TEMA 3. PROGRAMACIÓN BASADA EN LENGUAJE DE MARCAS CON CÓDIGO EMBEBIDO.

Objetivos

- Conocer y utilizar los mecanismos disponibles de decisión.
- Aprender a utilizar bucles y su sintaxis.
- Estudiar la forma de almacenar y recuperar conjunto de datos utilizando variables compuestas (arrays)
- Aprender a utilizar y definir funciones.
- Comprender los diferentes métodos de recuperación de información enviada por un cliente.
- Aprender a procesar la información enviada por un cliente a través de formularios.

Contenidos

3.1.- Sentencias Condicionales.

Permiten evaluar condiciones y ejecutar ciertas instrucciones si la condición es verdadera y ejecutar otras instrucciones si la condición es falsa.

Existen dos tipos de sentencias condicionales: sentencia if, sentencia switch.

Sentencias if.

Indica si debe ejecutar una parte de código en base al valor lógico de una expresión condicional.

Existen hay tres tipos de estructuras if.

a) if.

Si se cumple una condición ejecuta unas sentencias.

```
if (condicion) {
      sentencias;
}
```

Las llaves no son obligatorias en caso de que sólo se ejecute una sentencia.

b) if else.

Si se cumple una condición ejecuta unas sentencias y el caso contrario ejecuta otras.

```
if (condicion) {
      sentencias;
} else {
      sentencias;
}
```

c) if else if.

Si se cumple una condición ejecuta unas sentencias y el caso contrario comprueba otra condición ejecutando unas sentencias si se cumple y así sucesivamente.

```
if (condicion) {
      sentencias;
} else if (condicion) {
      sentencias;
} elseif (condicion) {
      sentencias;
} else {
      sentencias;
}
```

Ejemplos:
Si el sueldo del empleado es mayor de 1500 euros calcular una retención del 12% sobre el sueldo.

```
$sueldo = 1600;
if ($sueldo > 1500) {
      $retencion = $sueldo * 0.12;
      echo "La retención es de: " . $retencion;
}
```

Si el sueldo del empleado es mayor de 1500 euros calcular una retención del 12% sobre el sueldo en caso contrario calcular una retención del 5%.

```
$sueldo = 1200;
if ($sueldo > 1500) {
      $retencion = $sueldo * 0.12;
} else {
      $retencion = $sueldo * 0.05;
}
echo  "La retención es de: " . $retencion;
```

Si el sueldo del empleado es menor de 1200 euros calcular una retención del 12% sobre el sueldo en caso contrario si el sueldo es mayor o igual a 1200 y menor o igual a 2500 calcular una retención del 18% sobre el sueldo y si no calcular una retención del 25% sobre el sueldo.

```
$sueldo = 1800;
if ($sueldo < 1200) {
      $retencion = $sueldo * 0.12;
```

```
} else if ($sueldo >= 1200 && $sueldo <= 1500){
      $retencion = $sueldo * 0.18;
} else {
      $retencion = $sueldo * 0.25;
}
echo "La retención es de: " . $retencion;
```

Sentencias switch.

Compara el valor de una variable con una serie de valores conocidos. Si uno de los valores coincide con el valor de la variable se ejecuta el código asociado a dicho valor conocido.

```
switch (expresion){
      case valor1: sentencias1; break;
      case valor1: sentencias1; break;
            :     :     :     :
      case valorn: sentenciasn; break;
      [default: sentenciasdef;]
}
```

Ejemplo:
Si un empleado tiene categoría laboral 1 se le aplica un 10% de retención sobre el sueldo, si tiene categoría laboral 2 se le aplica un 15% de retención sobre el sueldo, si tiene categoría laboral 3 se le aplica un 20% de retención sobre el sueldo, si tiene categoría laboral 4 se le aplica un 25% de retención sobre el sueldo el cualquier otro caso se le aplica un 30% de retención sobre el sueldo.

```
$sueldo = 1800;
$catLaboral = 3;
$retencion;
switch ($catLaboral){
      case 1: $retencion = $sueldo * 0.10; break;
      case 2: $retencion = $sueldo * 0.15; break;
      case 3: $retencion = $sueldo * 0.20; break;
      case 4: $retencion = $sueldo * 0.25; break;
      default: $retencion = $sueldo * 0.30;
}
echo "La retención es de: " . $retencion;
```

3.2.- Sentencias bucle.

Permiten ejecutar un fragmento de código de forma repetida mientras la condición sea verdadera. Las estructuras de repetición o bucles son utilizadas cuando unas sentencias han de ser ejecutadas cero, una o más veces.

Hay que tener cuidado con los bucles infinitos que no terminan nunca pues la página seguirá ejecutando el bloque de instrucciones y el navegador se quedará colgado.

a) Bucle while.

El bucle **while** se utiliza cuando se tiene que ejecutar un grupo de sentencias un número determinado de veces. Las sentencias podrían no llegar a ejecutarse ya que inicialmente podría no cumplirse la condición de ejecución de bucle.

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

Ejemplo:

```
$contador = 1;
while ($contador<=10){
      echo $contador."<br>";
      $contador++;
}
```

Ejemplo:

```
$contador = 0;
$poblacion = array ("Valencia", "Castellón", "Alicante");
while ($contador <=2) {
      echo $poblacion[$contador]."<br>";
      $contador++;
}
```

b) Bucle do ... while.

El bucle **do ... while** se utiliza cuando se tiene que ejecutar un grupo de sentencias un número determinado de veces. Las sentencias se ejecutan al menos una vez ya que la comprobación de la condición de salida del bucle se encuentra después de las sentencias del bucle.

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

Ejemplo:

```
$contador = 1;
do {
      echo $contador."<br>";
      $contador++;
} while ($contador<=10);
```

Ejemplo:

```
$contador = 0;
$poblacion = array("Valencia", "Castellón", "Alicante");
do {
      echo $poblacion[$contador]."<br>";
```

```
        $contador++;
    } while ($contador<=2);
```

b) Bucle for.

El bucle **for** se utiliza cuando se tiene que ejecutar un grupo de sentencias un número fijo y conocido de veces. La sentencia **for** se puede conseguir utilizando el bucle while.

```
for(inicializacion; condicion; incremento){
      sentencias;
}
```

Ejemplo:

```
for($contador = 1; $contador<=10; $contador++){
      echo $contador."<br>";
}
```

Ejemplo:

```
for($contador = 10; $contador>=1; $contador--){
      echo $contador."<br>";
}
```

Ejemplo:

```
for($contador = 1; $contador<=9; $contador+=2){
       echo $contador."<br>";
}
```

Ejemplo:

```
$poblacion = array("Valencia", "Castellón", "Alicante");
for($contador = 0; $contador<=2; $contador++){
      echo $poblacion[$contador]."<br>";
}
```

b) Bucle foreach.

El bucle **foreach** se utiliza para recorrer de forma sencilla los elementos de un array.

```
foreach(nombre_array as nombre_variable){
      sentencias;
}

foreach(nombre_array as nombre_indice => nombre_variable){
      sentencias;
}
```

Ejemplo:

```
$poblacion = array("Valencia", "Castellón", "Alicante");
foreach($poblacion as  $valor){
            echo $valor."<br>";
}
```

Ejemplo:

```
$poblacion = array("Valencia", "Castellón", "Alicante");
foreach($poblacion as  $pos => $valor){
            echo "La posición ".$pos." contiene ".$valor."<br>";
}
```

3.3.- Estructuras de salto.

a) Sentencias break y continue.

La sentencia **break** se puede utilizar tanto en estructuras de selección como en estructuras de repetición y permite salir de un bloque de sentencias.
La sentencia **continue** se puede utilizar solo en estructuras de repetición y permite saltar desde el bloque de sentencias a la sentencia de evaluación de la condición.

Ejemplo:

```
$contador = 0;
while ($contador < 10){
      $contador++;
      if ($contador==5) {
            break;
      }
      echo $contador."<br>";
}
```

Ejemplo:

```
$contador = 0;
while ($contador < 10){
      $contador++;
      if ($contador==5) {
            continue;
      }
      echo $contador."<br>";
}
```

3.4.- Tipos de Datos Complejos.

Los arrays son estructuras que permiten el almacenamiento de una serie de datos identificados por un índice o posición.
Las matrices son arrays multidemensionales, es decir, array que almacenan array de datos.

Declaración de un array.
Opción 1.

```
nombre_array = array(valor1, valor2, ... valorn);
```

Opción 2.

```
nombre_array = array(clave1=>valor1, clave2=>valor2,… claven=>valorn);
```

Ejemplos:

```
$empleados = array('Luis', 'Pedro', 'Ana');
```

```
$empleado = array('nombre'=>'Luis', 'edad'=>32,… 'sueldo'=>1548.63);
```

Declaración de una matriz o array multidimensional.
Opción 1.

```
nombre_array = array(   array(valor0_0, valor0_1, ... valor0_m),
                        array(valor1_0, valor1_1, ... valor1_m),
                        :     :     :     :     :     :
                        array(valorn_0, valorn_1, ... valorn_m)
                    );
```

Opción 2.

```
nombre_array = array(clave1=>array(),… claven=>array());
```

Ejemplos:

```
$alumnos = array( array(100, 'Pedro', 4.5),
                  array(140, 'Ana', 8.4),
                  array(190, 'Marta', 7.3)
                 );
```

```
$alumnos = array( 0 => array( 0=> 100, 1=> 'Pedro', 2=> 4.5),
                  1 => array(0 => 140, 1 => 'Ana', 2=> 8.4),
                  2 => array(0 => 190, 1=> 'Marta', 2=> 7.3)
                  );
```

Acceso a elementos del array.

Para acceder a un elemento del array lo haremos mediante el nombre de la variable y entre corchetes la posición del elemento.

$nombre_array[pos]

Ejemplo:

```
$empleado[2];     //    Ana
```

Acceso a elementos de una matriz o array multidimensional.

Para acceder a un elemento del array multidimensional o matriz lo haremos mediante el nombre de la variable y entre corchetes las posiciones del elemento.

$nombre_array[posx] [posy]

Ejemplo:

```
$empleados[1][2];       //    8.4
```

Acceder a todos los elementos del array.

Si queremos acceder a todos los elementos de un array lo haremos con la sentencia for o foreach para poder iterar sobre los índices del array y acceder a su valor en casa posición.

Ejemplo 1:

```
$empleados = array('Luis', 'Pedro', 'Ana');
foreach($empleados as $valor){
      echo "El elemento es ".$valor."<br>";
}
```

Ejemplo 2:

```
$empleados = array('Luis', 'Pedro', 'Ana');
foreach($empleados as $indice => $valor){
      echo "El elemento de la posición ".$indice." es ".$valor."<br>";
}
```

Ejemplo 3:

```
$empleado = array('nombre'=>'Luis', 'edad'=>32,… 'sueldo'=>1548.63);
foreach($empleados as $indice => $valor){
      echo "El elemento de la posición ".$indice." es ".$valor."<br>";
}
```

Ejemplo 4:

```
$alumnos = array( array(100, 'Pedro', 4.5),
                  array(140, 'Ana', 8.4),
                  array(190, 'Marta', 7.3)
                 );
foreach($alumnos as $indice => $valor){
      echo "Posición ".$indice.":";
      foreach($valor as $pos => $elemento){
            echo $elemento." ";
      }
      echo "<br>";
}
```

Borrar un array.

```
unset($nombre_array)
```

Ejemplo:

```
unset($empleados);
```

Borrar un elemento de una posición concreta del array.

```
unset($nombre_array[pos])
```

Ejemplo:

```
unset($empleado[1]);
```

Añadir un elemento al array.

```
$nombre_array[]=valor;
```

Ejemplo:

```
$empleado[]="Roberto";
```

Ejemplo:

```
$alumnos = array('Pedro', 'Ana', 'Marta');
unset($alumnos);
foreach($alumnos as $indice => $valor){
            echo $indice.": ".$valor."<br>";
}
echo "<br>";

$alumnos = array('Pedro', 'Ana', 'Marta');
unset($alumnos[1]);
foreach($alumnos as $indice => $valor){
            echo $indice.": ".$valor."<br>";
}
echo "<br>";

$alumnos = array('Pedro', 'Ana', 'Marta');
$alumnos[]="Roberto";
foreach($alumnos as $indice => $valor){
            echo $indice.": ".$valor."<br>";
}
```

Operaciones con Arrays.

a) Búsqueda en un array.

Existen dos tipos de búsqueda de datos dentro de un array: secuancial y binaria.

La búsqueda secuencial es más sencilla y pero menos eficiente. La búsqueda binaria es más compleja, mucho más eficiente y actúa sobre array previamente ordenados.

Búsqueda secuencial.

Compara cada elemento del array con el elemento buscado y termina cuando encontramos el elemento buscado devolviendo la posición o cuando termina de recorrer el array devolviendo que el elemento no fue encontrado.

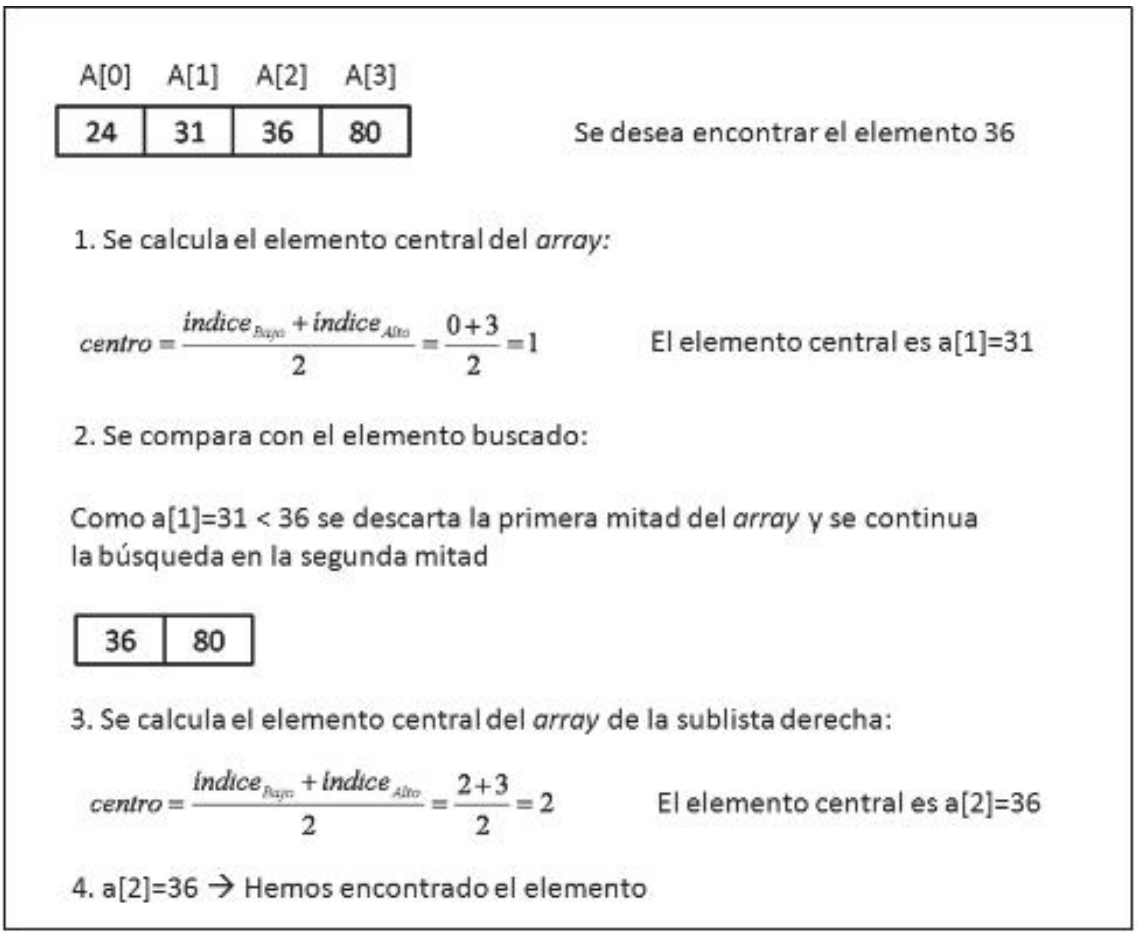
Ejemplo:

```
$pos=-1;
$nombre= "Ana";
```

```
$alumnos = array('Pedro', 'Ana', 'Marta', "Roberto", "Rosa",
"Santiago");
foreach($alumnos as $indice => $valor){
      if ($valor==$nombre) {
            $pos=$indice;
            break;
      }
}
if ($pos >= 0)
      echo "Elemento encontrado en la posición ".$pos."<br>";
else
      echo "Elemento no encontrado<br>";
```

Búsqueda binaria.

El array debe estar ordenado y debemos dividir el array en dos mitades y comprobar si el elemento del centro es el que buscamos. Si no determinamos en que mitad debemos seguir buscando y volvemos a realizar el mismo proceso hasta que los índices se crucen o se encuentre el elemento.



Ejemplo:

```
	$elementos = 7;
	$inf=0;
	$sup=$elementos-1;
	$medio=($inf+$sup)/2;
	echo $medio;
	$alumnos = array ("Ana", "Luis", "María", "Montse", "Raquel",
"Rosa", "Sara");
	$valor_buscado = "Raquel";

	while($inf<=$sup && $alumnos[$medio]!=$valor_buscado) {
		if($valor_buscado<$alumnos[$medio])
			$sup=$medio-1;
		else
			$inf=$medio+1;
		$medio=($inf+$sup)/2;
	}

	if($valor_buscado==$alumnos[$medio])
		echo"El valor se encuentra en la posición ".$medio."<br>";
	else
		echo"El valor NO se encuentra<br>";
```

Afortunadamente en PHP disponemos de funciones para buscar elementos en un array sin necesidad de codificar nuestro propio algoritmo.

```
array_search($valor_buscado, $nombre_array);
```

Devuelve la posición que encontró el elemento en el array o false si no lo encontró.

Ejemplo:

```
$alumnos=array("Ana","Luis","María","Montse","Raquel","Rosa","Sara");
$valor_buscado = "Ana";
$busca = array_search($valor_buscado, $alumnos);
if ($busca===false)
	echo "Elemento no encontrado";
else
	echo "La posición es ".$busca;
```

b) Ordenación de arrays.

Podemos ordenar arrays los elementos del array utilizando diferentes sentencias:

Sentencia sort.
Ordena los valores en orden ascendente alterando el índice asociado.

```
sort ($nombre_array)
```

Ejemplo:

```
$numeros = array(5,8,2,4,1,9,7,0);
sort($numeros);
foreach($numeros as $indice => $valor){
      echo $indice.": ".$valor."<br>";
}
```

Sentencia rsort.
Ordena los valores en orden descendente alterando el índice asociado.

```
rsort ($nombre_array)
```

Ejemplo:

```
$numeros = array(5,8,2,4,1,9,7,0);
rsort($numeros);
foreach($numeros as $indice => $valor){
      echo $indice.": ".$valor."<br>";
}
```

Sentencia asort.
Ordena los valores en orden ascendente respetando el índice asociado.

```
asort ($nombre_array)
```

Ejemplo:

```
$numeros = array(5,8,2,4,1,9,7,0);
asort($numeros);
foreach($numeros as $indice => $valor){
      echo $indice.": ".$valor."<br>";
}
```

Sentencia arsort.
Ordena los valores en orden descendente respetando el índice asociado.

```
arsort ($nombre_array)
```

Ejemplo:

```
$numeros = array(5,8,2,4,1,9,7,0);
arsort($numeros);
foreach($numeros as $indice => $valor){
      echo $indice.": ".$valor."<br>";
}
```

Sentencia ksort.

Ordena los índices del array en orden ascendente respetando los valores asociados en cada posición.

```
ksort ($nombre_array)
```

Ejemplo:

```
$numeros=array(7=>5, 6=>8, 9=>2, 3=>4, 2=>1, 8=>9, 5=>7, 4=>0);
ksort($numeros);
foreach($numeros as $indice => $valor){
      echo $indice.": ".$valor."<br>";
}
```

Sentencia krsort.

Ordena los índices del array en orden decendente respetando los valores asociados en cada posición.

```
krsort ($nombre_array)
```

Ejemplo:

```
$numeros=array(7=>5, 6=>8, 9=>2, 3=>4, 2=>1, 8=>9, 5=>7, 4=>0);
krsort($numeros);
foreach($numeros as $indice => $valor){
      echo $indice.": ".$valor."<br>";
}
```

c) Recorrido de arrays no secuenciales.

PHP cuenta con el siguiente conjunto de funciones para el recorrido de arrays no secuenciales:

current.

Devuelve el valor de la posición actual del puntero dentro del array. Devuelve false cuando el puntero está al final del array o cuando el array no contiene ningún elemento.

```
current($nombre_array)
```

Ejemplo:

```
$numeros=array(7=>5, 6=>8, 9=>2, 3=>4, 2=>1, 8=>9, 5=>7, 4=>0);
echo current($numeros);
```

pos.

Es idéntica a la función anterior.

```
pos($nombre_array)
```

Ejemplo:

```
$numeros=array(7=>5, 6=>8, 9=>2, 3=>4, 2=>1, 8=>9, 5=>7, 4=>0);
echo pos($numeros);
```

key.

Devuelve el índice de la posición actual del array pasado como argumento.

```
key($nombre_array)
```

Ejemplo:

```
$numeros=array(7=>5, 6=>8, 9=>2, 3=>4, 2=>1, 8=>9, 5=>7, 4=>0);
echo key($numeros);
```

next.

Devuelve el valor del elemento siguiente al actual (si existe) y avanza el puntero interno una posición. En caso de que el elemento actual sea el último del array, devuelve false.

```
next($nombre_array)
```

Ejemplo:

```
$numeros=array(7=>5, 6=>8, 9=>2, 3=>4, 2=>1, 8=>9, 5=>7, 4=>0);
echo next($numeros);
```

prev.

Devuelve el valor del elemento anterior al actual (si existe) y retrocede el puntero interno una posición. En caso de que el elemento actual sea el primero del array, devuelve false.

```
prev($nombre_array)
```

Ejemplo:

```
$numeros=array(7=>5, 6=>8, 9=>2, 3=>4, 2=>1, 8=>9, 5=>7, 4=>0);
```

```
echo next($numeros);
echo "<br>";
echo prev($numeros);
```

end.

Coloca el puntero interno en el último elemento de un array y devuelve el valor asociado.

```
end($nombre_array)
```

Ejemplo:

```
$numeros=array(7=>5, 6=>8, 9=>2, 3=>4, 2=>1, 8=>9, 5=>7, 4=>0);
echo end($numeros);
```

reset

Devuelve el valor del primer elemento del array y sitúa el puntero interno en su primera posición.

```
reset($nombre_array)
```

Ejemplo:

```
$numeros=array(7=>5, 6=>8, 9=>2, 3=>4, 2=>1, 8=>9, 5=>7, 4=>0);
echo end($numeros);
echo "<br>";
echo reset($numeros);
```

d) Número de elementos de un array.

Mediante sizeof podemos saber el número de elementos de un array pasado como parámetro. Si el parámetro modo tiene el valor 1 cuenta los elementos de un array multidimensionales.

```
sizeof($nombre_array[, modo])
```

Ejemplo 1:

```
$numeros=array(7=>5, 6=>8, 9=>2, 3=>4, 2=>1, 8=>9, 5=>7, 4=>0);
echo sizeof($numeros);
```

Ejemplo 2:

```
$coches = array (
      array("Volvo",22,18),
      array("BMW",15,13),
      array("Saab",5,2),
      array("Land Rover",17,15)
      );
echo sizeof($coches)."<br>";
echo sizeof($coches,1)."<br>";
```

e) Insertar elementos al final de un array.

Podemos insertar elementos al final de un array mediante array_push().

```
array_push($nombre_array, $valor)
```

Ejemplo:

```
$numeros=array(7=>5, 6=>8, 9=>2, 3=>4, 2=>1, 8=>9, 5=>7, 4=>0);
array_push($numeros, 10);
foreach($numeros as $indice => $valor){
            echo $indice.": ".$valor."<br>";
}
```

f) Insertar elementos al principio de un array.

Podemos insertar elementos al principio de un array mediante array_unshift(). Reordena los índices.

```
array_unshift($nombre_array, $valor)
```

Ejemplo:

```
$numeros=array(7=>5, 6=>8, 9=>2, 3=>4, 2=>1, 8=>9, 5=>7, 4=>0);
array_unshift($numeros, 25);
foreach($numeros as $indice => $valor){
            echo $indice.": ".$valor."<br>";
}
```

g) Inserta, eliminar y sustituir elementos de un array.

Podemos eliminar, insertar o sustituir elementos de un array con array_splice. Elimina elementos de un array, sustituyéndolos opcionalmente por los elementos de otro array (sustituto). Elimina y si está indicado, sustituye los elementos que están situados a partir de la posición indicada por despl. Devuelve los elementos eliminados. Los significados de los diferentes valores que pueden tomar despl y tam se especifican en la siguiente tabla:

array_splice(matriz, despl [,tam [array sustituto])

despl	
Valor	**Significado**
positivo	Posición de comienzo de la sustitución/eliminación
negativo	Posición de comienzo de la sustitución/eliminación desde el final
tam	
Valor	**Significado**
positivo	Número de elementos a eliminar/sustituir
negativo	Último elemento a eliminar/sustituir desde el final
nulo	Se eliminan/sustituyen todos los elementos hasta el final del *array*

Ejemplo Insertar:

```
$numeros=array(5, 8, 2, 4, 1, 9, 7, 0);
$nuevos=array(20,30,40);
array_splice($numeros, 3, 0, $nuevos);
foreach($numeros as $indice => $valor){
        echo $indice.": ".$valor."<br>";
}
```

Ejemplo Eliminar:

```
$numeros=array(5, 8, 2, 4, 1, 9, 7, 0);
array_splice($numeros, 3, 2);
foreach($numeros as $indice => $valor){
        echo $indice.": ".$valor."<br>";
}
```

Ejemplo Eliminar y devuelve elementos eliminados:

```
$numeros=array(5, 8, 2, 4, 1, 9, 7, 0);
$eliminados = array_splice($numeros, 3, 2);
foreach($numeros as $indice => $valor){
        echo $indice.": ".$valor."<br>";
}
echo "<br>";
foreach($eliminados as $indice => $valor){
        echo $indice.": ".$valor."<br>";
}
```

Ejemplo Sustituye:

```
$numeros=array(5, 8, 2, 4, 1, 9, 7, 0);
$nuevos=array(20,30,40);
array_splice($numeros, 3, 2, $nuevos);
foreach($numeros as $indice => $valor){
        echo $indice.": ".$valor."<br>";
}
```

Ejemplo Sustituye y devuelve elementos sustituidos:

```
$numeros=array(5, 8, 2, 4, 1, 9, 7, 0);
$nuevos=array(20,30,40);
$eliminados = array_splice($numeros, 3, 2, $nuevos);
foreach($numeros as $indice => $valor){
        echo $indice.": ".$valor."<br>";
}
echo "<br>";
foreach($eliminados as $indice => $valor){
        echo $indice.": ".$valor."<br>";
}
```

h) Devolver un subarray de un array.

Podemos devolver parte de un array con array_slice. Devuelve los elementos del array que están situados a partir de una posición determinada por desplazamiento. Opcionalmente, podemos indicar el total de elementos que queremos.

`array_slice(matriz, desplazamiento [, tamaño]):`

Los parámetros desplazamiento y tamaño pueden ser valores positivos o negativos. Esto da lugar a las siguientes interpretaciones reflejadas en la tabla de abajo:

desplazamiento	
Valor	**Significado**
positivo	Posición de comienzo
negativo	Posición de comienzo desde el final
tamaño	
Valor	**Significado**
positivo	Número de elementos a considerar
negativo	Último elemento a considerar desde el final
nulo	Se consideran todos los elementos hasta el final del *array*

Ejemplo parte de array desde posición hasta final:

```
$numeros=array(5, 8, 2, 4, 1, 9, 7, 0);
$parte_numeros = array_slice($numeros, 3);
foreach($parte_numeros as $indice => $valor){
            echo $indice.": ".$valor."<br>";
}
```

Ejemplo parte de array desde posición un número de elementos:

```
$numeros=array(5, 8, 2, 4, 1, 9, 7, 0);
$parte_numeros = array_slice($numeros, 3, 2);
foreach($parte_numeros as $indice => $valor){
            echo $indice.": ".$valor."<br>";
}
```

i) Array en orden inverso.

Devuelve el array pasado como parámetro pero con sus componentes en orden inverso.

`array_reverse($nombre_array)`

Ejemplo:

```
$numeros=array(5, 8, 2, 4, 1, 9, 7, 0);
$inverso = array_reverse($numeros);
foreach($inverso as $indice => $valor){
            echo $indice.": ".$valor."<br>";
}
```

j) Array valores enteros consecutivos.

Devuelve un array con los valores enteros comprendidos entre el primer argumento y el segundo que recibe la función, ambos inclusive.

```
range($limite_inf, $limite_sup)
```

Ejemplo:

```
$numeros=range(30, 45);
foreach($numeros as $indice => $valor){
            echo $indice.": ".$valor."<br>";
}
```

k) Array de frecuencia de valores.

Recibe como argumento un array y devuelve otro cuyos índices son los mismos que los del array original y sus valores asociados son la frecuencia con la que se repiten dichos valores en el array original.

```
array_count_values($nombre_array)
```

Ejemplo:

```
$numeros = array(1, 2, 3, 1, 3, 4, 3, 4, 1, 2, 2, 1, 2, 2, 2);
$frecuencia = array_count_values($numeros);
foreach($frecuencia as $indice => $valor){
            echo $indice.": ".$valor."<br>";
}
```

l) Comprobar si existe un elemento en un array.

Podemos saber si un elemento está contenido en un array con In_array.

```
in_array($elemento, $nombre_array)
```

Ejemplo 1:

```
$numeros = array(1, 2, 3, 1, 3, 4, 3, 4, 1, 2, 2, 1, 2, 2, 2);
if (in_array(10, $numeros))
      echo "Existe";
else
      echo "No Existe";
```

Ejemplo 2:

```
$numeros = array(1, 2, 3, 1, 3, 4, 3, 4, 1, 2, 2, 1, 2, 2, 2);
if (in_array(4, $numeros))
      echo "Existe";
else
      echo "No Existe";
```

m) Combinar varios arrays.

Podemos combinar 2 o más arrays en otro array con array_merge().

```
array_merge($nombre_array1,.. $nombre_arrayn)
```

Ejemplo:

```
$alumnos1 = array("Luis", "Pedro", "Ana");
$alumnos2 = array("Marta", "Rosa", "Angel");
$alumnos = array_merge($alumnos1, $alumnos2);
foreach($alumnos as $indice => $valor){
            echo $indice.": ".$valor."<br>";
}
```

n) Devolver el valor de todos los elementos del array.

Podemos devolver el valor de todos los elementos de un array con array_values().

```
array_values($nombre_array);
```

Ejemplo:

```
$alumnos = array("Luis", "Pedro", "Ana", "Marta", "Rosa", "Angel");
print_r(array_values($alumnos));
```

o) Devolver el número de elementos del array.

Podemos devolver el número de elementos de un array con count(). Si el parámetro modo tiene el valor 1 cuenta los elementos de un array multidimensionales.

```
count($nombre_array[, modo]);
```

Ejemplo 1:

```
$alumnos = array("Luis", "Pedro", "Ana", "Marta", "Rosa", "Angel");
echo count($alumnos);
```

Ejemplo 2:

```
$coches = array (
      array("Volvo",22,18),
      array("BMW",15,13),
      array("Saab",5,2),
      array("Land Rover",17,15)
      );
echo count($coches)."<br>";
echo count($coches,1)."<br>";
```

3.5.- Principios de Subprogramación.

Un subprograma es un fragmento de código que tiene una funcionalidad específica, permitiendo que sea modular y reutilizable.

Existen dos tipos de subprogramas: las funciones que se ejecutan y devuelven un valor y los procedimientos que se ejecutan sin devolver un valor.

a) Función. La ejecución devuelve un valor. Utilizamos la sentencia **return** para que la función devuelva el valor.

```
function nombre_funcion($arg1, $arg2, …) {
      sentencias;
      return $valordevuelto;
}
```

Ejemplo:

```
function sumar($num1, $num2) {
      $suma = $num1 + $num2;
      return $suma;
}
```

b) Procedimientos. La ejecución no devuelve un valor.

```
function nombre_funcion($arg1, $arg2, …) {
      sentencias;
}
```

Ejemplo:

```
function sumar($num1, $num2) {
      $suma = $num1 + $num2;
      echo $suma;
}
```

Las funciones y procedimientos se suelen definir en la cabecera del documento entre las etiquetas <head></head> para asegurarnos que están cargadas en memoria antes de usarlas.

Ejecución de procedimientos y funciones.

Para usar las funciones y procedimientos solamente necesitamos llamarlos por su nombre y pasarle los parámetros necesarios para que se puedan ejecutar.

En el caso de funciones podemos asignar a una variable o visualizar su ejecución.

Ejemplo de uso de la función sumar:

```
resultado = sumar(7, 5);
echo resultado;
```

Ejemplo de uso de la función sumar:

```
echo sumar(7, 5);
```

Ejemplo de uso del procedimiento sumar:

```
sumar(7, 5);
```

Paso de Parámetros.

En ocasiones, necesitaremos definir funciones que aceptan parámetros o argumentos. De esta forma conseguimos que la función sea más utilizable dentro del propósito general para el que está definida. Por ejemplo, podemos querer utilizar una función múltiples veces a lo largo del programa, pero algunos de sus valores internos pueden variar con cada llamada. La mejor solución a este problema es definir una función que pueda aceptar argumentos en su llamada para cada valor con el que queramos trabajar.

Parámetros por valor

El parámetro que recibe la función es una copia del valor de la variable pasada como parámetro; de esta forma, las modificaciones que puedan hacerse dentro del cuerpo de la función a la variable parámetro no afectan al valor final de la variable pasada como argumento.

```
nombre_funcion($valor);
```

Parámetros por referencia

En el caso de que queramos que los cambios que se producen en el cuerpo de la función afecten a la variable que se pasó como argumento en la llamada a la función deberemos pasar el parámetro por referencia. Para indicar qué parámetros se pasan por referencia, hay que marcarlos en la definición de la función, anteponiendo el símbolo & al nombre del parámetro.

```
nombre_funcion(&$valor);
```

Parámetros por defecto

Parámetros opcionales en la llamada a las funciones. De este modo, este tipo de parámetros toma un valor predefinido cuando, desde la llamada a la función, no se les ha proporcionado ningún argumento.

Para definir un parámetro por omisión, hay que, además de nombrar el parámetro, escribir el operador de asignación = y, a continuación, el valor que vaya a recibir el parámetro en caso de no especificarse en la llamada. Cuando se usan parámetros por defecto, éstos tienen que situarse los últimos en la declaración, es decir, a la derecha de cualquier parámetro normal.

```
nombre_funcion($valor, $valor2, $valor3 = valor_predeterminado);
```

Devolución de valores

El valor devuelto desde una función puede ser asignado a una variable o utilizado dentro de una expresión. Una función devuelve un único valor: para devolver múltiples valores, deberemos utilizar un array. Para poder hacerlo, se utiliza la palabra reservada **return** acompañada de una expresión. En este instante, la función deja de ejecutarse para devolver el flujo de ejecución al punto del programa donde se llamó a la función.

Si después de **return** hay más líneas de código, dichas líneas no se ejecutarán nunca, por eso, es habitual que aparezca como última instrucción del cuerpo de la función.

```
function calculo($valor1, $valor2){
$resultado = $valor1 * $valor2;
return $resultado;
}
echo calculo(5, 8);
```

Funciones con número variable de parámetros

PHP permite definir funciones en las que el número de parámetros es variable. No se necesita de una sintaxis específica, pero su funcionamiento se basa en el siguiente conjunto de funciones definidas en PHP:

- func_num_args ( ): Devuelve el número de argumentos pasados a la función.
- func_get_args (): Devuelve un array con los argumentos pasados a la función.
- func_get_arg ( ): Devuelve un elemento de la lista de argumentos pasados a la función, Los argumentos comienzan en la posición 0, al igual que los arrays. Si se solicita un argumento de una posición que no existe, devuelve false.

Ejemplo funciones sobre parámetros y paso de argumentos.

```
  function importe_facturas(){
    $totalfacturas=0;
    $num_args = func_num_args();
    echo $num_args."<br>";
    $datos=func_get_args();
    foreach($datos as $indice => $valor){
      echo "Argumento ".$indice." tiene el valor ".$valor."<br>";
    }
    echo "Parámatro tercero ".func_get_arg(2)."<br>";

    foreach($datos as $indice => $valor){
      $totalfacturas+=$valor;
    }

    return $totalfacturas;
  }

  echo "El importe de las facturas es ".importe_facturas(1200, 600,
150)."<br>";

  // funcion paso parámetro por valor.

  function incrementa($num1){
    $num1 += 50;
    return $num1;
  }

  $num1 = 100;
  echo incrementa($num1)."<<<<<<".$num1."<br>";

  // funcion paso parámetro por referencia.

  function incrementa2(&$num1){
    $num1 += 50;
    return $num1;
  }

  $num1 = 100;
  echo incrementa2($num1)."<<<<<<".$num1."<br>";

  // funcion paso parámetro por omisión.

  function dto($importe, $dto = 0.10){
    return $importe * $dto;
  }

  echo "El descuento es ".dto(100,0.20)."<br>";
  echo "El descuento es ".dto(100)."<br>";
?>
```

Funciones predefinidas del lenguaje.

Las funciones predefinidas son aquellas que se integran por defecto dentro del lenguaje.

a) Funciones integradas para cadenas.

Método	Descripción	Sintaxis
`echo()`	Imprime una cadena.	`echo(string)` `echo "hola"; ➜ "hola"`

`explode()`	Rompe un *string* en trozos, utilizando el delimitador y lo guarda en un *array*.	`explode(delimitador,string[,lim ite])`
		`explode(",","Hola mundo");`
		`➜Array([0]=>Hola, [1]=> mundo)`

`implode()`	Convierte un *array* en un *string*.	`implode(delimitador,nombre_array)`
		`$a = array('lunes', 'martes');`
		`$b=implode(" ",$a); ➜ b= " lunes martes"`

`ltrim()`	Elimina los espacios u otros caracteres predefinidos en el lado izquierdo de una cadena.	`ltrim(string[,caracteres predefinidos])` `ltrim(" hola "); ➜ "hola "`
`rtrim()`	Elimina los espacios u otros caracteres predefinidos en el lado derecho de una cadena.	`rtrim(string[,caracteres predefinidos])` `rtrim(" hola "); ➜ " hola"`
`str_repeat()`	Repite un *string* un número específico de veces.	`str_repeat(string, nºrepeticiones)` `str_repeat("#",2);➜ "##"`
`str_replace()`	Reemplaza una parte de un *string* por otra cadena.	`str_replace(cadena_vieja,cadena_ nueva,string,[count])` `str_replace("hola","adios", "hola a todos");➜ "adiós a todos"`
`strcmp()`	Compara dos *string*. Devuelve 0 si son iguales y distinto de 0 si son distintos.	`strcmp(string1,string2)` `strcmp("hola", "hola"); ➜0`
`strlen()`	Devuelve la longitud de un *string*.	`strlen(string)` `strlen("hola");➜4`
`strrev()`	Devuelve un *string* invertido.	`strrev(string)` `strrev("hola");➜"aloh"`

`strstr()`	Busca la primera ocurrencia de un *string* en otro.	`strstr(string1,string2)`
		`strstr("hola a todos", "todos");`
		`➜ "todos"`

`strtolower()`	Convierte a minúsculas un *string*.	`strtolower(string)` `strtolower("HOLA");➜ "hola"`
`strtoupper()`	Convierte a mayúsculas un *string*.	`strtoupper(string)` `strtoupper("hola");➜ "HOLA"`
`trim()`	Elimina los espacios en blanco y otros caracteres predefinidos a un lado y a otro del *string*.	`trim(string[,caracteres predefinidos])` `trim(" hola "); ➜"hola"`

Método	Descripción
strpos($cadena, $cadena_buscada[, $posicion_inicio])	La función PHP strpos() devuelve la posición de la primera coincidencia de la palabra o carácter buscado en una cadena de texto (string). Sensible a Mayúsculas y Minúsculas.
stripos($cadena, $cadena_buscada[, $posicion_inicio])	La función PHP strpos() devuelve la posición de la primera coincidencia de la palabra o carácter buscado en una cadena de texto (string). NO sensible a Mayúsculas y Minúsculas.

Método	Descripción
strrpos($cadena, $cadena_buscada[, $posicion_inicio])	La función PHP strrpos() devuelve la posición de la última coincidencia de la palabra o carácter buscado en una cadena de texto (string). Sensible a Mayúsculas y Minúsculas.
strripos($cadena, $cadena_buscada[, $posicion_inicio])	La función PHP strrpos() devuelve la posición de la última coincidencia de la palabra o carácter buscado en una cadena de texto (string). NO sensible a Mayúsculas y Minúsculas.
substr ($cadena, $posición [, $longitud])	Devuelve una subcadena a partir de la posición indicada empezando desde 0 y si es negativa empieza desde el final. Se extraen los caracteres indicados por longitud. Si longitud es no se indica devuelve hasta el final.
substr_count(cadena, patron)	Devuelve el número de apariciones de una subcadena dentro de una cadena. Diferencia entre mayúsculas y minúsculas.
str_pad (cadena , long, carácter, método)	Rellena una cadena con un carácter de relleno hasta que la cadena resultante tenga la longitud deseada siguiendo un método de relleno entre los siguientes valores. Método (STR_PAD_BOTH, STR_PAD_LEFT, STR_PAD_RIGHT)
ucfirst(cadena)	Convierte en mayúscula el primer carácter de una cadena de caracteres.
ucwords(cadena)	Convierte en mayúscula el primer carácter de cada palabra en una cadena de caracteres.
str_word_count(cadena)	Cuenta el número de palabras de una cadena de caracteres.

b) Funciones integradas para arrays.

Método	Descripción	Sintaxis
`array()`	Crea un *array*.	`array(clave => valor)` `$a=array("a"=>"lunes","b"=>"martes");`

`array_merge()`	Combina uno o más *arrays* en otro *array*.	`array_merge(array1[,array2]…)`
		`$a1=array("a"=>"lunes","b"=>"martes");`
		`$a2=array("c"=>"miercoles");`
		`array_merge($a1,$a2);➔array("a"=>"lunes","b"=>"martes","c"=>"miercoles")`

Método	Descripción	Sintaxis
array_pop()	Elimina el último elemento del *array*.	array_pop(nombre_array) $a=array("a","b","c"); array_pop($a);➔ array("a","b")
array_push()	Añade uno o más elementos a un *array* por el final.	array_push(nombre_array, valor1, valor2…) $a=array("a","b","c"); array_push($a, "d"); ➔array("a","b","c","d")
array_values()	Devuelve el valor de todos los elementos.	array_values(nombre_array) $a1=array("a"=>"lunes","b"=>"martes"); array_values($a1);➔ array([0]=>"lunes", [1]=>"martes")
count()	Devuelve el nº de elementos que hay en un *array*.	count(nombre_array) $a=array("a","b","c"); count($a);➔3
in_array()	Informa de si un elemento está dentro del *array*. Devuelve un valor booleano	in_array(valor,nombre_array[,tipo_búsqueda]) $a=array("a","b","c"); in_array("a",$a); ➔ true

Ejemplo de uso de funciones predefinidas sobre cadenas:

```
<?php
	// explode(delimitador, cadena);
	$texto = "Valencia,Castellón,Alicante";
	$poblacion = explode(",",$texto);
	foreach($poblacion as $valor){
		echo $valor."<br>";
	}

	// implode(delimitador,array)
	$cadena = implode("-",$poblacion);
	echo $cadena."<br>";

	// ltrim(cadena), rtrim(cadena), trim(cadena),
	// strlen(canena)
	$cadena = "     Aula Campus          ";
	echo strlen($cadena)."<br>";
	echo strlen(ltrim($cadena))."<br>";
	echo strlen(rtrim($cadena))."<br>";
	echo strlen(trim($cadena))."<br>";

	// str_repeat(caracter, num_repeticiones)
	echo str_repeat("*",20)."<br>";

	// str_replace(cadena_vieja, cadena_nueva, cadena, veces)
	$cadena = "El coche más vendido es de la marca Mercedes. La
marca Mercedes es una marca Alemana.";
	echo str_replace("Mercedes", "Audi", $cadena, $cantidad)."<br>";
	echo $cantidad."<br>";
```

```
	// strcmp(cadena1, cadena2)
	$cadena1 = "Aula Campus";
	$cadena2 = "Aula campus";
	if ((strcmp($cadena1, $cadena2))==0)
		echo "Cadenas Iguales<br>";
	else
		echo "Cadenas Distintas<br>";

	// strrev(cadena)
	echo strrev($cadena1)."<br>";

	// strstr(cadena, cadenabuscada);
	echo strstr("Aula Campus","Campus")."<br>";

	// strtolower(cadena)
	$cadena = "AULA CAMPUS";
	echo strtolower($cadena)."<br>";

	// strtoupper(cadena)
	$cadena = "aula campus";
	echo strtoupper($cadena)."<br>";

	// strpos($cadena, $cadena_buscada[, $posicion_inicio])
	// substr ( $cadena, $posición [, $longitud ])
	$cadena="Esto es una prueba";
	$pos = strpos($cadena, "es");
	echo substr($cadena,$pos)."<br>";
	echo substr($cadena,$pos,6)."<br>";

?>
```

3.6.- Acceso a la información del cliente web.

Los métodos GET y POST permiten el intercambio de información entre el cliente y el servidor. La principal diferencia entre los métodos GET y POST radica en cómo codifican la información.

a) Método GET.

El método GET envía las variables dentro de la URL de la página.
La URL tiene el formato

```
protocolo://dominio/directorio/fichero
```

por ejemplo

```
http://www.elcorteingles.es/catalogo/zapatos.php
```

El método GET envía la información concatenando al final de la URL el símbolo ? y cada una de las variables con su valor separadas con el símbolo &.

```
http://www.elcorteingles.es/catalogo/zapatos.php?cod=100&color=negro
```

b) Método POST.

En el método POST la información va codificada en el cuerpo de la petición por lo que se envía oculta. Se utilizará este sistema para autenticación de formularios o mantenimiento de información en una base de datos, ya que si utilizamos el método GET cualquier persona podría manipular la URL.

En general el método GET se utiliza para recuperar información y el método POST para enviar información.

Definición de formularios.

Los formularios son métodos para recolectar información rellenada por el usuario. Los formularios permiten autenticar usuarios, insertar o modificar bases de datos o especificar criterios de búsqueda.

a) Formulario mediante método GET.

formulario_get.html

```
<html>
  <head>
    <title>Formulario GET</title>
  </head>
  <body>
    <form action="envio_get.php" method="GET">
      Introduzca su nombre:<input type="text" name="nombre"><br>
      Introduzca su edad:<input type="text" name="edad"><br>
      <input type="submit" value="Enviar">
    </form>
  </body>
</html>
```

Para recuperar la información enviada por el formulario utilizamos la variable especial $_GET que almacena los datos de las variables pasadas con el método GET. Esta variable especial $_GET es un array en el que el índice es el nombre del atributo **name** de cada elemento del formulario y dentro almacena el valor introducido por el usuario en el formulario.

```
$_GET['nombre_elemento_formulario']
```

Ejemplo:

```
$_GET['edad'];
```

Ejemplo Recuperar información enviada por el formulario:

```
<?php
  $nombre = $_GET['nombre'];
  $edad = $_GET['edad'];
  echo "El nombre es ".$nombre." y tiene ".$edad." años";
?>
```

b) Formulario mediante método POST.

formulario_post.html

```
<html>
  <head>
    <title>Formulario POST</title>
  </head>
  <body>
    <form action="envio_post.php" method="POST">
      Introduzca su nombre:<input type="text" name="nombre"><br>
      Introduzca su edad:<input type="text" name="edad"><br>
      <input type="submit" value="Enviar">
    </form>
  </body>
</html>
```

Para recuperar la información enviada por el formulario utilizamos la variable especial $_POST que almacena los datos de las variables pasadas con el método POST. Esta variable especial $_POST es un array en el que el índice es el nombre del atributo **name** de cada elemento del formulario y dentro almacena el valor introducido por el usuario en el formulario.

```
$_POST['nombre_elemento_formulario']
```

Ejemplo:

```
$_POST['edad'];
```

Ejemplo Recuperar información enviada por el formulario:

```
<?php
  $nombre = $_POST['nombre'];
  $edad = $_POST['edad'];
  echo "El nombre es ".$nombre." tiene ".$edad." años";
?>
```

c) Recuperación de datos mediante la variable $_REQUEST.

La variable $_REQUEST permite recuperar la información enviada por ambos métodos GET y POST y contiene la información de las variables $_GET y $_POST.

```
<?php
  $nombre = $_REQUEST['nombre'];
  $edad = $_REQUEST['edad'];
  echo "El nombre es ".$nombre." tiene ".$edad." años";
?>
```

3.7.- Funciones de Fecha y Hora.

a) time ().

Devuelve la marca de tiempo correspondiente al instante en que se ejecuta.

Ejemplo:

```
$fechahora=time();
echo $fechahora."<br>";
```

b) checkdate (mes, dia, año).

Verifica si la fecha que se le pasa como parámetro es una fecha correcta. Si la fecha es correcta la función devuelve verdadero; en caso contrario, la función devuelve falso.

Ejemplo:

```
if (checkdate(2, 29, 2017))
      echo "La fecha existe<br>";
else
      echo "La fecha NO existe<br>";
if (checkdate(2, 29, 2020))
      echo "La fecha existe<br>";
else
      echo "La fecha NO existe<br>";
```

c) mktime (hora, min, seg, mes, dia, año [, is_dst]).

Esta función devuelve la marca de tiempo (el número de segundos transcurridos desde el 1 de enero de 1970 a las 00:00:00), correspondiente a la fecha y hora pasadas a la función como parámetros. Esta función es especialmente útil para realizar cálculos matemáticos con las fechas o validaciones de ellas.

Ejemplo:

```
$fechanac=mktime(20, 45, 18, 5, 12, 1987);
echo $fechanac."<br>";
```

d) date (formato [, timestamp])

Esta función nos permite darle un formato específico a una cadena que contendrá una fecha y una hora. Acepta como parámetros una cadena de formato y un parámetro timestamp.

Los caracteres distintos a los que aparecen en la tabla, que estén dentro de la cadena formato, se imprimirán tal cual aparecen. Para que los caracteres utilizados en la cadena formato se puedan imprimir, es necesario enmascararlos, es decir, deben ir precedidos del carácter /.

valores	descripción
a	"a.m." o "p.m."
A	"A.M." o "P.M."
d	Día del mes con dos dígitos (de "01" a 31")
D	Día de la semana con tres caracteres
F	Nombre del mes
h	Hora en formato "01" a "12"
H	Hora en formato "00" a "23"
g	Hora en formato "1" a "12" (sin cero)
G	Hora en formato "0" a "23" (sin cero)
i	Minutos de "00" a "59"
j	Día del mes en formato "1" a "31"
l	Día de la semana, en texto completo
L	1: si es un año bisiesto 0: si no es un año bisiesto
m	Mes de "01" a "12"
M	Mes con tres caracteres
n	Mes de "1" a "12" (sin cero inicial)
s	Segundos de "00" a "59"
S	Sufijo ordinal en inglés ("th", "nd")
t	Número de días del mes dado, de "28" a "31"
U	Segundos transcurridos desde el valor de inicio (01-01-1970)
w	Día de la semana de "0" (domingo) a "6" (sábado)
Y	Año con cuatro dígitos
y	Año con dos dígitos
z	Día del año de "0" a "365"
Z	Diferencia horaria en segundos de "43200" a "43200"

Ejemplo:

```
$fechanac=mktime(20, 45, 18, 5, 12, 1987);
echo date("d/m/Y H:i:s")."<br>";
echo date("d/m/Y H:i:s", $fechanac)."<br>";
```

e) getdate ([timestamp]).

Esta función devuelve un array asociativo que contiene información sobre la fecha y hora asociadas a la marca de tiempo timestamp, pasada como parámetro. En caso de no pasar ningún parámetro a la función, ésta obtendrá la marca de tiempo del instante en que se ejecuta.

clave	descripción
seconds	Identifica los segundos.
minutes	Identifica los minutos.
hours	Identifica las horas.
mday	Identifica el día del mes.
wday	Identifica, en número, el día de la semana.
mon	Identifica, en número, el mes.
year	Identifica, en número, el año.
yday	Identifica, en número, el día del año.
weekday	Identifica, en texto, el día de la semana.
month	Identifica, en texto, el mes.

Ejemplo:

```
$arrayfecha = getdate($fechanac);
foreach($arrayfecha as $key => $valor){
            echo $key.": ".$valor."<br>";
}

$arrayfechahoy = getdate();
foreach($arrayfechahoy as $key => $valor){
            echo $key.": ".$valor."<br>";
}
```

f) strtotime (formato, timestamp]).

Convierte una cadena con formato de fecha en el número de segundos desde el 1 de enero del 1970 00:00:00.

Ejemplo:

```
echo "Fecha actual:" . strtotime("now") . "<br>";
echo "Fecha pasada '15 May 2015':".strtotime("15 May 2015")."<br>";
echo "Fecha actual + 1 hora:" .strtotime("+1 hours") . "<br>";
echo "Fecha actual + 1 día:" .strtotime("+1 day") . "<br>";
echo "Fecha actual + 1 semana:" .strtotime("+1 week") . "<br>";
echo "Fecha actual + 1 mes:" .strtotime("+1 month") . "<br>";
echo "Fecha actual + 1 año:" .strtotime("+1 year") . "<br>";
echo "Fecha actual + 1 año + 1 mes + 1 semana + 1 día + 1 hora:"
    .strtotime("+1 year +1 month +1 week +1 day +1 hours") . "<br>";
echo "Próximo lunes:" .strtotime("next monday") . "<br>";
echo "El pasado lunes:" .strtotime("last monday") . "<br>";
echo "Próxima semana:" .strtotime("next week") . "<br>";
echo "Próximo mes:" .strtotime("next month") . "<br>";
echo "Próximo año:" .strtotime("next year") . "<br>";
echo "Semana pasada-->" .strtotime("last week") . "<br>";
echo "Mes pasado-->" .strtotime("next month") . "<br>";
echo "Año pasado-->" .strtotime("next year") . "<br>";
```

g) strftime (formato, timestamp]).

Esta función permite dar un formato específico de hora y fecha a la marca de tiempo que se le pasa como parámetro. En caso de no proporcionar este parámetro, se tomará por defecto la marca de tiempo correspondiente al instante en que se ejecuta la función. Los formatos posibles a tener en cuenta se especifican en la siguiente tabla:

valores	descripción
%a	Nombre del día de la semana (abreviado)
%A	Nombre del día de la semana
%b	Nombre del mes (abreviado)
%B	Nombre del mes
%c	Fecha y hora en el idioma actual
%d	Día del mes de 00 a 31
%H	Hora de 00 a 23
%I	Hora de 01 a 12
%j	Día del año de 001 a 366
%m	Mes de 01 a 12
%M	Minutos de 00 a 59
%p	"am" o "pm", según corresponda a la hora dada
%S	Segundos de 00 a 59
%w	Día de la semana de 0 a 6 (0 se corresponde con domingo)
%W	Número de semana en el año (el primer lunes del año es el primer día de la primera semana)
%x	Fecha sin hora
%X	Hora sin fecha
%y	Año de 00 a 99
%Y	Año en cuatro dígitos
%Z	Zona horaria
%%	%

Ejemplo:

```
setlocale("LC_TIME", "spanish");
echo strftime("%A, %d de %B del %Y a las %H:%M:%S")."<br>";
echo strftime("%A, %d de %B del %Y a las %H:%M:%S",$fechanac)
    ."<br>";
```

Ejemplo calculo de Edad a partir de la fecha de nacimiento:

```
$fecha="20/07/1986";
$dia=substr($fecha,0,2);
$mes=substr($fecha,3,2);
$anyo=substr($fecha,6,4);
if (date("md")<$m.$d){
    $edad=date("Y")-$y-1;
}else{
    $edad=date("Y")-$y;
}
```