

## TEMA 2

### PHP FUNCIONES

#### 1. FUNCIONES

- Parámetros por Valor
- Parámetros Por Referencia
- Parámetros Por Defecto

#### 2. Reutilizar Código. Include y Require

#### 3. FUNCIONES DE PHP

#### 4. LIBRERÍAS DE FUNCIONES DE USUARIO

## 1.- FUNCIONES

PHP, además de su vertiente **Orientada a Objetos**, nos permite aplicar esquemas de programación estructurada como puede ser la división en **módulos** (funciones) de un problema.

Ventajas de la **programación modular**:

- Reutilizar código
- Disminuir costes
- Aumentar la fiabilidad y uniformidad de nuestro código

**Sintaxis:**

```
<?php
```

```
function nombre_funcion ($arg_1, $arg_2, ..., $arg_n)
```

```
{
```

```
Sentencias;
```

```
return $valor_devuelto;    // Opcional
```

```
}
```

```
?>
```

**Conceptos:**

- **Cabecera**

La declaración empezará con la palabra clave **function** seguida de *nombre\_función* que será el identificador

- **Parámetros formales :**

Serán los representados por las variables **\$arg\_1, \$arg\_2, ...**

También se puede declarar funciones que aceptan un número de parámetros variable

**Nota:**

Por medio de las funciones de ayuda **func\_num\_args()**, **func\_get\_arg()** y **func\_get\_args()** se puede saber cuántos parámetros se han pasado y qué valores tienen.

Ejemplo de **func\_num\_args()**: <http://php.net/manual/es/function.func-num-args.php>

Ejemplo de **func\_get\_args()**: <http://php.net/manual/es/function.func-get-args.php>

- **Cuerpo de la función:**

Cualquier instrucción válida de PHP puede aparecer en el cuerpo de la función, que estará comprendido entre las claves, incluso otras funciones y definiciones de clases.

- **Valor de retorno:**

Los valores se devolverán usando la palabra reservada opcional **return** esta interrumpirá la ejecución de las instrucciones dentro de la función, finalizando la misma, pudiendo devolverse cualquier tipo de valor incluso matrices y objetos. Su inclusión es **opcional**

**Nota:**

- Si una función devuelve un valor, **return** suele ser la última instrucción
- Una función podría no devolver nada, por lo cual no incluiría la palabra reservada **return**

## INVOCACIÓN DE FUNCIONES

Una vez definida una función, esta puede ser invocada desde un bloque de código php tantas veces como se requiera.

**Sintaxis:**

```
$retorno = nombre_funcion(val_1, val_2, ... , val_n);
```

- Si la función devuelve algún valor este tendría que ser asignado a una variable.
- val\_1, val\_2, ... serán los **parámetros reales** y podrán ser literales, constantes, variables e incluso otros llamadas a funciones, aunque esto último no es una buena práctica.
- Los **nombres de función**, al contrario que las variables, **no discriminan entre mayúsculas** y minúsculas, en la hora de ser invocada.

## EJEMPLOS DE FUNCIONES:

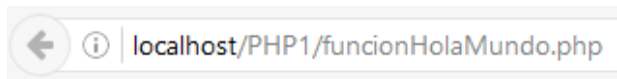
### Sin parámetros:

*funcionHolaMundo.php*

```
<?php
function hola_mundo()
{
    echo "¡Hola Mundo!";
}

/* PROGRAMA PRINCIPAL*/
// invocamos a la función
hola_mundo();
?>
```

### Ejecución:



¡Hola Mundo!

**Con parámetros:**

*funcionMediaAritmeticaDosValores.php*

```
<html>
<head>
<title>Ejemplo Función</title>
</head>
<body>
<?php
    function media_aritmetica($a, $b)
    {
        $media=($a+$b)/2;
        return $media;
    }
$num1=4;
$num2=6;
echo "La media aritmética de $num1 y $num2 es: ";
echo media_aritmetica($num1,$num2),"<br>";
?>
</body>
</html>
```

## Uso de return:

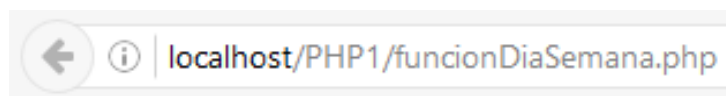
Podemos devolver un resultado con la instrucción **return valor**, lo cual produce la terminación de la función devolviendo un valor.

*funcionDiaSemana.php*

```
<?php
function dia_semana()
{
    $semana = array("lunes", "martes", "miércoles",
        "jueves", "viernes", "sabado", "domingo");
    $dia = $semana[rand(0, 6)];
    return $dia;
}

/* invocación */
$dia_cine = dia_semana();
echo "El próximo $dia_cine voy al cine.";
?>
```

Salida:



El próximo sabado voy al cine.

## Número de parámetros pasados a una función

### *funcionNumArgumentos.php*

```
<?php
function foo()
{
    $numargs = func_num_args();
    echo 'Número de argumentos:'. $numargs.'<br>';
}
/*Programa principal*/
foo(1, 2, 3,4,5,6,7);
?>
```

## Valores de los argumentos pasados a una función

### *funcionNumArgumentosValores.php*

```
function foo2()
{
    $numargs = func_num_args();
    echo 'Número de argumentos:'. $numargs.'<br><br>';
    echo 'Acceso a un determinado argumento:<br>';
    if ($numargs >= 2)
    {
        echo 'El segundo argumento es: '.func_get_arg(1).'<br><br>';
    }
    echo 'Acceso a un todos los argumentos:<br>';
    $arg_list = func_get_args();
    for ($i = 0; $i < $numargs; $i++) {
        echo 'El argumento '.$i.' es: ' . $arg_list[$i] . '<br>';
    }
}

/*Programa Principal*/
foo2(10, 20, 30,40);
?>
```



## **PASO DE PARÁMETROS A UNA FUNCIÓN**

Puede realizarse de dos formas:

- Por valor
- Por Referencia

## PASO DE PARÁMETROS POR VALOR

Es aquel caso en el cual el programa que invoca a la función usa una variable propia en la que facilita información a la función y la función hace una **copia** de este contenido en una **variable** de la función. De este modo, un cambio en esta variable no afecta al valor de la variable usada en el programa que ha invocado a la función.

Por **defecto**, los parámetros de una función **se pasan por valor** (de forma que si cambias el valor del argumento dentro de la función, no se ve modificado fuera de ella).

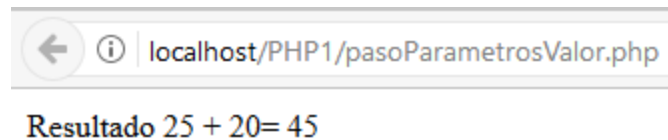
Ejemplo:

*pasoParametrosValor.php*

```
<?php
// Paso de parámetros por valor
function sumaValor($sumando1, $sumando2)
{
    $res_suma=$sumando1+$sumando2;
    return $res_suma; // Opcional
}

$a = 25;
$b = 20;
$res=sumaValor($a, $b);
echo "Resultado de $a + $b = ".$res."<br>";
?>
```

Salida:



Resultado 25 + 20= 45

## PASO DE PARÁMETROS POR REFERENCIA

En este caso el programa que invoca a la función y la propia función usan la misma **variable**, de forma que una modificación de la misma afecta a los dos. En realidad bloque invocador y función comparten la dirección de memoria en la cual está el contenido del parámetro.

Si quieres que un parámetro de una función siempre se pase por referencia, puedes anteponer un **ampersand (&)** al nombre del parámetro en la definición de la función.

Esto puede ser **útil** en casos en que necesitamos **más de un parámetro de salida**. Y no queramos devolverlos mediante una matriz, al quedar más confuso.

Ejemplo:

*pasoParametrosReferenciaSuma.php*

```
function sumaValoresRefer($a,&$bRes)
{
    $resultado=$a+$bRes;
    $bRes=$resultado;
    echo "No hace falta hacer Return, pero lo usamos<br>";
    return $resultado;
}

/* programa principal */
$n1=10;
$n2=30;
$res=sumaValoresRefer($n1,$n2);
echo "El resultado está en n2 y es: <strong>$n2</strong><br>";
echo "El resultado también está en res y es: <strong>$res</strong><br>";
```

## PASO DE PARÁMETROS POR DEFECTO

Los parámetros pueden tomar un valor por si no se los asigna otro en el llamamiento.

- Destacar que cuando se usan **parámetros por defecto**, estos tienen que **estar situados a la derecha** de cualquier parámetro sin valor por defecto.
- Tendremos que tener cuidado al utilizarlos, puesto que pueden dar lugar al hecho que una función haga más de una cosa

**Ejemplo:**

*pasoParametrosPorDefecto.php*

```
<?php
// ejemplo de paso de parámetros por defecto a la función
function porDefecto( $param1="You", $param2='my', $param3 = 'Blue' ){
    return $param1 . ' ' . $param2 . ' ' . $param3;
}
/* invocación de la función */
$argum1='Hello';
$argum2='World';
$argum3='Moon';
echo "Invocamos con tres argumentos<br>";
echo porDefecto ( $argum1, $argum2, $argum3 ); // 'hello world Moon'
echo "<br><br>Invocamos con dos argumentos<br>";
echo porDefecto ( $argum1, $argum2 ); // 'hello world Blue'
echo "<br><br>Ahora Invocamos con un solo argumento<br>";
echo porDefecto ( $argum1); // 'hello my Blue'
echo "<br><br>Ahora sin argumentos<br>";
echo porDefecto (); //You my Blue
?>
```

## FUNCIONES CON DECLARACIONES DE TIPOS (TYPE HINTING)

- Las funciones obligan al hecho que los **parámetros** sean de cierto tipo
- Si el valor dado es de un tipo incorrecto se generará un error
- Tiene que anteponerse el tipo del parámetro antes del nombre
- Se puede hacer que una declaración acepte valores NULL si el valor predeterminado del parámetro se establece a NULL

**NOTA:** porque funciona lo TYPEHINTING, tiene que especificarse en propiedades del proyecto, que trabajamos con PHP7, no versiones anteriores

## TIPOS VÁLIDOS

Tipo	Descripción	Versión de PHP mínima
nombre de clase/interfaz	El parámetro debe ser una <a href="#"><i>instanceof</i></a> del nombre de la clase o interfaz dada.	PHP 5.0.0
<i>self</i>	El parámetro debe ser una <a href="#"><i>instanceof</i></a> de la misma clase donde está definido el método. Esto solamente se puede utilizar en clases y métodos de instancia.	PHP 5.0.0
<a href="#">array</a>	El parámetro debe ser un <a href="#">array</a> .	PHP 5.1.0
<a href="#">callable</a>	El parámetro debe ser un <a href="#">callable</a> válido.	PHP 5.4.0
<a href="#">bool</a>	El parámetro debe ser un valor de tipo <a href="#">boolean</a> .	PHP 7.0.0
<a href="#">float</a>	El parámetro debe ser un número de tipo <a href="#">float</a> .	PHP 7.0.0
<a href="#">int</a>	El parámetro debe ser un valor de tipo <a href="#">integer</a> .	PHP 7.0.0
<a href="#">string</a>	El parámetro debe ser un <a href="#">string</a> .	PHP 7.0.0

Ejemplo:

*funcionEjemploTypeHinting.php*

```
function sumaTypeHinting (int $a, int $b)
{
    return $a+$b;
}
```

Invocación:

```
/*PROGRAMA PRINCIPAL*/
echo "<strong>Programa Principal</strong><br>";

$num1=5;
$num2=4;

$resultado=sumaTypeHinting ( $num1,$num2);

echo "<H4>Resultado: ", $resultado, " </H4>";
```

Salida:

Programa Principal

**Resultado:9**

Invocación:

```
$num1=5.7;
$num2=5.2;
```

Salida:

Ahora forzará a trabajar con enteros, no con reales, por lo cual no tendrá en cuenta la parte decimal.

**Resultado:10**

**FUNCIONES. Con Declaraciones de la MENA del VALOR DEVUELTO (TYPE HINTING Devolución)**

- Las funciones obligan a la salida sea de cierto tipo

**NOTA:** porque funciona, tiene que especificarse en propiedades del proyecto, que trabajamos con PHP7, no versiones anteriores

**TIPOS VÁLIDOS**

Los mismos que en la tabla anterior

Tipo	Descripción	Versión de PHP mínima
nombre de clase/interfaz	El parámetro debe ser una <a href="#"><i>instanceof</i></a> del nombre de la clase o interfaz dada.	PHP 5.0.0
<i>self</i>	El parámetro debe ser una <a href="#"><i>instanceof</i></a> de la misma clase donde está definido el método. Esto solamente se puede utilizar en clases y métodos de instancia.	PHP 5.0.0
<a href="#">array</a>	El parámetro debe ser un <a href="#">array</a> .	PHP 5.1.0
<a href="#">callable</a>	El parámetro debe ser un <a href="#">callable</a> válido.	PHP 5.4.0
<a href="#">bool</a>	El parámetro debe ser un valor de tipo <a href="#">boolean</a> .	PHP 7.0.0
<a href="#">float</a>	El parámetro debe ser un número de tipo <a href="#">float</a> .	PHP 7.0.0
<a href="#">int</a>	El parámetro debe ser un valor de tipo <a href="#">integer</a> .	PHP 7.0.0
<a href="#">string</a>	El parámetro debe ser un <a href="#">string</a> .	PHP 7.0.0

Ejemplo:

*funcionEjemploTypeHintingDevolucion.php*

```
function sumaTypeHintingDevolucion (int $a, int $b):int
{
    return $a+$b;
}
```

Invocación:

```
/*PROGRAMA PRINCIPAL*/
echo "Programa Principal<br>";

$num1='5';
$num2=4;

$resultado= sumaTypeHintingDevolucion( $num1,$num2);

echo "<H4>Resultado:$resultado</H4>";
```

Salida:

Programa Principal

**Resultado:9**



## 2.- REUTILIZAR CÓDIGO. INCLUDE Y REQUIRE

Funciones `include()` y `require()` → <http://php.net/manual/es/function.include.php>

- La sentencia `include` sirve para incluir y evaluar el archivo especificado entre comillas.

**Nota:** existe `require` es idéntico a `include` excepto que **`require`**, en caso de fallo **detiene** lo script, mientras que `include` solo emitirá una advertencia (**`E_WARNING`**) y permite que continúo la carga de la página php.

Ejemplo:

### Ejemplo #1 Ejemplo básico de `include`

```
vars.php
<?php

$color = 'verde';
$fruta = 'manzana';

?>

test.php
<?php

echo "Una $fruta $color"; // Una

include 'vars.php';

echo "Una $fruta $color"; // Una manzana verde

?>
```

- Inicialmente las variables `$fruta` y `$color` no tienen contenido, por eso se escribe solo la cadena: Una
- Después, al hacer `include 'vars.php'`, estas variables ya toman un valor y por eso, se escribe la cadena Una manzana verde

### 3.- FUNCIONES DE PHP

Existen muchas bibliotecas (librerías) de funciones PHP, como por ejemplo :

- Funciones de manipulación de cadenas.
- Funciones de fecha y hora
- Funciones de arrays.
- Funciones de ficheros.
- Funciones matemáticas
- Funciones de bases de datos
- Funciones de red.

Algunas bibliotecas requieren la instalación de componentes adicionales

## FUNCIONES DE MANIPULACIÓN DE CADENAS.

- Una cadena puede ser accedida como un array de caracteres
- Podemos hacer uso de cualquier tipo de bucle por recorrer la cadena

Ejemplo:

```
$cad="Cadena de prueba";  
  
$longCad=strlen($cad);  
echo "Visualizando la cadena carácter a carácter...<br><br>";  
  
for ($i = 0; $i < $longCad; $i++)  
{  
    echo $cad[$i];  
}
```

Salida:



Visualizando la cadena carácter a carácter...

Cadena de prueba

## **FUNCIONES VISUALIZACIÓN DE CADENAS.**

PHP dispone de funciones que muestran el contenido de variables con y sin formato específico:

<b>echo()</b>	Es la más utilizada, muestra cadenas de caracteres en la salida estándar. No acepta formato de salida.  Sintaxis: <code>echo string arg1 [, string arg2];</code> Sintaxis: <code>echo (string arg1 [, string arg2]);</code>
<b>print()</b>	Es la más sencilla, y muestra el contenido de una cadena de caracteres en la salida estándar. No acepta formato de salida.  Sintaxis: <code>printf(string cadena);</code>
<b>printf()</b>	Realiza la misma acción que la función anterior, con la diferencia que ésta si acepta argumentos de formato de salida.  Sintaxis: <code>printf(string formato [, cualquier valor , ...]);</code>

## FUNCIONES DE ALTERACIÓN DEL CONTENIDO DE CADENAS

En este apartado veremos algunas de las funciones más utilizadas para manipular el contenido de las variables string.

<b>chop()</b>	Devuelve una cadena de caracteres a la que se han eliminado los caracteres en blanco y el de nueva línea que aparece al final de la cadena.  Sintaxis: <code>string chop(string cadena);</code>
<b>ltrim()</b> <b>rtrim()</b> <b>trim()</b>	Eliminan los caracteres en blanco que aparecen a la izquierda, a la derecha y a la izquierda y la derecha, respectivamente.  Sintaxis: <code>string ltrim(string cadena);</code>
<b>str_pad()</b>	Ajusta el tamaño de una cadena de caracteres a una longitud determinada, permitiendo especificar el carácter de relleno. Realiza la misma acción que la función anterior, con la diferencia que ésta si acepta argumentos de formato de salida.  Sintaxis: <code>string str_pad(string cadena, int longitud [, string relleno [, int lugar]]);</code>
<b>strtolower()</b>	Convierte todos los caracteres alfabéticos a minúsculas.
<b>strtoupper()</b>	Convierte todos los caracteres alfabéticos a mayúsculas.
<b>ucfirst()</b>	Convierte todos los caracteres alfabéticos a minúsculas, excepto el primero de la cadena.
<b>ucwords()</b>	Convierte todos los caracteres alfabéticos a minúsculas, excepto el primer carácter de cada palabra.  Sintaxis: <code>string strtoupper(string cadena);</code>

<b>str_replace ()</b>	Recibe una cadena de caracteres y devuelve otra en la que se han sustituido todas las apariciones de una subcadena2 por otra subcadena1.  Sintaxis: string str_replace(string subcadena1, string subcadena2, string cadena);
<b>strstr()</b>	Permite establecer una correspondencia entre los diferentes caracteres a traducir y sus sustitutos.  Sintaxis: string strstr(string cadena, string originales, string traducidos);  ejemplo: \$nom = strstr(\$nom, "áéíóúÁÉÍÓÚ", "aeiouAEIOU");

**FUNCIONES DE ACCESO AL CONTENIDO DE CADENAS**

<b>strlen()</b>	Devuelve un número entero que indica cuántos caracteres tiene la cadena que recibe como parámetro.  Sintaxis: <code>int strlen(string cadena);</code>
<b>strchr()</b>	Devuelve la subcadena que comienza en la primera aparición del carácter indicado.  Sintaxis: <code>string strchr(string cadena, char caracter);</code>
<b>strstr()</b>	Permite localizar una subcadena dentro de una cadena original. Es sensible a las mayúsculas y minúsculas. Existe <code>stristr()</code> que tiene la misma funcionalidad pero no es sensible.  Sintaxis: <code>string strstr(string cadena, string subcadena);</code>
<b>strpos()</b> <b>strrpos()</b>	Indica la posición de la primera (última) ocurrencia de una cadena en otra.  Sintaxis: <code>int strpos(string cadena, string subcadena [, int pos_inicial]);</code>
<b>substr()</b>	Devuelve la porción de cadena original que empieza en una determinada posición y que tiene una determinada longitud.  Sintaxis: <code>string substr(string subcadena1, int comienzo, int longitud);</code>

<b>strcmp()</b>	Permite comparar dos cadenas, siendo sensible a mayúsculas y minúsculas.
<b>strcasecmp()</b>	Permite comparar dos cadenas, NO siendo sensible a mayúsculas y minúsculas.
<b>strncmp()</b>	Permite comparar los n primeros caracteres de dos cadenas.  Sintaxis: <code>int strcmp(string cadena1, string cadena2);</code>



## OTRAS FUNCIONES DE CADENAS

- **explode():** divide una cadena en elementos, en base a cierto separador, generando una matriz.
- **chr(n):** presa como argumento un entero (el código ASCII) y devuelve el carácter que corresponde.
- **ord(carácter):** presa como argumento un carácter y devuelve el código ASCII.
- **strip-tags():** elimina marcas de código HTML o PHP
- **nl2br():** tomada una cadena de texto con caracteres de nueva línea y la convierte de tal forma que quedo igual en HTML, insertando para lo cual marcas < br /> en cada salto.
- **addslashes():** añade las barras invertidas para interpretar correctamente los campos delimitados por “ o ‘ al operar con bases de datos .
- **stripslashes():** elimina las barras invertidas
- **htmlspecialchars():** se ocupa de los caracteres <>,&, etc.
- **htmlspecialchars():** se ocupa de la sustitución de caracteres nacionales que, por la codificación de caracteres de la página, tienen que aparecer como entidades .
- **HTML\_entity\_decode():** se pueden convertir las entidades HTML de una cadena en sus respectivas representaciones como caracteres .

## FUNCIONES DE FECHA Y HORA

Cuando se solicita la hora a un ordenador, realmente no se obtiene la hora, los minutos y los segundos de un día de un mes y de un año determinado, se obtienen los segundos transcurridos desde el inicio de la era de UNIX (día 1 de enero de 1970 a la 00:00:00 GMT), a este valor se le denomina "timestamp"

**time():** Devuelve el número de segundos transcurridos desde el inicio de la era de UNIX (1 de enero de 1970 a las 00:00:00 GMT). Este entero que representa la marca de tiempo (*timestamp*) correspondiendo al instante en que se ejecuta esta función.

Ejemplo: *time.php*

```
$numSegs= time();  
$numHoras=$numSegs/3600;  
$numDias=$numHoras/24;  
$numAnyos=$numDias/365;  
echo 'Nº segundos desde inicio hora UNIX: '.$numSegs."<br>";  
echo 'Nº años aproximados desde inicio hora UNIX: '.$numAnyos."<br>";
```

Este número entero es poco útil, en la hora de mostrar el tiempo, la forma normal de trabajar continúa siendo con días, meses, años, horas, minutos y segundos. Y para lo cual PHP dispone de una función que se encarga de obtener la fecha del sistema presentada de una forma determinada, es *getdate()*.

- **getdate():** Permite representar la fecha en formato días, meses, años, horas, minutos y segundos. Y para lo cual Esta función devuelve una matriz asociativa con la información de la fecha y hora del sistema.

Los elementos de esta matriz son:

<b>seconds</b>	Número de segundos de la hora actual.
<b>minutes</b>	Número de minutos de la hora actual.
<b>hours</b>	Número de horas de la hora actual.
<b>mday</b>	Día correspondiente del mes.
<b>wday</b>	Día de la semana en valor numérico, el 0 es el domingo, ...
<b>mon</b>	Valor numérico del mes.
<b>year</b>	Valor numérico del año.
<b>weekday</b>	Cadena que indica el día de la semana (ingles).
<b>month</b>	Cadena que indica el mes del año (ingles).
<b>minutes</b>	Número de minutos de la hora actual.

Ejemplo: *getDate.php*

```
$hoy = getdate();
$dia=$hoy['mday'];
$mes=$hoy['mon'];
$anyo=$hoy['year'];
$hora=$hoy['mon'];
$min=$hoy['minutes'];
$secs=$hoy['seconds'];
$fecha=$dia.'/'.$mes.'/'.$anyo;
echo "Fecha hoy: $fecha".<br>";
$horaCompleta=$hora.':'.$min.':'.$secs;
echo "Hora es: $horaCompleta";
```

- **date():** Devuelve una cadena de caracteres que se corresponden con la fecha a la cual se ha aplicado un determinado formato.

Para la definición del formato disponemos de las siguientes opciones, no se han mostrado todas las opciones de que dispone, solo las más utilizadas:

<b>d</b>	Día del mes con dos dígitos.
<b>D</b>	Día de la semana con tres letras (ejemplo Mon).
<b>F</b>	Nombre del mes completo.
<b>h</b>	Hora con dos dígitos, con formato 01 a 12.
<b>H</b>	Hora con dos dígitos, con formato 00 a 23.
<b>i</b>	Minutos con dos dígitos.
<b>l</b>	El minúscula, aparece el nombre completo del día de la semana (ingles).
<b>m</b>	Mes con dos dígitos.
<b>M</b>	Mes con tres letras (ejemplo Mar).
<b>s</b>	Segundos con dos dígitos
<b>Y</b>	Año con cuatro dígitos
<b>y</b>	Año con dos dígitos

Ejemplo: *date.php*

```
echo 'Ahora: ' . date('d/m/Y') . "<br>";

$semanaSiguiente = time() + (7 * 24 * 60 * 60);
// 7 días; 24 horas; 60 minutos; 60 segundos
echo 'Semana Siguiente: ' . date('d/m/Y', $semanaSiguiente) . "<br>";
```

- **strftime()**: Muestra el formato a una fecha.

Los nombres del mes y del día de la semana, siguen los valores establecidos por la función **setlocale()**.

El formato quedará definido por los siguientes valores:

<b>%a</b>	Nombre del día de la semana abreviado (según idioma).
<b>%A</b>	Nombre del día de la semana completo (según idioma).
<b>%b</b>	Nombre del mes abreviado.
<b>%B</b>	Nombre del mes completo.
<b>%c</b>	Representación de fecha y hora (según idioma).
<b>%d</b>	Día del mes en formato 01 a 31.
<b>%H</b>	Hora como un número de 00 a 23.
<b>%I</b>	Hora como un número de 01 a 12.
<b>%m</b>	Mes como un número de 01 a 12.
<b>%M</b>	Minuto en número.
<b>%S</b>	Segundos en número.
<b>%x</b>	Representación por defecto de la fecha sin la hora.
<b>%X</b>	Representación por defecto de la hora sin la fecha.
<b>%y</b>	Año en número de 00 a 99.
<b>%Y</b>	Año con cuatro dígitos.

Ejemplo:

<http://php.net/manual/es/function.strftime.php>

- **checkdate()**: comprueba si una determinada combinación de día , mes y año representa una fecha válida.

La función tiene que recibir tres parámetros numéricos correspondientes al mes, en el día y en el año y devuelve true si los parámetros resultan una fecha válida y false en cualquier otro caso.

La sintaxis es la siguiente:

```
int checkdate(int mes, int día, int anio);
```

Ejemplo:

Ver documentación PHP

## FUNCIONES DE ARRAYS.

### Funciones de ordenación de arrays:

vectoresOrdenar.php

- **sort()** : Ordena y reindexa un array

```
//Ordenar desde el menor al mayor
$alumnos = array("Pepe", "Juan", "Marcelo", "Alberto", "Gerardo");
sort($alumnos);
foreach ($alumnos as $key => $val)
{
    echo "alumnos[" . $key . "] = " . $val . "<br>";
}
```

Salida:

### Ordenar Vectores

```
alumnos[0] = Alberto
alumnos[1] = Gerardo
alumnos[2] = Juan
alumnos[3] = Marcelo
alumnos[4] = Pepe
```

- **rsort()**: Ordena y reindexa un array de manera decreciente (r=reverse)
- **ksort()**: En arrays asociativos ordena por el índice o **clave**.
- **krsort()**: Igual que Ksort, pero de manera inversa por clave (r=reverse).
- **asort()**: En arrays asociativos ordena por valor.
- **arsort()**: Igual que asort, pero de manera inversa por valor (r=reverse).
- **array\_count\_values()**: Calcula la frecuencia de cada uno de los elementos de un array
- **array\_search(elemento, array)**: Busca elemento en un array. Devuelve posición o FALSE

Ejemplo:

```
$capitales = array("Portugal" => "Lisboa", "Italia" => "Roma", "Francia" => "París");  
$elem_buscado="Roma";  
$posicion=array_search($elem_buscado, $capitales);  
echo "$elem_buscado está en la posición $posicion<br>";
```

Salida:

Roma está en la posición Italia



- **count()**: Cuenta los elementos de un array

```
$num_elementos=count($capitales);  
echo "El número de elementos del array es $num_elementos<br>;"
```

- **unset()**: Permite eliminar o desasignar elementos de un array

```
$a = array(1 => 'uno', 2 => 'dos', 3 => 'tres');  
echo "<br>Mostramos contenido inicial del vector<br>";  
print_r($a);  
unset($a[2]);  
  
echo "<br><br>Lo volvemos a mostrar tras hacer unset()<br>";  
$b = array_values($a);  
print_r($b);
```

Salida:

```
Mostramos contenido inicial del vector  
Array ( [1] => uno [2] => dos [3] => tres )
```

```
Lo volvemos a mostrar tras hacer unset()  
Array ( [0] => uno [1] => tres )
```

## FUNCIONES PARA RECORRER UNA MATRIZ ASOCIATIVA:

Ya se ha visto:

Función	resultado
<b>each()</b>	Recupera el par formado por la clave y el valor del elemento actual y además avanza una posición el puntero de la matriz.  Sintaxis: <code>array each(array matriz);</code>  Devuelve false cuando no quedan elementos por tratar
<b>List()</b>	Asigna los valores del elemento actual de una matriz a las variables que se hayan pasado com parametro.  sintaxis: <code>void list(\$var1, \$var2, ...);</code>
<b>reset()</b>	Hace que el puntero interno apunte a la primera posición de la matriz.
<b>end()</b>	Hace que el puntero interno apunte a la última posición de la matriz.
<b>next()</b> <b>prev()</b>	Permite ir al siguiente / anterior elemento. En caso de estar al principio o al final devuelve el valor falso.
<b>current()</b>	Devuelve el contenido del elemento actual. Esta función no desplaza el puntero interno de posición y devuelve el valor false cuando se encuentre después del último elemento o si la matriz no tiene elementos.

- **reset(array), end(), next (array), prev (array):** Permiten moverse a manera de puntero a principio, siguiente y anterior, de un array
- **each (array):** devuelve el valor del elemento actual, en este caso, el valor del elemento actual y su clave, y desplaza el puntero al siguiente, cuando llega al final devuelve FALSO, y acaba el bucle **while()**.
- **list(vable1, vble2, vble3,...):** Asigna valores a unas lista de variables . Devuelve un vector-. Es más bien un operador de asignación .

**OTRAS FUNCIONES:**

- **print\_r (vble):** Visualiza la variable en formato legible.
  - Si variable string, integer o float: visualiza el propio valor.
  - Si variable es array: Visualiza en formato [índice] => valor

Ejemplo:

```
$array_simple = array(10, 20, 30, 40);
$array_asociativo = array("Portugal" => "Lisboa", "Italia" => "Roma", "Francia" => "París");

echo "print_r para un array SIMPLE<br>";
print_r($array_simple);
echo "<br>print_r para un array ASOCIATIVO<br>";
print_r($array_asociativo);
```

Salida:

```
print_r para un array SIMPLE
Array ( [0] => 10 [1] => 20 [2] => 30 [3] => 40 )
print_r para un array ASOCIATIVO
Array ( [Portugal] => Lisboa [Italia] => Roma [Francia] => París )
```

- **In\_array()**: para comprobar si existe o no un valor en un array

Ejemplo:

```
$valor=30;
$res=in_array($valor, $array_simple);
if($res==true)
{
    echo "<br>";
    echo "Encontrado $valor en el vector <br>";
    print_r($array_simple);
}
else
    echo "No se ha encontrado $valor en el vector <br> print_r($array_simple)";
?>
```

Salida:

Encontrado 30 en el vector

Array ( [0] => 10 [1] => 20 [2] => 30 [3] => 40 )

## 4.- LIBRERÍAS DE USUARIO

El uso de librerías (*bibliotecas*) nos permite agrupar varias funciones y variables en un mismo fichero, de forma que después podemos incluir esta librería en diferentes páginas y disponer de esas funciones fácilmente.

Las librerías no son más que archivos *php* que se pueden incluir en cualquier otro archivo.

Permiten que todo lo definido en estos archivos pueda ser accedido desde los programas en las cuales se han incluido.

El contenido de un archivo de librería puede ser:

- Funciones
- Variables
- Constantes
- Código

La instrucción para incluir una librería en nuestra página:

***include("nombre de librería ")***

Ejemplo:

Creamos un archivo con las funciones:

*libreriaVisualizaDatosPersonales.php*

```
function visualizaNombre($nombre)
{
    echo 'Mi nombre es: '.$nombre.'<br>';
}

function visualizaDireccion($direc, $poblac, $cp)
{
    echo 'Vivo en: '.$direc.'<br>';
    echo $poblac.' ( '.$cp.' ) '<br><br>';
}

function calculaEdad($anyoNac)
{
    $horaActual=getdate();
    // getdate devuelve array asociativo
    $actualYear=$horaActual["year"];
    $edad= $actualYear-$anyoNac ;
    return $edad;
}
```

Creemos un archivo que contenga el **PROGRAMA PRINCIPAL** que invoca a la las funciones:

*libreriaVisualizaDatosPersonalesPpal.php*

```
include 'libreriaVisualizaDatosPersonales.php';

// asignamos valores
$nom='Pepe Gómez';
$dir='C\ Albufera, 20';
$pob='Cheste';
$cpostal='46300';
$anyoNac=1997;

//invocamos a las funciones
visualizaNombre($nom);
visualizaDireccion($dir,$pob,$cpostal);
$miEdad=calculaEdad($anyoNac);

echo 'Mi edad es: '.$miEdad;
```

*Se verá en ejercicios*