Funciones en PHP

Lenguajes Interpretado en el Servidor

Objetivos

- Tener claridad del propósito de trabajar con funciones en PHP.
- Adquirir dominio en la creación de funciones definidas por el programador con PHP.
- Tener conocimiento de las principales funciones de PHP para resolver tareas frecuentes.
- Utilizar parámetros o argumentos en la definición de funciones.
- Devolver valores desde las funciones.
- Utilizar las funciones o constructores *include()* y *require()* para agregar modularidad a las apliaciones.

Contenido

- ¿Qué es una función?
- Declaración de funciones en PHP.
- ¿Cómo se ejecutan las funciones?
- Llamadas a función.
- Paso de argumentos o parámetros a las funciones.
- Tipos de parámetros en las funciones.
  - Parámetros por valor.
  - Parámetros por referencia.
  - Parámetros por defecto.
- Devolución de valores desde las funciones.
  - Devolución de valores escalares.
  - Devolución de matrices.
  - Devolución de referencias.

Contenido

- Ámbito o alcance de las variables.
  - Ámbito local.
  - Ámbito global.
  - Variables estáticas.
- Funciones con número variable de argumentos.
- Funciones variables (o dinámicas).
- Funciones recursivas.
- Funciones incorporadas de PHP.
- Funciones para el manejo de matrices.
- Reutilización de código y modularidad.
  - Funciones *include()*, *require()* y sus variantes.
  - Utilización de *include()*.
  - Utilización de *require()*.

¿Qué es una función?

- Una función es un **bloque de código independiente** al que **se identifica con un nombre** y cuyo **propósito es realizar una tarea específica** y, opcionalmente, **devolver un valor**.
- Estos bloques de código se pueden invocar desde cualquier punto de un ***script*** PHP mediante un identificador (nombre que se le ha asignado a la función).
- Al utilizar funciones **aumenta la eficiencia del código**, volviéndolo **más fácil de depurar**.

¿Qué es una función?

- Las funciones permiten ejecutar un bloque de código varias veces sin que ello signifique volver a escribir todas las instrucciones cada vez que se utilicen.
- Una **función bien diseñada** debe actuar como un **pequeño programa independiente** que **no tiene conciencia de las variables externas a ella**. Pese a ello se puede utilizar la palabra reservada *global* para que la función pueda tener acceso al valor de una variable del exterior.

Declaración de funciones en PHP

- En PHP las funciones pueden aparecer en cualquier parte del ***script*** y pueden ser invocadas desde cualquier punto del mismo. Esto no era así en la versión 3 de PHP.

Ejemplo #1 Pseudo código para demostrar el uso de funciones

```php
<?php
function foo($arg_1, $arg_2, /* ..., */ $arg_n)
{
    echo "Función de ejemplo.\n";
    return $valordevuelto;
}
?>
```

Declaración de funciones en PHP

- Una función en PHP se declara utilizando la siguiente construcción sintáctica:

```
function nombre_funcion([parametro1[,parametro2,…]]){
    [//Bloque de sentencias;]
}
```

Donde:

`function` es la palabra reservada de PHP para crear o definir una función. Recuerde que por ser palabra reservada se puede escribir sin tener en cuenta las mayúsculas y las minúsculas; sin embargo, para establecer un estilo de programación consistente, se recomiendo utilizar siempre minúsculas.

Declaración de funciones en PHP

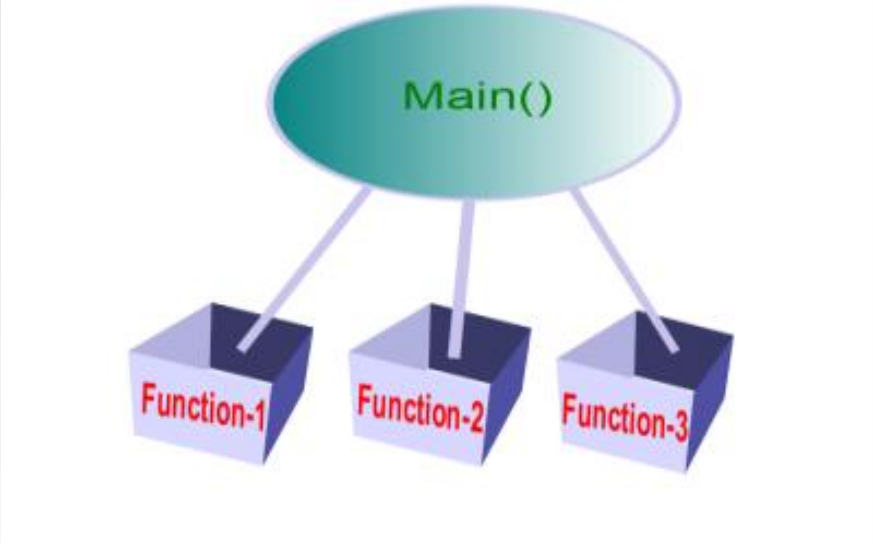
`nombre_funcion` representa el identificador con el que se hará referencia al bloque de código comprendido entre las llaves de la función.

`parametro1,parametro2,...` representan uno o varios de los parámetros o argumentos que reciben los valores con los que la función es llamada o invocada. Si se utilizan varios parámetros estos deben ir separados por comas.

`Bloque de sentencias` son las instrucciones que componen el cuerpo de la función y que están destinadas a realizar una tarea específica.

¿Cómo se ejecutan las funciones?

- Las funciones no se ejecutan automáticamente al llegar a la parte de la implementación dentro del código, como ocurre con las instrucciones secuenciales.
- **Para ejecutar una función** esta **debe ser invocada o llamada**, ya sea **desde el flujo de ejecución principal** o **desde las instrucciones de otra función**.



Llamadas a función

- La llamada a una función se realiza de forma similar a como se realiza esta misma tarea en lenguajes de programación como C/C++, Java, Perl e incluso JavaScript.
- La sintaxis de una llamada a función es como la siguiente:

```
nombre_funcion([parametro1[, …]]);
```

- En el momento en que se realiza la llamada a una función el control del programa pasa a la primera línea del código de la función.

Llamadas a función

- Una vez que se terminan de ejecutar las instrucciones que componen el cuerpo de la función el control del programa regresa al punto desde donde fue hecha la llamada a la función y continúa la ejecución del *script*.
- Si al hacer la llamada a la función se le pasan valores como argumentos, estos ocupan el lugar de las variables definidas como parámetros de la función.
- Al terminar la función, si se devuelve un valor, este es devuelto al punto de la llamada donde puede asignarse a una variable o utilizarse para alguna operación específica dentro de una expresión.

Paso de argumentos a la función

- Es habitual crear funciones que acepten argumentos o parámetros con el propósito de hacer que la función sea reutilizable.
- El tipo de los argumentos para las funciones puede ser de cualquiera de los tipos de dato válidos para PHP, incluso matrices. Sin embargo, una vez realizada la llamada a la función el tipo de dato enviado es uno en concreto para cada argumento de la función.

include()
require()

Ejemplo de paso de argumentos a una función

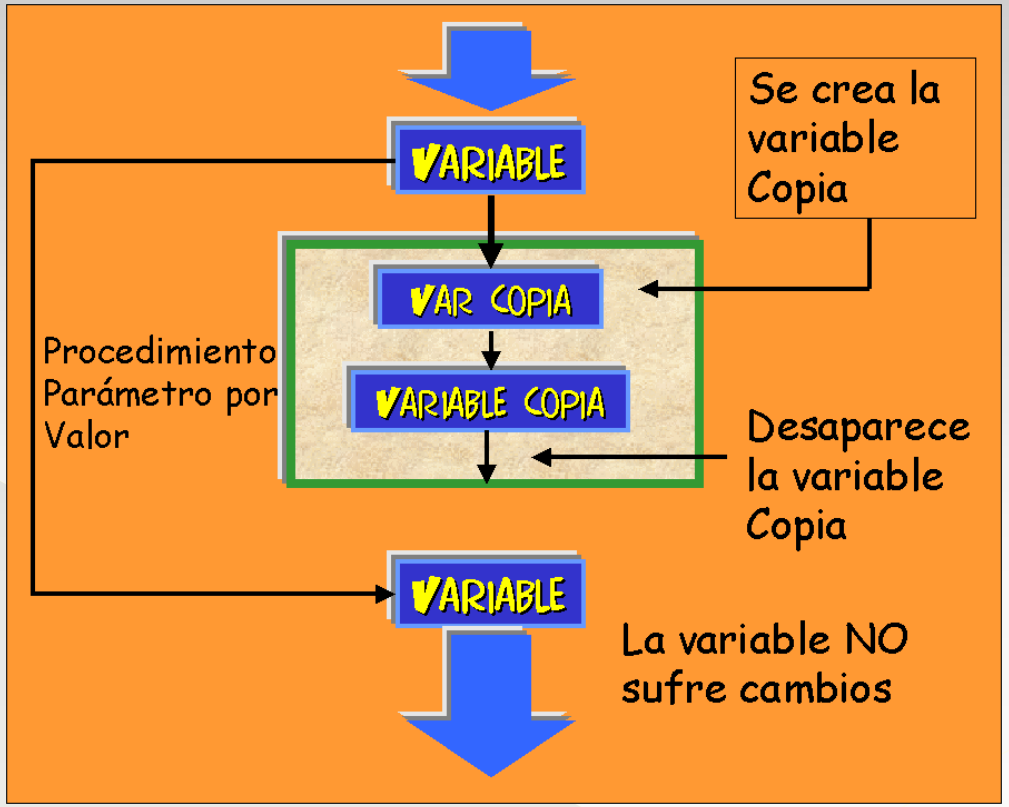
```
function createtable($data){
    echo '<table border="1">';
    reset($data);
    $value = current($data);
    while($value){
        echo "<tr><td>$value</td></tr>\n";
        $value = next($data);
    }
    echo '</table>';
}
```

La llamada a la función debería hacerse así:

```
$nombres = array('Jorge Santos','Pilar Iglesias');
createtable($nombres);
```

Tipos de parámetros en las funciones

- **Parámetros por valor**. Cuando se pasa una variable como argumento por valor a una función, lo que se recibe realmente es una copia del valor almacenado en la variable. Esto significa que las modificaciones hechas dentro de la función al valor recibido como parámetro no alteran el valor original de la variable enviada.

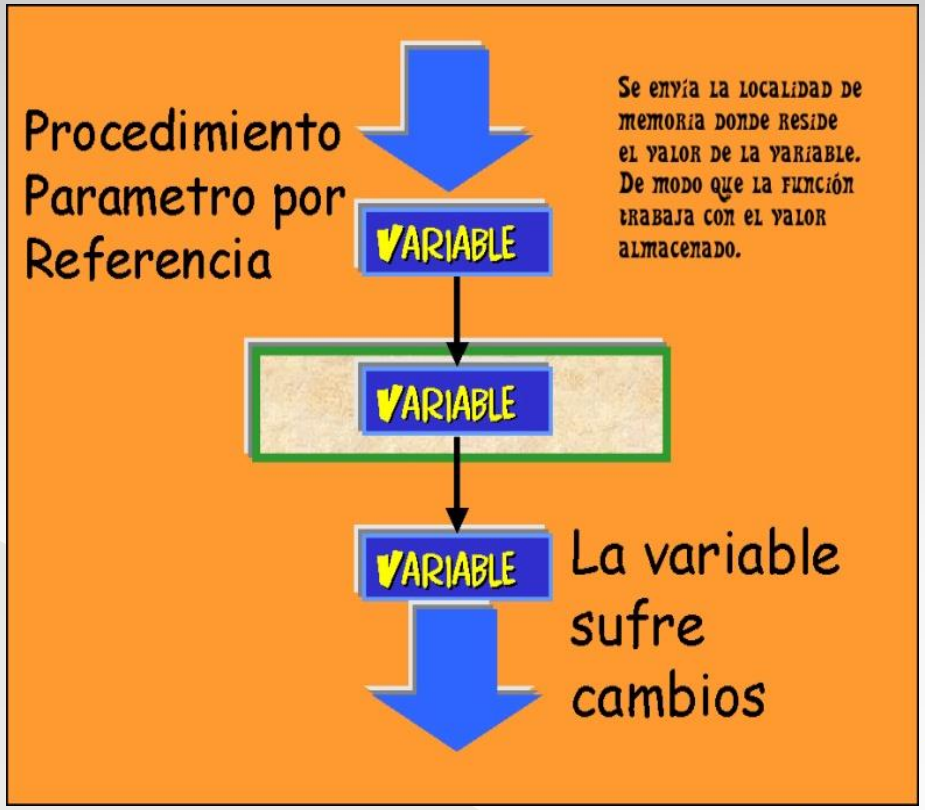


Tipos de parámetros en las funciones

```php
 <?php
function concatenar($cadena){
    $cadena .= "este es el complemento.";
}
$cad = "Esta es una cadena y ";
concatenar($cad);
?>
```

Tipos de parámetros en las funciones

- **Parámetros por referencia**. Cuando se pasa una variable por referencia como argumento de una función, lo que realmente se está haciendo es pasarle una referencia de la dirección de memoria donde está almacenado ese valor. Por lo tanto, se está trabajando con el valor de la variable directamente, no con una copia. Es por esto que los cambios que se hagan dentro de la función al parámetro, afectarán el valor original de la variable.



Tipos de parámetros en las funciones

```php
<?php
function añadir_algo(&$cadena)
{
    $cadena .= 'y algo más.';
}
$cad = 'Esto es una cadena, ';
añadir_algo($cad);
echo $cad;    // imprime 'Esto es una cadena, y algo más.'
?>
```

Tipos de parámetros en las funciones

- **Parámetros por defecto**. Un parámetro por defecto es una forma en que PHP permite crear argumentos opcionales dentro de la definición de una función. Cuando se define un parámetro por defecto, si en la llamada a la función no se especifica el parámetro, toma el valor predefinido en la definición o firma de la función. Los parámetros por defecto deben ser los últimos en la definición de la función, de lo contrario obtendremos un comportamiento inesperado cuando llamemos a la función.

Tipos de parámetros en las funciones

```php
<?php
function hacercafé($tipo = "capuchino")
{
    return "Hacer una taza de $tipo.\n";
}
echo hacercafé();
echo hacercafé(null);
echo hacercafé("espresso");
?>
```

Devolución de valores desde funciones

- Las funciones por lo general devolverán un valor que será utilizado en el punto del ***script*** desde donde se realiza la llamada.
- Por esta razón la llamada a la función será asignada casi siempre a una variable o estará presente dentro de una expresión que implique una comparación o cálculo.
- Por ejemplo:

```
$n1 = 12; $n2 = 25; $n3 = 18;
$mayor = larger($n1, $n2, $n3);
```

O bien,

```
echo "El número mayor es: " . larger($n1, $n2, $n3);
```

Devolución de valores desde funciones

- Las funciones por lo general devolverán un valor que será utilizado en el punto del ***script*** desde donde se realiza la llamada. No osbtante también podemos hacer que las funciones devuelvan múltiples valores o que devuelvan referencias.
- Las formas más comunes en que PHP permite devolver múltiples valores es usando matrices, ya sea escalares o asociativas.
- En cuanto a la devolución de referencias, se requiere que en la definición del código de la función se anteponga el símbolo ampersand (&) al nombre de la función.

Devolución de múltiples valores

- La forma más práctica de devolver múltiples valores desde una función es utilizando matrices o arreglos. Estos pueden ser de todos los tipos de matrices soportados por PHP y de una o más dimensiones.
- Veamos el siguiente ejemplo:

```
$numeros = array(1,2,3,4,5,6);
printf("<pre>Antes: ");
print_r($numeros);
$cuadrado_numeros = square($numeros);
printf("utf8_decode(Después: "));
print_r($cuadrado_numeros);
printf("</pre><br />");
//… continúa
```

Devolución de múltiples valores

- La declaración de la función:

```
//… viene de la diapositiva anterior
//La función recibe una matriz como argumento que se modificará dentro
//de la misma y luego se retornará modificada.
function square($numbers){
  for($i=0; $i<count($numbers); $i++){
    $numbers[$i] *= $numbers[$i];
  }
  return $numbers;
}
```

```
Antes: Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
    [5] => 6
)
```

```
Después: Array
(
    [0] => 1
    [1] => 4
    [2] => 9
    [3] => 16
    [4] => 25
    [5] => 36
)
```

Devolución de referencias

- Las funciones también podrían devolver referencias en lugar de valores.
- Para hacer esto basta con anteponer el símbolo de ampersand (&) justo antes del nombre de la función cuando se está creando su definición.
- El efecto que tiene esto es que cuando la función devuelve la variable con la sentencia return, pasa una referencia a esa variable en lugar del valor de su valor y con esa referencia se trabaja en el punto desde donde se realizó la llamada a la función.

Devolución de referencias

- Ejemplo:

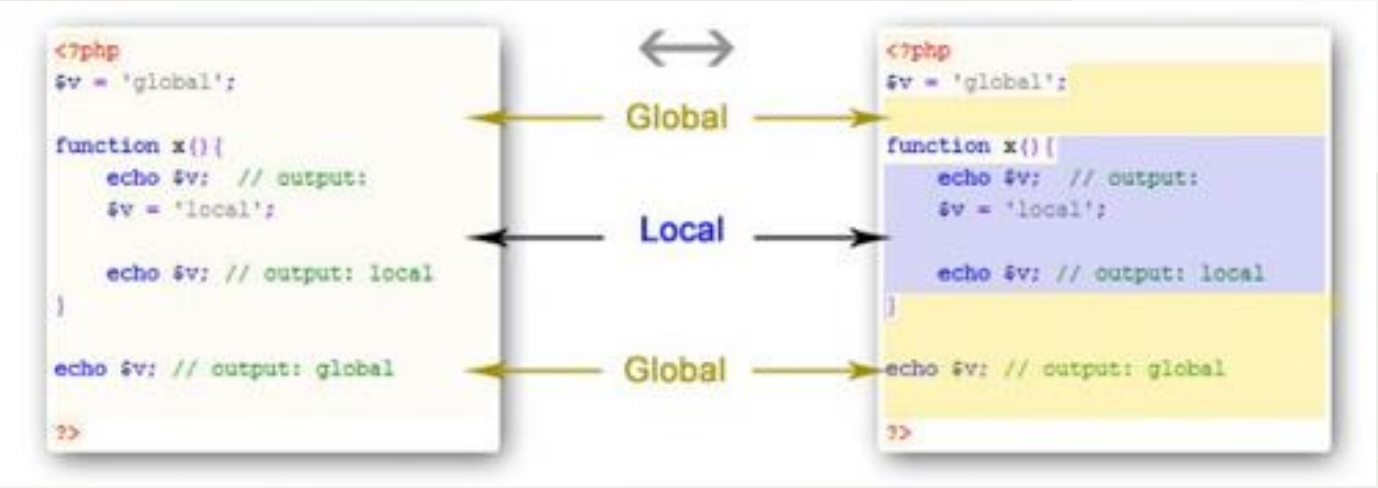
```
//Note que al nombre de la función se le ha antepuesto el símbolo '&'
function &numero_aleatorio(){
   //Generar un número aleatorio del 1 al 100
   $var = rand(1,10);
   return $var;
 }

 $minumero = 5;
 echo "<h3>" . $minumero . "</h3>\n";

 //Invocando la función
 $numeroreferencia = &numero_aleatorio();
 $numeroreferencia++;

 //Imprimiendo los resultados
 //Note que a la llamada a la función se le ha antepuesto el símbolo '&'
 $minumero = &numero_aleatorio();
 echo "<h3>" . $minumero . "</h3>\n";
```

Ámbito o “alcance” de las variables

- Llegados a este punto es conveniente volver a discutir sobre las variables para entender claramente las partes del programa que pueden visualizar o acceder a una variable.
- La duración de la vida de una variable es el tiempo durante cual existe y se puede por tanto, acceder a esta asignándole un valor o leyendo el valor que almacena.
- En PHP se definen tres tipos de ámbito o alcance para las variables: a) local, b) global, y c) estático.

Ámbito local

- Las variables que son definidas dentro de las funciones tienen un ámbito o alcance local, lo que significa que sólo son accesibles dentro de la función en la que fueron definidas.
- Estas variables dejan de existir o, lo que es lo mismo, dejan de ser accesibles cuando la función termina.



Ámbito global

- Dentro de las funciones no se puede acceder a las variables que han sido definidas fuera de ellas.
- Una forma de poder hacer que una función pueda acceder a variables externas a ella es pasar estas variables como argumento por referencia, ya que así la función trabajará con la variable misma, si se hiciera por valor trabajaría con una copia de la misma.

Ámbito global

- Para acceder desde una función a una variable definida fuera de ésta, sin tener que pasarla por referencia es utilizar la palabra reservada global anteponiéndola al nombre de la variable. Así: **`global $salario;`**
- Otra forma de acceder a una variable global desde una función es utilizando la matriz superglobal $GLOBALS[], que contiene todas las variables de ámbito o alcance global presentes en el script actual.

Variables estáticas

- Las variables que se declaran dentro de las funciones dejan de existir una vez que la función termina.
- El proceso se puede resumir así, las variables son locales dentro de la función, se crean cuando la función es invocada y se destruyen cuando la ejecución de la función termina.
- Una **variable estática** no es visible fuera de la función, pero, al contrario de la variable local, no se destruye cuando la función se termina de ejecutar.
- Cuando se inicializa una variable estática, ésta no pierde su valor de una llamada a otra, siempre recuerda el valor que tenía antes.

Variable estática

- Lo explicado anteriormente implica que la siguiente función al ser llamada en distintas ocasiones siempre mostraría el mismo resultado para la variable $contador.

Ejemplo:

```
function contador(){
  $contador = 0;
  $contador = $contador + 1;
  return $contador;
}
//Llamada a la función contador()
for($i=1; $i<=10; $i++){
  echo contador() . ","; //Mostrará 1,1,1,1,…,1 (10 veces)
}
```

Variable estática

- Si se desea que la función recuerde el valor de $contador en cada llamada; es decir, que conserve el valor que tenía en la llamada anterior, debemos anteponer la palabra stactic.

Ejemplo:

```
function contador(){
  static $contador = 0;
  $contador = $contador + 1;
  return $contador;
}
//Llamada a la función contador()
for($i=1; $i<=10; $i++){
  echo contador() . ","; //Mostrará: 1,2,3,4,…,10
}
```

Funciones con número variable de argumentos

- PHP permite definir funciones en las que el número de parámetros no está fijado a un número específico.
- Este tipo de función recibe como parámetro una lista de valores de longitud variable.
- El funcionamiento de este tipo de funciones requiere del uso de un conjunto de funciones definidas en PHP, que son:
- **`func_num_args()`**: devuelve el número de argumentos pasados a la función.

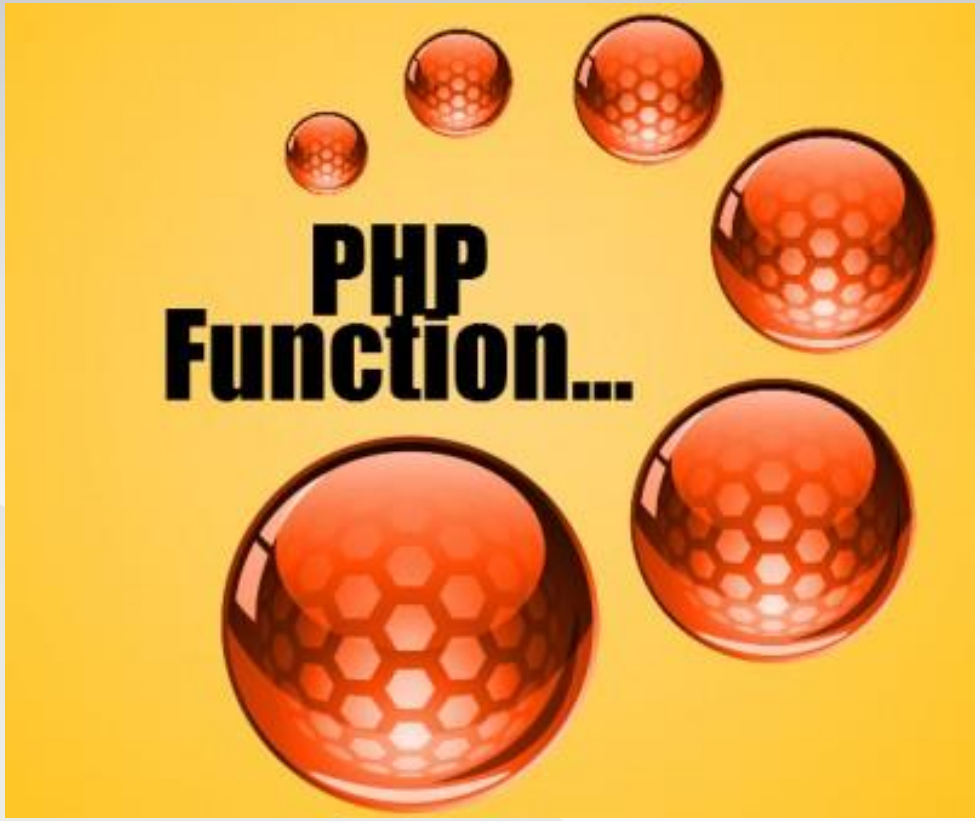
Funciones con número variable de argumentos

- **`func_get_args()`**: devuelve una matriz (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 en las matrices. Si se solicita un argumento de una posición que no existe, devuelve *false*.

```php
function parse_args($args){

if( empty($args) ):
    throw new Exception("\n\nfunction parse_args() expects you to pass a param\n\n");
elseif( is_array( func_get_arg(0) )  ):
/* PHP natively maintains an array of parameters passed.
   Within the array of arguments passed, we can test the data type. */
/* return right away if param itself is an array */
    return func_get_args();
elseif( !is_string($args) ):
    throw new Exception("\n\nfunction parse_args() expects a string, ". gettype($args). " passed\n\n");
endif;

// array to store our parsed args
$parsed = array();
```

Funciones variables

- Esta es una construcción del lenguaje PHP que permite utilizar variables para almacenar el nombre de funciones, de modo que al añadir a esa variable paréntesis al final podrá invocar indirectamente a la función almacenada.
- Esto permite, entre otras cosas, realizar tareas como retrollamadas (o llamadas de retorno: *callbacks*) y tablas de llamada.



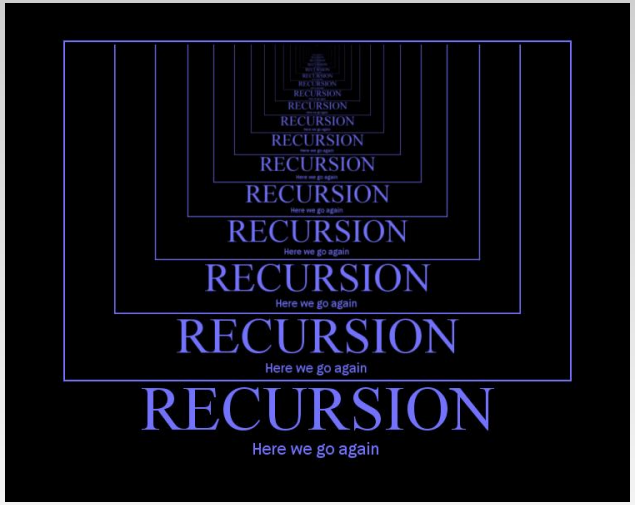
Funciones variables

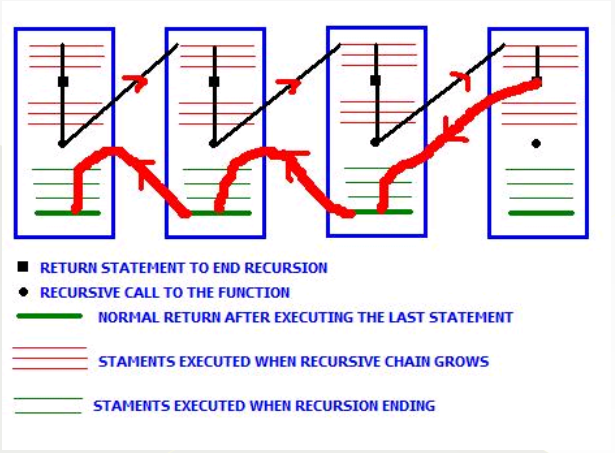
- Ejemplo:

```
function saludo_manana(){
  return ("Buenos días");
}
function saludo_tarde(){
  return ("Buenas tardes");
}
function saludo_noche(){
  return ("Buenas noches");
}
$turno = "tarde";
//Crear la variable con el nombre de la función
$function_variable = "saludo_" . $turno;
//Se le agregar paréntesis a la variable con el nombre de la función
echo $function_variable();
```

Funciones recursivas

- Una función recursiva es una función que se llama a sí misma en algún punto del cuerpo de una función.
- Debe tener especial cuidado al desarrollar la función recursiva ya que debe existir una condición de paro adecuada para evitar que la función se llame a si misma indefinidamente produciendo una falla que cuelgue el sistema.



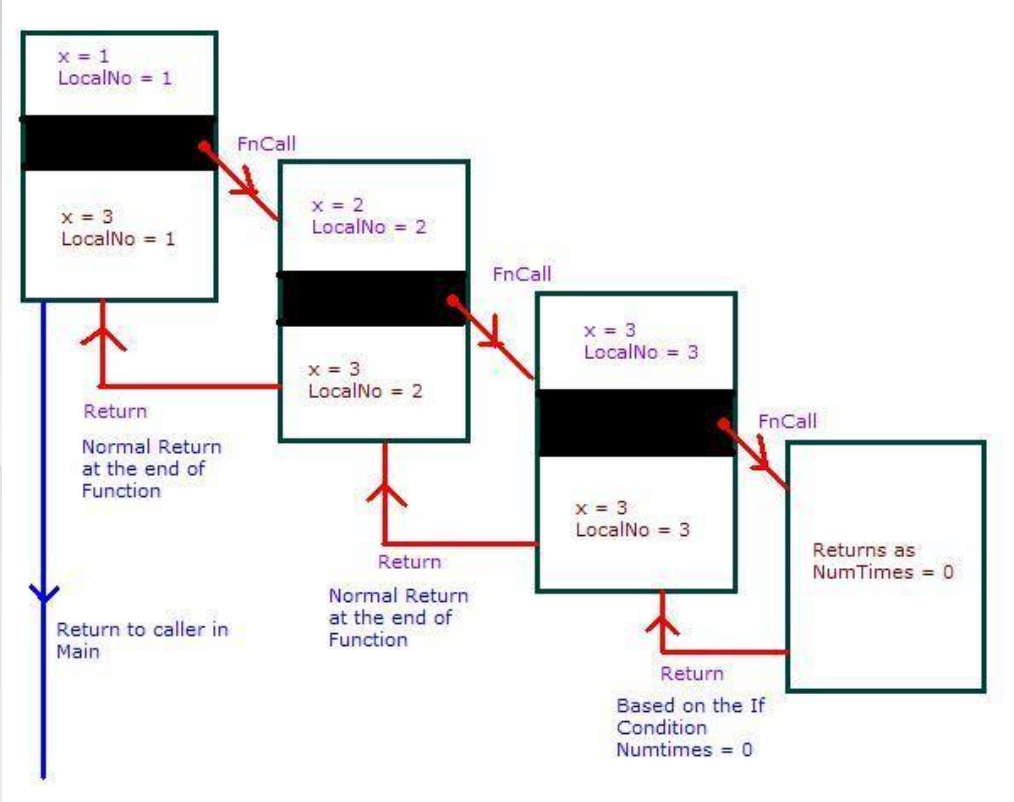


Ejemplo de función recursiva

```
function factorial($num){
        if($num == 1){
              return $num;
        }
        return $num * factorial($num-1);
 }
```

Llamada a la función:

```
$numero = 8;
 $msg = "El factorial de " . $numero;
 $msg .=  " es: " . factorial($numero);
 echo $msg;
```



Funciones incorporadas de PHP

- Existen un sinfín de funciones incorporadas en PHP que puede consultar detenidamente en el sitio oficial en el apartado Referencia de funciones: http://php.net/manual/es/funcref.php
- Encontrará las funciones organizadas en distintas categorías, sin embargo, puede realizar la búsqueda de una función concreta directamente con el buscador que se proporciona en el sitio web oficial de PHP.
- Las categorías de funciones que más hemos utilizado hasta este punto en la materia son: a) funciones matemáticas, b) funciones para manejo de fechas, c) funciones de cadenas, d) funciones para entrada/salida, e) funciones para manejo de variables, etc.

Funciones matemáticas de PHP

- min(). Se utiliza para encontrar el valor numérico más bajo, que puede ser proporcionado como un único argumento en una matriz o como una serie de valores escalares separados por comas.

```
mixed min(array $values);
mixed min(mixed $value1, mixed $value2[,… mixed $valueN]);
```

- max(). Encuentra el valor más alto de un conjunto de valores que puede ser pasado como un único argumento en forma de matriz o como una serie de valores escalares separados por coma.

```
mixed max(array $values);
mixed max(mixed $value1, mixed $value2[,… mixed $valueN]);
```

Funciones matemáticas de PHP

- round(). Devuelve el valor redondeado del primer argumento recibido, opcionalmente, con el número de decimales indicado en el segundo argumento y en la forma indicada por el tercer argumento.

```
float round(float $val[, int $precisión=0[, int $modo=PHP_ROUND_HALF_UP]]);
```

- ceil(). Redondea cantidades no enteras siempre hacia arriba. Esto significa que siempre devuelve el valor entero mayor al valor recibido como argumento.

```
float ceil(float $value);
```

Funciones matemáticas de PHP

- floor(). Redondea el valor recibido como argumento hacia abajo; es decir, devuelve el máximo de los valores enteros menores o iguales al valor pasado como argumento.

```
float floor(float $value);
```

- pow(). Devuelve el resultado de elevar un número a una potencia. La función recibe ambos datos como argumentos, el primero es la base y el segundo es la potencia a la que se desea elevar.

```
float pow(float $base, float $exponente);
```

- sqrt(). Obtiene la raíz cuadrada del valor pasado como argumento a la función.

```
float sqrt(float $argumento);
```

Funciones matemáticas de PHP

- rand(). Genera un número entero aleatorio. Puede ser invocada sin argumentos el número aleatorio obtenido puede ser tan grande como lo soporte el sistema operativo sobre el que se ejecute PHP. Si se desea un número aleatorio comprendido entre dos valores concretos, deberá pasarlos como argumentos.

```
int rand([int $min[, int $max]]);
```

- srand(). Genera un número aleatorio a partir de una semilla que puede proporcionarse como argumento. Si no se proporciona PHP genera una aleatoriamente.

```
void srand([int $semilla]);
```

Funciones para manejo de fechas

- time(). Retorna la fecha Unix actual, conocida también como marca de tiempo. Es importante configurar bien la zona horaria en el archivo de configuración de PHP para obtener resultados confiables al usar esta función.
- La sintaxis:

  int time(void);
- La función devuelve el momento actual medido como el número de segundos desde la época Unix (1° de ene 1970).

NOTA: fecha o época Unix es una fecha concreta a partir de la cual se cuentan los segundos, lo que da por resultado una nueva medida de tiempo que usan los sistemas operativos Unix o Linux, o incluso, lenguajes de programación como PHP.

Funciones para manejo de fechas

- Ejemplo con time():

```
$semanaSiguiente = time() + (7 * 24 * 60 * 60);
                  // 7 días; 24 horas; 60 minutos; 60 segundos
echo 'Ahora:              '. date('Y-m-d') ."\n";
echo 'Semana Siguiente: '. date('Y-m-d', $semanaSiguiente) ."\n";
// o usar strtotime():
echo 'Semana Siguiente: '. date('Y-m-d', strtotime('+1 week')) ."\n";
```

- La salida sería:

```
Ahora:              2005-03-30
Semana Siguiente: 2005-04-06
Semana Siguiente: 2005-04-06
```

Funciones para manejo de fechas

- date(). Permite dar formato a la fecha/hora local. Esto permite formatear la fecha en una cadena de texto más comprensible para el ser humano.
- Sintaxis:

```
string date(string $format[, int $timestamp = time()]);
```

- Devuelve una cadena formateada con la fecha, según el formato indicado en el primer argumento $format. Se puede indicar la marca de tiempo con que se desea trabajar o dejar que PHP tome la fecha y hora actual en caso de omitir este segundo argumento $timestamp.

Funciones para manejo de fechas

- Existen numerosas opciones de formato que se pueden proporcionar en el primer argumento de esta función. Se recomienda su revisión en el sitio web oficial de PHP (http://php.net/manual/es/function.date.php).
- Ejemplo:

```
<?php
  echo "Hoy es ", date("D"), ", ", date("d"), " de ", date("F"), " de ",
date("Y") , "<br />\n";
  echo date("\S\o\\n \l\a\s h:i:s a"), "<br />\n";
?>
//La salida sería
Hoy es Wed, 09 de Febrary de 2015
Son las 01:53:21 pm
```

Funciones de cadena

- strlen(). Obtiene el número de caracteres de la cadena o, lo que es lo mismo, su longitud.

```
int strlen(string $string);
```

- trim(). Elimina los espacios en blanco al inicio y al final de una cadena. Existen versiones específicas de esta función como ltrim() y rtrim() que eliminan los espacios en blanco al inicio y al final de la cadena, respectivamente. Si se proporciona un segundo argumento pueden eliminarse otro tipo de caracteres distintos a los considerados espacios en blanco.

```
string trim(string $str[, string $character_mask]);
```

Funciones de cadena

- strtolower(). Convierte todos los caracteres de la cadena pasada como argumento a minúsculas.

```
int strtolower(string $string);
```

- strtoupper(). Convierte todos los caracteres de la cadena pasada como argumento a mayúsculas.

```
int strtoupper(string $string);
```

Funciones de cadena

- strrev(). Invierte los caracteres de una cadena. Debe proporcionar una cadena como argumento a la función que retornará la cadena completamente invertida.

```
int strrev(string $string);
```

- strcmp(). Realiza una comparación segura a nivel binario de dos cadenas pasadas como argumentos. La función devolverá un valor menor que 0 si el primer argumento es menor que el segundo, un valor mayor que 0 si resulta mayor o 0 si son iguales.

```
int strcmp(string $str1, string $str2]);
```

Funciones de cadena

- implode(). Une los elementos de una matriz en una sola cadena, pudiendo incluir algún carácter de separación entre cada elemento del array unido. A este carácter se le denomina pegamento y si no se proporciona se utiliza una cadena vacía para unir los elementos en una cadena.

```
string implode(string $glue, array $matriz);
```

- explode(). Divide una cadena en varias subcadenas que son retornadas en una matriz. El primer argumento debe ser el carácter, presente en la cadena que se buscará para particionar la cadena en partes, el segundo argumento debe ser la cadena. Si el primer argumento es una cadena vacía la función devolverá False.

```
array explode(string $delimiter, string $string[, int $limit]);
```

Funciones de cadena

- strpos(). Encuentra la posición numérica (comenzando en 0) de la primera ocurrencia de una subcadena dentro de otra cadena. La idea es encontrar un segmento de cadena dentro de otra cadena más larga, obviamente, y de ser localizada la función devolverá la posición a partir de la cual fue encontrada la coincidencia. Puede especificarse con el tercer argumento que la búsqueda no inicie desde el comienzo de la cadena completa.

```
mixed strpos(string $string, mixed $needle[, int $offset=0]);
```

- Ejemplo:

```
$mystring = 'abc';
$findme   = 'a';
$pos = strpos($mystring, $findme);
if ($pos === false) {
    echo "La cadena '$findme' no fue encontrada en la cadena '$mystring'";
} else {
    echo "La cadena '$findme' fue encontrada en la cadena '$mystring'";
    echo " y existe en la posición $pos";
}
```

Funciones de cadena

- number_format(). Formatea un número de varias cifras con millares agrupados y la parte decimal con el número que se le indique en el tercer argumento. La función puede aceptar uno, dos o los cuatro parámetros, pero nunca tres.

```
number_format(float $number[, int $decimales=0]);
number_format(float $number, int $decimales=0, string $point, string
$separator);
```

Funciones de PHP para el manejo de matrices

- PHP ofrece un amplio conjunto de funciones para trabajar con matrices.
- Las más útiles serán las funciones para imprimir los datos del *array*, para extraer datos, para recorrer por sus elementos, para posicionar el puntero interno al inicio o al final y para obtener el número total de elementos de la matriz.

Comprobar si una variable es matriz

- Puede determinar si una variable es matriz o no, antes de comenzar a trabajar con ella. Para esto se utiliza la función *is_array()*.
- **`is_array($matriz)`**: Comprueba si la variable pasada como argumento es una matriz o no. Devuelve true si la variable del argumento es matriz y false si resulta que no lo es.

```
$arr = array("Kabul", "Balkh", "Herat",
"Qandahar");

if(is_array($arr)){
    echo "YES";
}else{
    echo "NO";
}
> YES
```

Obtener el número de elementos de la matriz

- PHP proporciona dos funciones para obtener el número de elementos de una matriz, ya sea indexada numéricamente o asociativa. Esas funciones son count() y sizeof().
- **`count($matriz)`**: Devuelve el número de elementos que contiene una matriz y en general cualquier tipo de dato compuesto.
- **`sizeof($matriz)`**: Esta función es prácticamente un alias de la función count(), de modo que igualmente devuelve el número total de elementos de la matriz.

```
$animals = array ('dog', 'cat', 'fish');
echo count($animals);
echo sizeof($animals);
```

```
3
3
```

Buscar un elemento dentro de la matriz

- Puede buscar un valor concreto dentro de los elementos de una matriz.
- **`in_array($valor, $matriz[, $strict])`**: Comprueba si un valor está presente dentro de los elementos de una matriz. De acuerdo a la documentación la comparación es flexible por defecto, de modo que si se compara un valor '5' con 5, la función devolverá true, pero si se establece el tercer argumen ión en true, la comparación

```
$arr = array("Kabul", "Balkh", "Herat",
"Qandahar");

if(in_array("Kabul",$arr)){
    echo "Found";
}else{
    echo "Not Found";
}
> Found
```

Comprobar si una clave o índice existe en la matriz

- Así como se puede determinar si un valor concreto está presente en la matriz, también podemos determinar si una clave o índice está presente en la matriz. Para esto se utiliza la función *array_key_exists()*.
- **`array_key_exists($clave, $matriz)`**: Verifica si la clave o índice pasado como primer argumento está presente en la $matriz pasada como segundo argumento.

```
$a = array ('a' => NULL, 'b' => 2);
echo array_key_exists ('a',$a);
```

Funciones de manejo de puntero o cursor interno de la matriz

- **`current($matriz)`**: Devuelve el valor del elemento situado en la posición actual del puntero interno o cursor de la matriz. Devuelve false cuando apunta al final del arreglo.
- **`key($matriz)`**: Devuelve el índice de la posición actual de la matriz pasada como argumento. Este valor será un número en caso de tratarse de un ***array*** indexado numéricamente o una cadena de caracteres si se trata de un ***array*** asociativo.
- **`next()`**: Avanza el puntero o cursor interno de la matriz hacia el siguiente elemento, devolviendo el valor de

Funciones de manejo de puntero o cursor interno de la matriz

```php
$array = array('foo' => 'bar', 'baz',
'bat' => 2);
reset($array);
while (key($array) !== null) {
   echo key($array).": ".current($array);
   next($array);
}
```

Manipulación de matrices utilizando funciones de PHP

- **`reset($matriz)`**: Sitúa el cursor o puntero interno de la matriz en el primer elemento y devuelve el valor de dicho elemento. Si la matriz está vacía, la función devolverá el valor *false*.
- **`prev($matriz)`**: Devuelve el valor del elemento anterior al actual (si existe) y retrocede al puntero interno una posición. En caso de que el elemento actual sea el primero, devuelve *false*.
- **`end($matriz)`**: Coloca el cursor o puntero interno de la matriz en el último elemento de un ***array*** escalar o asociativo. Devuelve el valor almacenado en el último elemento y si la matriz está vacía en su lugar,

Manipulación de matrices utilizando funciones de PHP

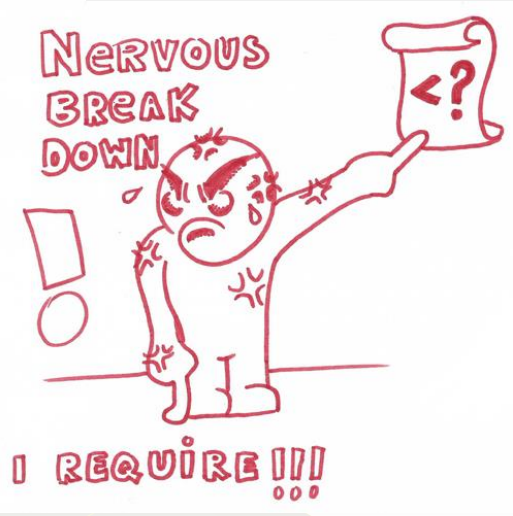
```php
$frutas = array("Manzana", "Fresa", "Uva", "Naranja", "Mandarina");
echo "<ul>\n";
end($frutas);
do{
    $fruta = current($frutas);
    echo "\t<li>\n";
    echo "\t\t" . $fruta . "\n";
    echo "\t</li>\n";
}while(prev($frutas));
echo "</ul>\n";
```

Reutilización de código y modularidad

- A medida vayan creciendo sus scripts PHP en complejidad y número de líneas de código, comenzará a darse cuenta que aunque utilice funciones el proyecto empezará a resultar bastante complejo de manejar.
- Además, notará que muchas de las funciones que desarrolle le pueden resultar útiles para otros proyectos igual o más complejos que el actual, lo cual le vendría bien para ahorrar tiempo de desarrollo.
- La técnica para la reutilización de código es simple, se crea un archivo donde guarda todas las funciones que le puedan resultar útiles y las incluye en los proyectos futuros utilizando una de dos funciones: include() o require().

Funciones include(), require() y sus variantes

- Mediante el uso de las funciones *include()* y *require()* es posible cargar un archivo dentro de la secuencia de comando (*script*) PHP.
- El archivo incluido puede contener todo lo que es válido dentro de una secuencia de comandos de PHP, como variables, instrucciones completas, funciones y clases de PHP. Sin embargo, es más frecuente que sean utilizados archivos que son considerados librerías de funciones o bibliotecas de clases.

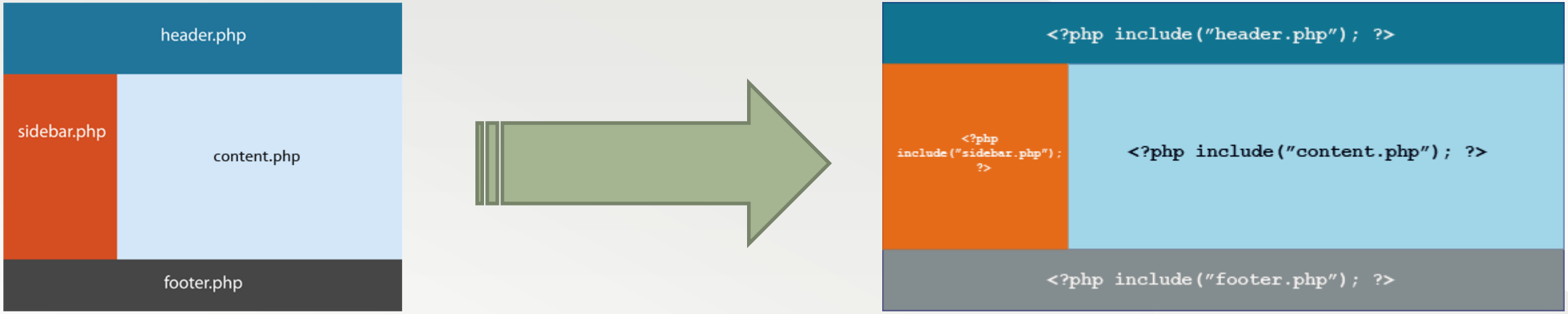


Funciones include(), require() y sus variantes

- Las instrucciones *include()* y *require()* son prácticamente idénticas, lo único que las diferencia es la forma en que procesan los errores; es decir, mientras *include()* devuelve una advertencia (warning) en caso de producirse una falla, *require()* devuelve un error fatal (fatal error).
- Existen dos variantes de estas funciones denominadas *include_once()* y *require_once()*, respectivamente.
- El propósito de estas dos variantes es evitar conflicto si se intenta incluir varias veces el mismo archivo. De modo que al usar include_once() o require_once() se asegura que el archivo se incluya una sola vez.

Utilización de include()

- Puede utilizar *include()* para incluir librerías de funciones, bibliotecas de clase o simplemente partes de secuencias de comando que le permitan modularizar la aplicación.
- Un ejemplo típico de utilización es dividir la página en varias secciones lógicas, por ejemplo en header.php con la cabecera de la página, sibebar.php con el menú de navegación principal, content.php con el área de contenido principal del sitio y footer.php con el pie de la página.



Utilización de require()

- En el caso de *require()*, puede utilizarse de la misma forma que *include()*, la única diferencia será que en caso de producirse una falla obtendrá un error fatal que interrumpirá la ejecución del script. En el caso del include el error sólo emitirá una advertencia y permitirá continuar la ejecución de las instrucciones que siguen a continuación de la que produjo la advertencia (warning).

```
<?php require 'includes/header.php'; ?>
<!-- end #menu -->
        <div id="page">
            <div id="page-bgtop">
                <div id="page-bgbtm">
```

FIN

Lenguajes Interpretados en el Servidor