

Subprograma

Un subprograma es un fragmento de código que tiene una funcionalidad específica. Este permite que el código sea modular y lo podamos reutilizar.

Tipos

Funciones: subprogramas que devuelven un valor como resultado de su ejecución.

Procedimientos: son aquellos que se ejecutan sin devolver ningún tipo de valor.

Función

Sintaxis:

- **PHP:**

```
function nombre($arg1,$arg2,...){  
    instrucciones;  
    return $valorDevuelto;  
}
```

- **ASP:**

```
function nombre (arg1,arg2,...)  
    instrucciones  
    Nombre= valordevuelto  
end function
```

Función

Sintaxis (continuación):

JSP:

```
tipo devuelto nombre (tipo1 arg1,tipo2 arg2,... ){  
    instrucciones;  
    return valorDevuelto  
}
```

Procedimiento

Sintaxis:

PHP:

```
function nombre($arg1,$arg2,...){  
    instrucciones;  
}
```

ASP:

```
sub nombre (arg1,arg2,...)  
    instrucciones  
end sub
```

Procedimiento

Sintaxis:

JSP:

```
void nombre(tipo1 arg1,tipo2 arg2,...){  
    instrucciones;  
}
```

Llamadas a Función/Procedimiento

Sintaxis:

PHP:

nombre (\$arg1,\$arg2,...);

ASP:

nombre (arg1,arg2,...)

Sólo procedimientos:

call nombre (arg1,arg2,...)

JSP:

nombre (arg1,arg2,...);

PHP- Funciones

Una función puede ser definida empleando la siguiente sintaxis:

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

PHP- Funciones

Cualquier código PHP válido puede aparecer dentro de una función, incluso otras funciones y definiciones de clases.

Los nombres de las funciones siguen las mismas reglas que las demás etiquetas de PHP:

- Un nombre de función válido comienza con una letra o guión bajo, seguido de cualquier número de letras, números o guiones bajos.
- En rigor se define por la expresión regular:

`[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*`

Los nombres de las funciones son insensibles a mayúsculas-minúsculas (case-insensitive).

Aunque es recomendable llamar a las funciones tal y como aparecen en sus declaraciones.

PHP - Funciones

Otras características de las funciones:

- No es posible 'desdeclarar' funciones previamente declaradas
- No es posible redeclarar funciones.
- No admite sobrecarga de funciones.

PHP – Argumentos de funciones

- Cualquier información puede ser pasada a las funciones mediante la lista de argumentos.
- **Lista de argumentos:** expresiones delimitadas por comas. Los argumentos son evaluados de izquierda a derecha.
- PHP admite el paso de argumentos por valor (predeterminado), paso por referencia, y valores de argumentos predeterminados (default).
- Las Listas de argumentos de longitud variable también están soportadas.

PHP – Argumentos de funciones

- **Valores predeterminados**

```
function foo( $a, $b =3, $c='Hola') { };
```

- **Llamadas válidas:**

```
foo(1)
```

```
foo("Gato", 5);
```

```
foo(7, "segundo pararámetro, '3er param.');" ;
```

PHP - Argumentos de funciones

- Si usamos el parámetro n-enésimo, también deberemos explicitar todos los anteriores.

```
foo( 1 , , "3er.parám"); // NO VÁLIDO
```

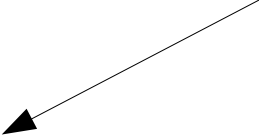
PHP - Paso por referencia

- Por defecto, los argumentos de las funciones son pasados por valor (así, si el valor del parámetro dentro de la función cambia, este no cambia fuera de la función).
- Para permitir a una función modificar sus argumentos, éstos deben pasarse por referencia.
- Para hacer que un argumento de una función sea siempre pasado por referencia hay que anteponer al nombre del argumento el signo 'et' (&) en la definición de la función.

PHP - Paso por referencia

Ejemplo:

Paso por valor



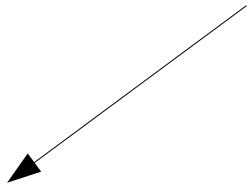
```
<?php
function resetCounter($c) {
    $c = 0;
}
$counter=0;
$counter++;
$counter++;
$counter++;

echo "$counter<br/>";    //Muestra "3"
resetCounter($counter);
echo "$counter<br/>";    //Muestra "3"
?>
```

PHP - Paso por referencia

Ejemplo:

Paso por referencia



```
<?php
function resetCounter(&$c) {
    $c = 0;
}
$counter=0;
$counter++;
$counter++;
$counter++;

echo "$counter<br/>";    //Muestra "3"
resetCounter($counter);
echo "$counter<br/>";    //Muestra "0"
?>
```

PHP - Paso por referencia

Ejemplo:

```
<?php
    $a=3;

    function bar(&$b) {
        $b = 8;
    }

    bar($a);
    echo 'El valor de $a es: ' . $a; // Muestra 'El valor de $a es: 8'
?>
```


PHP - Paso por referencia

Ejemplo:

```
<?php
    $a=3;

    function bar(&$b) {
        $b = 8;
    }

    $aref= & $a; // recibe la referencia de $a
    $aref++;     // $a vale 4

    bar($aref);
    echo 'El valor de $a es: ' . $a; //muestra 'El valor de $a es: 8'
?>
```

PHP - Devolver valores

Una función PHP usa la palabra clave ***return*** para devolver valores.

```
function foo(){  
    // hacer algo aquí  
    return valor;  
}
```

'valor' puede ser cualquier expresión, por ejemplo, valor literal (como 1 o false), un nombre de variable (como \$resultado), o una expresión como ($\$x * 3 / 7$).

PHP - Devolver valores

Cuando el motor PHP encuentra la sentencia ***return***, inmediatamente sale de la función y devuelve valor al código que la invocó.

Incluso puede utilizar la sentencia ***return*** sin incluir un valor a devolver.

```
function foo() {  
    // hacer algo  
    return;  
}
```

PHP - Devolver valores

Devolver valores es opcional. Una misma función puede retornar un valor en cualquier parte del código, o nunca, indistintamente.

```
function foo(){  
    If (expresion) {echo "hola Mundo"; return valor}  
    echo "No devuelvo nada";  
}
```

PHP – Devolver referencias

Al igual que podemos pasar parámetros por referencia, una función también puede devolver una referencia en lugar de un valor.

Para ello situaremos un ampersand ('&') antes del nombre de la función en su definición.

```
function &foo(){  
    . . .  
}  
$mivariable = &foo();
```

Debido a que PHP devuelve automáticamente los objetos por referencia, probablemente sea una opción que no utilicemos con frecuencia. Ver:

<http://php.net/manual/es/language.references.return.php>

PHP - Funciones condicionales

Una función es condicional cuando solo existe si se cumple una determinada condición.

PHP - Funciones condicionales

```
<?php
    $hacer_algo = true;
    /* No podemos llamar a foo() desde aquí, no existe aún, pero podemos llamar a bar() */

    bar();

    if ($hacer_algo) {
        function foo()
        {
            echo "<p>No existo hasta que la ejecución del programa llegue hasta mí.\n";
        }
    }
    /* Ahora podemos llamar de forma segura a foo() ya que $hacer_algo= true */

    if ($hacer_algo) foo();

    function bar()
    {
        echo "Existo desde el momento inmediato que comenzó el programa.\n";
    }
?>
```

PHP- Funciones anidadas

Todas la funciones tienen ámbito global.

Se pueden llamar desde fuera de una función incluso si fueron definidas dentro, y viceversa.

```
<?php

function eXTerNa(){
    echo "<p>Función externa llamada desde una función anidada.";
}

function foo()
{
    function bar() { echo "No existo hasta que se llame a foo().\n"; externa(); }
}

/* No podemos llamar aún a bar() ya que no existe. */

foo();

/* Ahora podemos llamar a bar(), el procesamiento de foo() la ha hecho accesible. */

bar();

?>
```


PHP- Funciones recursivas

PHP admite recursividad, (funciones que se llaman a sí mismas).

Así opera una función recursiva:

- 1)El código llamante llama a la función recursiva.
- 2)Si se encuentra en el caso base (o condición de parada) la función hace cualquier procesamiento requerido, y luego termina.
- 3)De otro modo, la función hace cualquier procesamiento requerido, luego se llama a sí misma para continuar la recursión (caso recursivo).

PHP- Funciones recursivas

Ejemplo de función recursiva:

```
<?php
function factorial ($n){
    if ($n==1) return 1;
    else{
        return ($n * factorial ($n-1));
    }
}
echo "Factorial de 5 es: " . factorial(5);
?>
```

Nota: Las llamadas a funciones/métodos recursivos con más de 100-200 niveles de recursividad pueden agotar la pila y ocasionar la finalización del script en curso. Especialmente, las recursividades infinitas están consideradas un error de programación. Actualmente ya no admite más de 256 niveles (**depende de la versión de PHP**), lo que finalizaría la ejecución del script.

PHP - Funciones variables

- Si un nombre de variable tiene paréntesis anexos a él, PHP buscará una función con el mismo nombre que lo evaluado por la variable, e intentará ejecutarla. Entre otras cosas, esto se puede usar para implementar llamadas de retorno, tablas de funciones, y así sucesivamente.
- Las funciones variables no funcionarán con constructores de lenguaje como **echo**, **print**, **unset()**, **isset()**, **empty()**, **include**, **require** y similares. Utilizaremos funciones de envoltura para hacer uso de cualquiera de estos constructores como funciones variables.

PHP- Funciones de variable

Ejemplo:

```
<?php
function foo() {
    echo "En foo()<br />\n";
}

function bar($arg = "")
{
    echo "En bar(); el argumento era '$arg'.<br />\n";
}

// Esta es una función de envoltura alrededor de echo
function hacerecho($cadena)
{
    echo $cadena;
}

$func = 'foo';
$func();      // Esto llama a foo()

$func = 'bar';
$func('prueba'); // Esto llama a bar()

$func = 'hacerecho';
$func('prueba'); // Esto llama a hacerecho()
?>
```

PHP- Funciones de variable

Ejemplo:

```
<?php
    $trigonometricas= array("sin", "cos", "tan");
    $grados= 30;

    foreach ($trigonometricas as $trig){
        echo "$trig($grados)= " . $trig(deg2rad($grados)) . "<br/>";
    }
?>
```

Funciones PHP ¿Cuántas hay?

Un huevo :-)

```
<?php
    $funcs = get_defined_functions();
    echo count( $funcs ['internal'] );
?>
```

Esto visualizará las funciones del core y extensiones instaladas. Resultado: 1671
Incluyendo todas las demás: unas 5845 en marzo 2015.

PHP- Funciones

No incluidas en este curso:

- Funciones anónimas
- Funciones variádicas.
- Clousures