U.T. 3: Fundamentos de la inserción de código en páginas web (I) : arrays

Estructuras de control

- PHP suministra dos sentencias condicionales:
  - if […] else […] elseif […]
  - switch […] case
- Estructuras repetitivas:
  - while
  - for
  - foreach
  - do … while

Estructuras de control

```
<?php
print "<B><U>Sentencia switch</U></B><BR>";
$i = 2;
switch ($i) {
     case 0:
         print "i es igual a 0<BR>";
     case 1:
         print "i es igual a 0 o 1<BR>";
     case 2:
         print "i es igual 0, 1 o 2<BR>";
     default:
 print "Sólo un break impediría que se imprima esta línea<BR>";
}
?>
```

Estructuras de control

```
<?php
   print "<BR>Ejemplo 2do.switch<BR>";
   // Este switch es similar a un if/elseif
   $i = 2;
   switch ($i) {
     case 0:
         print "i es igual a 0<BR>";
     break;
     case 1:
         print "i es igual a 0 o 1<BR>";
     break;
     default:
 print "Sólo un break impediría que se imprima esta línea<BR>";
    }
?>
```

Estructuras de control

```
<?php
    print "<BR>Ejemplo if idéntico a sentencia switch<BR>";
    // este if/elseif es idéntico funcionalmente
    // al switch anterior

    if ($i == 0)
    {
     print "i es igual a 0<BR>";
    }
    elseif ($i == 1)
    {
     print "i es igual a 1<BR>";
    }
    else
    {
     print "Ambas expresiones fueron falsas<BR>";
    }
    print "fin del ejemplo<BR>";
?>
```

Estructuras de control

- Estructura selectiva **if-else-elseif**

```
if (condición)
   sentencia
```

```
if (condición)
   sentencia 1
else
   sentencia 2
```

```
if (condición1)
   sentencia 1
else if (condición2)
   sentencia 2
...
else if (condición n)
   sentencia n
else
   sentencia n+1
```

- Mismo comportamiento que en C
- Las sentencias compuestas se encierran entre llaves
- elseif puede ir todo junto

Estructuras de control

- Estructura selectiva **switch**

```
switch (expresión)
{
   case valor_1:
    sentencia 1
    break;
   case valor_2:
    sentencia 2
    break;
   …
   case valor_n:
    sentencia n
     break;
   default
    sentencia n+1
}
```

- Mismo comportamiento que en C, sólo que la expresión del case puede ser integer, float o string.

Estructuras de control

- Estructura repetitiva **while**

```
while (condición)
{
   sentencias;
}
```

  - Mismo comportamiento que en C y java.

- Estructura repetitiva **do … while**

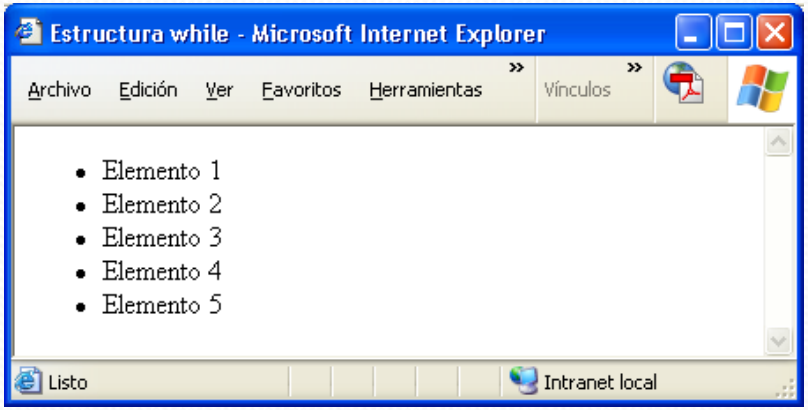
```
do{
   sentencias;
}while (condición);
```

- Mismo comportamiento que en C y java.

Estructuras de control

- Ejemplo de estructura repetitiva while

```
<?PHP
   print ("<UL>\n");
   $i=1;
   while ($i <= 5)
   {
    print ("<LI>Elemento $i</LI>\n");
    $i++;
   }
   print ("</UL>\n");
?>
```



Estructuras de control

- Estructura repetitiva **for**

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

  - Mismo comportamiento que en C y java.

- Estructura repetitiva **foreach**

```
foreach (exprMatriz as valor)
   sentencias;
Foreach (expMatriz as clave=>valor)
```

- Mismo comportamiento que en java.
- Diseñado para recorrer matrices.

Estructuras de control

- Ejemplo de estructura repetitiva for

```
<?PHP
   print ("<UL>\n");
   for ($i=1; $i<=5; $i++)
    print ("<LI>Elemento $i</LI>\n");
   print ("</UL>\n");
?>
```

Estructuras de control

- Ejemplo de estructura repetitiva foreach

```
<?php
    print "<B><U>Sentencia foreach</U></B><BR>";
    print "<BR>Primer ejemplo de foreach<BR>";
    $matriz1 = array("PHP 3", "PHP 4", "PHP 5");
    foreach ($matriz1 as $var1) {
       print "Elemento de matriz 1: $var1<br>";
   }
   print "<BR>Segundo ejemplo de foreach<BR>";
   $matriz2["PHP 3"] = 1998;
   $matriz2["PHP 4"] = 2000;
   $matriz2["PHP 5"] = 2004;
   foreach ($matriz2 as $clave => $var1) {
       print "Elemento de matriz 2: clave  $clave  año
$var1<br>";
   }
?>
```

Estructuras de control

- Sentencia **break**:

```
break  n;
```

- Nos permite salir de una estructura de control o bucle de manera directa.
- Si la acompañamos de un número entero (opcional), indicamos el número de niveles de la estructura anidada que queremos saltar.

Estructuras de control

- Sentencia ***continue***:

  continue [n];

- Nos permite abandonar la iteración vigente de una estructura de control.
- Si la acompañamos de un número entero (opcional), indicamos el número de niveles de la estructura anidada que queremos saltar.

Arrays

```
<HTML>
    <HEAD>
        <TITLE>Definición de matrices</TITLE>
    </HEAD>
    <BODY>
        <CENTER><H3>Uso del constructor array()</H3>
        <?php
            $Estad = array(1=>"Alemania", "Austria",5=> "Bélgica");
        ?>
        <TABLE BORDER="1" CELLPADDING="1" CELLSPACING="2">
            <TR ALIGN="center" >
                <TD>Elemento</TD>
                <?php
                    foreach ($Estad as $clave => $valor)
                        echo"<TD>$clave</TD>";
                ?>
            </TR>
            <TR ALIGN="center" >
                <TD>Valor</TD>
                <?php
                    foreach ($Estad as $clave => $valor)
                        echo "<TD> $valor </TD>";
                ?>
            </TR>
        </TABLE>
    </BODY>
</HTML>
```

Arrays

- Un array es una variable que almacena una secuencia de valores.
- Puede tener un número variable de elementos.
- Cada elemento puede tener un valor.
- Este valor puede ser simple (número, texto, etc.) o compuesto (otro array).
- Un array que contiene otro/s array se llama ***multidimensional***.
- PHP admite:
  - Array Escalares → los índices son números
  - Array Asociativos → los índices son cadenas

Arrays

- Sintaxis:
  - array ([clave =>] valor, ...)
  - La clave es una cadena o un entero no negativo.
  - El valor puede ser de cualquier tipo válido en PHP, incluyendo otro array
  - Ejemplos:

    $color = array ('rojo'=>101, 'verde'=>51, 'azul'=>255);

    $medidas = array (10, 25, 15);
  - Acceso:

    $color['rojo'] // No olvidar las comillas

    $medidas[0]
  - El primer elemento es el 0

Arrays

- Como el resto de variables, los arrays no se declaran, ni siquiera para indicar su tamaño.
- Pueden ser dispersos (se implementan como tablas hash).
  - Los índices de sus elementos no tienen porque ser consecutivos.

    $vec[1] = '1º elemento';

    $vec[8] = '2º elemento';
- En realidad contienen un mapeo entre *claves y valores (arrays* asociativos)

  Array array([index]=>[valor], [index2]=>[valor], ...);
  - Los índices no tienen porque ser números

    $vec['tercero'] = '3º elemento‘;

Arrays

- Los arrays no son homogéneos.
  - Sus elementos pueden ser de cualquier tipo (incluso otros arrays) y ser de tipos diferentes.

    $vec[5] = '4º elemento';

    $vec[1000] = 5.0;

Creando y eliminando arrays

- Hay dos formas de crear un array:
  - Asignación directa.
    - Se añaden sus elementos uno a uno, indicando el índice (mediante []).
    - Si el array no existía se crea.

```
$vec[5] = '1° elemento'; $vec[1] = "2° elemento";
$vec[] = '3° elemento'; $vec[6]= "3° elem..
```

- Utilizando el constructor **array().**
  - Se añaden entre paréntesis los primeros elementos del array. El primero de todos tiene asignado el índice cero.

```
$vec = array ( 3, 9, 2.4, 'Juan');
// $vec[0] = 3; $vec[1] = 9; $vec[2] = 2.4;...
```

  - Se pueden fijar el índice con el operador **=>**

```
$vec = array ( 2=>3 ,9, 2.4, 'nombre'=>'Juan');
// $vec[2] = 3;$vec[3]=9;.. $vec['nombre']="Juan
```

Creando y eliminando arrays

- Ejemplo2:

```
$unarray = array("dia" => 15, 1 => "uno");
```

- Ejemplo3:

```
$otro = array("unarray" => array(0=>14, 4=>15),
        "nombre" => "Una tabla");
```

- Para eliminar un elemento del array hay que emplear unset()
  - unset($miarray['nombre']);
  - unset($miarray);
- Los arrays no se imprimen con echo, sino con *print_r*:

```
print_r ($unarray) ;
```

Arrays: Arrays escalares

- **Ejemplo1:**

```
$trimestre1 = array(1 => 'Enero', 'Febrero', 'Marzo');
print_r($trimestre1);
```

Genera:

```
Array (    [1] => Enero,
[2] => Febrero,
[3] => Marzo)
```

Array que empieza en 1 en vez de 0

Arrays: Arrays escalares

- **Ejemplo 2:**

```
$array = array(1,1,1,1,1, 8=>1 ,4=>1,19, 3=>13);
print_r($array);
```

Genera:

```
Array(    [0] => 1, [1] => 1,
[2] => 1, [3] => 13,
[4] => 1, [8] => 1,
[9] => 19 )
```

El valor 13 sobreescribe al anterior de la posición 3.

El valor 19 se aloja en la pos 9, que es la siguiente a la última utilizada (8).

Arrays: Arrays escalares

- Mostrar el contenido del array (for)

```
$ciudad = array("París", "Madrid", "Londres");
for ($i=0;$i<=count($ciudad); $i++){
   echo $ciudad[$i]; echo "<br>";
}
```

- Mostrar el contenido del array (foreach)

```
$ciudad = array("París", "Madrid", "Londres");
foreach ($ciudad as $ciudades){
  echo $ciudades; echo "<br>";
}
```

- Inicializar un array

```
$vec = array();
```

Arrays: Arrays asociativos

- La clave o índice es un String.
- Pueden definirse:
  - Mediante la función array():

```
$precios = array("Azúcar" => 1,  "Aceite" => 4,  "Arroz" => 0.5);
$capitales = array("Francia"=>"París",  "Italia"=>"Roma");
```

  - Por referencia:

```
$precios["Azúcar"] = 1;
$precios["Aceite"] = 4;
$precios["Arroz"] = 0.5

$capitales["Francia"]="París";
$capitales ["Italia"]="Roma";
```

Arrays: Arrays asociativos

- Mostrar el contenido del array asociativo

```
$precios = array("Azúcar" => 1, "Aceite" => 4, "Arroz" => 0.5);
echo "<ul>";
foreach ( $precios as $producto => $precio ){
echo "<li>"."Producto: ".$producto." Precio: ".$precio."</li>";
}
echo "</ul>";
```

- Producto: Azúcar Precio: 1
- Producto: Aceite Precio: 4
- Producto: Arroz Precio: 0.5

- Otra forma

```
while (list($clave,$valor) = each ($precios)){
        echo "Producto: ".$clave."Precio: ".$valor."<br />";
}
```

Arrays: Arrays multidimensionales

- Son arrays en los que al menos uno de sus valores es, a su vez, otro array.
- Pueden ser escalares o asociativos

```
$pais=array(
        "espana"=>array(
       "nombre"=>"España",
       "lengua"=>"Castellano",
       "moneda"=>"euro"),
        "francia" =>array(
       "nombre"=>"Francia",
       "lengua"=>"Francés",
       "moneda"=>"euro"));
```

Arrays *multidimensionales*

pais	idioma	moneda
España	Castellano	Peseta
Francia	Francés	Franco

Arrays: Arrays multidimensionales

```
001  <?php
002          $juan=array('Juan Félix Mateos',185,90);
003          $ana=array('Ana Irene Palma',172,57);
004          $alumnos=array($juan,$ana);
005          print_r( $alumnos);
006  ?>
```

```
001  <?php
002          $alumnos=array(
003                  array('Juan Félix Mateos',185,90),
004                  array('Ana Irene Palma',172,57));
005          print_r( $alumnos);
006  ?>
```

Arrays de dos dimensiones

- Realiza el código php necesario para visualizar la siguiente tabla. Utiliza un array unidimensional:

País	Capital	Extensión	Habitantes
Alemania	Berlín	557046	78420000
Austria	Viena	83849	7614000
Bélgica	Bruselas	30518	9932000

Arrays: Recorrer un array

- **Recorrer un array secuencial:**
  - Usar count($matriz) y un bucle
    - **int count (mixed $array)**
    - Devuelve el número de elementos que contiene el array.

```
$matriz=array('lunes','martes','miércoles','jueves','viernes','sábado',‘
domingo');
echo count($matriz);
```

- Otra función para el tamaño de la matriz
  - sizeof($matriz)
    - Devuelve el número de elementos

Arrays: Recorrer un array

- **Recorrer un array no secuencial o asociativo:**
  - A través de funciones que actúan sobre un puntero interno:
    - **current()** - devuelve el valor del elemento que indica el puntero
    - **pos()** - realiza la misma función que current
    - **reset()** - mueve el puntero al primer elemento del array
    - **end()** - mueve el puntero al último elemento del array
    - **next()**- mueve el puntero al elemento siguiente
    - **prev()** - mueve el puntero al elemento anterior
    - **count()** - devuelve el número de elementos de un array
    - **key()** – devuelve el índice de la posición actual

[Ejemplo 053]

Arrays: Recorrer un array

- Ejemplo:

```
$semana = array("lunes", "martes", "miércoles", "jueves",  "viernes",                "sábado",
"domingo");
echo count($semana);  //7
reset($semana);                          //situamos el puntero en el 1ºelemento
echo current($semana);                //lunes
next($semana);
echo pos($semana);    //martes
end($semana);
echo pos($semana);    //domingo
prev($semana);
echo current($semana);                //sábado
echo key($semana);    // 5
```

Arrays: Recorrer un array

- **list($var1, $var2, $var3, ...)**
  - Asigna valor a una lista de variables en una sola operación. Solo arrays numéricos

```
list($var1, $var2)= array("Lunes", "Martes");
$var1="Lunes", $var2="Martes"
```

- **each($unarray)**
  - En cada iteración devuelve el par clave/valor actual y avanza el cursor. Devuelve un array de 4 elementos:

    0,key → la clave

    1,value → el valor

```
$semana = array("lunes", "martes");
foreach ($semana as $k=>$v){
echo "<pre>";
print_r(each($semana));
echo "</pre>";}
```

```
Array
(
    [1] => lunes
    [value] => lunes
    [0] => 0
    [key] => 0
)

Array
(
    [1] => martes
    [value] => martes
    [0] => 1
    [key] => 1
)
```

Arrays: Recorrer un array

- Otra forma de recorrer un array:

```
$unarray = array('uno', 'dos', 'tres');
reset($unarray);
while (list($clave, $valor) = each($unarray))
echo "$clave => $valor\n";
```

Arrays: Operadores para arrays

- Ot:

Ejemplo	Nombre	Resultado
$a + $b	Unión	Unión de $a y $b.
$a == $b	Igualdad	TRUE si $a i $b tienen las mismas parejas clave/valor.
$a === $b mismo	Identidad	TRUE si $a y $b tienen las mismas parejas clave/valor en el orden y de los mismos tipos.
$a != $b	Desigualdad	TRUE si $a no es igual a $b.
$a <> $b	Desigualdad	TRUE si $a no es igual a $b.
$a !== $b	No-identidad	TRUE si $a no es idéntica a $b.

El operador + devuelve el array del lado derecho añadido al array del lado izquierdo; para las claves que existan en ambos arrays, serán utilizados los elementos del array de la izquierda y serán ignorados los elementos correspondientes del array de la derecha.

Arrays: Recorrer un array

- **array_keys ($unarray [,valor_a_buscar])**
  - Devuelve las claves del array en otro array
  - Si hay ***val_a_buscar,*** sólo devuelve las claves de ese valor

```
$array = array(0=>100,  "color"=>"rojo");
print_r(array_keys($array));
$array = array("azul","red","green","azul",
"azul");
print_r(array_keys($array, "azul"));
```

```
Array
(
    [0] => 0
    [1] => color
)
Array
(
    [0] => 0
    [1] => 3
    [2] => 4
)
```

- **array_values ($unarray)**
  - Devuelve todos los valores del array en orden numérico.

```
$matriz = array("talla" => "XL",
        "color" =>"dorado");
print_r(array_values($matriz));
```

```
Array
(
    [0] => XL
    [1] => dorado
)
```

Arrays: Recorrer un array

- **array_keys ($unarray [,valor_a_buscar])**
  - Devuelve las claves del array en otro array
  - Si hay ***val_a_buscar,*** sólo devuelve las claves de ese valor

```
$array = array(0=>100,  "color"=>"rojo");
print_r(array_keys($array));
$array = array("azul","red","green","azul",
"azul");
print_r(array_keys($array, "azul"));
```

```
Array
(
    [0] => 0
    [1] => color
)
Array
(
    [0] => 0
    [1] => 3
    [2] => 4
)
```

- **array_values ($unarray)**
  - Devuelve todos los valores del array en orden numérico.

```
$matriz = array("talla" => "XL",
        "color" =>"dorado");
print_r(array_values($matriz));
```

```
Array
(
    [0] => XL
    [1] => dorado
)
```

Arrays: Buscar un elemento

- **array preg_grep(string patron, array $matriz)**
  - Devuelve un array con los elementos que cumplen el criterio fijado por *patron*.

  El patrón debe ser delimitado por el carácter / en el inicio y fin.

  ' /patron/'
- **array_search(valor, $matriz)**
  - Permite buscar un valor en un array y si lo encuentra devuelve su clave, sino devuelve NULL.

Arrays: Expresiones regulares

- PHP permite utilizar **funciones** para expresiones regulares.
- Una expresión regular permite **comparar un patrón frente a un texto**, para comprobar si el texto contiene lo especificado en el patrón.
- Ejemplos de patrones de búsqueda:
  - Patrón: **in**
    Coindicen:

    intensidad

    c**in**ta

    interior
  - Patrón: **[mp]adre**
    Coindicen:

Mi **madre** se llama Luisa

Tu **padre** es jardinero

Expresiones regulares: Sintaxis básica

- **El punto**
  - El punto representa **cualquier carácter**. Desde la A a la Z (en minúscula y mayúscula), del 0 al 9, o algún otro símbolo.

    ca.a coincide con *ca**n**a*, *ca**m**a*, *ca**s**a*, *ca**j**a*, etc…

    No coincide con *ca**st**a* ni *caa*
- **Principio y fin de cadena**
  - Si queremos indicar al patrón qué es el principio de la cadena o qué es el final, debemos hacerlo con **^ para inicio y $ para final**.

    "*^olivas*" coincide con "***olivas** verdes*",

    pero no con "*quiero olivas*"

Expresiones regulares: Sintaxis básica

- **Cuantificadores**
  - Para indicar que cierto elemento del patrón va a repetirse un **número indeterminado de veces**, usaremos **+** (una o más veces) o ***** (cero o más veces) .

    "gafas+" coincide con "*gafa**ssss***"

    pero no con "*gafa*"

    *"clo*aca"* coincide con "*claca*", "*cloaca*", "*clooooooooaca*", etc..

Expresiones regulares: Sintaxis básica

- El interrogante indica que un elemento **puede que esté (una vez) o puede que no**:

  "coches?" coincide con *"coche"* y con *"coches"*

- Las llaves **{ }** definen la **cantidad de veces que va a repetirse el elemento :**

  "abc{4}" coincide con "*abcccc*"

  pero no con "*abc*" ni "*abcc*", etc…

  "abc{3,}" coincide con "*abc*", "*abcc*", "*abccc*",

  pero no con "*abcccc*"

Expresiones regulares: Sintaxis básica

- Si un parámetro queda vacío, significa "un **número indeterminado**". Por ejemplo:

  "x{5,}" la x ha de repetirse 5 veces, o más.

- **Rangos**
  - Los corchetes **[]** incluidos en un patrón permiten especificar el **rango de caracteres** válidos a comparar.

"c[ao]sa" coincide con "*casa*" y con "*cosa*"

"[a-f]" coincide con todos los caracteres alfabéticos de la "*a*" a la "*f*"

"[0-9][2-6][ANR]" coincide con "*12A*", "*35N*", "*84R*", etc.. pero no con "*21A*", ni "*33L*", ni "*3A*", etc…

Expresiones regulares: Sintaxis básica

- Dentro de los corchetes el símbolo **^** es un negador, es decir:

  "[^A-z]" coincidirá con cualquier texto que NO tenga ningún carácter alfabético (ni minúsculas ni mayúsculas)

  "[^@] " coincide con cualquier carácter excepto "@" y "*espacio*"

- **Alternancia**
  - Para alternar entre varias opciones usaremos el símbolo **|.** Si una de las opciones coincide el patrón será cierto.

    "ale(ma|mi)n(ia|es)" coincide con "*aleman**ia***" y con "*aleman**es***"
    "(norte|sur|este|oeste)" coincide con cualquiera de los puntos cardinales.

Expresiones regulares: Sintaxis básica

- **Agrupadores**
  - Los paréntesis nos sirven para agrupar un subconjunto de caracteres.

    "(abc)+" coincide con "*abc*", "*abcabc*", "*abcabcabc*", etc
    "ca(sca)?da" coincide con "*cascada*" y con "*cada*"
- **Escapar caracteres**
  - Si queremos utilizar caracteres especiales en el patrón sin que se interprete como metacaracter, tendremos que "escaparlo". Esto se hace poniendo una barra invertida justo antes:

    "\." o "*"

Expresiones regulares

Ejemplos de expresiones regulares

https://maugelves.com/expresiones-regulares-en-php/

Expresiones regulares que debes conocer

https://code.tutsplus.com/es/tutorials/8-regular-expressions-you-should-know--net-6149

Las Expresiones Regulares más usadas en PHP

http://blog.educacionit.com/2016/12/19/las-expresiones-regulares-mas-usadas-en-php/

Arrays: Buscar un elemento

- **in_array(valor, $matriz, $strict)**
  - Devuelve *True* o *False* en función de la existencia o no de un valor en el array. Si *$strict* es *True se* tendrá en cuenta el tipo de los valores.
  - Es case-sensitive.

```
$a = array('1.10', 12.4, 1.13);
if (in_array('12.4', $a, true)) {
echo "'12.4' Encontrado con chequeo STRICT\n";}
if (in_array(1.13, $a, true)) {
echo "1.13 Encontrado con chequeo STRICT\n";}
```

[Ejemplo 055]

Arrays: Buscar un elemento

- **array_count_values($matriz)**
  - Cuenta las veces que aparece cada elemento de un array en ese array

```
$matriz = array(1, "hola", 1, "mundo", "hola");
array_count_values($matriz);     // devuelve array(1=>2,"hola"=>2,"mundo"=>1)
```

Arrays : Modificar un array

- **mixed array_pop (array &$matriz)**
  - Extrae y devuelve el último elemento del array. Obsérvese que esta función actúa sobre el array original como indica el hecho de que reciba el argumento implícitamente por referencia.

```
$matriz=array('lunes','martes','miércoles','jueves','viernes','sábado','domingo');
echo array_pop($matriz);
var_dump($matriz);
```

[Ejemplo 044]

Arrays : Modificar un array

- **int array_push(array &$matriz, $var1, $var2, …)**
  - Inserta los elementos $var al final del array y devuelve el número de elementos que contiene el array aumentado.

```
$matriz=array('lunes','martes','miércoles','jueves','viernes','sábado');
echo (array_push($matriz,'domingo'));
var_dump($matriz);
```

[Ejemplo 045]

Arrays : Modificar un array

- **mixed array_shift (array &$matriz)**
  - Extrae el primer elemento de la matriz, desplazando todos los elementos restantes hacia adelante. Devuelve el elemento extraído.

```
$matriz=array('lunes','martes','miércoles','jueves','viernes','sábado','domingo');
echo array_shift($matriz);
var_dump($matriz);
```

[Ejemplo 043]

- **array_unshift ($mat, $elem1, $elem2, …)**
- Permite añadir uno o más elementos por el inicio de la matriz indicada como parámetro. Devuelve el nuevo número de elementos del array.

Arrays : Modificar un array

- **array_walk (&matriz, func_usuario [, parametro])**
  - Nos permite aplicar una función definida por el usuario a cada uno de los elementos de un array.
  - La función *func_usuario()* recibe, al menos, dos parámetros
    - El valor del elemento
    - Su clave asociada
  - Una vez aplicada la función, el puntero interno del array se encontrará al final de él.

```
function aEuros(&$valor,$clave){
$valor=$valor/166.386;
}
array_walk($precios,'aEuros');
```

[Ejemplo 054]

Producto	Precio
prod1	1500 Ptas.
prod2	1000 Ptas.
prod3	800 Ptas.
prod6	100 Ptas.
prod7	500 Ptas.

Producto	Precio
prod1	9.02 €
prod2	6.01 €
prod3	4.81 €
prod6	0.60 €
prod7	3.01 €

Arrays : Modificar un array

- **array array_replace (array $matriz_destino , array $matriz_origen)**

  - Devuelve un array que es el resultado de sobrescribir/añadir sobre matriz destino los elementos de matriz origen (los que coinciden en índice se sobrescriben, y los que no se añaden). No afecta a las matrices que recibe como argumento.

```
$matriz_destino=array('altura'=>185,'peso'=>85);
$matriz_origen=array('pelo'=>'moreno','peso'=>95);
var_dump(array_replace($matriz_destino, $matriz_origen));
```

Arrays : Modificar un array

- **array_merge($mat1, $mat2, $mat3)**
  - Une las matrices indicadas como parámetros, empezando por la primera. Elimina los elementos con claves duplicadas en arrays asociativos (dejando la última leída). En arrays numéricos se generan nuevas claves.

    [Ejemplo 047]

  - También podemos unir matrices con el operador + . Elimina claves duplicadas (dejando el primer elemento leído).

    [Ejemplo 046]

Arrays : Modificar un array

- **array_merge_recursive($mat1,$mat2,$mat3)**
  - Permite combinar matrices sin perder elementos. Devuelve la matriz resultado de la suma. Con las claves duplicadas genera una nueva matriz para ese elemento.

    [Ejemplo 048]

- **array_pad($mat, $cantidad, $relleno)**
  - Permite añadir elementos de relleno en el inicio (negativo) y fin del array (positivo). Devuelve la matriz resultado.

    [Ejemplo 049]

Arrays : Modificar un array

- **array array_slice (array $matriz , int $inicio, int $cantidad)**
  - Devuelve un sub-array de $matriz a partir del *inicio* indicado y con la cantidad de elementos indicada.
  - Si cantidad no se especifica devuelve todos los elementos desde *inicio* hasta el final.

```
$vec=array(10,6,7,8,23);  [Ejemplo  042]
$res=array_slice($vec,1,3);     // $res= 6,7,8
```

Inicio	
Positivo	Posición del primer elemento contando desde el principio.
Negativo	Posición de comienzo desde el final
Cantidad	
Positivo	Número de elementos a considerar

Arrays : Modificar un array

- **array array_splice (array &$matriz , int $inicio, int $cantidad, mixed $reemplazo)**
  - Elimina de matriz *cantidad* elementos contados a partir del elemento *inicio*, los sustituye por los elementos del array *reemplazo* y los devuelve en un array. Si los índices son numéricos los reajusta.

```
$matriz=array('lunes','martes','miércoles','jueves','viernes','sábado',
'domingo');
var_dump (array_splice($matriz,1,2));
var_dump($matriz);

$matriz=array('altura'=>185,'peso'=>85,'pelo'=>'moreno');
var_dump(array_splice($matriz,1,2));
var_dump($matriz);
```

Arrays : Modificar un array

- **string implode (string $delimitador , array $matriz)**
  - Convierte *matriz* en una cadena de caracteres separando sus elementos con la cadena indicada en *delimitador*.

```
$matriz=array(7,'julio',2011);
echo implode(' de ',$matriz);
```

Arrays : Modificar un array

- Intersección de matrices.
  - **array_intersect($mat1,$mat2,$mat3)**
    - Devuelve una matriz con los elementos comunes a las matrices indicadas. La comparación se hace con el operador identidad (===)

      [Ejemplo 050]
  - **array_intersect_assoc($mat1,$mat2,$mat3)**
    - Devuelve una matriz con los elementos comunes utilizando el operador identidad (===). En la comparación se tienen en cuenta también las claves.

      [Ejemplo 051]

Arrays : Modificar un array

- Creación de una matriz con los elementos únicos de otra:
  - **array_unique ($mat)**
    - Crea una nueva matriz a partir de otra original, tomando sólo los elementos no duplicados de ésta. Utiliza el operador de identidad en la comparación.

      [Ejemplo 052]

  - **array_combine ($mat1,$mat2)**
    - Crea un nuevo array a partir de otros dos. Un array le sirve para tomar las claves y el otro para tomar los valores correspondientes. Los dos arrays deben tener el mismo número de elementos.

Arrays : Modificar un array

- **array_reverse ($array, true)**
  - Devuelve el array invertido.
  - Si el 2º parámetro es true, conserva las claves

```
$entrada = array ("php", 4, "rojo");
$resultado = array_reverse ($entrada);
$resultado_claves = array_reverse($entrada, true);
```

```
Array
(
[0] => rojo
[1] => 4
[2] => php
)
Array
(
[2] => rojo
[1] => 4
[0] => php
)
```

- **range (low, high, paso)**
  - Crea una matriz que contiene un rango de elementos
  - *paso* indica el salto

```
$numeros=range(5,9); □ (5,6,7,8,9)
$numeros2=range(0,50,10); □ (0,10,20,30,40,50)
$letras=range(a,f); □ (a,b,c,d,f)
```

Arrays : Modificar un array

- **compact(var1,var2,....,varN)**
  - Crea un vector asociativo cuyas claves son los nombres de las variables y los valores el contenido de las mismas.

```
$ciudad="miami";
$edad="23";
$vec=compact("ciudad","edad");
Es equivalente a:
$vec=array("ciudad"=>"miami","edad"=>"23");
```

- **shuffle ($array)**
  - Desordena en forma aleatoria los elementos de un array.

Arrays : Ordenar un array

- Ordenación de matrices:
  - **bool sort (array &$array [, int $sort_flags = *SORT_REGULAR*])**
    - Ordena un array de menor a mayor
  - **bool rsort (array &$array [, int $sort_flags = *SORT_REGULAR*])**
    - Ordena un array en orden inverso (de mayor a menor)
  - **bool asort (array &$array [, int $sort_flags = *SORT_REGULAR*])**
    - Ordena un array manteniendo la correlación de los índices con los elementos asociados.
  - **bool arsort (array &$array [, int $sort_flags = *SORT_REGULAR*])**
    - Ordena un array en orden inverso, manteniendo la correlación de los índices con los elementos asociados.
  - **bool ksort (array &$array [, int $sort_flags = *SORT_REGULAR*])**
    - Ordena un array por clave, manteniendo la correlación entre la clave y los datos.

Arrays : Ordenar un array

- Ordenación de matrices:
  - **bool krsort (array &$array [, int $sort_flags = *SORT_REGULAR*])**
    - Ordena un array por clave en orden inverso, manteniendo la correlación entre la clave y los datos.
  - **bool usort (array &$array , callable $value_compare_func)**
    - Ordena un array usando una función de comparación definida por el usuario. Se asignan nuevas claves a los elementos ordenados.
  - **bool uksort (array &$array , callable $key_compare_func)**
    - Ordena las claves de un array usando una función de comparación proporcionada por el usuario.

Arrays : Ordenar un array

- Ordenación de matrices:
  - **bool uasort (array &$array , callable $value_compare_func)**
    - Ordena un array de manera que los índices mantienen sus correlaciones con los elementos del array asociados, usando una función de comparación definida por el usuario.
  - **bool array_multisort (array &$arr [, mixed $arg = *SORT_ASC* [, mixed $arg = *SORT_REGULAR* [, mixed $...]]])**
    - Ordenar varios arrays al mismo tiempo, o un array multi-dimensional por una o más dimensiones. Las claves asociativas (string) se mantendrán, aunque las claves numéricas son re-indexadas.

[Ejemplo 055]

Arrays predefinidos

- No precisan ser definidos como globales dentro de una función.
- En las versiones anteriores a PHP 4.1 => **$GLOBALS**
- En las versiones PHP 4.1 o posteriores, también:

$_GET, $_POST, $_COOKIE, $_REQUEST, $_ENV, $_SERVER y $_SESSION

```
<?php
    function f0() {
    global $HTTP_GET_VARS;
    $nom = $HTTP_GET_VARS['nombre'];
    $nom = $_GET['nombre']; // Mismo resultado.
    return $nom;
    }
?>
```

Arrays predefinidos

- Arrays predefinidos dentro de cualquier aplicación PHP:
  - $HTTP_GET_VARS o **$_GET** (PHP 4.1)**:** contiene los valores enviados por el método GET.

```
print('Variables enviadas por el método GET');
foreach($_GET as $nom_variable=>$valor)
    echo "$nombre_variable = $valor<br>\n";
```

  - $HTTP_POST_VARS o **$_POST** (PHP 4.1): contiene los valores enviados por el método POST.
  - $HTTP_COOKIE_VARS o **$_COOKIE** (PHP 4.1): contiene los valores de las cookies enviadas por el cliente.
  - **$_REQUEST** (PHP 4.1): Contienen todas las variables incluidas en los tres anteriores

Arrays predefinidos

- $HTTP_SERVER_VARS o **$_SERVER** (PHP 4.1): variables asociadas a la cabecera HTTP o a variables del servidor.

```
$acepta = $HTTP_SERVER_VARS ['HTTP_ACCEPT'];
$cliente = $_SERVER['HTTP_USER_AGENT'];
$camino_raiz = $_SERVER['DOCUMENT_ROOT'];
$fichero_php = $_SERVER['PHP_SELF'];//SCRIPT_NAME
```

- $HTTP_SESSION_VARS o **$_SESSION** (PHP 4.1): variables de la sesión.
- $HTTP_ENV_VARS o **$_ENV** (PHP 4.1): variables de entorno.

```
$camino = $_ENV['PATH'];
```

- **$GLOBALS**: contiene todas las anteriores, además del resto de variables globales.

print_r ($_SERVER)

Array

(

[HTTP_ACCEPT] => image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms -excel, application/vnd.ms-powerpoint,

application/msword, */*

[HTTP_REFERER] => http://www.ignside.net/man/php/arrays.php

[HTTP_ACCEPT_LANGUAGE] => es

[HTTP_ACCEPT_ENCODING] => gzip, deflate

[HTTP_USER_AGENT] => Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322; InfoPath.1)

[HTTP_HOST] => www.ignside.net

[HTTP_CONNECTION] => Keep-Alive

[PATH] => C:\Python23\.;C:\Perl\bin\;C:\Archivos de programa\Windows Resource

Kits\Tools\;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\Archivos de programa\ATI Technologies\ATI Control Panel;C:\Archivos de

programa\Archivos comunes\GTK\2.0\bin;;c:\php;C:\Archivos de programa\MySQL\MySQL Server 5.0\bin

[SystemRoot] => C:\WINDOWS

print_r ($_SERVER)

[COMSPEC] => C:\WINDOWS\system32\cmd.exe

[PATHEXT] => .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.pyo;.pyc;.pyw;.py

[WINDIR] => C:\WINDOWS

[SERVER_SIGNATURE] => Apache/2.0.54 (Win32) PHP/5.1.2 DAV/2 Server at www.ignside.net Port 80

[SERVER_SOFTWARE] => Apache/2.0.54 (Win32) PHP/5.1.2 DAV/2

[SERVER_NAME] => www.ignside.net

[SERVER_ADDR] => 192.168.1.6

[SERVER_PORT] => 80

[REMOTE_ADDR] => 147.158.228.75

[**DOCUMENT_ROOT**] => E:/realbeta // Directorio raíz del servidor web

[SERVER_ADMIN] => irvmail@teleline.es

[SCRIPT_FILENAME] => E:/realbeta/man/php/ejemplos/print_r.php

[REMOTE_PORT] => 2445

[GATEWAY_INTERFACE] => CGI/1.1

[SERVER_PROTOCOL] => HTTP/1.1

[REQUEST_METHOD] => GET

[QUERY_STRING] =>

[REQUEST_URI] => /man/php/ejemplos/print_r.php

[SCRIPT_NAME] => /man/php/ejemplos/print_r.php

[**PHP_SELF**] => /man/php/ejemplos/print_r.php // Directorio actual de ejecución del script

print_r ($_SERVER)

```
<?php
    foreach( $_SERVER  as  $value ) {
            echo "Valor: $value<br>\n";
    }
?>
```

```
<?php
    foreach($_SERVER  as   $key => $value)
    {
            echo "<b>".$key ."</b> tiene el valor de ". $value ."<br>";
    }
?>
```