cipsa.net

PHP & MySQL

Anexo 3.- Estructuras

DISTRIBUIDO POR:

CENTRO DE INFORMÁTICA PROFESIONAL S.L.

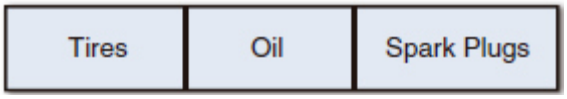
C/ URGELL, 100
08011 BARCELONA
TFNO: 93 426 50 87

C/ RAFAELA YBARRA, 10
48014 BILBAO
TFNO: 94 448 31 33

www.cipsa.net

Matrices

Una matriz es una estructura de datos que permite almacenar en una variable múltiples valores. Mientras que una variable convencional sólo contiene un único valor de un tipo, las matrices permiten almacenar múltiples valores incluyendo incluso otras matrices.



Representación de una matriz con tres cadenas de caracteres por valores.

Cada posición de la matriz se identifica por un valor denominado *índice*. Estos índice permiten acceder a cada valor almacenado en la matriz para obtenerlo o modificarlo.

Tipos de matrices.

PHP define dos tipos de matrices en función del tipo de valores empleados como índices para acceder a los valores

Matrices Escalares

Son aquellas que emplean como índices valores numéricos. El índice de la matriz se corresponde con un valor entero comprendido *0* y *n-1* ambos incluidos donde *n* es el número de valores almacenados en la matriz o *longitud*.

```php
<?php
        // Declaración
        $matriz = Array();
        // Inicialización
        $matriz[0] = "Yuri";
        $matriz[1] = "Roger";
        $matriz[2] = "Ivan";
        $matriz[3] = "Gregory";
        // Visualizacion del primer elemento almacenado.
        echo "Primer elemento: $matriz[0]<br/>";
        // Visualizacion del último elemento almacenado.
        echo "Último elemento: $matriz[3]<br />";
?>
```

También es posible declarar las matrices escalares inicializándolas con sus valores al mismo tiempo:

```php
        // Declaración explícita de matriz.
        $matriz = Array("Yuri", "Roger", "Ivan", "Gregory");
```

En ambos casos, los valores e índices contenidos en *$matriz* serían los mismos:

```
array (size=4)
  0 => string 'Yuri' (length=4)
  1 => string 'Roger' (length=5)
  2 => string 'Ivan' (length=4)
  3 => string 'Gregory' (length=7)
```

Matrices Asociativas

En una matriz asociativa los índices son sustituidos por ***claves***. Las ***claves*** son valores de tipo cadena que se asocian con cada valor almacenado en la matriz permitiendo su obtención y modificación.

```php
<?php
        // Declaración
        $matriz = Array();
        // Inicialización
        $matriz["Francia"] = "Paris";
        $matriz["Italia"] = "Roma";
        $matriz["Inglaterra"] = "Londres";
        $matriz["España"] = "Madrid";
        // Obteniendo la capital de Francia
        echo "La capital de francia es: ".$matriz["Francia"]."<br />";
?>
```

Al igual que en el caso de las matrices escalares también es posible declarar las matrices asociativas indicando sus pares clave-valor al mismo tiempo:

```php
        $matriz = array(
                        'Francia' => 'Paris',
                        'Italia' => 'Londres',
                        'Inglaterra' => 'Londres',
                        'Estaña' => 'Madrid'
        );
```

En ambos casos, el conjunto de claves-valores almacenados en la matriz es el mismo:

```
array
  'Francia' => string 'Paris' (length=5)
  'Italia' => string 'Roma' (length=4)
  'Inglaterra' => string 'Londres' (length=7)
  'EspaÃ±a' => string 'Madrid' (length=6)
```

Matrices multidimensionales

Las matrices multidimensionales son aquellas que en vez de almacenar un valor simple en cada índice o clave, almacenan a su vez otras matrices. En función del número de matrices anidadas pueden ser bidimensionales, tridimensionales…, etc.

En el caso de *una matriz que contiene matrices de valores*, hablamos de una *matriz bidimensional* o *tabla*:

Code	Description	Price
TIR	Tires	100
OIL	Oil	10
SPK	Spark Plugs	4

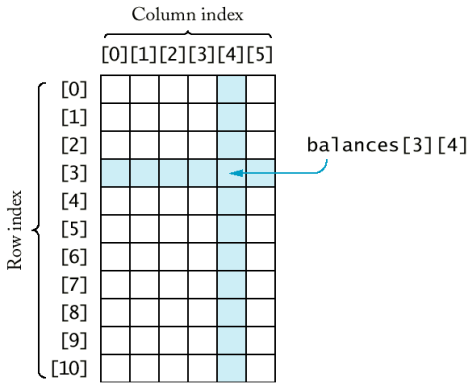
Representación de una matriz bidimensional o tabla.

```php
<?php
        $productos = Array(
                Array('TIR', 'Tires', 100),
                Array('OIL', 'Oil', 10),
                Array('SPK', 'Spark Plugs', 5)
        );

        // OObtencion precio del primer artículo.
        echo $productos[0][2]."<br/>";          // Muestra '100'
        // Obtención descripcion del segundo artículo
        echo $productos[1][1]."<br/>";          // Muestra 'Oil'
        // Obtención código de referencia del tercer artículo.
        echo $productos[2][0]."<br/>";          // Muestra 'SPK'
?>
```

```
array (size=3)
  0 =>
    array (size=3)
      0 => string 'TIR' (length=3)
      1 => string 'Tires' (length=5)
      2 => int 100
  1 =>
    array (size=3)
      0 => string 'OIL' (length=3)
      1 => string 'Oil' (length=3)
      2 => int 10
  2 =>
    array (size=3)
      0 => string 'SPK' (length=3)
      1 => string 'Spark Plugs' (length=11)
      2 => int 5
```

Los valores almacenados en una tabla son accesibles a través de dos índices. El primer índice se refiere a la matriz principal (identifica una fila) y el segundo a un valor en la matriz contenida en tal posición (identifica una columna)



Representación visual del acceso a un valor contenido en una matriz bidimensional o tabla.

Para facilitar el uso de este tipo de matrices también puede declararse una matriz de matrices asociativas:

```php
<?php
        $productos = Array(
                Array('Code' => 'TIR', 'Description' => 'Tires', 'Price' => 100),
                Array('Code' => 'OIL', 'Description' => 'Oil', 'Price' => 10),
                Array('Code' => 'SPK', 'Description' => 'Spark Plugs', 'Price' => 5)
        );

        // OObtencion precio del primer artículo.
        echo $productos[0]['Price']."<br/>";            // Muestra '100'
        // Obtención descripcion del segundo artículo
        echo $productos[1]['Description']."<br/>";      // Muestra 'Oil'
        // Obtención código de referencia del tercer artículo.
        echo $productos[2]['Code']."<br/>";             // Muestra 'SPK'
?>
```

Manejo de matrices

Asignación de valores a una matriz

La asignación de valores a una matriz se realiza con una asignación indicando entre corchetes la posición en la que quiere almacenarse el valor (matriz escalar), o la clave asociada al valor (matriz asociativa):

```
// Asigna el valor "hola" a la posición 2 de la matriz
$matriz[2] = 'hola';
// Asigna el nombre a la clave "12345678W"
$matriz['12345678W'] = 'Roger Petrov';
```

Obtención de valores de una matriz

La obtención del valor almacenado en una posición de la matriz se lleva acabo utilizando la posición o clave con la que se almacenó.

```
// Obtiene el valor en la posición 2 de la matriz.
$valor = $matriz[2];
// Obtiene el valor asociado a la clave "12345678W"
$valor = $matriz['12345678W'];
```

Al solicitar el valor de una posición o clave sin valor asignado se obtiene un valor ***null***. Esto puede comprobarse con la función ***is_null()***. También puede comprobarse si existe un valor en una determinada posición o asociado a cierta clave con ***isset().***

```
// Comprobacion -> ¿Existe algún valor en la posición 10?
if ( isset($matriz[10]) == false ) {
        echo "No hay ningún valor en la posicion 10";
}
```

En el caso de las matrices escalares es posible obtenerse a la vez todos los valores de la matriz mediante la función ***list()***. Esta recupera los valores en una serie de variables pasadas como parámetros.

En el ejemplo, las variables *$ds*, *$d*, y *$m*, se cargan automáticamente con los valores de los tres primeros elementos existentes en la matriz. El número de variables debe coincidir con el de valores almacenados.

```
$matriz = array('Lunes',  20, 'Febrero');
// Recuperacion de valores en variables $ds, $d y $m.
list($ds, $d, $m ) = $matriz;
echo "Hoy es $ds $d $m";
```

El resultado mostrado por pantalla sería:

```
Hoy es Lunes 20 Febrero
```

Recorrido de matrices

Recorrido matrices con bucle '*for*'

El recorrido de los valores de una matriz es una de las operaciones más comunes. En el caso de matrices escalares con posiciones consecutivas: 0,1,2,3,4…., etc; puede emplearse una estructura repetitiva ***for*** y la función ***count()***:

```
// Declaracion de la matriz
$matriz = array( 10, 5, 20, 34, 15 );
// Bucle de recorrido de posiciones
for($i = 0; $i < count($matriz); $i++) {
        echo "matriz[$i] = $matriz[$i]<br />";
}
```

La función ***count()*** devuelve la cantidad de valores almacenados en la matriz. La variable de control (*$i*) recorre todos los valores entre 0 y el anterior a la longitud de la matriz. El resultado mostrado por pantalla es:

```
matriz[0] = 10
matriz[1] = 5
matriz[2] = 20
matriz[3] = 34
matriz[4] = 15
```

Recorrido matrices con bucle '*foreach*'

En el caso de matrices escalares con posiciones no consecutivas, o matrices asociativas puede emplearse la estructura repetitiva ***foreach***. Esta implementa un bucle que se repite para cada uno de los valores de una matriz junto con sus posiciones o claves.

Puede emplearse para recuperar únicamente los valores:

```
foreach( <matriz> as <valor> ) {
        // código de bucle
}
```

También para obtener tanto los valores como sus posiciones o claves:

```
foreach( <matriz> as <clave o posición> =><valor> ) {
        // código de bucle
};
```

Ejemplo: El siguiente código muestra en pantalla cada uno de los valores almacenados en la matriz $matriz indicando su posición.

```
$matriz = array();
$matriz[1] = 1;
$matriz[2] = 4;
$matriz[5] = 25;
$matriz[10] = 100;
// Bucle de recorrido de matriz
foreach( $matriz as $clave => $valor) {
        echo "posición: $clave => $valor.</br>";
}
```

El resultado mostrado sería:

```
posición: 1 => 1.
posición: 2 => 4.
posición: 5 => 25.
posición: 10 => 100.
```

El mismo código nos valdría para obtener las claves y valores de una matriz asociativa:

```php
$matriz = array(
                'ID'=>'1AB223',
                'Name'=>'Roger',
                'Telf'=>918398493);
// Bucle de recorrido de matriz
foreach( $matriz as $clave => $valor) {
        echo "posición: $clave => $valor.</br>";
}
```

El siguiente código mostraría por pantalla:

```
posición: ID => 1AB223.
posición: Name => Roger.
posición: Telf => 918398493.
```

Las matrices multidimensionales se recorren empleando varias estructuras *foreach* embebidas. Los siguientes códigos son válidos para recorrer y mostrar todos los valores contenidos de una tabla de 10 x 10 valores:

```php
// Recorrido de matriz bidimensional mediante bucles for.
// Bucle de recorrida de matriz principal
for($f = 1; $f <= 10; $f++ ) {
        // Bucle de recorrido de matriz anidada en posición de la matriz principal.
        for( $c = 1; $c <= 10; $c++ ) {
                echo "<td>".$matriz[$f][$c]."</dr>";
        }
}
// Recorrido de matriz bidimensional mediante foreach
// Bucle de recorrido de matriz principal
foreach( $matriz as $submatriz ){
        // Bucle de recorrido de matriz anidada
        foreach( $submatriz as $valor){
                echo "<td>".$valor."</td>";
        }
}
```

Funciones de navegación en matrices

PHP define una serie de funciones que permiten navegar por los valores de una matriz hacia delante o hacia detrás, o saltar directamente al primero o al último. Para ello cada matriz posee un *cursor* que apunta a un valor de la misma por defecto el primero.

Las siguientes funciones permiten mover el cursor y obtener el valor apuntado:

- ***current()*** → Devuelve el valor apuntado por el cursor.
- ***key()*** → Devuelve la posición/clave del valor apuntado por el cursor.
- ***next()*** → **Mueve el cursor a la siguiente posición/clave** y devuelve el valor almacenado. En caso de no haber siguiente valor *devuelve un valor lógico falso*.
- ***prev()*** → **Mueve el cursor a la anterior posición/clave** y devuelve el valor almacenado. En caso de no haber anterior valor *devuelve un valor lógico falso.*
- ***end()*** → **Mueve el cursor a la última posición/clave** y devuelve su valor.
- ***reset()*** → **Mueve el cursor a la primera posición/clave** y devuelve su valor.

```php
$transport = array('pie', 'bici', 'coche', 'avión');

$valor = current($transport); // 'pie';
$valor = next($transport);    // 'bici';
$valor = next($transport);    // 'coche';
$valor = prev($transport);    // 'bici';
$valor = end($transport);     // 'avión';
$valor = reset($transport);   // 'pie'
```

El siguiente código muestra la implementación de una función mostrar() que recibe como parámetro una matriz y muestra todas sus posiciones/claves y valores empleando las funciones de navegación vistas.

```php
function mostrar( $matriz ) {
        // Comprobacion. Tiene la matriz un primer elemento?
        if (reset($matriz) !== false ) {
                do {
                        // Visualizacion del valor y clave del elemento apuntado.
                        echo "[".key($matriz)."] -> ".current($matriz)."</br>";
                        // Desplazamiento al siguiente elemento, si existe.
                } while ( next($matriz) !== false);
                echo "END<br/>";
        } else {
                echo "MATRIZ VACIA<br />";
        }
}
```

(*) En el caso de funciones que retornan cualquier tipo de valor (mixed), deben emplearse los operadores de comparación con tipo (===), y (¡==) para comprobar si devuelven un valor lógico cierto/falso. Empleando los operadores convencionales (== / ¡=), se corre el peligro de identificar como falso un valor numérico 0, y como cierto cualquier otro valor.

Funciones de manipulación de matrices

Obtención de valores / claves

La función ***array_slice()*** extrae múltiples valores de una matriz en otra. Esta función posee dos sintaxis:

```
<matriz_resultado> = array_slice(<matriz>, <pos_origen>);
```

Devuelve una matriz con todos los elementos de la matriz de origen contenidos desde la posición indicada en *<pos_origen>* hasta el último.

```
<matriz_resultado> = array_slice(<matriz>, <pos_origen>, <longitud>);
```

Devuelve una matriz con la cantidad de valores indicados en *<longitud>* a partir del valor contenido en la posición *<pos_origen>.*

```php
$matriz = array(0,10,20,30 );
// Devuelve en $submatriz dos valores a partir de la posición 2 incluida de $matriz
$submatriz = array_slice($matriz, 2,2);
var_dump( $submatriz );
```

El resultado mostrado por pantalla es:

```
array (size=2)
  0 => int 20
  1 => int 30
```

En el caso de las matrices asociativas a veces resulta útil poder obtener todas las claves y los valores en dos matrices diferentes. PHP dispone para ello de las funciones ***array_values()*** y ***array_keys().***

La sintaxis de ambas funciones es idéntica:

```
<matriz> = array_values( <matriz> );

<matriz> = array_keys( <matriz> );
```

```php
$datos = array('A'=>0, 'B'=>1, 'C'=>2, 'D'=>3, 'E'=>4, 'F'=>5);

// Asigna a $claves una matriz con todas las claves de $datos
$claves = array_keys($datos);

// Asigna a $valores una matriz con todo los valores de $datos
$valores = array_values($datos);
```

Búsqueda de elementos.

En multitud de ocasiones es necesario saber si un valor está contenido en algún elemento de una matriz, o la posición que ocupa en la misma. Estas operaciones pueden realizarse perfectamente mediante un bucle ***foreach***, o un bucle ***do-while*** empleando las funciones de procesado de matrices.

PHP ofrece sin embargo una serie de funciones que permiten averiguar si un determinado valor está en algún elemento de una matriz, y si así es; en qué posición o qué clave lleva asociado.

La función ***in_array()*** permite conocer si el valor dado como argumento está contenido en la matriz. El primer parámetro *($needle)* es el valor a buscar y el segundo parámetro (*$haystack*) la matriz en la que buscarlo. El último parámetro (*$strict*) es opcional e indicar si debe comprobarse también el tipo del valor buscado:

```
bool in_array ( mixed $needle , array $haystack [, bool $strict = FALSE ]
```

La función ***array_key_exists()*** permite conocer si existe algún valor almacenado en una matriz asociativa (*$array*) con la clave indicado en el parámetro (*$key*):

```
bool array_key_exists ( mixed $key , array $array )
```

La función ***array_search()*** permite obtener la posición o clave asociada al valor indicado como parámetro. En caso de no existir retorna un valor lógico falso. Los parámetros son idénticos a los de la función ***in_array()***:

```
mixed array_search ( mixed $needle ,
                     array $haystack [, bool $strict = false ] )
```

Ejemplo: Sea la siguiente función asociativa:

```
$personas = array(
        "28394485G" => "Jhonny Mendez",
        "45980234H" => "Roger Petrov",
        "34389430J" => "Yuri Ivanovich",
        "49083984L" => "Raul Gothasy");
```

La siguiente función ***mostrarDNI()*** recibe la matriz anterior y un nombre como parámetro y muestra el DNI de la persona o el mensaje “No existe” si no hay nadie registrado con el DNI indicado.

```
function mostrarDNI( $nombre, $matriz ) {
        if ( in_array($nombre, $matriz) === true ) {
                $dni = array_search($nombre, $matriz, true);
                echo "El DNI es : ".$dni."<br />";
        } else {
                echo "NO existe";
        }
}
```

Filtrado de elementos.

En muchas ocasiones cuando se trabaja con matrices es necesario realizar una selección sobre los valores de una *matriz de origen* pasando aquellos que cumplen una determinada condición a *una matriz de resultados*. Para esto es necesario realizar una selección o ***filtrado***. Para ello se emplea la función ***array_filter()***:

```
array array_filter ( array $array
                     [, callable $callback
                     [, int $flag = 0 ]] )
```

El primer parámetro (*$array*) es la matriz cuyos valores se quieren filtrar. El segundo parámetro (*$callback*) es una *función de evaluación* que determina qué valores pasan a la matriz resultado. Esta función se ejecutará por cada valor almacenado en la matriz. El último parámetro (*$flag*) determina qué recibe la función de evaluación según los siguientes valores:

- ***Valor predeterminado (0)*** → La función de evaluación recibe cada valor almacenado en la matriz de origen.
- ***Constante ARRAY_FILTER_USE_KEY*** → La función evaluación recibe cada clave/posición almacenada en la matriz.
- ***Constante ARRAY_FILTER_USE_BOTH*** → La función de evaluación recibe tanto cada valor como la clave o posición asociada.

Ejemplo: Sea la siguiente matriz de personas con el nombre como clave y su edad como valor:

```
$personas = array(
                "Jhonny" => 20,
                "Roger" => 25,
                "Yuri" => 17,
                "Artyom" => 26,
                "Vasili" => 29,
                "Ivan" => 23
);
```

El siguiente código devuelve tanto la clave (el nombre) como el valor (la edad) de todas aquellas personas mayores de 25 años:

```
// Función de evaluación
function filtro( $valor ) {
        return $valor >= 25;
}

$resultados = array_filter($personas, "filtro");
```

```
array (size=3)
  'Roger' => int 25
  'Artyom' => int 26
  'Vasili' => int 29
```

La función de evaluación recibe cada valor almacenado en la matriz y retorna un valor lógico indicando si pasa a la matriz de resultados (*true*) o no (*false*).

Ejemplo: Sea la siguiente matriz de personas con el DNI como clave y su nombre como valor:

```
$personas = array(
                "28394485G" => "Jhonny Mendez",
                "45980234J" => "Roger Petrov",
                "34389430J" => "Yuri Ivanovich",
                "49083984L" => "Raul Gothasy");
```

El siguiente código devuelve tanto la clave (el DNI) como el valor (el nombre) de todas aquellas personas cuyo DNI termina por la letra "J":

```
// Función de evaluación
function filtro( $clave ) {
        return substr($clave, -1) == "J";
}

$resultados = array_filter($personas, "filtro", ARRAY_FILTER_USE_KEY);
```

```
array (size=2)
  '45980234J' => string 'Roger Petrov' (length=12)
  '34389430J' => string 'Yuri Ivanovich' (length=14)
```

En este caso, la función de evaluación recibe como parámetro la clave de cada valor.

Ejemplo: Sea la siguiente tabla compuesta por una matriz asociativa con el DNI cada persona como clave, y una matriz escalar con su nombre y edad como valor:

```
$personas = array(
                "28394485G" => array( "Jhonny Mendez", 20),
                "45980234J" => array("Roger Petrov", 25),
                "34389430J" => array("Yuri Ivanovich", 19),
                "49083984L" => array("Raul Gothasy", 39)
);
```

El siguiente código devuelve la clave (DNI), y valor (nombre y edad) de todas aquellas personas con DNI terminado en "J" y su edad mayor o igual a 20 años:

```
// Función de evaluación
function filtro( $valor, $clave ) {
        return substr($clave, -1) == "J" && $valor[1] > 20;
}

$resultados = array_filter($personas, "filtro", ARRAY_FILTER_USE_BOTH);
```

En este último caso, la función de evaluación recibe tanto el valor como la clave o posición asociada de cada valor.

Mapeo de elementos

El mapeo de elementos consiste en aplicar una transformación a cada valor de una matriz o varias matrices almacenando el resultado en otra matriz. La matriz resultante contendrá el mismo número de elementos que las matrices de origen, pero no necesariamente los mismos valores o claves.

PHP dispone de la función ***array_map()***.

```
array array_map ( callable $callback
                  , array $array1
                 [, array $... ] )
```

La función recibe como parámetros una *función de mapeo* encargada de retorna un valor para la matriz de resultado por cada valor de las matrices de origen. El resto de parámetros son las matrices de origen.

Ejemplo: Sea la siguiente tabla que almacena el nombre y edad de una serie de personas guardando su DNI como clave.

```
$personas = array(
                "28394485G" => array("nombre" => "Jhonny Mendez", "edad" => 20),
                "45980234J" => array("nombre" => "Roger Petrov", "edad" => 25),
                "34389430J" => array("nombre" => "Yuri Ivanovich", "edad" => 16),
                "49083984L" => array("nombre" => "Raul Gothasy", "edad" => 39)
);
```

El siguiente código muestra cómo obtener una matriz asociativa en la que por cada DNI se obtiene un valor lógico indicando si el individuo es mayor de edad:

```
// Función de mapeo
function mapeo( $valor) {
        return $valor[“edad”]>18;
}

$resultados = array_map("mapeo", $personas);
```

```
array (size=4)
  '28394485G' => boolean true
  '45980234J' => boolean true
  '34389430J' => boolean false
  '49083984L' => boolean true
```

La función de mapeo “*mapeo()*” recibe un único parámetro por el que se recibe cada valor de la matriz dada *$personas* dada como argumento a la función ***array_map()***.

Ejemplo: Sean las siguientes tablas con valores:

```
$bases = array(2,4,8,7,6,2,3,6,1,7);
$potencias = array(2,2,4,5,3,2,3,5,6);
```

El siguiente código obtiene una matriz con los resultados de elevar cada valor de la matriz *$bases* a la potencia indicada en la misma posición de la matriz *$potencias*.

```php
// Función de mapeo
function mapeo( $base, $pot) {
        return pow($base, $pot);
}

$resultados = array_map("mapeo", $bases, $potencias);
```

```
array (size=10)
  0 => int 4
  1 => int 16
  2 => int 4096
  3 => int 16807
  4 => int 216
  5 => int 4
  6 => int 27
  7 => int 7776
  8 => int 1
  9 => int 1
```

En este caso, la función de mapeo "*mapeo()*" recibe dos parámetros; por cada una de las matrices de origen dadas como argumento a la función ***array_map()***. El parámetro $*base* recibe cada valor de $*bases*, y el parámetro $*pot* cada valor de $*potencias*.

En caso de no indicarse ninguna función de mapeo (indicando *null*), se devuelve una matriz de matrices con los valores correlativos de todas las matrices de origen:

```php
$resultados = array_map(null, $bases, $potencias);
```

```
array (size=10)
  0 =>
    array (size=2)
      0 => int 2
      1 => int 2
  1 =>
    array (size=2)
      0 => int 4
      1 => int 2
  2 =>
    array (size=2)
      0 => int 8
      1 => int 4
  3 =>
    array (size=2)
      0 => int 7
      1 => int 5
  4 =>
    array (size=2)
      0 => int 6
      1 => int 3
  5 =>
    array (size=2)
      0 => int 2
      1 => int 2
  6 =>
    array (size=2)
      0 => int 3
      1 => int 3
  7 =>
    array (size=2)
      0 => int 6
      1 => int 5
  8 =>
    array (size=2)
      0 => int 1
      1 => int 6
  9 =>
    array (size=2)
      0 => int 7
      1 => null
```

Modificación de elementos

La transformación de elementos permite aplicar una función para modificar los valores de una matriz. Para ello PHP define la función ***array_walk():***

```
bool array_walk ( array &$array
                  , callable $callback
                 [, mixed $userdata = NULL ] )
```

La función de retrollamada debe incluir dos parámetros para la clave y el valor de cada elemento de la matriz pasado este último como referencia para ser modificable.

Ejemplo: El siguiente código muestra una matriz convencional cuyos valores son elevados al cuadrado por la función *array_walk()* empleando la función *potencia()*.

```
// Matriz de valores
$valores = [1,2,3,4,5,6];

 // Funcion de retrollamada
 function potencia( &$valor, $clave ) {
        // Eleva al cuadrado cada valor en la matriz
        $valor = $valor * $valor;
 }

var_dump($valores);

array_walk($valores, "potencia");

var_dump($valores);
```

```
C:\xampp7\htdocs\prueba\inicio.php:6:
array (size=6)
  0 => int 1
  1 => int 2
  2 => int 3
  3 => int 4
  4 => int 5
  5 => int 6

C:\xampp7\htdocs\prueba\inicio.php:16:
array (size=6)
  0 => int 1
  1 => int 4
  2 => int 9
  3 => int 16
  4 => int 25
  5 => int 36
```

La función *potencia()* recibe un primer parámetro $*valor* con el valor de la matriz, y un segundo parámetro $*clave* con la clave correspondiente. El primer parámetro es pasado por referencia, por lo que su modificación modifica el valor de la matriz.

Comparación de funciones *array_map(), array_filter(),* y *array_walk().*

- La función ***array_map()*** genera una nueva matriz a partir de los valores de una o varias matrices de origen. Esta función no modifica las matrices de origen. La matriz de destino tiene un valor por cada valor de las matrices de origen.

- La función ***array_filter()*** genera una nueva matriz filtrando los valores de otra matriz de origen. La matriz resultante tendrá parte de los valores de la matriz de origen.

- La función ***array_walk()*** modifica los valores de una matriz de origen sin modificar su número. No añade valores ni elimina valores, sólo los modifica.

Cálculo de valores

El cálculo de valores consiste en aplicar una operación a una matriz que devuelve como resultado un valor, por ejemplo; el sumatorio de todos sus valores, el máximo, el mínimo, la media, o el nº de veces que aparece un determinado valor.

Para todo este tipo de operaciones puede emplearse la función ***array_reduce()***.

```
mixed array_reduce ( array $array
                   , callable $callback
                  [, mixed $initial = NULL ] )
```

El primer parámetro (*$array*) es la matriz a procesar. El segundo parámetro (*$callback*) es una función de cálculo. Esta función se invoca por cada valor recibiendo el propio valor y el resultado retornado por la anterior ejecución. El tercer parámetro (*$initial*) es opcional e indica el valor recibido en la primera ejecución de la función de cálculo.

Ejemplo: Supóngase la siguiente matriz de valores:

```
$valores = array(2,4,8,7,6,2,3,6,1,7);
```

El siguiente código calcula el sumatorio de los valores contenidos en la matriz:

```
function sumar( $resultado, $valor) {
        // Al resultado anterior le suma el del elemento actual
        $resultado += $valor;
        return $resultado;
}
// Devuelve el sumatorio de todos los valores
$resultado = array_reduce($valores, "sumar", 0);
```

La función de cálculo (*sumar*) recibe cada valor de la matriz en el parámetro ($valor) y el parámetro ($resultado) con el valor retornado por la anterior ejecución de la función, al que se le suma el valor actual. El valor inicial es 0.

El siguiente código devuelve el valor más grande contenido en la matriz.

```
function maximo( $resultado, $valor) {
        // Se asigna el valor actual si es mayor que el recibido
        $resultado = ($valor > $resultado ) ? $valor : $resultado;
        return $resultado;
}

// Devuelve el máximo de todos los valores contenidos
$resultado = array_reduce($valores, "maximo", $valores[0]);
```

En este caso la función de cálculo (máximo) pasa a la siguiente ejecución de la propia función el valor mayor de entre los dos parámetros recibidos. El valor inicial se toma del primer elemento de la matriz.

Funciones anónimas (*closures*) .

PHP permite emplear funciones como argumentos. Este es el caso de algunas funciones de manejo de matrices como *array_reduce()*, *array_filter()*, *array_map()* o *array_walk()*. Para simplificar el uso de funciones que requieren funciones como argumentos, desde PHP 5.3 pueden emplearse funciones anónimas o *closures*:

Ejemplo: El siguiente código muestra el uso de la función *array_reduce()* para obtener el sumatorio de todos los valores de la matriz *$valores* empleando la función *sumar()* como argumento:

```
$valores = array(2,4,8,7,6,2,3,6,1,7);

function sumar( $resultado, $valor) {
        // Al resultado anterior le suma el del elemento actual
        $resultado += $valor;
        return $resultado;
}
// Devuelve el sumatorio de todos los valores
$resultado = array_reduce($valores, "sumar", 0);
```

Este código puede simplificarse declarando el código de la función *sumar()* como una función anónima en la propia llamada a *array_reduce()*.

```
$valores = array(2,4,8,7,6,2,3,6,1,7);

// Devuelve el sumatorio de todos los valores
$resultado = array_reduce($valores,
                function( $resultado, $valor) {
                        // Al resultado anterior le suma el del elemento actual
                        $resultado += $valor;
                        return $resultado;
                }, 0);
```

Uso de variables globales en funciones anónimas.

Para emplear una variable externa en una función anónima puede emplearse la palabra clase *use* indicando el nombre de la variable a emplear:

Ejemplo: El siguiente código muestra el uso de la función *array_walk()* para multiplixar todos los valores de la matriz *$valores* por el valor de la variable global *$producto*.

La variable *$producto* es accesible para la función anónima si es indicada en la partícula *use($producto)*.

```
// Matriz de valores
$valores = array(2,4,8,7,6,2,3,6,1,7);
// Valor de producto
$producto = 2;
array_walk($valores,
                function( &$valor, $clave ) use ($producto) {
                        $valor *= $producto;
                }
);
```

Modificación de matrices

Eliminación de un valor en la matriz

La eliminación de un elemento en una matriz puede realizarse mediante la función ***unset()*** pasando como argumento la posición o clave del elemento a eliminar.

Ejemplo: El siguiente código elimina el valor en la posición 2 de la matriz $valores. Fíjate que la propia posición 2 deja de existir:

(*) Al eliminar un valor se elimina también su posición/clave. No se produce una reordenación de los valores.

```php
// Matriz de valores
$valores = array(2,4,8,7,6,2,3,6,1,7);
// Eliminación del valor en la posición 2
unset($valores[2]);
```

Antes:

```
array (size=10)
  0 => int 2
  1 => int 4
  2 => int 8
  3 => int 7
  4 => int 6
  5 => int 2
  6 => int 3
  7 => int 6
  8 => int 1
  9 => int 7
```

Después: (*No existe posición n.2)

```
array (size=9)
  0 => int 2
  1 => int 4
  3 => int 7
  4 => int 6
  5 => int 2
  6 => int 3
  7 => int 6
  8 => int 1
  9 => int 7
```

Extracción de valores de una matriz.

La extracción de un valor de la matriz implica su obtención *y eliminación de la matriz.* Esta operación puede realizarse para obtener el elemento situado en la primera o en la última posición de la matriz según la función empleada:

Para extraer *el primer valor de la matriz* se emplea la función: ***array_shift()***. La función recibe como único parámetro la matriz origen y devuelve el valor extraído.

```php
$valores = array("valor_0", "valor_1", "valor_2");
var_dump($valores);

// Extrae el primer valor de la matriz y lo retorna.
$valor = array_shift($valores);
echo "Valor extraido es $valor</br>";

var_dump($valores);
```

La función ***array_shift()*** provoca el reordenamiento de los elementos al extraer el primer valor en matrices escalares:

```
array (size=3)
  0 => string 'valor_0' (length=7)
  1 => string 'valor_1' (length=7)
  2 => string 'valor_2' (length=7)

Valor extraido es valor_0

C:\xampp7\htdocs\prueba\inicio.php:8:
array (size=2)
  0 => string 'valor_1' (length=7)
  1 => string 'valor_2' (length=7)
```

En el caso de las matrices asociativas, la extracción del primer elemento con la función ***array_shift()*** no provoca un reordenamiento de los valores:

```
        $valores = array("k1"=>"v1",
                         "k2"=>"v2",
                         "k3"=>"v3");
        var_dump($valores);
        // Extrae el primer valor de la matriz y lo retorna.
        $valor = array_shift($valores);
        echo "Valor extraido es $valor</br>";
        var_dump($valores);
```

```
array (size=3)
  'k1' => string 'v1' (length=2)
  'k2' => string 'v2' (length=2)
  'k3' => string 'v3' (length=2)

Valor extraido es v1

C:\xampp7\htdocs\prueba\inicio.php:10
array (size=2)
  'k2' => string 'v2' (length=2)
  'k3' => string 'v3' (length=2)
```

Para extraer el último valor de una matriz puede emplearse la función ***array_pop()***. Esta recibe como único argumento una matriz y retorna el valor extraído:

```
        $valores = array("k1"=>"v1",
                         "k2"=>"v2",
                         "k3"=>"v3");
        var_dump($valores);
        // Extrae el último valor de la matriz y lo retorna.
        $valor = array_pop($valores);
        echo "Valor extraido es $valor</br>";
        var_dump($valores);
```

```
C:\xampp7\htdocs\prueba\inicio.php:6:
array (size=3)
  'k1' => string 'v1' (length=2)
  'k2' => string 'v2' (length=2)
  'k3' => string 'v3' (length=2)

Valor extraido es v3

C:\xampp7\htdocs\prueba\inicio.php:10:
array (size=2)
  'k1' => string 'v1' (length=2)
  'k2' => string 'v2' (length=2)
```

Inserción de valores en una matriz

La inserción de elementos en una matriz puede hacerse asignando un valor a una posición mediante el índice correspondiente entre corchetes. PHP dispone además de funciones especiales que permiten insertar valores al principio y final de una matriz.

La función ***array_unshift()*** permite insertar un elemento al principio de la matriz provocando el reordenamiento del resto de valores. La función requiere como primer parámetros la matriz y el valor a insertar:

```php
$matriz = array(0,10,20,30 );
var_dump($matriz);

$valor = 5;
// Inserta el valor indicado a la cabeza de la matriz
array_unshift($matriz, $valor);
var_dump($matriz);
```

```
C:\xampp7\htdocs\prueba\inicio.php:4:
array (size=4)
  0 => int 0
  1 => int 10
  2 => int 20
  3 => int 30

C:\xampp7\htdocs\prueba\inicio.php:9:
array (size=5)
  0 => int 5
  1 => int 0
  2 => int 10
  3 => int 20
  4 => int 30
```

El valor indicado en la función se añade en la primera posición de la matriz desplazando al resto de valores existentes a la posición siguiente.

Para insertar un valor al final de la matriz se emplea la función: ***array_push()***. La función recibe como parámetros la matriz y el valor/es a insertar.

```php
$matriz = array(0,10,20,30 );
var_dump($matriz);

$valor = 5;
// Inserta el valor indicado al final de la matriz
array_push($matriz, $valor);
var_dump($matriz);
```

```
C:\xampp7\htdocs\prueba\inicio.php:4:
array (size=4)
  0 => int 0
  1 => int 10
  2 => int 20
  3 => int 30

C:\xampp7\htdocs\prueba\inicio.php:9:
array (size=5)
  0 => int 0
  1 => int 10
  2 => int 20
  3 => int 30
  4 => int 5
```

Concatenación de matrices.

En PHP es posible concatenar los valores de dos matrices distintas en una sola empleando la función ***array_merge()***. Esta función recibe un número indefinido de matrices como parámetro y devuelve una matriz resultado que contiene todos los valores de las matrices indicadas.

```php
$matriz_1 = array(0,10,20,30 );
$matriz_2 = array(40,50);

// Crea una matriz concatenando los valores de la matrices indicadas
$resultado = array_merge($matriz_1, $matriz_2);
var_dump($resultado);
```

```
array (size=6)
  0 => int 0
  1 => int 10
  2 => int 20
  3 => int 30
  4 => int 40
  5 => int 50
```

La matriz resultante contiene todos los valores de las matrices pasadas por parámetro ordenados consecutivamente.

Relleno de matrices

El relleno de matrices consiste en crear una matriz a partir de otra insertando valores al principio o al final hasta alcanzar una determinada longitud. Para ello se emplea la función ***array_pad***():

```
array array_pad ( array $matriz ,
                  int $longitud , mixed $valor )
```

El parámetro <*matriz_origen*> es la matriz a partir de cuyos valores se va a crear la matriz resultado, añadiendo el valor indicado por el parámetro <*valor*> tantas veces como sea necesaria hasta alcanza la longitud indicada por el parámetro <*longitud*>. Si la longitud se expresa en positivo, los valores se añaden al final de la matriz, si es negativo se añaden al principio.

```php
$matriz = array(10,20,30 );

// Matriz que añade ceros al final hasta alcanzar longitud 6
$resultado_1 = array_pad($matriz, 6, 0);
echo "matriz resultado_1</br>";
var_dump($resultado_1);

// Matriz que añade ceros al prinicip hasta alcanzar longitud 6
$resultado_2 = array_pad($matriz, -6, 0);
echo "matriz resultado_2</br>";
var_dump($resultado_2);
```

En este ejemplo, ambas matrices resultantes contienen una longitud de 6 valores incluyendo los de la matriz original y tantos valores 0 como son necesarios insertados al principio o al final.

El resultado mostrado por pantalla sería:

```
matriz resultado_1

C:\xampp7\htdocs\prueba\inicio.php:7:
array (size=6)
  0 => int 10
  1 => int 20
  2 => int 30
  3 => int 0
  4 => int 0
  5 => int 0

matriz resultado_2

C:\xampp7\htdocs\prueba\inicio.php:11:
array (size=6)
  0 => int 0
  1 => int 0
  2 => int 0
  3 => int 10
  4 => int 20
  5 => int 30
```

Intersección de matrices

La intersección de matrices permite obtener una matriz con los valores coincidentes en otras dos. La comparación de los valores se realiza por su valor y tipo (empleando el operador condicional ***===***).

PHP define dos funciones para la intersección de funciones:

```
array array_intersect ( array $array1 , array $array2 [, array $... ] )
array array_intersect_assoc ( array $array1 , array $array2 [, array $... ] )
```

La función ***array_intersect()*** devuelve una matriz resultado con los valores coincidentes en ambas matrices. La función ***array_intersect_assoc()*** es más restrictiva puesto que sólo incluye en la matriz resultado los valores coincidentes presentes en la misma posición de la matriz.

```
$matriz1 = array(1,2,3,4,5);
$matriz2 = array(2,7,3,9,6);

// Devuelve los valores presentes en ambas matrices
$resultado_intersect = array_intersect($matriz1, $matriz2);
var_dump($resultado_intersect);

// Devuelve una matriz con los valores presentes en ambas matrices
// coincidentes en tipo y posicion
$resultado_asociativo = array_intersect_assoc($matriz1, $matriz2);
var_dump($resultado_asociativo);
```

(*) En el caso de emplear matrices asociativas, los valores coincidentes se incluyen en la matriz resultante con las claves presentes en la primera matriz.

```
C:\xampp7\htdocs\prueba\inicio.php:27:
array (size=2)
  1 => int 2
  2 => int 3

C:\xampp7\htdocs\prueba\inicio.php:31:
array (size=1)
  2 => int 3
```

Eliminación de repeticiones

La eliminación de repeticiones permite obtener una matriz a partir de otra eliminando todos los valores repetidos. PHP dispone de la función ***array_unique()*** que recibe como único parámetro la matriz original:

```php
$matriz = array(3,4,5,5,1,2,3,4,2,3,3,4);
// Obtencion de matriz eliminando resultados.
$resultado = array_unique($matriz);
var_dump($resultado);
```

La matriz resultante tendrá los siguientes valores:

```
C:\xampp7\htdocs\prueba\inicio.php:8:
array (size=5)
  0 => int 3
  1 => int 4
  2 => int 5
  4 => int 1
  5 => int 2
```

Ordenación de matrices

PHP posee una serie de funciones específicas para reordenar los elementos contenidos en una matriz en función de determinados criterios. Todas estas funciones reciben como parámetro la matriz a ordenar.

- ***sort()*** / ***rsort()*** → Ordena los valores de una matriz escalar de menor a mayor, y de mayor a menor respectivamente. *Si se aplica a una matriz asociativa únicamente devuelve sus valores, no sus claves*.

```php
$matriz = array("Jonas", "Lucas", "Pedro", "Raul", "Manuel");
// Ordenar de matriz de menor a mayor
sort($matriz);
var_dump($matriz);
// Ordena matriz de mayor a menor
rsort($matriz);
var_dump($matriz);
```

```
C:\xampp7\htdocs\prueba\inicio.php:8:
array (size=5)
  0 => string 'Jonas' (length=5)
  1 => string 'Lucas' (length=5)
  2 => string 'Manuel' (length=6)
  3 => string 'Pedro' (length=5)
  4 => string 'Raul' (length=4)

C:\xampp7\htdocs\prueba\inicio.php:11:
array (size=5)
  0 => string 'Raul' (length=4)
  1 => string 'Pedro' (length=5)
  2 => string 'Manuel' (length=6)
  3 => string 'Lucas' (length=5)
  4 => string 'Jonas' (length=5)
```

(*) La ordenación se hace tanto si los valores son numéricos como si son cadenas alfabéticamente.

- ***asort()*** / ***arsort()*** → Ordena los elementos de una matriz asociativa por sus valores. La ordenación se hace de menor a mayor y de mayor a menor respectivamente.

```php
$valores = array("00A"=>"Alberto","11B"=>"Manuel","22C"=>"Pedro");
// Retorna los valores y claves ordenados de menor a mayor
asort($valores);
var_dump($valores);
// Retorna los valores y claves ordenados de mayor a menor
arsort($valores);
var_dump($valores);
```

```
C:\xampp7\htdocs\prueba\inicio.php:7:
array (size=3)
  '00A' => string 'Alberto' (length=7)
  '11B' => string 'Manuel' (length=6)
  '22C' => string 'Pedro' (length=5)

C:\xampp7\htdocs\prueba\inicio.php:10:
array (size=3)
  '22C' => string 'Pedro' (length=5)
  '11B' => string 'Manuel' (length=6)
  '00A' => string 'Alberto' (length=7)
```

- ***ksort()*** / ***krsort()*** → Ordena los elementos de una matriz asociativa por sus claves de menor a mayor, y de mayor a menor respectivamente.

```php
$valores = array("00A"=>"Alberto","11B"=>"Manuel","22C"=>"Pedro");
// Retorna los valores y claves ordenados de menor a mayor por las claves
ksort($valores);
var_dump($valores);
// Retorna los valores y claves ordenados de mayor a menor por las claves
krsort($valores);
var_dump($valores);
```

```
C:\xampp7\htdocs\prueba\inicio.php:7:
array (size=3)
  '00A' => string 'Alberto' (length=7)
  '11B' => string 'Manuel' (length=6)
  '22C' => string 'Pedro' (length=5)

C:\xampp7\htdocs\prueba\inicio.php:10:
array (size=3)
  '22C' => string 'Pedro' (length=5)
  '11B' => string 'Manuel' (length=6)
  '00A' => string 'Alberto' (length=7)
```

- ***array_reverse()*** → Invierte el orden de los elementos presentes en la matriz. La función recibe la matriz original como argumento y devuelve una matriz resultado con los valores ordenados en orden inverso.

```php
$valores = array("00A"=>"Alberto","11B"=>"Manuel","22C"=>"Pedro");
// Invierte el orden de los valores.
$resultado = array_reverse($valores);
var_dump($resultado);
```

```
C:\xampp7\htdocs\prueba\inicio.php:7:
array (size=3)
  '22C' => string 'Pedro' (length=5)
  '11B' => string 'Manuel' (length=6)
  '00A' => string 'Alberto' (length=7)
```

PHP dispone por último de la función ***usort()*** para ordenar matrices utilizando una función externa que determine el modo en que se comparan los valores. De este modo es posible ordenar una matriz bajo cualquier criterio.

```
bool usort ( array &$array , callable $value_compare_func )
```

La función de comparación (*$value_compare_func*) dada como argumento debe recibir dos parámetros y retornar un valor entero mayor, menor o igual a cero según la comparación de los valores recibidos.

El siguiente código muestra la ordenación de menor a mayor de los valores numéricos de una empleando la función ***usort()*** y una función de comparación *cmp()*.

```php
// Matriz de valores numéricos.
$valores = [3, 2, 5, 6, 1];

// Funcion de comparación
function cmp($a, $b)
{
        if ($a == $b) {
                return 0;
        }
        return ($a < $b) ? -1 : 1;
}

// Ordenacion de la matriz empleando la funcion de comparacion cmp().
usort($valores, "cmp");
var_dump($valores);
```

```
C:\xampp7\htdocs\prueba\inicio.php:16:
array (size=5)
  0 => int 1
  1 => int 2
  2 => int 3
  3 => int 5
  4 => int 6
```

Paso de parámetros a funciones mediante matrices.

Las matrices pueden utilizarse como parámetros para las funciones. Al igual que cualquier parámetro pueden pasarse por valor o por referencia. Se indica el paso por referencia cuando la función debe modificar la función dada como argumento. Se indica el paso por valor cuando la matriz dada como argumento no es modificada.

Ejemplo: El siguiente código muestra una función *incrementar()* que recibe una matriz como parámetro por referencia (*$&matriz*) y la modifica sumando el segundo parámetro (*$valor*) a todos sus valores:

```php
// Función que recibe una matriz como parámetro por referencia
function incrementar( &$matriz, $valor ) {
        for ( $i = 0; $i < count($matriz); $i++ ) {
                $matriz[$i]+=$valor;
        }
}

// Código principal
$m = array(1,2,3,4,5,6);        // Declaración matriz
echo "Antes de invocar incrementar</br>";
var_dump($m);
// Modifica la matriz sumando 10 a todos sus valores.
incrementar($m, 10);
echo "Después de invocar incrementar</br>";
var_dump($m);
```

Antes de invocar incrementar

```
C:\xampp7\htdocs\prueba\inicio.php:12:
array (size=6)
  0 => int 1
  1 => int 2
  2 => int 3
  3 => int 4
  4 => int 5
  5 => int 6
```

Después de invocar incrementar

```
C:\xampp7\htdocs\prueba\inicio.php:15:
array (size=6)
  0 => int 11
  1 => int 12
  2 => int 13
  3 => int 14
  4 => int 15
  5 => int 16
```

El uso de matrices como parámetros permite varias ventajas:

- Pueden utilizarse para pasar un número variable de valores a una función definiendo únicamente un parámetro de entrada.

- Pueden utilizarse para implementar funciones capaces de devolver más de un valor modificando los valores de una matriz pasada por referencia.

Ejercicios

1.- Crea una función que dada una matriz escalar de una sola dimensión muestre por pantalla una tabla con las posiciones y los valores comprendidos en la matriz.

2.- Crea una función que dada una matriz asociativa muestre en una tabla la relación de claves y valores almacenados en la matriz.

3.- Crea una función que admite como parámetro una matriz escalar de una dimensión y muestre por pantalla:

- El valor máximo contenido y su posición en la matriz.
- El valor mínimo contenido y su posición en la matriz.
- El valor media de todos los valores contenidos en la matriz.

4.- Crea una función que reciba como parámetro una cadena de texto con el nombre de un color, y retorne otra cadena con el código del color correspondiente en HTML. Para ello la función debe implementar internamente una matriz asociativa con los siguientes valores:

Clave	Código
Azul	#0000FF
Rojo	#FF0000
Magenta	#FF00FF
Verde	#00FF00
Cyan	#00FFFF
Amarillo	#FFFF00
Blanco	#FFFFFF

5.- Crea una función que devuelva una matriz escalar bidimensional con los valores de la tabla de multiplicar del 1 al 10. A continuación crea otra función que reciba como parámetro la matriz y muestre sus valores en una tabla de la siguiente forma:

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

6.- Crea una página PHP que define las siguientes matrices:

- $lista1=(‘a’,’b’,’c’,’d’,’e’)
- $lista2=(‘v’,’w’,’x’,’y’,’z’)

A continuación implementa las siguientes operaciones y muestra las matrices por pantalla para verificar el funcionamiento.

- Combina en el array “lista1” los valores que éste contiene junto con los del array “lista2”.
- Agrega al array “lista1” el elemento ‘6’ por la derecha y el elemento ‘1’ por la izquierda.
- Elimina del array “lista1” los elementos insertados en el apartado anterior.
- Añade al final del array “lista1” los elementos: ‘7’,’8’,’9’.
- Elimina el último elemento del array "lista1" y muestra tanto el valor extraído como el array resultante.
- Elimina el primer elemento del array “lista1” y muestra tanto el valor extraído como el array resultante.
- Muestra el array "lista1" en orden inverso
- Muestra el array "lista1" ordenado de menor a mayor, y de mayor a menor
- Indica si el elemento ‘e’ se encuentra contenido en el array “lista1”, y la posición que ocupa.

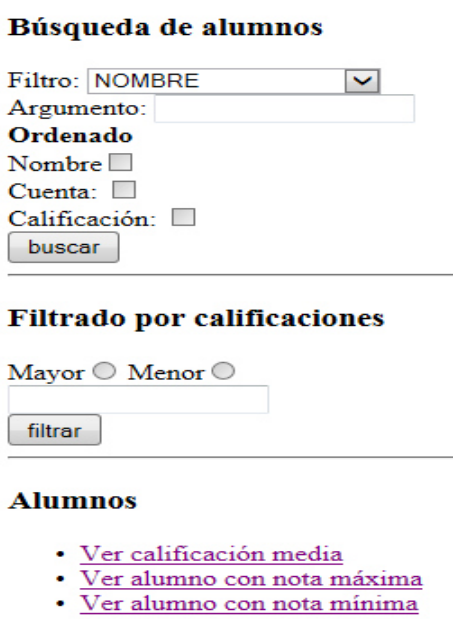
Práctica

Crea una aplicación web que permita seleccionar y obtener datos de los alumnos de un centro. La aplicación debe constar de los siguientes ficheros:

alumnos.php → script de PHP que consta de los datos de los alumnos almacenados en una matriz. Se trata de una matriz escalar que guarda por cada alumno una matriz asociativa con su nombre, apellido1, apellido2, cuenta y calificación correspondiente:

```php
<?php //ARRAY $usuarios
$alumnos[]=array('nombre' => 'Alberto', 'apellido1' => 'Polanco', 'apellido2' => 'Sánchez','cuenta' => 'albe', 'calificacion' => 7.5);
$alumnos[]=array('nombre' => 'Alvaro', 'apellido1' => 'Jiménez', 'apellido2' => 'Lopez','cuenta' => 'alva', 'calificacion' => 6.0);
$alumnos[]=array('nombre' => 'Aranzazu', 'apellido1' => 'Pérez', 'apellido2' => 'Laborda','cuenta' => 'aran', 'calificacion' => 3.5);
$alumnos[]=array('nombre' => 'Aritz', 'apellido1' => 'Hernáez', 'apellido2' => 'García','cuenta' => 'arit', 'calificacion' => 6);
$alumnos[]=array('nombre' => 'Asier', 'apellido1' => 'Pérez', 'apellido2' => 'Tamayo','cuenta' => 'asie', 'calificacion' => 7);
$alumnos[]=array('nombre' => 'Eder', 'apellido1' => 'Moledo', 'apellido2' => 'Villalba','cuenta' => 'eder', 'calificacion' => 6.5);
$alumnos[]=array('nombre' => 'Eneko', 'apellido1' => 'de la Torre', 'apellido2' => 'Sánchez','cuenta' => 'enek', 'calificacion' => 5.5);
$alumnos[]=array('nombre' => 'Francisco Borja', 'apellido1' => 'Navarro', 'apellido2' => 'Río','cuenta' => 'fran', 'calificacion' => 5);
$alumnos[]=array('nombre' => 'Gaizka', 'apellido1' => 'Lorenzo', 'apellido2' => 'Jiménez','cuenta' => 'gaiz', 'calificacion' => 4.5);
$alumnos[]=array('nombre' => 'Iñaki', 'apellido1' => 'Angulo', 'apellido2' => 'Tarancón','cuenta' => 'inak', 'calificacion' => 2);
$alumnos[]=array('nombre' => 'Iñaki', 'apellido1' => 'Demón', 'apellido2' => 'Fernandez','cuenta' => 'inai', 'calificacion' => 7);
$alumnos[]=array('nombre' => 'Iñigo', 'apellido1' => 'Casado', 'apellido2' => 'Rico','cuenta' => 'inig', 'calificacion' => 8);
$alumnos[]=array('nombre' => 'Iratxe', 'apellido1' => 'Urriolabeitia', 'apellido2' => 'Zabala','cuenta' => 'irat', 'calificacion' => 9.5);
$alumnos[]=array('nombre' => 'Isaac Jacob', 'apellido1' => 'Pazos', 'apellido2' => 'López','cuenta' => 'isaa', 'calificacion' => 7.5);
$alumnos[]=array('nombre' => 'Jon', 'apellido1' => 'Aróstegui', 'apellido2' => 'Brizuela','cuenta' => 'jona', 'calificacion' => 6);
$alumnos[]=array('nombre' => 'Jon', 'apellido1' => 'Elorriaga', 'apellido2' => 'Bilbao','cuenta' => 'jone', 'calificacion' => 6.5);
$alumnos[]=array('nombre' => 'Lander', 'apellido1' => 'Eguskiza', 'apellido2' => 'Lamelas','cuenta' => 'land', 'calificacion' => 3);
$alumnos[]=array('nombre' => 'Luis Fernando', 'apellido1' => 'Coca', 'apellido2' => 'Cabrera','cuenta' => 'luis', 'calificacion' => 4);
$alumnos[]=array('nombre' => 'Miguel', 'apellido1' => 'Rodrigo', 'apellido2' => 'Ortega','cuenta' => 'migu', 'calificacion' => 5.5);
$alumnos[]=array('nombre' => 'Sergio', 'apellido1' => 'Castro', 'apellido2' => 'Bravo','cuenta' => 'serg', 'calificacion' => 6.5);
$alumnos[]=array('nombre' => 'Xabier', 'apellido1' => 'Isla', 'apellido2' => 'Rodriguez','cuenta' => 'xabi', 'calificacion' => 7);
```

formulario.html → Página HTML normal compuesta por los siguientes elementos:




```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h3>Búsqueda de alumnos</h3>
 <form action="busqueda.php" method="get">
        Filtro: <select name="filtro">
                <option value="1">NOMBRE</option>
                <option value="2">PRIMER APELLIDO</option>
                <option value="3">SEGUNDO APELLIDO</option>
                <option value="4">CUENTA</option>
        </select>
        <br/>
        Argumento: <input type="text" name="argumento">
        <br/><b>Ordenado</b><br/>
        Nombre<input type="checkbox" name="orden" value="nombre"><br />
        Cuenta: <input type="checkbox" name="orden" value="cuenta"><br />
        Calificación: <input type="checkbox" name="orden" value="calificacion"><br />
        <input type="submit" value="buscar" name="accion">
 </form>
<hr>
<h3>Filtrado por calificaciones</h3>
 <form action="busqueda.php" method="get">
        Mayor<input type="radio" name="filtro" value="mayor" checked>
        Menor<input type="radio" name="filtro" value="menor">
        <br />
        <input type="number" name="argumento" max=10 min=0><br/>
        <input type="submit" value="filtrar" name="accion">
 </form>
<hr>
<h3>Alumnos</h3>
<ul>
        <li><a href="busqueda.php?acc=media">Ver calificación media</a></li>
        <li><a href="busqueda.php?acc=maximo">Ver alumno con nota máxima</a></li>
        <li><a href="busqueda.php?acc=minimo">Ver alumno con nota mínima</a></li>
</ul>
</body>
</html>
```

busqueda.php → Página web PHP que recibe los datos enviados por el formulario concatenados en la URL y muestra los datos solicitados como contenido. Para ello debe mostrarse una tabla con las siguientes columnas: CUENTA, NOMBRE, APELLIDO1, APELLIDO2, CALIFICACIÓN, con tantas filas como alumnos deban mostrarse.

Las posibles configuraciones de parámetros enviadas en la URL por el formulario pueden ser las siguientes:

- *Si accion = 'buscar'*

  filtro → Tipo de filtrado. Posibles valores:
  1 = Por nombre
  2 = Por apellido 1
  3 = Por apellido 2
  4 = Por cuenta

  argumento → Valor a filtrar

  orden → Campo por el que se ordenan los valores. Posibles valores:
  1 = Nombre
  2 = Cuenta
  3 = Calificación
  (*) Puede no aparecer si no se selecciona ningún orden.

- *Si accion = 'filtrar'*

  filtro → Tipo de filtrado de calificaciones. Posibles valores:
  "mayor" = Filtrar calificaciones superiores a la indicada
  "menor" = Filtrar calificaciones inferiores a la indicada

  argumento → Calificación a partir de la que filtrar

- *Si accion = 'media'*

  Debe mostrarse la nota media de las calificaciones del alumnado.

- *Si accion = 'maximo'*

  Debe mostrarse los datos del alumno con mayor calificación

- *Si accion = 'minimo'*

  Debe mostrarse los datos del alumno con menor calificación

Si no está presente el parámetro 'accion' simplemente se entiende que la página no ha sido solicitada desde el formulario y se debe redireccionarse al mismo.

Cadenas de Caracteres.

Las cadenas de caracteres son un tipo de dato que permiten representar textos de manera semejante a una matriz de caracteres. El manejo de las cadenas de caracteres se emplea en PHP para generar y componer código HTML para el navegador de cliente.

Declaración de cadenas de caracteres

La declaración de cadenas de texto se realiza mediante el operador de asignación. El literal puede aparecer entre comillas dobles o simples.

Si los literales se indican con comillas dobles pueden incluir identificadores de variables que son sustituidos por sus valores. Si se indican con comillas simples contienen exactamente lo escrito.

```
$nombre = "Roger";
 echo "Hola $nombre";     // Muestra Hola Roger
 echo "Hola ".$nombre;    // Muestra Hola Roger
 echo 'Hola $saludo';     // Muestra Hola $saludo
```

La definición de una cadena puede realizarse en varias líneas para almacenar secciones completas de código HTML. Para ello se emplea el operador especial **<<<** seguido de una etiqueta. A partir de ahí; todas las líneas contenidas hasta la reaparición de la etiqueta se toman como contenido de la cadena:

```
$html = <<<INICIO
        <p><b>ESTO ES UN SIMPLE PARRAFO DE HTML</br>
        QUE INCLUYE UNAS CUANTAS LÍNEAS DE TEXTO</br>
        EMPLEANDO EL OPERADOR ESPECIAL </b></p>
INICIO;
 echo $html;
```

En el ejemplo; todo las líneas escritas entre ***<<<INICIO*** y ***INICIO*** se asignan como valor a la cadena *$html*.

Proyección de cadenas

Las funciones de visualización de datos permiten incluir en el código HTML el valor de variables o literales para que sean visibles en el navegador del usuario.

PHP define la sentencia ***echo*** para incluir el valor de una variable o literal en el código HTML de respuesta al usuario.

PHP define además las siguientes funciones especializadas:

- `int print ( string $arg )` → Función que imprime el valor de una variable pasada como parámetro.

- `int printf ( string $format [, mixed $args [, mixed $... ]] )` →Función que imprime el valor o variable pasada como parámetro aplicando un determinado formato.

- `string sprintf ( string $format [, mixed $args [, mixed $... ]] )` → Función que devuelve una cadena con el valor de varias variables aplicando un formato.

- `int vprintf ( string $format , array $args )` → Función que imprime el valor de los elementos contenidos en una matriz pasada como parámetro aplicando un formato.

Las funciones anteriores admiten una cadena de formato como parámetro (*$format*) que permite determinar cómo deben mostrarse los valores.

Una cadena de formato se compone de *descriptores*. Cada descriptor indica la forma en que se representa un determinado valor al convertirlo en cadena para su visualización. Cada *descriptor* se expresa mediante una serie de signos y letras en un orden determinado:

%[<relleno><alineación><longitud><.precisión>]<tipo>
* Los indicadores entre corchetes son opcionales.

- ***<relleno>:*** Indicador opcional del carácter a utilizar para completar la longitud fijada de un campo. Por defecto el carácter de relleno utilizado es el espacio.

- ***<alineación>*:** Indicador opcional de alineamiento. Se pone un guión '-' para alineamiento a la izquierda. Por defecto el alineamiento es a la derecha.

- ***<longitud>:*** Indicador opcional de la longitud mínima del campo en total. Si el campo tiene menos caracteres, se añaden caracteres de relleno hasta completar la longitud indicada. Los caracteres se añaden a la izquierda o derecha en función de la alineación. Si tiene más caracteres no se añade nada.

- ***<precisión>:*** Indicador opcional del número de valores decimales a visualizar en el caso de valores de tipo decimal exclusivamente.

- ***<tipo>:*** Indica el tipo de valor a visualizar: Los indicadores más importantes son:

Carácter	*Tipo*
d	Decimal
b	Binario
o	Octal
x, X	Hexadecimal
c	ASCII
f	Coma flotante
s	Cadena

Ejemplo: Las siguientes sentencias ejemplarizan el uso de la función ***printf()*** con diferentes cadenas de formato orientadas a la proyección de valores numéricos:

```
$n =  43951789;
$u = -43951789;
$c = 65; // ASCII 65 is 'A'

// Repetir el signo %, permite que se visualice.
printf("%%b = '%b'<br/>", $n); // Valor binario
printf("%%c = '%c'<br/>", $c); // Valor como carácter ASCII del valor.
printf("%%d = '%d'<br/>", $n); // Valor como entero
printf("%%e = '%e'<br/>", $n); // Valor en notación científica
printf("%%u = '%u'<br/>", $n); // Valor entero sin signo.
printf("%%u = '%u'<br/>", $u); // Valor entero sin signo.
printf("%%f = '%f'<br/>", $n); // Valor decimal.
printf("%%o = '%o'<br/>", $n); // Valor en Octal ( base 8 )
printf("%%s = '%s'<br/>", $n); // Valor como cadena de texto.
printf("%%x = '%x'<br/>", $n); // Valor hexadecimal ( base 16 ) minúsculas
printf("%%X = '%X'<br/>", $n); // Valor hexadecimal en mayúsculas.

printf("%%+d = '%+d'<br/>", $n); // Valor con signo.
printf("%%+d = '%+d'<br/>", $u); // Valor con signo.
```

```
%b = '10100111101010011010101101'
%c = 'A'
%d = '43951789'
%e = '4.395179e+7'
%u = '43951789'
%u = '4251015507'
%f = '43951789.000000'
%o = '247523255'
%s = '43951789'
%x = '29ea6ad'
%X = '29EA6AD'
%+d = '+43951789'
%+d = '-43951789'
```

Para más información sobre el formateo de datos en cadenas de texto puede consultarse la página del sitio web oficial de PHP:

http://php.net/manual/en/function.sprintf.php

Ejemplo: Proyección de valores de tipo cadena:

```
$s = "monkey";
$t = "many monkeys";

printf("[%s]<br/>",      $s); // Valor como cadena por defecto.
printf("[%10s] <br/>",    $s); // Valor con 10 caracteres justificado a derecha.
printf("[%-10s] <br/>",   $s); // Valor con 10 caracteres justificado a izquierda.
printf("[%010s] <br/>",   $s); // Valor con 10 caracteres rellenado con ceros.
```

```
[monkey]
[ monkey]
[monkey ]
[0000monkey]
```

Ejemplo: Proyección de valores de tipo fecha.

La fecha del sistema se obtiene mediante la función ***getdate()*** que devuelve una matriz asociativa con las diferentes partes de la fecha asociadas a las siguientes claves:

Clave	Descripción	Ejemplo de valores devueltos
"seconds"	Representacion numérica de los segundos	*0* a *59*
"minutes"	Representacion numérica de los minutos	*0* a *59*
"hours"	Representacion numérica de las horas	*0* a *23*
"mday"	Representacion numérica del día del mes	*1* a *31*
"wday"	Representacion numérica del día de la semana	*0* (para Domingo) hasta *6* (para Sábado)
"mon"	Representacion numérica de un mes	*1* hasta *12*
"year"	Representación numérica del año con 4 dígitos.	Ejemplos: *1999* o *2003*
"yday"	Representacion numérica del día del año	*0* hasta *365*
"weekday"	Representación cadena del nombre del día.	*Sunday* hasta *Saturday*
"month"	Representación cadena del nombre del mes	*January* hasta *December*
0	Los segundos desde la Época Unix, similar a los valores devueltos por time() y usados por date().	Dependiente del Sistema, típicamente -*2147483648* hasta*2147483647*.

```
$fecha = getdate();
$year = $fecha["year"];
$month = $fecha["mon"];
$day = $fecha["mday"];

printf("%04d-%02d-%02d", $year, $month, $day);
```

```
2016-12-29
```

Ejemplo: Proyección de valores decimales

```
$money1 = 68.75;
$money2 = 54.35;
$money = $money1 + $money2;
echo $money."</br>";
// Valor decimal 9 caracteres en total con 2 para decimales rellenando con 0.
printf("%09.2f", $money);
```

```
123.1
000123.10
```

Manipulación de cadenas

Cada uno de los caracteres de una cadena puede obtenerse del mismo modo que en una matriz empleando el operador de indización '[]' e indicando la posición del carácter deseado mediante un valor entero comprendido entre 0 y n-1; donde n es la longitud de la cadena.

```
        $palabra = "HOLA MUNDO";
        for( $i = 0; $i < strlen($palabra); $i++) {
             echo $palabra[$i]."</br>";
        }
```

La función ***strlen()*** devuelve la longitud en caracteres de la cadena dada como argumento. El código HTML generado mostrará lo siguiente:

Operadores de cadenas.

El operador de concatenación ***(.):*** Permite concatenar dos cadenas en una nueva:

```
$hola = "HOLA ";
$nombre = "PEPE";
$saludo = $hola.$nombre;
echo $saludo;           // Muestra el resultado "HOLA PEPE"
```

El operador de comparación ***(==).*** Devuelve un valor booleano indicando si dos cadenas son idénticas:

```
$cadena1 = "pepe";
$cadena2 = "luis";

if ( $cadena1 == $cadena2 ) {
        echo "SON IGUALES";
} else {
        echo "SON DISTINTAS";    // Mostrará que son distintas.
}
```

(*) Este modo de comparación es sensible a mayúsculas y minúsculas por lo que las cadenas "HOLA" y "hola" no se considerarían iguales.

```
    ?>
```

Funciones de comparación/ordenación de cadenas

PHP dispone de las siguientes funciones que permiten comparar y ordenar cadenas.

```
int strcmp ( string $str1 , string $str2 )
```

La función ***strcmp()*** es la más básica para comparar y ordenar cadenas. Recibe dos cadenas como parámetro y devuelve un valor entero con los siguientes significados:

- El valor es 0 si ambas cadenas son iguales. *La función es sensible a mayúsculas y minúsculas como el operador (==)*.
- El valor es menor que 0 si la cadena *str1* es anterior que *str2*
- El valor es mayor que 0 si la cadena *str1* es posterior que *str2*

El siguiente código muestra una matriz de nombres de la que se obtiene el primero nombre por orden alfabético:

```
// Matriz de nombres
$nombres = [ "Roger", "Yuri", "Ivanov", "Petrov", "Sergei", "Andrei"];
// Tomamos el primer nombre como el primero alfabéticamente.
$primero = $nombres[0];
// Bucle de recorrido de nombres de la matriz
foreach( $nombres as $nombre) {
        printf("Comparamos strcmp(%s, %s) = %d<br/>", $primero, $nombre, strcmp($primero,
$nombre));
        // Comparamos el mayor hasta el momento con cada nombre
        if ( strcmp($primero, $nombre) > 0) {
                // Si el nombre obtenido es anterior alfabéticamente no los copiamos.
                $primero = $nombre;
        }
}
// Visualizamos el valor
printf("%s<br/>", $primero);
```

```
Comparamos strcmp(Roger, Roger) = 0
Comparamos strcmp(Roger, Yuri) = -1
Comparamos strcmp(Roger, Ivanov) = 1
Comparamos strcmp(Ivanov, Petrov) = -1
Comparamos strcmp(Ivanov, Sergei) = -1
Comparamos strcmp(Ivanov, Andrei) = 1
Andrei
```

La función ***strcasecmp()*** es equiparable a ***strcmp()*** pero **no** es sensible a mayúsculas y minúsculas, por lo que detectaría como iguales cadenas como “HOLA” y “hola”:

```
int strcasecmp ( string $str1 , string $str2 )
```

La comparación entre cadenas con las funciones ***strcmp()*** y ***strcasecmp()*** se realiza a nivel binario convirtiendo las cadenas a valores numéricos y comparándolos. Este modo de ordenación es propio de los ordenadores pero un poco distinto al modo lexicográfico propio de las personas (también llamado *modelo natural*).

La función ***strnatcamp()*** compara dos cadenas empleando el modelo de comparación natural. Sus parámetros de entrada y valor de salida sigue la misma norma que las otras dos funciones anteriores:

```
int strnatcmp ( string $str1 , string $str2 )
```

El siguiente código muestra las diferencias al ordenar una misma matriz de cadenas empleando el modelo binario y el natural:

```php
$arr1 = $arr2 = ["img12.png", "img10.png", "img2.png", "img1.png"];

echo "Ordenacion empleando método binario de strcmp()";
usort($arr1, "strcmp");
var_dump($arr1);

echo "Ordenación empleando el modelo natural de strnatcmp()";
usort($arr2, "strnatcmp");
var_dump($arr2);
```

Ordenacion empleando método binario de strcmp()

```
C:\xampp7\htdocs\prueba\inicio.php:6:
array (size=4)
  0 => string 'img1.png' (length=8)
  1 => string 'img10.png' (length=9)
  2 => string 'img12.png' (length=9)
  3 => string 'img2.png' (length=8)
```

Ordenación empleando el modelo natural de strnatcmp()

```
C:\xampp7\htdocs\prueba\inicio.php:9:
array (size=4)
  0 => string 'img1.png' (length=8)
  1 => string 'img2.png' (length=8)
  2 => string 'img10.png' (length=9)
  3 => string 'img12.png' (length=9)
```

Formateo de cadenas

Eliminación de caracteres innecesarios.,

PHP dispone de una serie de funciones que limpian las cadenas de ciertos caracteres considerados innecesarios (espacios en blanco, tabulaciones, saltos de línea) situados al principio y/o fin de las cadena:

string rtrim (string $str [, string $character_mask])

- Elimina por defecto espacios en blanco, saltos de carro y tabulaciones del final de la cadena pasada como primer argumento. El segundo parámetro es opcional y permite indicar los caracteres a eliminar. Retorna la cadena resultante.

string trim (string $str [, string $character_mask = " \t\n\r\0\x0B"])

- Equivalente a ***rtrim(),*** pero elimina los caracteres situados al principio de la cadena.

string ltrim (string $str [, string $character_mask])

- Combina el efecto de las dos funciones anteriores en una sola eliminando caracteres innecesarios del principio y final de la cadena dada como argumento y devolviendo la resultante.

El siguiente código muestra el uso de las tres funciones y su efecto sobre una cadena de origen con espacios en blanco innecesarios al principio y final de la misma:

```
$cadena = "  0000  ";
echo "[".$cadena."]</br>";              // Muestra '[  0000  ]'
$cadena_trim = trim($cadena);
$cadena_rtrim = rtrim($cadena);
$cadena_ltrim = ltrim($cadena);
echo "[".$cadena_trim."]</br>"; // Muestra '[0000]'
echo "[".$cadena_rtrim."]</br>";// Muestra '[  0000]'
echo "[".$cadena_ltrim."]</br>";// Muestra '[0000  ]'
```

Cambio entre mayúsculas y minúsculas.

PHP incluye algunas funciones que permiten pasar de mayúsculas a minúsculas y viceversa una cadena de caracteres dada:

*string **strtolower** (string $string)*
Esta función retorna la cadena dada como argumento convertida en minúsculas.

*string **strtoupper** (string $string)*
Esta función retorna la cadena indicada como argumento convertida a mayúsculas.

Eliminación de caracteres HTML

PHP dispone también de funciones importantes para formatear una cadena cara a su proyección como parte de una página web.

La función ***htmlspecialchars()*** sustituye todos los caracteres especiales HTML presentes en la cadena dada como argumento por otros equivalentes para su visualización como parte de una página web evitando que el navegador los interprete erróneamente como HTML. La cadena resultante es devuelta como resultado:

```
$cadena = "Indicar el <titulo>";
echo $cadena."<br/>";
echo htmlspecialchars($cadena)."<br/>";
```

Indicar el
Indicar el <titulo>

La primera sentencia echo muestra la cadena "*Indicar el*", el resto "*<titulo>*" es interpretado erróneamente por el navegador como una etiqueta HTML. Al incluir la función ***htmlspecialchars()*** en el siguiente *echo()* se muestra correctamente la cadena completa.

Relleno de caracteres

El relleno de caracteres consiste en añadir una determinada cantidad de caracteres al principio o al final de una cadena hasta alcanzar una determinada longitud.

PHP dispone de la función ***str_pad()*** que permite añadir un determinado carácter a una cadena hasta alcanzar una longitud y devuelve la cadena resultante.

```
string str_pad ( string $input
                 ,int $pad_length
               [, string$pad_string = " "
               [, int $pad_type = STR_PAD_RIGHT ]] )
```

- El parámetro *($input*) es la cadena origen que queremos rellenar.
- El parámetro *($pad_length*) determina la longitud que deseamos que alcance rellenándola.
- El parámetro *($pad_string*) es opcional y determina el carácter con el que se va a rellenar la cadena. Si no se indica nada se emplean espacios en blanco por defecto.
- El parámetro (*$pad_type*) es también opcional y determina dónde se van a ingresar los caracteres de relleno, los posibles valores vienen dados por las siguientes constantes.
  - *STR_PAD_BOTH* → Añade los caracteres al principio y final
  - *STR_PAD_LEFT* → Añade los caracteres al principio
  - *STR_PAD_RIGHT* → Añade los caracteres al final. Es el valor por defecto.

El siguiente código muestra el relleno de una misma cadena (*$cadena*) añadiendo guiones bajos "_" hasta alcanzar los 10 caracteres de longitud:

```php
$cadena = "12345";

// Cadena original
echo "[".$cadena."]</br>";
// Relleno por ambos extremos
echo "[".str_pad($cadena, 10, "_", STR_PAD_BOTH)."]</br>";
// Relleno por la izquierda
echo "[".str_pad($cadena, 10, "_", STR_PAD_LEFT)."]</br>";
// Relleno por la derecha
echo "[".str_pad($cadena, 10, "_", STR_PAD_RIGHT)."]</br>";
```

```
[12345]
[__12345___]
[_____12345]
[12345_____]
```

Unión y división de cadenas

División de cadenas

La división de cadenas permite como su nombre indica dividir una cadena de datos en diferentes fragmentos empleando un determinado carácter como separador.

Una cadena de datos es una cadena normal y corriente en la que se introducen varias informaciones distintas separadas por un carácter:

```
Roger, Petrov, C\Ivanov,34, 666-767908
```

PHP dispone para esta operación de la función ***explode():***

```
array explode ( string $delimiter
                , string $string
               [, int $limit = PHP_INT_MAX ] )
```

El parámetro (*$delimiter*) representa la cadena a dividir. El parámetro (*$string*) representa la cadena separadora. El último parámetro (*$limit*) es opcional e indica el número de fragmentos que se desean obtener. Si no se indica se devuelven tantos fragmentos como haya. La función retorna una matriz con los fragmentos obtenidos.

El siguiente código muestra la división de la cadena (*$cadena*) por las comas (,) devolviendo como resultado una matriz (*$datos*) con cada uno de los valores contenidos en la cadena original delimitados por las comas.

```php
$cadena = "Roger,Petroviano,23,0123456782";
$datos = explode(",", $cadena );
var_dump($datos);
```

```
array (size=4)
  0 => string 'Roger' (length=5)
  1 => string 'Petroviano' (length=10)
  2 => string '23' (length=2)
  3 => string '012345678Z' (length=10)
```

En algunas ocasiones las cadenas se utilizan para albergar una secuencia de datos separados por un carácter delimitador, tal como comas, o puntos. Este puede ser el caso de archivos de texto en formato .CSV por ejemplo.

PHP dispone de la función ***strtok()*** que permite procesar una cadena de datos obteniendo cada valor uno por uno mediante un bucle.

```
string strtok ( string $str , string $token )
```

El primer parámetro (*$str*) se corresponde con la cadena de datos que queremos dividir, y el segundo parámetro (*$token*) es el carácter divisor. La función devuelve el primer fragmento en forma de cadena.

El siguiente código muestra la división de una cadena obteniendo los valores uno a uno mediante un bucle *while*:

```
$cadena = "Roger,Petroviano,23,012345678Z";
// Obtencion del primer fragmento
$valor = strtok($cadena, ",");
while ($valor !== false) {              // Hay algo?
        printf("%s<br/>", $valor);
        // Obtencion del siguiente fragmento
        $valor = strtok(",");
}
```

```
Roger
Petroviano
23
012345678Z
```

Tras la primera llamada a ***strtok()*** puede seguirse a la función indicando sólo el segundo parámetro para obtener los siguientes fragmentos. La función retorna al final un valor lógico *falso* cuando no hay más fragmentos.

Unión de cadenas

La unión de cadenas consiste en componer una cadena con los valores presentes en una matriz utilizando un determinado carácter como delimitador de manera inversa a la función *explode().* Para ello PHP incluye la función ***implode()****:*

```
string implode ( string $glue , array $pieces )
```

El primer parámetro (*$glue*) indica el carácter a utilizar para unir cada cadena contenida en la matriz indicada como segundo parámetro ($pieces). El resultado es una cadena con todas las cadenas de la matriz unidas.

```php
$datos = array('Roger', 'Petrov', '666-948596');
$cadena = implode(",", $datos);
echo $cadena;
```

```
Roger,Petrov,666-948596
```

Extracción de subcadenas

La extracción de subcadenas consiste en obtener una cadena parte de otra cadena indicando el carácter de partida y el número de caracteres a copiar. Para ello puede emplearse la función de PHP ***substr()***:

```
string substr ( string $string
                , int $start
                [, int $length ] )
```

La función retorna una cadena tomando los caracteres de la cadena indicada como primer parámetro ($*string*) a partir del nº de carácter indicado (inclusive) en el segundo parámetro (*$start*) contando el número de caracteres del tercer parámetro ($*length*).

```php
$cadena = "0123456789";

echo substr($cadena, 0, 2)."<br/>";             // Muestra '01'
echo substr($cadena, 2, 5)."<br/>";             // Muestra '23456'
```

Si se omite se devuelven todos los caracteres hasta el fin de la cadena original.

```php
echo substr($cadena, 5)."<br/>";        // Muestra '56789'
```

Si el nº del carácter inicial es negativo, se cuenta a partir del fin de la cadena de origen.

```php
echo substr($cadena, -2)."<br/>";       // Muestra '89'
```

Búsqueda de cadenas

La búsqueda de cadenas consiste en detectar si el contenido de una cadena (o un carácter), está presente en otra cadena.

PHP ofrece dos funciones básicas para esta tarea:

```
string strstr ( string $haystack
                , mixed $needle
                [, bool $before_needle = false ] )
```

La función ***strstr()*** busca una cadena aguja (*$needle*) en otra cadena pajar (*$haystack*). Si la cadena aguja está presente en la cadena pajar.

La función devuelve la parte final de la cadena pajar incluyendo la aguja si se omite o indica un valor falso al tercer parámetro (*$before_needle*). Si se indica un valor lógico verdadero devuelve la parte inicial de la cadena pajar sin incluir la cadena aguja:

```
$email  = 'name@example.com';
$domain = strstr($email, '@');          // Muestra la parte final de la cadena.
echo $domain."<br/>";                           // Muestra '@example.com'

$user = strstr($email, '@', true);      // Muestra la parte anterior de la cadena.
echo $user."<br/>";                             // Muestra 'name'
```

Si no se encuentra la cadena aguja en la cadena pajar la función retorna un valor falso.

```
mixed strpos ( string $haystack
               , mixed $needle
               [, int $offset = 0 ] )
```

La función ***strpos()*** busca también la cadena aguja (*$needle*) en la cadena pajar (*$haystack*). El parámetro desplazamiento (*$offset*) permite indicar el nº de carácter a partir del cual se inicia la búsqueda. Si se omite la búsqueda se efectúa a partir del primer carácter (*$offset = 0*)

A diferencia de *strstr()*, ésta función devuelve un valor numérico con la posición del primer carácter de la primera aparición de la cadena aguja en la cadena pajar:

```
// Se puede buscar por el caracter, ignorando cualquier cosa antes del offset
$newstring = 'abcdef abcdef';
$pos = strpos($newstring, 'a', 1); // $pos = 7, no 0
```

En caso de no encontrarse la cadena aguja la función retorna un valor lógico falso.

(*) Para comprobar si la función retorna un valor lógico falso debe emplearse preferentemente los operadores de comparación ===, o ¡==, puesto que si retorna un valor 0 puede tomarse erróneamente como un falso con los operadores == o ¡=.

Reemplazo de cadenas

El reemplazo de cadenas consiste en buscar todas las apariciones de una cadena en otra para sustituirlas por una tercera cadena. Este procedimiento es relativamente común para modificar cadenas sustituyendo unos caracteres por otros (puntos por comas o viceversa, vocales acentuadas por otras sin acentos…., etc), o eliminar ciertos caracteres o partes de una cadena.

PHP dispone para ello de la función ***str_replace()***.

```
mixed str_replace ( mixed $search
                  , mixed $replace
                  , mixed $subject
                  [, int &$count ] )
```

La función toma la cadena sujeto ($*subject*) y sustituye en ella todas las apariciones de la cadena de búsqueda ($*search*) por la cadena de reemplazo ($*replace*) retornando la cadena resultante. El parámetro opcional ($*count*) permite indicar el nº de apariciones a sustituir de la cadena de reemplazo en la cadena sujeto. Si se omite se sustituyen todas las apariciones.

```
$consulta = "SELECT * FROM PERSONAS WHERE ID = %1";
// Sustituye todas las apariciones de la cadena %1 por el valor 10
$query = str_replace("%1", 10, $consulta);
var_dump($query);
```

```
SELECT * FROM PERSONAS WHERE ID = 10
```

La función *str_replace()* también admite matrices como valores para los parámetros $*search* y $*replace*. En tal caso, se sustituye cada cadena de la matriz de búsqueda por la cadena en la misma posición de la matriz de reemplazos:

```
// Cadena sujeto
$consulta = "INSERT INTO PERSONAS VALUES( %1, %2, %3 )";
// Matriz de búsqueda
$params_search = array("%1", "%2", "%3");
// Matriz de reemplazos
$params_replace = array("'Roger'", "'Petrov'", 20);
$query = str_replace($params_search, $params_replace, $consulta);
echo $query;
```

```
INSERT INTO PERSONAS VALUES( 'Roger', 'Petrov', 20 )
```

Evaluación de expresiones regulares

Una expresión regular es una cadena que define un patrón que sólo un determinado conjunto de cadenas cumplen. Las expresiones regulares se construyen empleando una serie de símbolos y una sintaxis concreta.

Históricamente PHP ha sido compatible con expresiones regulares empleando la sintaxis POSIX y Perl. Sin embargo, a partir de PHP 5.3, POSIX se considera obsoleto.

La evaluación de una expresión regular comprueba si una cadena cumple o no con el formato definido por la expresión regular. Estas pueden emplearse para validar datos con un cierto formato tales como DNIs, direcciones de correo electrónico, códigos postales, nº cuentas bancarias… etc.

PHP incluye la función ***preg_match*** para comprobar expresiones regulares.

```
int preg_match ( string $pattern
                       , string $subject
                       [, array&$matches
                       [, int $flags = 0
                       [, int $offset = 0 ]]] )
```

La función evalúa si la cadena sujeto (*$subject*) cumple el patrón definido por la expresión regular (*$pattern*). Si la cadena cumple con la expresión regular la función retorna un valor 1, en caso contrario retorna 0. Si la expresión regular es incorrecta o está mal construida y no puede evaluarse retorna un valor lógico falso.

El siguiente código muestra una función que evalúa si el parámetro $mail contiene una dirección de correo electrónico correctamente formada:

```
function verificar_email($email)
{
        $expr_reg = "/^([a-zA-Z0-9])+([a-zA-Z0-9\._-])*@([a-zA-Z0-9_-])+([a-zA-Z0-9\._-]+)+$/";
        if(preg_match($expr_reg,$email) == true)
        {
                return true;
        }
        return false;
}
```

Ejemplos de expresiones regulares

Patrón	Cadena	¿Cumple?	Comentario
abc	awbwc	No	Los caracteres tienen que estar seguidos.
	34abc	Sí	No importa que hayan caracteres antes...
	cbabcba	Sí	... o después.
a2b	g1da2b3	Sí	Las expresiones regulares detectan letras, números, ...
áb	3áb4	Sí	... incluso acentos, ...
a\$b	1a$b2	Sí	... salvo los caracteres ^ . [$ ( ) | * + ? { \ € que deben llevar una contrabarra \ antes, además de \n (nueva línea) y \t (tabulador)
[aeiou]	bic	Sí	Los corchetes definen los caracteres admitidos en una posición ...
	bcd	No	
[^aeiou]	bic	Sí	... o no admitidos
	aei	No	
[p-t]	avr	Sí	Se pueden definir rangos de caracteres...
	av1	No	
[B-D]	PMD	Sí	... en minúsculas o mayúsculas ...
	AV1	No	
[0-9]	b9d	Sí	... o números
	bcd	No	
[[:alpha:]]			Cualquier carácter alfabético
[[:digit:]]			Cualquier número
[[:alnum:]]			Cualquier número o carácter alfabéticos
[[:punct:]]			Cualquier carácter que no sean letras y números (menos el euro)
[[:space:]]			Cualquier tipo de espacio en blanco
[[:upper:]]			Cualquier mayúscula
[[:lower:]]			Cualquier minúscula
^ab	cab	No	Los caracteres tienen que estar al principio
	abc	Sí	No importa que hayan caracteres después
ab$	abc	No	Los caracteres tienen que estar al final
	cab	Sí	No importa que hayan caracteres antes
^ab$	ab	Sí	Tiene que empezar y acabar por ab ...
	abab	No	... y no puede haber nada antes o después
ab?c	abcde	Sí	El carácter b puede estar entre a y c...
	acde	Sí	... o no estar entre a y c ...
	adcde	No	... pero no puede haber otro carácter
a.c	abc	Sí	El . representa cualquier carácter ...
	a c	Sí	... incluso el espacio el blanco, ...
	ac	No	pero no la ausencia del carácter
	abdc	No	o varios caracteres.

ab+c	abcde	Sí	El carácter b puede estar una vez...
	abbbcde	Sí	... o varias ...
	acde	No	... pero tiene que estar al menos una vez.
ab*c	abcde	Sí	El carácter b puede estar una vez...
	abbbcde	Sí	... o varias ...
	acde	Sí	... o ninguna.
ab{3}c	abbbc	Sí	Las llaves indican el número exacto de repeticiones del carácter, ...
	abbbbc	No	... no puede haber más ...
	abbc	No	... ni menos.
ab{2,4}c	abc	No	Se pueden definir rangos con límite inferior e inferior
	abbc	Sí	
	abbbc	Sí	
	abbbbc	Sí	
	abbbbbc	No	
ab{2,}c	abc	No	Se pueden definir rangos sin límite superior
a(bc){2}d	abcbcd	Sí	Los paréntesis definen agrupaciones de caracteres. En este caso bc tiene que aparecer repetido
a(bc)?d	abcd	Sí	Aquí bc puede estar ...
	ad	Sí	... o no estar, ...
	abd	No	... pero no puede aparecer sólo la b, o sólo la c u otro carácter
^a(b|d)c$	abc	Sí	Entre la a al principio y la c al final puede estar el carácter b...
	adc	Sí	... o el carácter d, ...
	abdc	No	... pero no los dos, ...
	ac	No	... ni ninguno de ellos.
^(ab)|(dc)$	abc	Sí	Está la pareja ab al principio ...
	adc	Sí	... o dc ...
	abdc	Sí	... o las dos, ...
	ac	No	... pero no ninguna
^(ab)$\|^(dc)$	abc	No	Está la pareja ab, pero sobra la c ...
	adc	No	... o está la pareja dc, pero sobra la a.
	dc	Sí	Está una de las dos

Ejercicios

1.- Crea un programa el cual, a partir de la cadena "*Estoy estudiando el capítulo relativo a las Cadenas de PHP*", muestre los siguientes datos. Para ello deberás utilizar las funciones de manipulación de cadenas vistas.

- La cantidad de caracteres que contiene la cadena
- La cantidad de caracteres que contiene la cadena sin contar espacios.
- La cadena expresada en mayúsculas.
- La longitud de la cadena
- El texto existente en la cadena tras la palabra "las"
- La posición de la palabra "relativo".
- La subcadena que se inicia en el carácter 8º y finaliza en el 15º
- La subcadena con los últimos cinco caracteres.
- La cadena con la palabra "capítulo" reemplazada por "tema".
- El número de "a" existentes en la cadena.
- El número de palabras contenidas en la cadena.
- Todas las palabras comprendidas en la cadena con una "o".

2.- Crea una función que reciba dos cadenas como parámetros y retorna un valor lógico indicando si ambas son iguales sin considerar diferencias entre mayúsculas y minúsculas.

3.- Crea una función que reciba como parámetro un valor numérico y devuelva una cadena con el dicho valor expresado con dos valores para la parte decimal como mínimo a rellenar con ceros si es necesario. La cadena resultante debe además incluir un carácter # al principio y final. Por ejemplo, el valor 153.4 debería mostrarse como: "#153.40#".

4.- Crea una función que reciba una cadena como parámetro y retorne una cadena con el mismo valor y longitud fija a 30 caracteres. Si la cadena dada es más larga debe recortarse, y si es más corta deben añadírsele tantos espacios como sean necesarios al final.

5.- Crea una función llamada *serialcheck* que compruebe los números de serie de unos productos que se componen de cuatro grupos de cinco cifras cada uno separados por guiones:

```
02394-45677-30950-34503
```

6.- La función recibe como parámetro una cadena y devuelve un valor lógico cierto o falso. Para ello la función comprueba que la cadena dada contiene 4 grupos de cifras separadas por guiones, que el primer y tercer valor son pares, y que el segundo y el último son impares. En tal caso retorna como resultado un valor lógico cierto.

Práctica

Se pide crear una página para mostrar los resultados de las elecciones catalanas. Los datos están ya almacenados en un conjunto de matrices que podemos encontrar en el fichero *elecciones.php*. Las matrices declaradas son las siguientes:

$nombre_partido → Matriz asociativa con las siglas de cada partido como clave y su correspondiente nombre como valor:

```
$nombre_partido["JxSí"]="JUNTS PEL SÍ";
$nombre_partido["C's"]="CIUTADANS-PARTIDO DE LA CIUDADANÍA";
$nombre_partido["PSC"]="PARTIT DELS SOCIALISTES DE CATALUNYA (PSC-PSOE)";
$nombre_partido["CatSíqueesPot"]="CATALUNYA SÍ QUE ES POT";
$nombre_partido["PP"]="PARTIT POPULAR";
$nombre_partido["CUP"]="CANDIDATURA D'UNITAT POPULAR";
$nombre_partido["unio.cat"]="UNIÓ DEMOCRÀTICA DE CATALUNYA";
$nombre_partido["PACMA"]="PARTIT ANIMALISTA CONTRA EL MALTRACTAMENT ANIMAL";
$nombre_partido["RECORTES CERO-ELS VERDS"]="RECORTES CERO-ELS VERDS";
$nombre_partido["GANEMOS"]="GUANYEM CATALUNYA";
$nombre_partido["PIRATA.CAT/XDT"]="PIRATES DE CATALUNYA - PER DECIDIR-HO TOT";
```

$circunscripciones → Matriz escalar con los nombres de las cuatro circunscripciones electorales:

```
$circunscripciones[]="Barcelona";
$circunscripciones[]="Girona";
$circunscripciones[]="Lleida";
$circunscripciones[]="Tarragona";
```

$comarcas → Matriz asociativa que almacena con el nombre de cada circunscripción como clave, una matriz con el nombre de todas las comarcas.

```
$comarcas["Barcelona"][]="Alt Penedès";
$comarcas["Barcelona"][]="Anoia";
$comarcas["Barcelona"][]="Bages";
$comarcas["Barcelona"][]="Garraf";
$comarcas["Girona"][]="Alt Empordà";
$comarcas["Girona"][]="Baix Empordà";
$comarcas["Girona"][]="Cerdanya (G)";
$comarcas["Girona"][]="Selva (G)";
$comarcas["Lleida"][]="Alta Ribagorça";
$comarcas["Lleida"][]="Alt Urgell";
$comarcas["Lleida"][]="Aran";
$comarcas["Lleida"][]="Berguedà (L)";
```

$ayuntamientos → Matriz asociativa que almacena con el nombre de cada comarca como clave, una matriz con el nombre de todos los ayuntamientos

```
$ayuntamientos["Alta Ribagorça"][]="Pont de Suert, el";
$ayuntamientos["Alta Ribagorça"][]="Vall de Boí, la";
$ayuntamientos["Alta Ribagorça"][]="Vilaller";
$ayuntamientos["Alt Camp"][]="Aiguamúrcia";
$ayuntamientos["Alt Camp"][]="Alcover";
$ayuntamientos["Alt Camp"][]="Alió";
```

$votos → Matriz asociativa que almacena como clave el nombre de cada ayuntamiento y una matriz asociativa como valor con las siglas de cada partida como clave y el nº de votos registrados como valor.

```
$votos["Abella de la Conca"]=array("censo" => 139, "JxSí" => 57, "C's" => 9,"PSC" => 11,
"CatSíqueesPot" => 3, "PP" => 5, "CUP" => 18, "unio.cat" => 3, "PACMA" => 0, "RECORTES CERO-ELS
VERDS" => 1, "GANEMOS" => 0, "PIRATA.CAT/XDT" => 0 );

$votos["Abrera"]=array("censo" => 8828, "JxSí" => 1685, "C's" => 1771,"PSC" => 1275,
"CatSíqueesPot" => 938, "PP" => 619, "CUP" => 453, "unio.cat" => 131, "PACMA" => 82, "RECORTES
CERO-ELS VERDS" => 51, "GANEMOS" => 0, "PIRATA.CAT/XDT" => 0 );

$votos["Àger"]=array("censo" => 484, "JxSí" => 271, "C's" => 18,"PSC" => 14, "CatSíqueesPot" =>
10, "PP" => 25, "CUP" => 45, "unio.cat" => 9, "PACMA" => 2, "RECORTES CERO-ELS VERDS" => 1,
"GANEMOS" => 0, "PIRATA.CAT/XDT" => 0 );

$votos["Agramunt"]=array("censo" => 3778, "JxSí" => 1940, "C's" => 252,"PSC" => 161,
"CatSíqueesPot" => 104, "PP" => 185, "CUP" => 152, "unio.cat" => 87, "PACMA" => 9, "RECORTES
CERO-ELS VERDS" => 1, "GANEMOS" => 3, "PIRATA.CAT/XDT" => 0 );

$votos["Aguilar de Segarra"]=array("censo" => 203, "JxSí" => 121, "C's" => 4,"PSC" => 0,
"CatSíqueesPot" => 5, "PP" => 4, "CUP" => 18, "unio.cat" => 10, "PACMA" => 0, "RECORTES CERO-
ELS VERDS" => 1, "GANEMOS" => 0, "PIRATA.CAT/XDT" => 0 );
```

Se pide: Crea una página web *datos.php* que puede ser llamada de los siguientes modos:

datos.php
datos.php?circ=2
datos.php?circ=2&com=3
datos.php?circ=2&com=10&ayto=7

La página debe mostrar cada vez que sea llamada los siguientes elementos:

Encabezado → La página debe mostrar siempre un encabezado con uno de los siguientes mensajes:

datos.php	→ "Circunscripciones".
datos.php?circ=2	→ "Comarcas de circunscripción *Lleida*".
datos.php?circ=2&com=3	→ "Ayuntamientos de comarca *Aran*"
datos.php?circ=2&com=10&ayto=7	→ "Ayuntamiento "*Vielha e Mijaran*"

Los valores de los componentes de la URL pueden obtenerse mediante la matriz superglobal $_GET. Las matrices deberán ser accesibles desde el interior de la función por lo que deberá emplearse la palabra clave '*global*'. El encabezado debe implementarse mediante una función llamada ***cabecera()***.

Navegación → La página debe mostrar una tabla con las circunscripciones, comarcas o ayuntamientos correspondientes según los valores recibidos. Estos valores deben ser a su vez enlaces que permitan seleccionar una circunscripción, comarca o ayuntamiento concreto según corresponda.

Para ello deben implementarse las siguientes funciones

- ***listarCircunscripciones()*** → Devuelve un listado de todas las circunscripciones donde cada valor debe ser un enlace a la propia página: "*datos.php?circ=X*".

- ***listarComarcas($cir)*** → Devuelve un listado de todas las comarcas de la circunscripción indicada como parámetro. Cada valor debe ser un enlace a la propia página con la URL: "*datos.php?circ=$cir&com=X*.

- ***listarAyuntamientos($cir, $cor)*** → Devuelve un listado de todos los ayuntamientos de la comarca indicada como parámetro. Cada valor debe ser un enlace a la propia página con la URL: "*datos.php?circ=$cir&com=$com&ayto=X*"

Crea una función ***menu()*** que invoque a las funciones anteriores en función de los parámetros recibidos a través de la URL.

Recuento totales → La página debe mostrar un recuento del censo total de habitantes y el sumatorio de los votos recibidos por cada partida de la circunscripción, comarca o ayuntamiento seleccionado por los parámetros de la URL.

Para ello debes crear las siguientes funciones:

- ***total_comarca($com)*** → devuelva una matriz asociativa con las claves de la matriz $votos, es decir, tendrá por claves 'censo' y las siglas de los partidos y por valores la suma de los censos y votos por partido de los ayuntamientos de la comarca indicada en el parámetro *$com* por su nombre.

(*) Necesitarás como contador un array con las claves correspondientes y lleno de ceros. Te pueden ayudar las funciones *array_combine()* y *array_fill().*

- ***total_circunscripcion($circ)*** → devuelve una matriz asociativa como la retornada por la función total_comarca() pero con el sumatorio del censo y votos por partido de todas las comarcas de la circunscripción indicada en el parámetro *$circ* por su nombre.

(*) Esta función puede implementarse de manera sencilla basándose en la anterior.

- ***mostrar_total($totales)*** → Genera una tabla HTML mostrando el censo y el sumatorio de votos de cada uno de los partidos contenido en la matriz devuelta por las funciones *total_comarca()* y *total_circunscripción()* indicadas anteriormente.

Recuento subtotales → La página debe mostrar subtotales de los resultados electorales en función los parámetros indicados en la URL:

datos.php	→ Devuelve subtotales por cada circunscripción
datos.php?circ=2	→ Devuelve subtotales por cada comarca de la circunscripción.
datos.php?circ=2&com=3	→ Devuelve subtotales por cada ayuntamiento de la comarca.
datos.php?circ=2&com=10&ayto=7	< No muestra nada >

Para ello deben implementarse las siguientes funciones.

- ***votos_por_circunscripcion()*** → Devuelve una matriz asociativa con el nombre de cada circunscripción y como valor la matriz de totales de censo y votos por partido correspondientes.

- ***votos_por_comarca($circ)*** → Devuelve una matriz asociativa con el nombre de cada comarca de la circunscripción indicada y como valor la matriz de totales de censo y votos por partido correspondientes.

- ***votos_por_ayuntamiento($circ,$com)*** → Devuelve una matriz asociativa con el nombre de cada ayuntamiento de la comarca de la circunscripción indicada y como valor la matriz de totales de censo y votos por partido correspondientes.

(*) Estas funciones pueden implementarse más sencillamente empleando *total_comarcas()* y *total_circunscripciones()* del punto anterior.

- ***mostrar_subtotales($subtotales)*** → Genera una tabla HTML mostrando los subtotales de las matrices generadas por las tres funciones anteriores

Ganador → La página debe mostrar al final el nombre del partido que más votos ha obtenido para la circunscripción, comarca o ayuntamiento seleccionada por la URL.