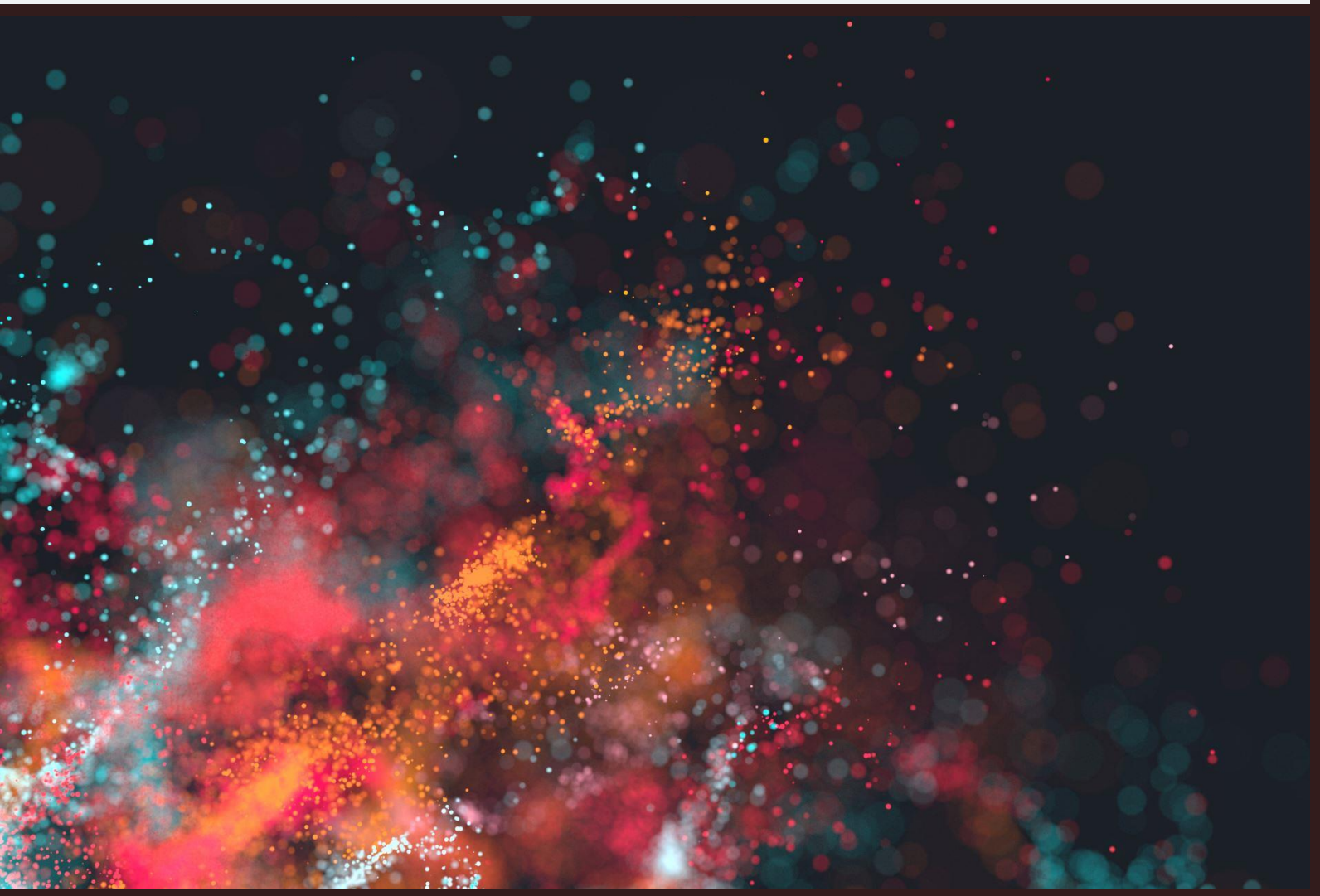


TEMA 2.4

Funciones

Funciones en PHP

https://www.php.net/manual/es/language.functions.php

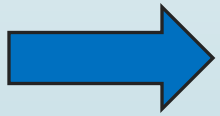
- PHP contiene multitud de funciones predefinidas, ya hemos empezado a utilizar algunas de las más habituales en temas anteriores.
- En la página de php.net podemos encontrar un manual con sus más de 1000 funciones predefinidas, así como información sobre cómo crear funciones propias (definidas por el usuario)
- El manual en español está disponible en la URL indicada arriba.

Definiendo funciones propias

- Para definir una función propia, utilizaremos la palabra **function**, luego pondremos el nombre de la función, y finalmente la lista de parámetros, si existen, entre paréntesis. Si retorna algún valor, lo hará mediante la sentencia **return**.

```php
function nombre_funcion ( $parametro1, $parametro2 ) {
    ... // Aquí el código de la función, podrá utilizar los parámetros
    return $resultado;
}
```

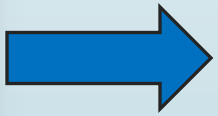
```php
<h2>Mi función saluda</h2>
<?php
  1 reference
  function saludos ($antes, $despues) {
    echo $antes . " saluditos " . $despues;
  }

  saludos("MI SALUDO: "," ...adiós");
?>
```



Funciones: return

- A las funciones que retornan valores no se les define tipo de datos, ya que PHP no es un lenguaje tipado.
- Si mi función va a retornar algún valor, el tipo de datos del **return** coincidirá con el del valor de la función.
- En el caso del siguiente ejemplo, la función retorna un número entero. Posteriormente, el echo lo convertirá a string para concatenarlo al resto del texto.

```php
<?php
    1 reference
    function suma ($sumando1, $sumando2) {
        return $sumando1 + $sumando2;
    }
    echo "La suma de 2 + 3 es: " . suma(2,3);
?>
```



La suma de 2 + 3 es: 5

Ejercicio 1:

Día de cine

Voy al cine el próximo domingo

1. Haz una función en PHP que retorne un día de la semana al azar entre lunes y domingo, en formato string.
2. Tu programa PHP realizará un echo indicando que vas al cine el día al azar indicado por la función.
3. Refresca varias veces el navegador y observa cómo va cambiando el día de la semana.

Nota: El resultado tendrá el aspecto de la izquierda

Pista: Puedes utilizar un array con los días de la semana

Funciones: Parámetros

- Los parámetros de una función van entre paréntesis. Al no ser PHP tipado, no se declara tipo de datos.
- Hay dos formas de pasar parámetros:
  - **Por valor:** El parámetro pasado por valor es una copia de la variable original utilizada en la llamada. La función utilizará el valor del parámetro, y podrá modificarlo, pero la variable original no cambia, puesto que no es manipulada.

```
function miFuncion ($parametroPorValor) {... } // El parámetro es una copia de la variable
```

  - **Por referencia:** La función accede directamente la variable original que se utiliza en la llamada. Por tanto, si modifica la variable, esta queda cambiada una vez retorna la función.

```
function miFuncion(&$parametroPorReferencia) {… } // El parámetro es una referencia a la variable
```

- Una función puede tener parámetros de los dos tipos. Si en su definición hay un "**&**" delante de algún parámetro, éste se está pasando por referencia.
- Antes de utilizar las funciones predefinidas, conviene que comprobemos en el manual de qué forma estamos pasando los parámetros. Por ejemplo, en la función **sort**, que ordena arrays:

```
sort(array &$array, int $flags = SORT_REGULAR): bool
```

Documentación en php.net

La función **sort** de PHP ordena arrays. El parámetro $array lleva **&**, va a ser modificado dentro. Después de la llamada, el array que pasemos por parámetro habrá cambiado y estará ordenado. El parámetro $flags es pasado por valor.

Parámetros por valor y por referencia: Ejemplo

```php
<?php
  1 reference
  function noIncremento ($numero) {
    $numero++;
  }
  1 reference
  function incremento(&$numero) {
    $numero++;
  }

  $valor1 = 10;
  $valor2 = 20;
  echo "Valor1 es $valor1 y valor2 es $valor2<br/>";
  noIncremento($valor1);
  incremento($valor2);
  echo "Después de las funciones, valor1 es $valor1 y valor2 es $valor2";
?>
```

$valor1 es pasado como parámetro *por valor* a la función **noIncremento**. El parámetro $numero es una copia de $valor1, otra variable diferente. Por tanto $valor1 no queda modificada.
$valor2 es pasado como parámetro *por referencia* a la función **incremento**. $numero es una referencia (un puntero) a la variable $valor2, por tanto se está manipulando $valor2 a través de su referencia $numero.

Valor1 es 10 y valor2 es 20
Después de las funciones, valor1 es 10 y valor2 es 21

Ejercicio 2:

Duplicando

```
Antes de la llamada, num1=10 y num2=20
Dentro de dupValor: 20
Dentro de dupReferencia: 40
Después de la llamada a dupValor num1=10
Después de la llamada a dupReferencia num2=40
```

1. Vas a hacer dos funciones, dupValor y dupReferencia que:
    1. Reciban un parámetro, dupValor por valor, y dupReferencia por referencia.
    2. Dupliquen el valor de ese parámetro
    3. Hagan un echo del resultado
2. Llama a la función dupValor, con el parámetro $num1, cuyo valor es 10.
3. Llama a la función dupReferencia, con el parámetro $num2, cuyo valor es 20.
4. Después de las llamadas, haz un echo de $num1 y $num2

Nota: El resultado tendrá el aspecto de la izquierda

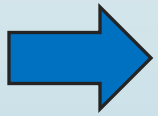
Parámetros opcionales: Valores por defecto

- En PHP podemos definir valores por defecto para un parámetro, igualándolo a un valor en la definición de una función:

```
function($parametro=3)
```

- Los parámetros con un valor por defecto son opcionales. Si los omitimos en la llamada, PHP utilizará el valor por defecto. Si no los omitimos, utilizará el valor indicado.
- Los parámetros con valor por defecto (opcionales) irán siempre al final.

```php
<?php
  4 references
  function hola ($nombre, $idioma="es") {
    if ($idioma == "es") return "Hola $nombre";
    else if ($idioma == "fr") return "Bonjour $nombre";
    return "Hello $nombre";
  }
  echo hola("Marta") . "<br/>";
  echo hola("Pascal","fr") . "<br/>";
  echo hola("Ang","??") . "<br/>";
?>
```



Hola Marta
Bonjour Pascal
Hello Ang

Variables dentro de una función: Locales y globales

- Cuando utilizamos una variable dentro de una función, el ámbito es interno a la función, es decir, el ámbito es **local**.
- Esto implica que, aunque exista fuera de la función otra variable con igual nombre, ambas son diferentes.
- Esto no se aplica a variables superglobales, cuyo ámbito es global y son la misma en todas las partes del código. Por ejemplo, $_GET.
- Existe la posibilidad de definir variables globales con la palabra **global**.
- **NO** se recomienda definir variables globales, pero en caso de que sea imprescindible, deberían utilizarse nombres largos, en mayúsculas, para que no haya confusión posible con otras de ámbito local.
- La mejor forma de compartir información en ámbitos interno y externo de una función es utilizar parámetros por valor o por referencia, o como valor de retorno de la función.

Funciones con parámetros variables

- Al igual que no se declara el tipo de datos que son los parámetros, en PHP es posible definir una función sin indicar cuantos (ni cuales) parámetros va a utilizar.
- Esto no significa que no tendrá parámetros. Puede tenerlos, tantos como se quiera. No hay limitación.
- Pongamos un **ejemplo**: Quiero hacer una función que sume varios números entre sí. No quiero limitación en el número de parámetros, es decir, puedo sumar 2 números o 200 números entre sí. A priori no lo se.
- Para hacer eso, defino la función sin parámetros, con los paréntesis vacíos

```
function sumaNumeros () {…} // No especifico parámetros al definir la función

sumaNumeros(1,34,66,2,88,7) // Llamo a la función con los números que quiero
```

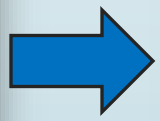
- Ahora bien, es necesario recuperar los valores de los parámetros dentro de la función, sino no podrá sumarlos. Para ello, se utilizan funciones de PHP que recuperan los parámetros.

Func_get_args, func_num_args y func_get_arg

Para obtener información sobre los parámetros que se le pasan a una función que no los tenga definidos (variables), PHP ofrece tres funciones predefinidas:

- **func_get_args()**: Obtiene un array, cada parámetro será un elemento.
- **func_num_args():** Obtiene el número de parámetros recibido por la función
- **func_get_arg($indice)**: Obtiene un parámetro concreto por índice del array de argumentos.

```php
function sumaNumeros () {
  // Si no hay parámetros retorna 0
  if (func_num_args()==0) return 0;
  $suma = 0;
  for ($i=0; $i<func_num_args(); $i++) {
    echo "Parámetro número $i = " . func_get_arg($i) . "<br/>";
    $suma = $suma + func_get_arg($i);
  }
  return $suma;
}
$suma = sumaNumeros(1,34,66,2,88,7);
echo "La suma de todos los números es : " . $suma . "<br/>";
```

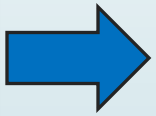


Parámetro número 0 = 1
Parámetro número 1 = 34
Parámetro número 2 = 66
Parámetro número 3 = 2
Parámetro número 4 = 88
Parámetro número 5 = 7
La suma de todos los números es : 198

Operador variádico: . . .

- Otra opción para resolver el número variable de parámetros es el **operador variádico**.
- Podemos utilizarlo en vez de func_get_arg , de forma más intuitiva, porque se visualiza en la definición de la función que puede haber un número indefinido de parámetros
- Lo que hace es "disfrazar" de array un parámetro, que podrá recoger varios parámetros de la llamada. Esta característica es nueva de PHP 5.6, no lo hay en versiones anteriores.

```php
<?php
function f($requerido, $opcional = null, ...$parámetros) {
    // $parámetros es un array que contiene los argumentos restantes.
    printf('$requerido: %d; $opcional: %d; número de parámetros: %d'."\n",
           $requerido, $opcional, count($parámetros));
}

f(1);
f(1, 2);
f(1, 2, 3);
f(1, 2, 3, 4);
f(1, 2, 3, 4, 5);
?>
```



```
$requerido: 1; $opcional: 0; número de parámetros: 0
$requerido: 1; $opcional: 2; número de parámetros: 0
$requerido: 1; $opcional: 2; número de parámetros: 1
$requerido: 1; $opcional: 2; número de parámetros: 2
$requerido: 1; $opcional: 2; número de parámetros: 3
```

Nota: En este ejemplo utilizamos, en vez de echo, la función printf, cuyo interfaz es similar a la del lenguaje C, también disponible en Java: printf (formato, argumentos)
Si quisiéramos la salida en navegador en vez de en consola, tendríamos que sustituir "/n" por "
".

Ejercicio 3:

Operador variádico

1. Crea una función suma que admita tres parámetros:
   1. **$desde**, parámetro obligatorio, indica un número inicial a sumar.
   2. **$hasta**, parámetro obligatorio.
      a. Si se indica $hasta, y el resultado de la suma supera ese $hasta, se retorna 0.
      b. Si $hasta es cero, se retorna la suma.
   3. **$sumandos**, parámetro variádico, este array contendrá el resto de los parámetros, que serán los sumandos.
2. Muestra en la **salida**: Los parámetros proporcionados, y el resultado de la suma (ver figura abajo), para los **$sumandos=[23, 65,10, 55,7]** en dos casos:
   1. **$desde=0 y $hasta=0**
   2. **$desde=60 y $hasta=200**

Salida en navegador

- Sumando: 23 -- Sumando: 65 -- Sumando: 10 -- Sumando: 55 -- Sumando: 7 - --> La suma total es 160
- Sumando: 23 -- Sumando: 65 -- Sumando: 10 -- Sumando: 55 -- Sumando: 7 - --> La suma supera el valor máximo

La función suma(0,0,23,65,10,55,7) retorna el valor: 160
La función suma(60,200,23,65,10,55,7) retorna el valor: 0

Parámetros con nombre

- Desde PHP 8, puede hacerse la llamada a una función indicando el nombre del parámetro. Esto permite cambiarlos de orden en la llamada.

```php
<?php
  1 reference
  function muestra($a, $b=2, $c=4) {
    echo "$a $b $c";
  }
  muestra(c: 3, a: 1);
?>
```

```
1 2 3
```

En la **definición** de la función:
- $a es un parámetro obligatorio
- $b y $c son opcionales

En la **llamada** a la función, al utilizar los parámetros con nombre, podemos cambiar el orden:
- Si hiciera **muestra(1,3)**, PHP asignaría 3 al parámetro $b, dejando $c por defecto.
- Al no tener que guardar el orden, puedo asignar un valor a $c sin asignar un valor a $b, que es opcional.

Esto **soluciona** el problema de varios parámetros opcionales o parámetros opcionales y variádicos.

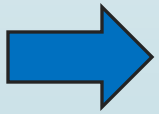
Funciones tipadas

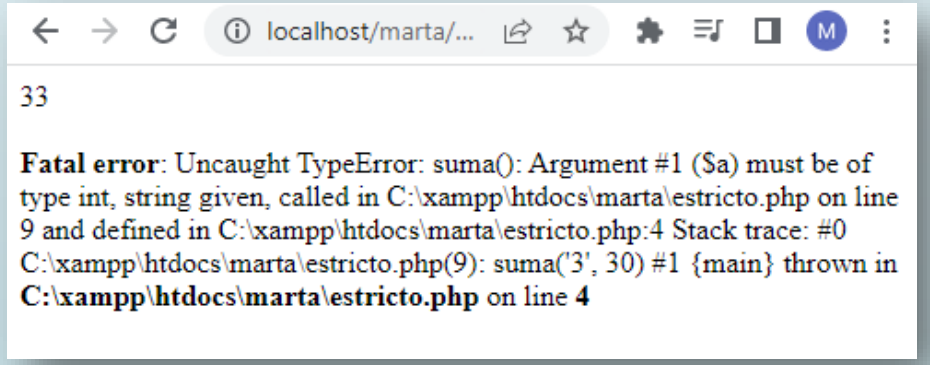
- Desde PHP 7, podemos definir tipos de datos tanto para las funciones como para los parámetros. A esto se le llama **strict_types** (tipado estricto).
- Si los tipos no coinciden, dará error.
- Los tipos contemplados son: int, float, string, bool, object y array.
- Si queremos utilizar tipado estricto, hay que definirlo en la primera línea de cada archivo php:

```php
<?php
    declare (strict_types=1);
```

- Si una función no retorna valor, se pone void.

```php
<?php
  declare (strict_types=1);

  3 references
  function suma (int $a, int $b) : int {
    return $a + $b;
  }

  echo suma(3,30) . "<br/>";
  echo suma("3", 30);
?>
```





Funciones anónimas

PHP permite el uso de funciones **anónimas**: Son funciones que no tienen nombre. También se les llama **closures** o **lambda**. Se utilizan en varios contextos:

1. Puedes asignar una definición de una función a una variable. No se hará la llamada a la función, sino que se hará referencia a la variable. Si había parámetros, se introducen al referenciar la variable (Ejemplo 1)

   Al caso anterior, podemos añadir un valor de parámetro previo a la definición, con la palabra **use** (Ejemplo 2)

2. Otra opción de uso es como parámetro de otra función. También se les llama "**callable**" o "**callback**". Muchas funciones predefinidas, como por ejemplo usort, tienen como parámetro una función "callback", es decir, una función que se aplicará en su interior (Ejemplo 3)

Ejemplo 1

```php
<?php
  $triple = function ($num) { return $num * 3;};
  echo "El triple de 3 es " . $triple(3);
?>
```

Ejemplo 2

```php
<?php
  $mas = 2;
  $tripleMas =
    function ($num) use ($mas) {return ($num * 3) + $mas;};
  echo "El triple más de 3 es " . $tripleMas(3);
?>
```

Ejemplo 3

```php
$arr = [10,3,70,21,54];
usort ($arr, function ($x , $y) {
   return $x > $y;
});
```

3 10 21 54 70

usort ordenará un array según sus valores usando una función de comparación definida por el usuario. En este caso se ordena de menor a mayor.

Parámetros callable: Función predefinida array_map

- La función array_map de PHP tiene dos parámetros: Una función callable definida por el usuario, y un array (o varios).
- Lo que hace es recorrer el array, y aplicar a cada elemento la función callable.
- Retorna otro array, con la transformación realizada a cada elemento.
- Podemos invocar array_map poniendo entre comillas el nombre de la función a aplicar (ejemplo1), o bien definiendo en ese momento dicha función, como función anónima (ejemplo 2)

Ejemplo 1: La función callable no es anónima, se indica su nombre entre comillas.

```php
function cubo($elemento) {
  return $elemento * 2;
}
$lista = [10, 20, 30];
$lista_cubo = array_map("cubo", $lista);
print_r($lista_cubo);
```

Ejemplo 2: La función callable es anónima. Se pasa de parámetro su definición.

```php
$lista = [10, 20, 30];
$lista_cubo = array_map (
  function($el){return $el**3;},
   $lista );
print_r($lista_cubo);
```

Salida en ambos casos

```
Array ( [0] => 1000 [1] => 8000 [2] => 27000 )
```

Ejercicio 4:

Función anónima

Figura A

El IVA de 200 es 42

Figura B

Precios: 145 200 87 23 56
Ivas: 30.45 42 18.27 4.83 11.76

1. Crea una **función anónima** con un parámetro $precio. La función retornará el iva correspondiente a ese parámetro.
2. El % de iva a aplicar no será un parámetro. Se indica en una variable $iva, asignada antes de la definición de la función anónima.
3. La función anónima se asignará a la variable $porcentaje, para su posterior referencia.
4. La salida del programa será la figura A de la izquierda.

Ampliación: Aplica una función anónima a un array de precios (ver figura B) y muestra la salida tal y como aparece en la figura.

Librerías de funciones y plantillas PHP/HTML

- **Librerías**: Podemos agrupar varias funciones en un fichero, a modo de librería reutilizable.
  - Para utilizar una librería, utilizando una de las siguientes sentencias:
  - **include(archivo)** : Incluye el archivo. Si no lo encuentra no da error.
  - **require(archivo)**: Incluye el archivo. Si no lo encuentra da error.
  - **include_once(archivo)**: Como include, pero si el archivo ya estaba incluido no vuelve a hacerlo.
  - **require_once(archivo)**: Como require, pero si el archivo ya estaba incluido no vuelve a hacerlo.
- **Plantillas**: Podemos crear archivos separados para escribir fragmentos HTML reutilizables en todos nuestros programas.
  - El archivo quedaría incluido en el lugar donde se haga el **include**.
  - Podemos tener fragmentos php/html de visualización de cabeceras y pies, y separar la visualización de otros fragmentos de código.

Ejemplo de plantillas en PHP

encabezado.php

```
1   <!DOCTYPE html>
2   <html>
3     <head>
4       <meta charset="utf-8">
5       <meta name="viewport" content="width=Device-width, initial-scale="1.0">
6       <title><?=$titulo?></title>
7     </head>
8     <body>
9
```

El archivo **encabezado.php** se reutiliza en todos los programas. La variable $titulo se asignará en cada programa, haciendo que la pestaña muestre el título correspondiente a cada uno.

contenido.php

```
<?php
  $titulo = "Ejemplo de plantilla";
  include("encabezado.php");
?>
<h1><?= $titulo ?></h1>
<?php
  include("pie.html");
?>
```

El archivo **contenido.php** es la plantilla básica de un programa cualquiera. Posteriormente añadiré el código correspondiente, asignando un título y completando la funcionalidad correspondiente.
Si hacemos request de este documento, incluirá el código de los otros dos, fusionando en el lugar del **include** correspondiente.

Salida en navegador

pie.php

```
<footer>Marta Olmedilla</footer>
</body>
</html>
```

El archivo **pie.html** es sólo html, finaliza la visualización de la salida. Lo reutilizaré en todos mis programas, mostrando mi nombre al pie.

Algunas funciones predefinidas típicas: Cadenas.

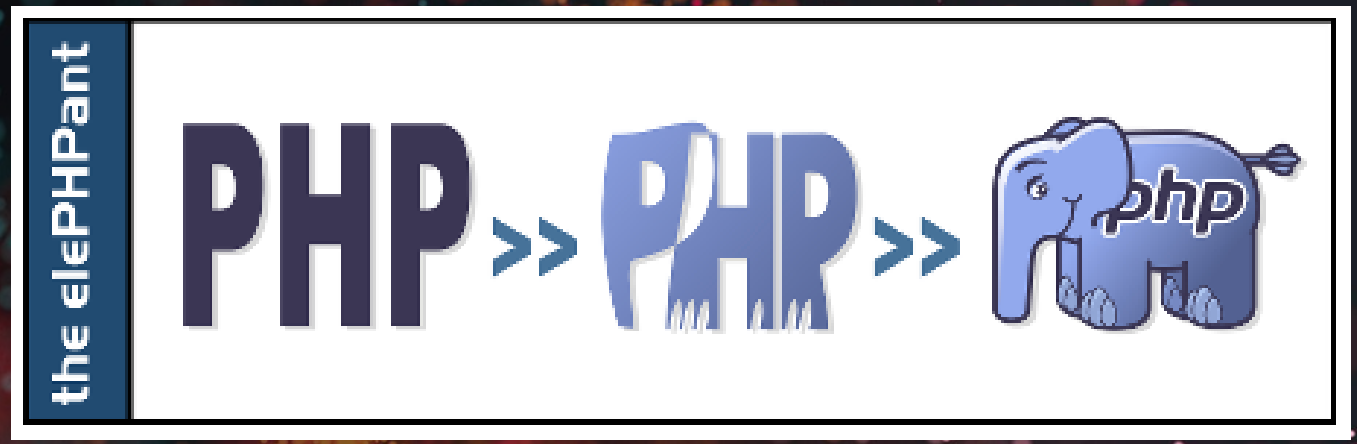
- **Limpiar y rellenar:**
  - trim: Elimina espacios sobrantes
  - ltrim: Elimina espacios a la izquierda
  - rtrim: Elimina espacios a la derecha
  - str_pad:Rellena hasta una longitud máxima con el carácter indicado.
- **Comparar y buscar:**
  - strcmp: compara dos strings
  - strcasecmp: pasa a minúsculas y compara dos strings
  - strncmp: Compara los n primeros caracteres de dos strings
  - strpos / strnpos: Busca una cadena y devuelve la posición de la primera/última ocurrencia
- **Subcadenas:**
  - explode: Separa por el carácter indicado una cadena en array de cadenas
  - implode/join: Pasa un array de cadenas a una cadena añadiendo un separador indicado
  - str_split / str_chunk: Separa una cadena en array de cadenas de la longitud fija indicada

Algunas funciones predefinidas típicas: Números.

- **Trigonométricas**
  - sin: seno
  - cos: coseno
  - tan: Tangente
  - Str_pad:Rellena hasta una longitud máxima con el carácter indicado.
- **Cálculo:**
  - sqrt: raíz cuadrada
  - log: logaritmo
  - decbin: Conversión decimal-binario
  - max: Máximo
  - min: Mínimo
- **Números aleatorios:**
  - rand: Obtiene un número aleatorio de entre el rango indicado

Algunas funciones predefinidas típicas: Tipos de datos.

- settype: Fuerza la conversión a un tipo de datos
- gettype: Obtiene el tipo de datos de una variable
- Is_int: Indica si la variable es entera
- Is_str: Indica si la variable es string
- Is_float: Indica si la variable es float
- Is_array: Indica si la variable es un array
- Is_object: Indica si la variable es de tipo object (objeto)



https://www.php.net/docs.php