bidimensional**, en caso de requerirse tres índices, será una matriz **tridimensional**, y así sucesivamente.

Matrices multidimensionales

- En una matriz multidimensional, los elementos de la matriz son también matrices.
- Para asignar elementos de una matriz bidimensional se pueden utilizar varios métodos. Nuevamente, se puede utilizar la asignación directa:

```
$programacion['lenguaje'][0] = "C++";
$programacion['lenguaje'][1] = "Java";
$programacion['lenguaje'][2] = "Delphi";
$programacion['lenguaje'][]  = "Perl";
```

- Al último elemento se le asignará como segundo índice el número 3, porque es el siguiente índice disponible. En este caso la matriz $programación['lenguaje'], tiene por elementos $programacion['lenguaje'][0], $programacion['lenguaje'][1], etc. Todos ellos matrices.

Matrices multidimensionales

- Otra forma de crear una matriz multidimensional es utilizando el constructor *array()*:

```
$paises = array("El Salvador" => array(
                                "capital" => "San Salvador",
                                "moneda" => "Dólar",
                                "extensión" => 20742
                               ),
                "Guatemala" => array(
                                "capital" => "Guatemala",
                                "moneda" => "Quetzal",
                                "extensión" => 108900
                              ),
                "Honduras" => array(
                                "capital" => "Tegucigalpa",
                                "moneda" => "Lempira",
                                "extensión" => 112492
                             ) );
```

Sentencia *foreach*

- Existe un tipo de sentencia repetitiva especialmente diseñada para recorrer las propiedades de un objeto o los elementos de una matriz.
- Debe tener presente que esta instrucción debe ser utilizada únicamente con matrices (*array*) u objetos (*objects*), nunca con otro tipo de dato. Obtendrá un error si intenta utilizarlo con otro tipo de dato o con una variable sin inicializar.
- Esta sentencia se incluyó a partir de la versión 4 de PHP.

Sintaxis de la sentencia *foreach*

```
foreach($nombre_array as $value){
    [sentencias;]
}
```

La primera forma, recorre el array **$nombre_array** asignando en cada iteración el elemento actual de la matriz a la variable **$value.**

Esta forma de la sentencia *foreach* es ideal para recorrer los elementos de una matriz indexada numéricamente.

```
$colors = array("red","green","blue","yellow");
foreach ($colors as $value) {
  echo "$value <br>";
}
```

Sintaxis de la sentencia *foreach*

```
foreach($nombre_array as $clave => $valor){
 //[sentencias;]
}
```

En la segunda forma, se recorren los elementos del arreglo y se asignan en cada iteración tanto el índice y el valor actual en las variables **$clave** y **$valor**, respectivamente.

Esta forma es ideal para recorrer una matriz asociativa, que posee cadenas como índices para acceder a los elementos de la matriz.

En ambos casos, al terminar una iteración, se desplaza el indicador de posición del array al elemento siguiente hasta llegar al último elemento.

Aspectos a considerar de la sentencia *foreach*

- Debe tener en cuenta que cuando la sentencia *foreach* inicia su ejecución, el puntero interno de la matriz se coloca automáticamente en el primer elemento del *array*.
- Lo anterior significa que no es necesario hacer una llamada a la función *reset()* que fuerza a que el puntero interno vaya al inicio.
- Este puntero interno avanza automáticamente después de cada lectura y que se inicia una nueva iteración por el ciclo o lazo; es decir, no se necesita de una variable contador para controlar el elemento de la matriz que se está leyendo como en los otros ciclos o lazos: *while*, *do-while* y *for*.

Sintaxis alternativa de la sentencia *foreach*

- Al igual que las otras estructuras de control que hemos revisado, la sentencia *foreach* también posee una sintaxis alternativa.

```
foreach($matriz as $value):
   //[Instrucciones foreach];
endforeach;
```

- Para el caso de utilizar sentencia *foreach* con índice y valor:

```
foreach($nombre_array as $clave => $valor):
 //[Instrucciones foreach;]
endforeach;
```

Acceso a los elementos de una matriz

- Se utilizan las estructuras repetitivas para acceder a los elementos de una matriz con el propósito de tener un acceso más eficiente a los elementos, ya sea para asignarles un valor, modificar el que tienen o simplemente obtener el valor para mostrarlo.
- Se pueden utilizar todas las sentencias repetitivas para acceder a los elementos de una matriz. Sin embargo, se utilizan con mayor frecuencia las sentencias *foreach* y *for*, puesto que facilitan el acceso a sus elementos.

Acceso a los elementos de una matriz

- PHP proporciona diversas formas de acceder a los elementos de una matriz.

```
echo $paises['El Salvador']['capital'];
```

- Sin embargo, la forma más eficiente de acceder a un arreglo es utilizando ciclos o lazos.

```
$materias = array("Internet", "Programación", "Redes");
for($i=0; $i<count($materias); $i++){
    echo "Materia ($i+1): {$materia[$i]}<br>\n";
}
```

```
$i = 0;
foreach($materias as $mat){
    echo "Materia " . ++$i . ": " . $mat ."<br>\n";
}
```

Acceso a los elementos de una matriz con *while*

```
<?php
$matriz = array("Cougar","Ford","Toyota","Mitsubishi","Nissan");
$i = 0;
while(each($matriz)):
    echo $matriz[$i];
    $i++;
endwhile;
?>
```

- Utilizar una sentencia *while* para recorrer los elementos de una matriz requiere que se utilice un contador para tener acceso a los elementos individuales de la matriz y el uso de la función *each* para extraer uno por uno los elementos en cada iteración hasta llegar al último, que será cuando la función devuelva un valor *false* que hará terminar el ciclo o lazo.

Acceso a los elementos de una matriz con *for*

```
<?php
$matriz = array("Cougar","Ford","Toyota","Mitsubishi","Nissan");
for($i=0;$i<count($matriz);$i++){
    echo $matriz[$i];
}
?>
```

- La clave para utilizar un *for* para acceder a los elementos de una matriz está en plantear la condición de paro utilizando la función *count()*, que debe tener una matriz como argumento. De este modo la función obtiene el número de elementos de la matriz para utilizarlo como límite del contador $i. Se suele sugerir que se asigne a una variable este cálculo del número de elementos de la matriz y luego utilizar esa variable como condición de paro en la condición de paro.

Acceso a los elementos de una matriz

- En caso de una matriz asociativa, para poder recuperar tanto el índice como el valor del elemento almacenado, debe hacer uso de una sentencia *foreach*, de la siguiente forma:

```
$paises = array("País" => "El Salvador", "Capital" => "San Salvador",
"Moneda" => "Dólar");
 foreach($paises as $indice => $valor){
    echo $indice . ": " . $valor . ";<br>\n";
 }
```

- También se puede acceder a los elementos de una matriz utilizando algunas funciones especiales como *list($key, $value)*.

Acceso a los elementos de una matriz con *foreach*

```
<?php
$matriz = array("Cougar","Ford","Toyota","Mitsubishi","Nissan");
foreach($matriz as $element){
  echo $element;
}
?>
```

- En una sentencia *foreach*, la clave es comprender que se va extrayendo elemento por elemento y se va asignando a la variable colocada después de la palabra reservada **as**. Puede realizarse cualquier operación con el valor actual extraído de la matriz.

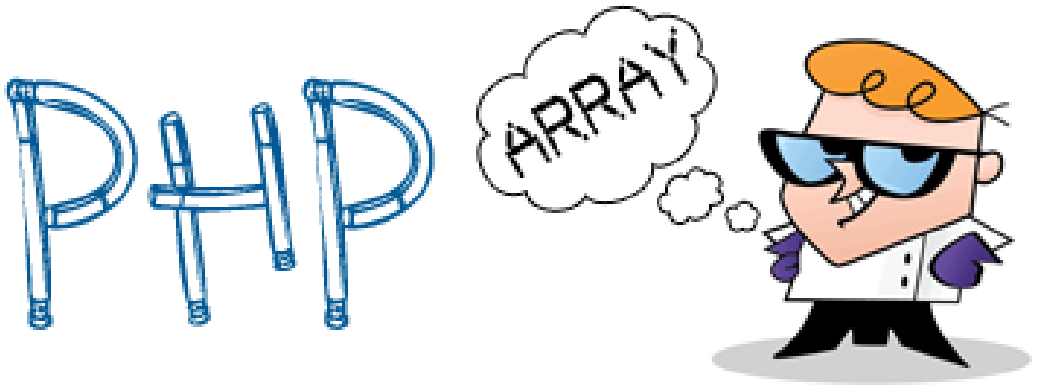
Acceso a los elementos de una matriz

```
$products = array(array('Code' => 'TIR',
                        'Description' => 'Llantas',
                        'Price' => 42),
                  array('Code' => 'OIL',
                        'Description' => 'Aceite',
                        'Price' => 4.25 ),
                  array('Code' => 'WAX',
                        'Description' => 'Cera',
                        'Price' => 2.50)
             );
for($row=0; $row<3; $row++){
    while(list($key,$value) = each($products[$row])){
       echo "$key ($row+1): $value" . "<br>\n";
    }
}
```

Demostraciones con ejemplos

Manipulación de matrices utilizando funciones de PHP

- PHP ofrece un amplio conjunto de funciones para trabajar con matrices.
- Las más útiles serán las funciones para imprimir los datos del *array*, para extraer datos, para recorrer por sus elementos, para posicionar el puntero interno al inicio o al final y para obtener el número total de elementos de la matriz.



Comprobar si una variable es matriz

- Puede determinar si una variable es matriz o no, antes de comenzar a trabajar con ella. Para esto se utiliza la función *is_array()*.
- **`is_array($matriz)`**: Comprueba si la variable pasada como argumento es una matriz o no. Devuelve true si la variable del argumento es matriz y false si resulta que no lo es.

```
$arr = array("Kabul", "Balkh", "Herat",
"Qandahar");

if(is_array($arr)){
    echo "YES";
}else{
    echo "NO";
}
> YES
```

Obtener el número de elementos de la matriz

- PHP proporciona dos funciones para obtener el número de elementos de una matriz, ya sea indexada numéricamente o asociativa. Esas funciones son count() y sizeof().
- **`count($matriz)`**: Devuelve el número de elementos que contiene una matriz y en general cualquier tipo de dato compuesto.
- **`sizeof($matriz)`**: Esta función es prácticamente un alias de la función count(), de modo que igualmente devuelve el número total de elementos de la matriz.

```
$animals = array ('dog', 'cat', 'fish');
echo count($animals);
echo sizeof($animals);
```

```
3
3
```

Buscar un elemento dentro de la matriz

- Puede buscar un valor concreto dentro de los elementos de una matriz.
- **`in_array($valor, $matriz[, $strict])`**: Comprueba si un valor está presente dentro de los elementos de una matriz. De acuerdo a la documentación la comparación es flexible por defecto, de modo que si se compara un valor '5' con 5, la función devolverá true, pero si se establece el tercer argumento opcional de la función en true, la comparación anterior resultaría falsa.

```
$arr = array("Kabul", "Balkh", "Herat",
"Qandahar");

if(in_array("Kabul",$arr)){
    echo "Found";
}else{
    echo "Not Found";
}
> Found
```

Comprobar si una clave o índice existe en la matriz

- Así como se puede determinar si un valor concreto está presente en la matriz, también podemos determinar si una clave o índice está presente en la matriz. Para esto se utiliza la función *array_key_exists()*.
- **`array_key_exists($clave, $matriz)`**: Verifica si la clave o índice pasado como primer argumento está presente en la $matriz pasada como segundo argumento.

```
$a = array ('a' => NULL, 'b' => 2);
echo array_key_exists ('a',$a);
```

Funciones de manejo de puntero o cursor interno de la matriz

- **`current($matriz)`**: Devuelve el valor del elemento situado en la posición actual del puntero interno o cursor de la matriz. Devuelve false cuando apunta al final del arreglo.
- **`key($matriz)`**: Devuelve el índice de la posición actual de la matriz pasada como argumento. Este valor será un número en caso de tratarse de un ***array*** indexado numéricamente o una cadena de caracteres si se trata de un ***array*** asociativo.
- **`next()`**: Avanza el puntero o cursor interno de la matriz hacia el siguiente elemento, devolviendo el valor de dicho elemento. Si la matriz no posee elementos o ya no hay elementos devuelve *false*.

Funciones de manejo de puntero o cursor interno de la matriz

```php
$array = array('foo' => 'bar', 'baz',
'bat' => 2);
reset($array);
while (key($array) !== null) {
   echo key($array).": ".current($array);
   next($array);
}
```

Manipulación de matrices utilizando funciones de PHP

- **`reset($matriz)`**: Sitúa el cursor o puntero interno de la matriz en el primer elemento y devuelve el valor de dicho elemento. Si la matriz está vacía, la función devolverá el valor *false*.
- **`prev($matriz)`**: Devuelve el valor del elemento anterior al actual (si existe) y retrocede al puntero interno una posición. En caso de que el elemento actual sea el primero, devuelve *false*.
- **`end($matriz)`**: Coloca el cursor o puntero interno de la matriz en el último elemento de un ***array*** escalar o asociativo. Devuelve el valor almacenado en el último elemento y si la matriz está vacía en su lugar, devuelverá el valor *false*.

Manipulación de matrices utilizando funciones de PHP

```php
$frutas = array("Manzana", "Fresa", "Uva", "Naranja", "Mandarina");
echo "<ul>\n";
end($frutas);
do{
    $fruta = current($frutas);
    echo "\t<li>\n";
    echo "\t\t" . $fruta . "\n";
    echo "\t</li>\n";
}while(prev($frutas));
echo "</ul>\n";
```

FIN