

## 5.1. Funciones en PHP

**1º DAW - Programación**  
*David Soler Talens*

# Funciones.



- **Es un bloque de código diseñado para realizar una tarea particular.**
  - También llamadas procedimientos o sub-rutinas
- **Una función se ejecuta cuando se la invoca**
  - También se denomina “llamar” a la función.
- **Ventajas**
  - *Re-uso de código* → reduce tiempo de codificación
  - Mejora la legibilidad
  - Reduce el tiempo de depuración
  - Simplifica el mantenimiento del código



# Declaración de funciones.

- La función puede ser **declarada en cualquier punto** del código y será **accesible desde cualquier punto** del código
- La declaración sigue la sintaxis

```
function nombre(parametro1, parametro2, ...) {  
    // codigo a ejecutar  
    return valor;  
}
```

- Los parámetros o argumentos son valores que recibe la función al ser llamada
- La palabra reservada 'return'
  - Posibilita que la función devuelva a su llamador, un valor
  - No es obligatorio, pero prácticamente siempre lo usaremos

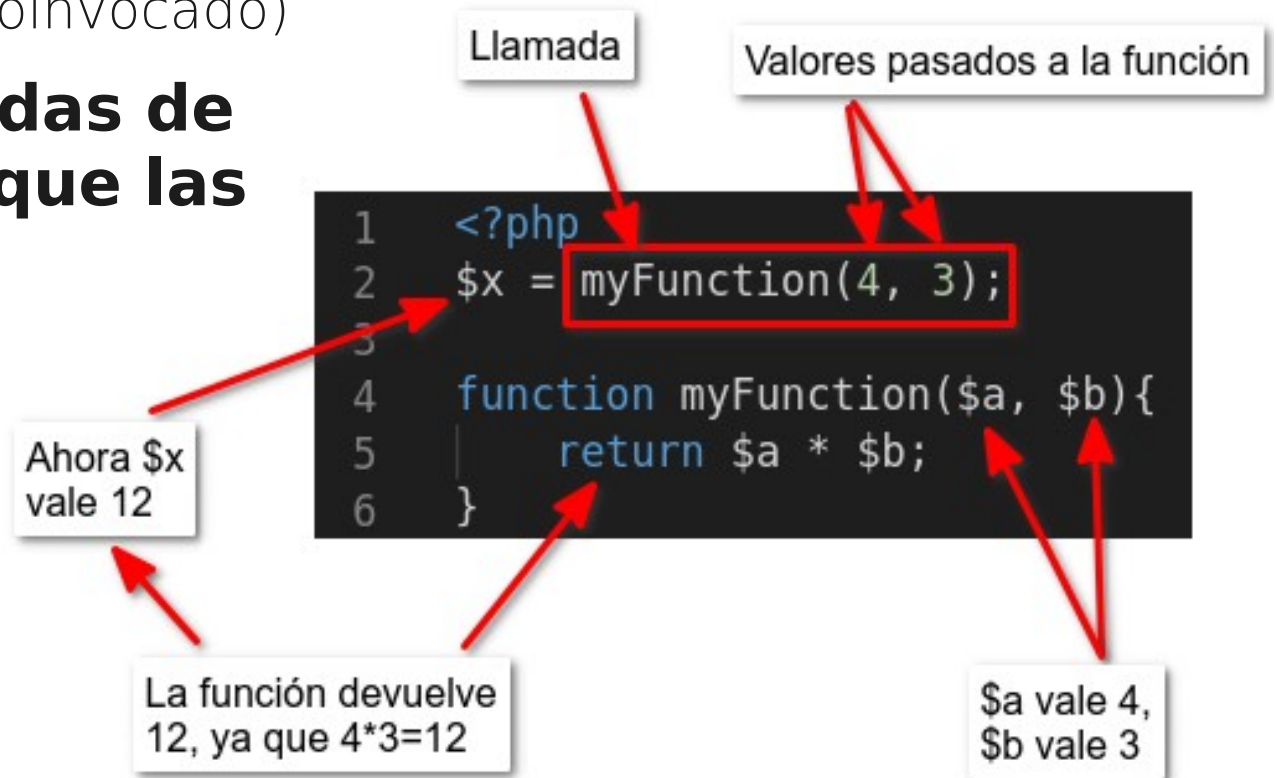
# Llamada a funciones.



- **Las funciones pueden ser llamadas:**

- Cuando se invoca (llama) desde el código PHP
- Automáticamente (autoinvocado)

- **Pueden ser utilizadas de la misma manera que las variables, en todo tipo de fórmulas, asignaciones y cálculos.**



# Parámetros (I).



- **Diferenciar entre:**

- Parámetros → identificadores en la definición de la función.
- Argumentos → valores reales pasados a (y recibidos por) la función.

- **Parámetros por defecto**

- Valor que toma el parámetro en caso de ser omitido el argumento relacionado
- Se pueden definir en la declaración de la función:

```
function myFunction($x, $y = 2) {  
    // código de la función  
}
```

- En caso de no tener valor por defecto, y que no se haya proporcionado argumento, ***PHP genera un error!***

# Parámetros (II).



- **Argumentos pasados por valor**

- Si una función cambia el valor de un parámetro *no cambia* el valor original del argumento.
- Los cambios en los argumentos *no son visibles fuera de la función*.
- Por defecto, los parámetros son pasados por valor

- **Argumentos pasados por referencia**

- Si una función cambia el valor de un parámetro *cambia* el valor original del argumento.
- Los cambios en los argumentos *son visibles fuera de la función*.
- Para determinar que un parámetro es pasado por referencia se antepone el símbolo '&' al identificador del mismo, por ejemplo *&\$variable*

# Parámetros (III).



## • Array de argumentos

- Array que contiene todos los argumentos pasados
- Nos permite pasar N argumentos sin necesidad de definirlos en la declaración de la función

```
1  <?php
2  foo(1, 2, 3);
3  function foo()
4  {
5      $numargs = func_num_args();
6      echo "Número de argumentos: $numargs <br>";
7      if ($numargs >= 2) {
8          echo "El segundo argumento es: " . func_get_arg(1) . "<br>";
9      }
10     $arg_list = func_get_args();
11     for ($i = 0; $i < $numargs; $i++) {
12         echo "El argumento $i es: " . $arg_list[$i] . "<br>";
13     }
14 }
```

Pasamos 3 argumentos en la llamada a la función

no definimos ningún parámetro

función nativa para obtener el número de argumentos pasados

función nativa para acceder a un argumento concreto

función nativa para obtener el array de argumentos

# Scope (I).



- **Accesibilidad de una variable**
  - Dentro del ámbito que se crearon
- **Las funciones**
  - Representan un *ámbito propio*
  - No pueden acceder a variables fuera de la función
  - La información con la que trabaja una función debe ser pasada a ésta por valor o referencia, a través de sus *parámetros*



# Scope (II).



- Para acceder a una variable externa a la función se usa *'global'*
- También podemos usar ***\$GLOBALS***
  - Array asociativo que contiene las variables globales

```
1  <?php
2  $a = 1;
3  $b = 2;
4
5  function Suma()
6  {
7      $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
8  }
9
10 Suma();
11 echo $b;
```

Accedemos por el identificador de la variable sin '\$'

\$b vale 3

```
1  <?php
2  $a = 1;
3  $b = 2;
4
5  function Suma()
6  {
7      global $a, $b;
8
9      $b = $a + $b;
10 }
11
12 Suma();
13 echo $b;
```

Hacemos accesible \$a y \$b dentro de la función

\$b vale 3

# Funciones nativas (I)



- **Las funciones y constantes nativas son las que incorpora el propio lenguaje**
- **Suelen cubrir necesidades o rutinas habituales de codificación**
- **La idea es *‘no re-inventar la rueda’***
- **Son de muchos tipos**
  - Matemáticas
  - Manejo de string
  - Manejo de fecha/hora
  - Manejo de arrays
  - Etc...

# Funciones nativas (II). Math.



- **pi()** → número PI
- **round()** → redondea a entero más próximo
- **ceil()** → redondea entero más próximo hacia arriba
- **floor()** → redondea entero más próximo hacia abajo
- **sqrt()** → raíz cuadrada
- **abs()** → valor absoluto
- **sin(), cos()** → seno y coseno
- **min(), max()** → mínimo y máximo de una sucesión numérica
- **rand()** → valor aleatorio

Referencia completa de  
Math en [www.w3schools.com](http://www.w3schools.com)



# Funciones nativas (III). String

- **strlen()** → longitud del string
- **strpos()** → busca un texto dentro de otro y retorna su posición
- **printf()** → imprime string con formato
- **substr()** → extrae parte de una string creando otro
- **str\_replace()** → reemplaza partes del texto por otras
- **strtoupper()** → convierte en mayúsculas
- **strtolower()** → convierte a minúsculas
- **trim()** → elimina los espacios vacíos al principio y al final del string
- **implode()** → convierte un array en string
- **explode()** → convierte un string en un array

Referencia completa de  
String en [www.w3schools.com](http://www.w3schools.com)