U.T. 2: Características del lenguaje PHP
Parte 2: Funciones y tipos de datos compuestos

Contenidos:

- Funciones
  - Inclusión de ficheros externos
  - Ejecución y creación de funciones
  - Argumentos
- Tipos de datos compuestos
  - Recorrer arrays
  - Funciones relacionadas con los tipos de datos compuestos

1.- Funciones

Cuando quieres repetir la ejecución de un bloque de código, puedes utilizar un bucle. Las **funciones** tienen una utilidad similar: **nos permiten asociar una etiqueta** (el nombre de la función) **con un bloque de código** a ejecutar. Además, al usar funciones estamos ayudando a estructurar mejor el código. Como ya sabes, las funciones permiten crear variables locales que no serán visibles fuera del cuerpo de las mismas.

Como programador puedes aprovecharte de la gran cantidad de funciones disponibles en PHP. De éstas, muchas están incluidas en el núcleo de PHP y se pueden usar directamente. Otras muchas se encuentran disponibles en forma de extensiones, y se pueden incorporar al lenguaje cuando se necesitan.

Con la distribución de PHP se incluyen varias extensiones. Para poder usar las funciones de una extensión, tienes que asegurarte de activarla mediante el uso de una directiva **extensión** en el fichero **php.ini**. Muchas otras extensiones no se incluyen con PHP y antes de poder utilizarlas tienes que descargarlas.

Para obtener extensiones para el lenguaje PHP puedes utilizar PECL. PECL es un repositorio de extensiones para PHP. Junto con PHP se incluye un **comando pecl** que puedes utilizar para instalar extensiones de forma sencilla:

pecl install nombre_extensión

En el curso puedes encontrar un enlace a información sobre PECL (PSM1)

1.1.- Inclusión de ficheros externos

Conforme vayan creciendo los programas que hagas, verás que resulta trabajoso encontrar la información que buscas dentro del código. En ocasiones resulta útil agrupar ciertos grupos de funciones o bloques de código, y ponerlos en un fichero aparte. Posteriormente, puedes hacer referencia a esos ficheros para que PHP incluya su contenido como parte del programa actual.

Para incorporar a tu programa contenido de un archivo externo, tienes varias posibilidades:

- **include**: Evalúa el contenido del fichero que se indica y lo incluye como parte del fichero actual, en el mismo punto en que se realiza la llamada. La ubicación del fichero puede especificarse utilizando una ruta absoluta, pero lo más usual es con una ruta relativa. En este caso, se toma como base la ruta que se especifica en la directiva **include_path** del fichero **php.ini**. Si no se encuentra en esa ubicación, se buscará también en el directorio del guión actual, y en el directorio de ejecución.

Ejercicio resuelto:

definiciones.php

```
<?php
        $modulo = 'DWES';
        $ciclo = 'DAW';
?>
```

programa.php

```
<?php
        print "Módulo $modulo del ciclo $ciclo<br />"; //Solo muestra "Modulo del ciclo"
        include 'definiciones.php';
        print " Módulo $modulo del ciclo $ciclo<br />"; // muestra "Modulo DWES del ciclo DAW"
?>
```

Cuando se comienza a evaluar el contenido del fichero externo, se abandona de forma automática el modo PHP y su contenido se trata en principio como etiquetas HTML. Por este motivo, es necesario delimitar el código PHP que contenga nuestro archivo externo utilizando dentro del mismo los delimitadores <?php y ?>.

En el curso de Moodle tienes un ejemplo de utilización de include (ER1)

1.1.- Inclusión de ficheros externos

- **include_once**: Si por equivocación incluyes más de una vez un mismo fichero, lo normal es que obtengas algún tipo de error (por ejemplo, al repetir una definición de una función). **include_once** funciona exactamente igual que **include**, pero solo incluye aquellos ficheros que aún no se hayan incluido.
- **require**: Si el fichero que queremos incluir no se encuentra, **include** da un aviso y continua la ejecución del guión. La diferencia más importante al usar **require** es que en ese caso, cuando no se puede incluir el fichero, se detiene la ejecución del guión.
- **require_once**. Es la combinación de las dos anteriores. Asegura la inclusión del fichero indicado solo una vez, y genera un error si no se puede llevar a cabo.

Muchos programadores utilizan la doble extensión .inc.php para aquellos ficheros en lenguaje PHP cuyo destino es ser incluidos dentro de otros, y nunca han de ejecutarse por sí mismos.

1.2.- Ejecución y creación de funciones

Ya sabes que para hacer una llamada a una función, basta con poner su nombre y unos paréntesis.

```
<?php
    phpinfo();
?>
```

Para crear tus propias funciones, deberás usar la palabra function.

```
<?php
    function precio_con_iva() {
            global $precio;
            $precio_iva = $precio * 1.21;
            print "El precio con IVA es ".$precio_iva;
    }
    $precio = 10;
    precio_con_iva();
?>
```

1.2.- Ejecución y creación de funciones

En PHP no es necesario que definas una función antes de utilizarla, excepto cuando está condicionalmente definida como se muestra en el siguiente ejemplo:

```
<?php
    $iva = true;
    $precio = 10;
    precio_con_iva();           // Da error, pues aquí aún no está definida la función
    if ($iva) {
        function precio_con_iva() {
                global $precio;
                $precio_iva = $precio * 1.21;
                print "El precio con IVA es ".$precio_iva;
        }
    }
    precio_con_iva();           // Aquí ya no da error
?>
```

Cuando una función está definida de una forma condicional sus definiciones deben ser procesadas antes de ser llamadas. Por tanto, la definición de la función debe estar antes de cualquier llamada.

1.3.- Argumentos

En el ejemplo anterior en la función usabas una variable global, lo cual no es una buena práctica. Siempre es mejor utilizar argumentos o parámetros al hacer la llamada. Además, en lugar de mostrar el resultado en pantalla o guardar el resultado en una variable global, las funciones pueden devolver un valor usando la sentencia **return**. Cuando en una función se encuentra una sentencia **return**, termina su procesamiento y devuelve el valor que se indica.

Puedes reescribir la función anterior de la siguiente forma:

```
<?php
    function precio_con_iva($precio) {
            return $precio * 1.21;
    }
    $precio = 10;
    $precio_iva = precio_con_iva($precio);
    print "El precio con IVA es ".$precio_iva
?>
```

Los argumentos se indican en la definición de la función como una lista de variables separada por comas. No se indica el tipo de cada argumento, al igual que no se indica si la función va a devolver o no un valor (si una función no tiene una sentencia **return**, devuelve **null** al finalizar su procesamiento).

1.3.- Argumentos

Al definir la función, puedes indicar valores por defecto para los argumentos, de forma que cuando hagas una llamada a la función puedes no indicar el valor de un argumento; en este caso se toma el valor por defecto indicado.

```
<?php
    function precio_con_iva($precio, $iva=0.21) {
            return $precio * (1 + $iva);
    }
    $precio = 10;
    $precio_iva = precio_con_iva($precio);
    print "El precio con IVA es ".$precio_iva
?>
```

Puede haber valores por defecto definidos para varios argumentos, pero en la lista de argumentos de la función todos ellos deben estar a la derecha de cualquier otro argumento sin valor por defecto.

1.3.- Argumentos

En los ejemplos anteriores los argumentos se pasaban por valor. Esto es, cualquier cambio que se haga dentro de la función a los valores de los argumento no se reflejará fuera de la función. Si quieres que esto ocurra debes definir el parámetro para que su valor se pase por referencia, añadiendo el símbolo & antes de su nombre.

```
<?php
    function precio_con_iva(&$precio, $iva=0.21) {
            $precio *= (1 + $iva);
    }
    $precio = 10;
    precio_con_iva($precio);
    print "El precio con IVA es ".$precio
?>
```

Anteriormente hiciste un ejercicio que mostraba la fecha actual en castellano. Con el mismo objetivo (puedes utilizar el código ya hecho), crea una función que devuelva una cadena de texto con la fecha en castellano, e introdúcela en un fichero externo. Después crea una página en PHP que incluya ese fichero y utilice la función para mostrar en pantalla la fecha obtenida. (Solución en Anexo I y en el curso de Moodle (ER2))

1.3.1.- Número variable de parámetros

- PHP permite una lista de valores de longitud variable como parámetro
- Funciones a usar:
  - **func_num_args()** → número de argumentos pasados a la función.
  - **func_get_args()** → array con los argumentos pasados a la función
  - **func_get_arg(num)** → el argumento que está en la posición num en la lista de argumentos. La primera posición es la 0

```
function prueba(){
        $num_args = func_num_args();
        echo "Numero de argumentos:$num_args<br/>\n";
        if ($num_args >= 2) {
                echo "El 2º argumento es:".func_get_arg(1)."<br/>\n";}
        $parametros=func_get_args();
        echo "Array con todos los argumentos:<br />\n";
        print_r($parametros);
}
prueba (1, 2, 3);
```

```
Numero de argumentos: 3
El 2º argumento es: 2
Array con todos los argumentos:

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

1.3.1.- Número variable de parámetros

- PHP 5.6: las listas de argumentos de las funciones pueden incluir el token ... para indicar que aceptan un número variable de parámetros.
- Los argumentos serán pasados a la variable dada como un array

```
<?php
function sum(...$números) {
	$acu = 0;
	foreach ($números as $n) {
		$acu += $n;
	}
	return $acu;
}
echo sum(1, 2, 3, 4); // El resultado sería 10
?>
```

1.3.2.- Funciones variables

- Definimos una variable: **$func=”prueba”;**
- Si llamamos a esa variable con paréntesis: $func() → PHP buscará una función con el mismo nombre que su contenido y la ejecutará si existe.
- Las funciones variables no funcionarán con echo(), print(), unset(), isset(), empty(), include(), require() y derivados.

```
function prueba($arg = ‘ ‘){
        echo "Estamos en la función prueba(); y el argumento es '$arg'.<br>\n";
}
$func = "prueba";
$func('hola'); // Esto llama prueba('hola')
```

1.3.3.- Funciones recursivas

- Una función se llama recursiva cuando en algún punto de su cuerpo se llama a sí misma.
- ¡Cuidado! puede llamarse a sí misma indefinidamente.
- Es muy importante la condición de salida.

```
<?php
function recursividad($a){
    if ($a < 20) {
        echo "$a\n";
        recursividad($a + 1);
    }
}
?>
```

1.3.4.- Funciones anónimas

- Las funciones anónimas, también se conocen como **clausuras** (closures).
- Permiten la creación de funciones que no tienen un nombre especificado.
- Las clausuras también se pueden usar como valores de variables.

```
<?php
    $saludo = function($nombre){
        printf("Hola %s\r\n", $nombre);};
    $saludo('Mundo');
    $saludo('PHP');
?>
```

1.3.5.- Funciones en PHP

- bool function_exists (string $function_name)
  - Comprueba la lista de funciones definidas, las incluidas (internas) y las definidas por el usuario, para ***function_name***.
  - Recibe el nombre de la función buscada como cadena.
  - Devuelve TRUE si function_name existe y es una función, si no, FALSE.

```
<?php
    if (function_exists('imap_open')) {
        echo "Las funciones de IMAP están disponibles.<br />\n";
    } else {
        echo "Las funciones de IMAP no están disponibles.<br />\n";
    }
?>
```

2.- Tipos compuestos

Un tipo de datos compuesto es aquel que te permite almacenar más de un valor. En PHP puedes utilizar dos tipos de datos compuestos: el **array** y el **objeto**. Los objetos los veremos más adelante; vamos a empezar con los arrays.

Un **array** es un tipo de datos que nos permite almacenar varios valores. Cada miembro del **array** se almacena en una posición a la que se hace referencia utilizando un valor clave. Las claves pueden ser numéricas o asociativas.

```
// array numérico
$modulos1 = array(0 => "Programación", 1 => "Bases de datos", ..., 9 => "Desarrollo web en entorno servidor");
// array asociativo
$modulos2 = array("PR" => "Programación", "BD" => "Bases de datos", ..., "DWES" => "Desarrollo web en entorno servidor");
```

En PHP existe la función **print_r**, que nos muestra todo el contenido del array que le pasamos. Es muy útil para tareas de depuración. Puedes encontrar un enlace a la documentación de esta función en el curso de Moodle.

Para hacer referencia a los elementos almacenados en un array, tienes que utilizar el valor clave entre corchetes:

```
$modulos1 [9]
$modulos2 ["DWES"]
```

2.- Tipos compuestos

Los arrays anteriores son vectores, esto es, arrays unidimensionales. En PHP puedes crear también arrays de varias dimensiones almacenando otro array en cada uno de los elementos de un array.

```
// array bidimensional
$ciclos = array(
    "DAW" => array ("PR" => "Programación", "BD" => "Bases de datos", ..., "DWES" => "Desarrollo web en entorno servidor"),
    "DAM" => array ("PR" => "Programación", "BD" => "Bases de datos", ..., "PMDM" => "Programación multimedia y de dispositivos móviles")
);
```

Para hacer referencia a los elementos almacenados en un array multidimensional, debes indicar las claves para cada una de las dimensiones:

```
$ciclos ["DAW"] ["DWES"]
```

2.- Tipos compuestos

En PHP no es necesario que indiques el tamaño del array antes de crearlo. Ni siquiera es necesario indicar que una variable concreta es de tipo array. Simplemente puedes comenzar a asignarle valores:

```
// array numérico
$modulos1 [0] = "Programación";
$modulos1 [1] = "Bases de datos";
...
$modulos1 [9] = "Desarrollo web en entorno servidor";

// array asociativo
$modulos2 ["PR"] = "Programación";
$modulos2 ["BD"] = "Bases de datos";
...
$modulos2 ["DWES"] = "Desarrollo web en entorno servidor";
```

Ni siquiera es necesario que especifiques el valor de la clave. Si la omites, el array se irá llenando a partir de la última clave numérica existente, o de la posición 0 si no existe ninguna:

```
$modulos1 [ ] = "Programación";
$modulos1 [ ] = "Bases de datos";
...
$modulos1 [ ] = "Desarrollo web en entorno servidor";
```

2.1.- Recorrer arrays

- **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);  // 7
```

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

2.1.- Recorrer arrays

Las **cadenas de texto** o **strings** se pueden tratar como arrays en los que se almacena una letra en cada posición, siendo 0 el índice correspondiente a la primera letra, 1 el de la segunda, etc.

```
// cadena de texto
$modulo = "Desarrollo web en entorno servidor";
// $modulo[3] == "a";
```

Para recorrer los elementos de un array, en PHP puedes usar un bucle específico: **foreach**. Utiliza una variable temporal para asignarle en cada iteración el valor de cada uno de los elementos del array. Puedes usarlo de dos formas. Recorriendo sólo los elementos:

```
$modulos = array("PR" => "Programación", "BD" => "Bases de datos", ..., "DWES" => "Desarrollo web en entorno servidor");
foreach ($modulos as $modulo) {
        print "Módulo: ".$modulo. "<br />"
}
```

O recorriendo los elementos y sus valores clave de forma simultánea:

```
$modulos = array("PR" => "Programación", "BD" => "Bases de datos", ..., "DWES" => "Desarrollo web en entorno servidor");
foreach ($modulos as $codigo => $modulo) {
        print "El código del módulo ".$modulo." es ".$codigo."<br />"
}
```

Haz una página PHP que utilice foreach para mostrar todos los valores del array $_SERVER en una tabla con dos columnas. La primera columna debe contener el nombre de la variable, y la segunda su valor. (Solución en Anexo II y en el curso de Moodle (ER3))

2.1.1.- Recorrer arrays

Pero en PHP también **hay otra forma de recorrer los valores de un array**. Cada array mantiene un **puntero** interno, que se puede utilizar con este fin. Utilizando funciones específicas, podemos avanzar, retroceder o inicializar el puntero, así como recuperar los valores del elemento (o de la pareja clave / elemento) al que apunta el puntero en cada momento. Algunas de estas funciones son:

Funciones para recorrer arrays.

Función	Resultado
reset	Sitúa el puntero interno al comienzo del array
next	Avanza el puntero interno una posición
prev	Mueve el puntero interno una posición hacia atrás
end	Sitúa el puntero interno al final del array
current	Devuelve el elemento de la posición actual
key	Devuelve la clave de la posición actual
each	Devuelve un array con la clave y el elemento de la posición actual. Además, avanza el puntero interno una posición.

2.1.1.- Recorrer arrays

Las funciones **reset**, **next**, **prev** y **end**, además de mover el puntero interno devuelven, al igual que **current**, el valor del nuevo elemento en que se posiciona. Si al mover el puntero te sales de los límites del array (por ejemplo, si ya estás en el último elemento y haces un **next**), cualquiera de ellas devuelve **false**. Sin embargo, al comprobar este valor devuelto no serás capaz de distinguir si te has salido de los límites del array, o si estás en una posición válida del array que contiene el valor "false".

La función **key** devuelve **null** si el puntero interno está fuera de los límites del array.

La función **each** devuelve un array con cuatro elementos. Los elementos **0** y **'key'** almacenan el valor de la clave en la posición actual del puntero interno. Los elementos **1** y **'value'** devuelven el valor almacenado.

Si el puntero interno del array se ha pasado de los límites del array, la función **each** devuelve **false**, por lo que la puedes usar para crear un bucle que recorra el array de la siguiente forma:

```
while ($modulo = each($modulos)) {

    print "El código del módulo ".$modulo[1]." es ".$modulo[0]."<br />"

}
```

Haz una página PHP que utilice estas funciones para crear una tabla como la del ejercicio anterior. (Solución en el Anexo III y en el curso de Moodle (ER4))

2.2.- Funciones relacionadas con los tipos de datos compuestos

Además de asignando valores directamente, **la función array permite crear un array con una sola línea de código**, tal y como vimos anteriormente. Esta función recibe un conjunto de parámetros, y crea un array a partir de los valores que se le pasan. Si en los parámetros no se indica el valor de la clave, crea un array numérico (con base 0). Si no se le pasa ningún parámetro, crea un array vacío.

```
$a = array();  // array vacío
$modulos = array("Programación", "Bases de datos", ..., "Desarrollo web en entorno servidor");
// array numérico
```

Una vez definido un array puedes añadir nuevos elementos (no definiendo el índice, o utilizando un índice nuevo) y modificar los ya existentes (utilizando el índice del elemento a modificar). También se pueden eliminar elementos de un array utilizando la función **unset**.

En el caso de los arrays numéricos, eliminar un elemento significa que las claves del mismo ya no estarán consecutivas.

```
unset ($modulos [0]);
// El primer elemento pasa a ser $modulos [1] == "Bases de datos";
```

2.2.- Funciones relacionadas con los tipos de datos compuestos

La función **array_values** recibe un array como parámetro, y devuelve un nuevo array con los mismos elementos y con índices numéricos consecutivos con base 0.

Para comprobar si una variable es de tipo array, utiliza la función **is_array**. Para obtener el número de elementos que contiene un array, tienes la función **count**.

Si quieres buscar un elemento concreto dentro de un array, puedes utilizar la función **in_array**. Recibe como parámetros el elemento a buscar y la variable de tipo array en la que buscar, y devuelve true si encontró el elemento o false en caso contrario.

```
$modulos = array("Programación", "Bases de datos", "Desarrollo web en entorno servidor");
$modulo = "Bases de datos";
if (in_array($modulo, $modulos)) printf "Existe el módulo de nombre ".$modulo;
```

Otra posibilidad es la función **array_search**, que recibe los mismos parámetros pero devuelve la clave correspondiente al elemento, o false si no lo encuentra.

Y si lo que quieres buscar es una clave en un array, tienes la función **array_key_exists**, que devuelve true o false.

En el curso de Moodle podrás encontrar un enlace a una lista completa de funciones para gestionar arrays. (PSM2)

2.2.1.- 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);
```

2.2.1.- 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);
```

2.2.1.- 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.

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

- 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.

2.2.1.- 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');
```

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 €

2.2.1.- 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));
```

2.2.1.- Modificar un array

- array_merge($mat1, $mat2, $mat3)
  - Une las matrices indicadas como parámetros, empezando por la primera. Elimina los elementos con claves duplicadas (dejando la última leída).

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

2.2.1.- 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.

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

2.2.1.- 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);
$res=array_slice($vec,1,3);      // $res= 6,7,8
```

Inicio	
Positivo	Posición del primer elemento contando desde el principio
Negativo	Posición de comienzo contando desde el final
Cantidad	
Positivo	Número de elementos a considerar
Negativo	Se detendrá a tantos elementos del final
Nulo	Se consideran todos los elementos hasta el final

2.2.1.- 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);
```

2.2.1.- 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);
```

2.2.1.- Modificar un array

- Intersección de matrices.
  - array_intersect($mat1,$mat2,$mat3)
    - Devuelve una matriz con los elementos comunes a las matrices indicadas. La comparación se hace con el operador identidad (===)
  - array_intersect_assoc($mat1,$mat2,$mat3)
    - Devuelve una matriz con los elementos comunes utilizando el operador identidad (===). En la comparación se tienen en cuenta también las claves.

2.2.1.- Modificar un array

- Creación de una matriz con los elementos únicos de otra:
  - array_unique ($mat)
    - Crea una nueva matriz a partir de otra original, eliminando las repeticiones de los elementos duplicados. Utiliza el operador de identidad en la comparación (==).
  - array_combine ($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.

2.2.1.- 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);
```

- 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)
```

2.2.1.- 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.

2.2.2.- Ordenar un array

- Ordenación de arrays:
  - 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.

2.2.2.- Ordenar un array

- Ordenación de arrays:
  - 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 $key_compare_func)
    - Ordena las claves de un array usando una función de comparación proporcionada por el usuario.
  - bool uasort (array &$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.

Anexo I.- Solución propuesta II

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 2 : Características del Lenguaje PHP -->
<!-- Ejemplo: Utilización include -->
<html>
    <head>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        <title>Fecha en castellano</title>
    </head>
    <body>
<?php
    include 'funciones.inc.php';
    print fecha();
?>
    </body>
</html>
```

Anexo I.- Solución propuesta II

```
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 2 : Características del Lenguaje PHP -->
<!-- Ejemplo: Utilización include -->
<!-- Fichero: funciones.inc.php -->
<?php
    // Función que devuelve un texto con la fecha actual en castellano
    function fecha()
    {
    date_default_timezone_set('Europe/Madrid');
    $numero_mes = date("m");
    $numero_dia_semana = date("N");
    switch($numero_mes)     {
        case 1: $mes = "Enero";
            break;
        case 2: $mes = "Febrero";
            break;
        case 3: $mes = "Marzo";
            break;
        case 4: $mes = "Abril";
            break;
        case 5: $mes = "Mayo";
            break;
        case 6: $mes = "Junio";
            break;
        case 7: $mes = "Julio";
            break;
        case 8: $mes = "Agosto";
            break;
        case 9: $mes = "Septiembre";
            break;
        case 10: $mes = "Octubre";
            break;
        case 11: $mes = "Noviembre";
            break;
        case 12: $mes = "Diciembre";
            break;
    }
    switch($numero_dia_semana) {
        case 1: $dia_semana = "Lunes";
            break;
        case 2: $dia_semana = "Martes";
            break;
        case 3: $dia_semana = "Miércoles";
            break;
        case 4: $dia_semana = "Jueves";
            break;
        case 5: $dia_semana = "Viernes";
            break;
        case 6: $dia_semana = "Sábado";
            break;
        case 7: $dia_semana = "Domingo";
            break;
    }
    return $dia_semana.", ".date("j")." de ".$mes." de ".date("Y");
    }
?>
```

Anexo II.- Solución propuesta III

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 2 : Características del Lenguaje PHP -->
<!-- Ejemplo: Tabla con los valores del array $_SERVER utilizando foreach -->
<html>
    <head>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        <title>Tabla</title>
        <style type="text/css">
            td, th {border: 1px solid grey; padding: 4px;}
            th {text-align:center;}
            table {border: 1px solid black;}
        </style>
    </head>
    <body>
        <table>
        <tbody>
            <tr>
                <th>Variable</th>
                <th>Valor</th>
            </tr>
<?php
    foreach ($_SERVER as $variable => $valor) {
print "<tr>";
print "<td>".$variable."</td>";
print "<td>".$valor."</td>";
print "</tr>";
    }
?>
        </tbody>
        </table>
    </body>
</html>
```

Anexo III.- Solución propuesta IV

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 2 : Características del Lenguaje PHP -->
<!-- Ejemplo: Tabla con los valores del array $_SERVER utilizando función each -->
<html>
    <head>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        <title>Tabla</title>
        <style type="text/css">
            td, th {border: 1px solid grey; padding: 4px;}
            th {text-align:center;}
            table {border: 1px solid black;}
        </style>
    </head>
    <body>
        <table>
        <tbody>
            <tr>
                <th>Variable</th>
                <th>Valor</th>
            </tr>
<?php
    reset($_SERVER);
    while ($valor = each($_SERVER)) {
        print "<tr>";
        print "<td>".$valor[0]."</td>";
        print "<td>".$valor[1]."</td>";
        print "</tr>";
    }
?>
        </tbody>
        </table>
    </body>
</html>
```