Unión Europea

Fondo Social Europeo

El FSE invierte en tu futuro

PHP

Uso de Funciones

Pepe

GENERALITAT VALENCIANA

Conselleria d'Educació, Investigació, Cultura i Esport

GOBIERNO DE ESPAÑA

MINISTERIO DE EDUCACIÓN, CULTURA Y DEPORTE

Continguts

1 Pròleg

Al igual que en otros lenguajes de programación (como por ejemplo JavaScript), las funciones en PHP nos van a permitir encapsular un conjunto de instrucciones bajo un nombre, y poderlo ejecutar en bloque cada vez que lo necesitemos, simplemente utilizando el nombre que le hayamos puesto a la función.

2 Definición de funciones

Para definir una función utilizaremos la instrucción `function`. Una función sólo podrá ser llamada desde el script donde fue declarada. En el siguiente ejemplo se define la función `mi_funcion`. Cada vez que la llamemos desde nuestro código se usará **echo** para añadir código HTML a la página devuelta al navegador:

```
function mi_funcion()
{
   echo '<strong>En la función</strong>';
}

...

mi_funcion(); //La llamada
```

Notar que estos dos fragmentos de código pueden ir en bloques PHP diferentes. Por ejemplo, podemos definir las funciones al inicio del código PHP, y luego llamarlas después:

```
<?php

function mi_funcion() {
    echo "Hola, buenas";
}

?>
<!DOCTYPE html>
...
<?php
    mi_funcion();
?>
```

3 Párametros

Los parámetros son el mecanismo por el cual podemos pasarle información a una función, en forma de variables, a la hora de llamarla. Por ejemplo, esta función suma los dos datos (numéricos) que se le pasan como parámetros:

```
function suma($a, $b) {
   echo $a + $b;
}
```

Y para utilizarla, simplemente la llamamos pasándole dos datos numéricos:

```
suma(2, 3);
```

Ejercicio 1:
Crea una página llamada **contador.php**. Crea una función llamada `cuenta($a, $b)` que reciba dos parámetros y vaya contando de un número al otro, separando los números por comas. Después, pruébala en el código PHP haciendo que cuente del 10 al 20.

3.1 Uso de parámetros por referencia

Si queremos pasar un parámetro por referencia, parámetros que sean modificable, en la declaración de la función, indicaremos los parámetros mediante el operador & para indicar la dirección de memoria de la variable.

```
<?php
function duplicarPorValor($argumento) {
    $argumento = $argumento * 2;
    echo "Dentro de la función: $argumento.<br>";
}
function duplicarPorReferencia(&$argumento) {
    $argumento = $argumento * 2;
    echo "Dentro de la función: $argumento.<br>";
}

$numero1 = 5;
echo "Antes de llamar: $numero1.<br>";
duplicarPorValor($numero1);
echo "Después de llamar: $numero1.<br>";
echo "<br>";
//Antes de llamar: 5.
//Dentro de la función: 10.
//Después de llamar: 5.
```

```php
$numero2 = 7;
echo "Antes de llamar: $numero2.<br>";
duplicarPorReferencia($numero2);
echo "Después de llamar: $numero2.<br>";
?>
//Antes de llamar: 7.
//Dentro de la función: 14.
//Después de llamar: 14.
```

Ejercicio 2:
Crea una página llamada `intercambia.php`. Añade dentro una función llamada intercambia que reciba 2 parámetros numéricos por referencia, y lo que haga sea intercambiar sus valores. Es decir, si recibe el parámetro `$a` y el valor de `$b` , y `$b` tome el valor de `$a`.

3.2 Parámetros por defecto / opcionales

Permiten asignar valores en la declaración, y posteriormente, dejar el argumento en blanco.

```php
<?php
function saluda($nombre, $prefijo = "Sr") {
    echo "Hola ".$prefijo." ".$nombre;
}

saluda("Salvador", "Mr");
saluda("Salvador");
saluda("Marina", "Srta");
```

La salida seria:

```
Hola Mr Salvador
Hola Sr Salvador
Hola Srta Marina
```

En el caso de convivir con otro tipo de parámetros, los parámetros que tienen el valor asignado por defecto siempre se colocan al **final**.

3.3 Parámetros variables

Podemos tener funciones donde en la declaración no indiquemos la cantidad de datos de entrada.

- $arrayArgs = `func_get_args();` → Obtiene un array con los parámetros
- $cantidad = `func_num_args();` → Obtiene la cantidad de parámetros recibidos
- $valor = `func_get_arg(numArgumento);` → Obtiene el parámetro que ocupa la posición numArgumento.

Estas funciones no se pueden pasar como parámetro a otra función (como funciones variable, que veremos más adelante). Para ello, debemos guardar previamente la función en una variable.

```php
<?php
function sumaParametros() {
    if (func_num_args() == 0) {
        return false;
    } else {
        $suma = 0;

        for ($i = 0; $i < func_num_args(); $i++) {
            $suma += func_get_arg($i);
        }

        return $suma;
    }
}

echo sumaParametros(1, 5, 9); // 15
?>
```

Desde PHP 5.6, se puede utilizar el operador **... (variadics)** el cual "disfraza" los parámetros como un array:

```php
function sumaParametrosMejor(...$numeros) {
    if (count($numeros) == 0) {
        return false;
    } else {
        $suma = 0;

        foreach ($numeros as $num) {
            $suma += $num;
        }

        return $suma;
    }
}

echo sumaParametrosMejor(1, 5, 9); // 15
```

3.4 Type hinting

A pesar de que PHP es un lenguaje débilmente tipado (es decir, no tenemos que especificar de qué tipo de dato son las variables y parámetros que utilizamos), sí que permite indicar el tipo de dato en los parámetros y tipo de retorno de las funciones, si lo queremos, para "obligar" a que los datos que se pasen y recojan sean de esos tipos.

```
// Comprueba si dos números son iguales
function iguales(int $param1, int $param2): boolean
{
    return $param1 === $param2;
}
```

3.5 Funciones anónimas

Igual que ocurre con otros lenguajes, como **JavaScript**, podemos definir funciones anónimas. Es decir, funciones sin nombre, que pueden asignarse a una variable, o utilizarse en un punto determinado del programa para invocarlas directamente.

```
<?php
$anonima = function() {
    echo "Hola";
};
$anonima();

$anonimaConParametro = function($nombre) {
    echo "Hola ".$nombre;
};
$anonimaConParametro("Salvador");

?>
```

Tenéis más información sobre funciones anónimas y flecha en el siguiente artículo (en inglés): Funciones anónimas y flecha en PHP

4 Biblioteca de funciones o Modularizando el código

Podemos agrupar un conjunto de funciones en un archivo, para permitir su reutilización. Posteriormente, se incluye con:

include(archivo); / include_once(archivo);

require(archivo); / require_once(archivo);

- Si **no** encuentra el archivo, **require** lanza un error fatal, **include** lo ignora.
- Las funciones **once** sólo se cargan una vez, si ya ha sido incluida previamente, **no** lo vuelve a hacer, evitando bucles.

Por ejemplo, colocamos las funciones en el archivo biblioteca.php:

```
<?php
function suma(int $a, int $b) : int {
    return $a + $b;
}

function resta(int $a, int $b) : int {
    return $a - $b;
}
?>
```

Y posteriormente en otro archivo:

```
<?php
    include_once("biblioteca.php");
    echo suma(10,20);
    echo resta(40,20);
?>
```

4.0.1 Plantillas mediante `include`

Mediante el uso de la instrucción include también podemos separar fragmentos de código PHP/HTML que queramos reutilizar en nuestros sitios web y crear un sistema muy sencillo de plantillas. Por ejemplo, vamos a separar una página en tres partes, primero la parte superior en encabezado.php:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title><?= $titulo ?></title>
</head>
<body>
```

La parte de abajo, por ejemplo, solo va a contener HTML y la colocamos en pie.html:

```
<footer>IES Salvador Gadea. DWES</footer>
</body>
</html>
```

Y luego nos centramos únicamente en el contenido que cambia en pagina.php:

```php
<?php
$titulo = "Página con includes";
include("encabezado.php");
?>
<h1><?= $titulo ?></h1>
<?php
include("pie.html");
?>
```

4.1 Separar la lógica de la presentación

Cuando estamos programando, siempre tenemos que hacer que cada fichero, clase, función … se encargue de una sola cosa. La más clara es separamos la lógica del programa (en nuestro caso en *php*) de la presentación al usuario (majoritariament con código *HTML*). Para poder separar estas dos funciones hacen falta sentencias anteriores para incluir ficheros.

```php
fruites.php
<?php

$color = 'verde';
$fruta = 'manzana';
include('fruites.view.php')

?>
```

```php
fruites.view.php

<html>
<head>
<title>Fruites</title>
</head>
<body>
    <h3>
        <?php"Una $fruta $color" ?>
    </h3>
</body>
</html>
```

5 Funciones predefinidas

El lenguaje ofrece un abanico de funciones ya definidas, agrupadas por su funcionalidad: https://www.php.net/manual/es/funcref.php

5.1 Strings

Todas las funciones se pueden consultar en https://www.php.net/manual/es/ref.strings.php

Algunas importantes son:

- `strlen`: obtiene la longitud de una cadena y devuelve un número entero
- `substr`: devuelve una subcadena de la cadena original
- `str_replace`: reemplaza caracteres en una cadena
- `strtolower` y `strtoupper`: Transforman una cadena de caracteres en la misma cadena en minúsculas o mayúsculas respectivamente.

```php
<?php
    $cadena = "El caballero oscuro";
    $tam = strlen($cadena);
    echo "La longitud de '$cadena' es: $tam <br />";

    $oscuro = substr($cadena, 13); // desde 13 al final
    $caba = substr($cadena, 3, 4); // desde 3, 4 letras
    $katman = str_replace("c", "k", $cadena);
    echo "$oscuro $caba ahora es $katman<br />";

    echo "Grande ".strtoupper($cadena);
?>
```

Si queremos trabajar con caracteres ASCII de forma individual, son útiles las funciones:

- `chr`: obtiene el carácter a partir de un ASCII
- `ord`: obtiene el ASCII de un carácter

```php
<?php
    function despues(string $letra): string {
        $asciiLetra = ord($letra);
        return chr($asciiLetra + 1);
    }

    echo despues("B");
?>
```

Si queremos **limpiar cadenas**, tenemos las funciones:

- `trim`: elimina los espacios al principio y al final
- `ltrim` / `rtrim` o `chop`: Elimina los espacios iniciales / finales de una cadena.
- `str_pad`: rellena la cadenas hasta una longitud especificada y con el carácter o caracteres especificados.

```php
<?php
    $cadena = " Programando en PHP ";
    $limpia = trim($cadena); // "Programando en PHP"

    $sucia = str_pad($limpia, 23, "."); // "Programando en PHP....."
?>
```

5.2 Comparando y buscando

La comparación de cadenas puede ser con conversión de tipos mediante == o estricta con ===. También funcionan los operadores < y > si ambas son cadenas. Al comparar cadenas con valores numericos podemos utilizar:

- `strcmp`: 0 iguales, <0 si a < b ó >0 si a>b
- `strcasecmp`: las pasa a minúsculas y compara
- `strncmp` / `strncasecmp`: compara los N primeros caracteres
- `strnatcmp`: comparaciones naturales

```php
<?php
    $frase1 = "Alfa";
    $frase2 = "Alfa";
    $frase3 = "Beta";
    $frase4 = "Alfa5";
    $frase5 = "Alfa10";

    var_dump( $frase1 == $frase2 ); // true
    var_dump( $frase1 === $frase2 ); // true
    var_dump( strcmp($frase1, $frase2) ); // 0
    var_dump( strncmp($frase1, $frase5, 3) ); // 0
    var_dump( $frase2 < $frase3 ); // true
    var_dump( strcmp($frase2, $frase3) ); // -1
    var_dump( $frase4 < $frase5 ); // false
    var_dump( strcmp($frase4, $frase5) ); // 4 => f4 > f5
    var_dump( strnatcmp($frase4, $frase5) ); // -1 => f4 < f5
?>
```

Si lo que queremos es buscar **dentro de una cadena**, tenemos:

- `strpos` / `strrpos`: busca en una cadena y devuelve la posición de la primera/última ocurrencia.
- `strstr` / `strchr (alias)`: busca una cadena y devuelve la subcadena a partir de donde la ha encontrado
- `stristr`: ignora las mayúsculas

```php
<?php
    $frase = "Quien busca encuentra, eso dicen, a veces";
    $pos1 = strpos($frase, ","); // encuentra la primera coma
    $pos2 = strrpos($frase, ","); // encuentra la última coma
    $trasComa = strstr($frase, ","); // ", eso dicen, a veces"
?>
```

Si queremos averiguar **que contiene las cadenas**, tenemos un conjunto de funciones de comprobaciones de tipo, se conocen como las funciones ctype que devuelven un booleano:

- `ctype_alpha` → letras
- `ctype_alnum` → alfanuméricos
- `ctype_digit` → dígitos
- `ctype_punct` → caracteres de puntuación, sin espacios
- `ctype_space` → son espacios, tabulador, salto de línea

```php
<?php
    $prueba1 = "hola";
    $prueba2 = "hola33";
    $prueba3 = "33";
    $prueba4 = ",.()[]";
    $prueba5 = " ,.()[]";

    echo ctype_alpha($prueba1)."<br>"; // true
    echo ctype_alnum($prueba2)."<br>"; // true
    echo ctype_digit($prueba3)."<br>"; // true
    echo ctype_punct($prueba4)."<br>"; // true
    echo ctype_space($prueba5)."<br>"; // false
    echo ctype_space($prueba5[0])."<br>"; // true
?>
```

5.3 Trabajando con subcadenas

Si queremos romper las cadenas en trozos, tenemos:

- `explode(separador,cadena)`: convierte en array la cadena mediante un separador.
- `implode(separador,array)` / `join`: pasa un array a cadena con un separador
- `str_split(cadena,x)` / `chunk_split`: pasa una cadena a una array/cadena cada X caracteres

```php
<?php
    $frase = "Quien busca encuentra, eso dicen, a veces";
    $partes = explode(",", $frase);
    /*
    $partes será un array con 3 elementos:
```

```
        [0]:Quien busca encuentra
        [1]: eso dicen
        [2]: a veces
    */
    $ciudades = ["Elche", "Aspe", "Alicante"];
    $cadenaCiudades = implode(">", $ciudades);

    $partes3cadena = chunk_split($frase, 3);
    // Qui
    // en
    // bus
    // ca
    // ...
    $partes3array = str_split($frase, 3);
    // ["Qui", "en ", "bus", "ca ", "enc",..]
?>
```

- `substr_count`: número de veces que aparece la subcadena dentro de la cadena
- `substr_replace`: reemplaza parte de la cadena a partir de su posición, y opcionalmente, longitud

```
<?php
    $batman = "Bruce Wayne es Batman";
    $empresa = substr($batman, 6, 5); // Wayne
    $bes = substr_count($batman, "B"); // 2
    // Bruce Wayne es camarero
    $camarero1 = substr_replace($batman, "camarero", 15);
    $camarero2 = substr_replace($batman, "camarero", -6); // quita 6 desde
        el final
    // Bruno es Batman
    $bruno = substr_replace($batman, "Bruno", 0, 11);
?>
```

5.4 Funciones para manejo de fechas y horas

Existen algunas funciones que pueden sernos muy útiles para manejar fechas y horas, y poderlas procesar o almacenar correctamente en bases de datos.

- `time()` : nos da el nº de segundos que han transcurrido desde el 1/1/1970. Esto nos servirá para establecer marcas temporales (timestamps), o para convertir después esta marca temporal en una fecha y hora concretas.
- `checkdate(mes, dia, año)` comprueba si la fecha formada por el mes, día y año que recibe como parámetros es correcta o no (da un valor de verdadero o falso).
- `date(formato, fecha)` obtiene una cadena de texto formateando la fecha con el formato indicado. Si no se indica ninguna fecha, se le aplicará el formato indicado a la fecha actual.

Dentro del formato, podemos usar diferentes patrones, dependiendo del tipo de formato que queramos. Usaremos los símbolos d/D (para día numérico o con letra), m/M (para mes numérico o abreviado), y/Y (año de dos o cuatro dígitos), h/H (hora de 12 o 24 horas), i (minutos), s(segundos), y otras variantes que se pueden consultar en https://www.php.net/manual/es/function.date.php

```
$fechaActual = time();
$textoFecha = date("d/m/Y H:i:s");
// Suponiendo que $fechaActual sea, por ejemplo, el 3 de noviembre de 2014
// a las 18:23:55, entonces $textoFecha sería '03/11/2014 18:23:55'
$esCorrecta = checkdate(15, 11, 2014);
// La variable $esCorrecta sería FALSE, porque 15 no es un mes válido
```

- `strtotime(texto)` convierte un texto que intenta representar una fecha en una fecha determinada. La fecha se representa en formato mes/día/año o mes-día-año, normalmente. Esta fecha sería incorrecta porque no existe el mes 21:

```
$fecha = strtotime("21/12/2013");
```

6 Funciones de Fichero

A veces nos puede resultar útil leer un fichero de texto o escribir información en él. Existen multitud de funciones en PHP para abrir un fichero, leerlo línea a línea, o leer o guardar un conjunto de bytes… Vamos a ver aquí sólo algunas de las funciones más útiles para manejo de ficheros.

- `readfile(fichero)` lee un fichero entero y lo vuelca al buffer de salida (es decir, a la página que se está generando, si estamos en una página PHP).
- `file(fichero)` lee un fichero entero y devuelve un array o lista, donde en cada posición hay una línea del fichero
- `file_get_contents(fichero)` lee un fichero entero y lo devuelve en una cadena (no en un array, como la anterior)
- `file_put_contents(fichero, texto)` escribe la cadena de texto en el fichero indicado, sobreescribiendo su anterior contenido si lo tenía. En el caso de que no queramos sobreescribir, sino añadir, pondremos un tercer parámetro con la constante FILE_APPEND.
- `file_exists(fichero)` devuelve TRUE o FALSE dependiendo de si el fichero indicado existe o no.
- `filesize(fichero)` devuelve el tamaño en bytes del fichero, o FALSE si no existe

```
<?php
$fichero = "libro.txt";
if (file_exists($fichero))
{
```

```
    $contenido = file_get_contents($fichero);
    $contenido .= "Fin"; //Añadimos la palabra FIN.
    file_put_contents($fichero, $contenido);
}
?>
```

La funcion `fopen()` desde PHP podemos abrir archivos que se encuentren en nuestros servidor o una URL.

A esta función hay que pasarle 2 parámetros; el nombre del archivo que queremos abrir y el modo en el que se abrirá

```
$fp = fopen("miarchivo.txt", "r");
```

Muchas veces no podemos abrir el archivo porque éste no se encuentra o no tenemos acceso a él, por eso es recomendable comprobar que podemos hacerlo

```
if (!$fp = fopen("miarchivo.txt", "r")){
    echo "No se ha podido abrir el archivo";
}
```

Modos de apertura de ficheros

- `r`: Modo lectura. Puntero al principio del archivo.
- `r+`: Apertura para lectura y escritura. Puntero al principio del archivo
- `w`: Apertura para escritura. Puntero al principio del archivo y lo sobreescribe. Si no existe se intenta crear.
- `w+`: Apertura para lectura y escritura. Puntero al principio del archivo y lo sobreescribe. Si no existe se intenta crear.
- `a`: Apertura para escritura. Puntero al final del archivo. Si no existe se intenta crear.
- `a+`: Apertura para lectura y escritura. Puntero al final del archivo. Si no existe se intenta crear.
- `x`: Creación y apertura para sólo escritura. Puntero al principio del archivo. Si el archivo ya existe dará error E_WARNING. Si no existe se intenta crear.
- `x+`: Creación y apertura para lectura y escritura. Mismo comportamiento que x.
- `c`: Apertura para escritura. Si no existe se crea. Si existe no se sobreescribe ni da ningún error. Puntero al principio del archivo.
- `c+`: Apertura para lectura y escritura. Mismo comportamiento que C.
- `b`: Cuando se trabaja con archivos binarios como jpg, pdf, png y demás. Se suele colocar al final del modo, es decir rb, r+b, x+b, wb...

7 Ejercicios

7.0.1 parametrosVariables.php

Crea las siguientes funciones:

Una función que devuelva el mayor de todos los números recibidos como parámetro variables:

function mayor(): int. Utiliza las funciones *func_get_args()*, *etc*… **No puedes usar la función max()**.

7.0.2 comprueba_hora.php

Crea una variable de texto con una hora en ella (por ejemplo, *"21:30:12"*), y luego procésala para extraer por separado la hora, el minuto y el segundo, y comprobar si es una hora válida. Por ejemplo, la hora anterior sí debería ser válida, pero si ponemos *"12:63:11"* no debería serlo, porque 63 no es un minuto válido.

7.0.3 matematicas.php:

Añade las siguientes funciones:

- `digitos(int $num): int` → devuelve la cantidad de dígitos de un número.
- `digitoN(int $num, int $pos): int` → devuelve el dígito que ocupa, empezando por la izquierda, la posición $pos.
- `quitaPorDetras(int $num, int $cant): int` → le quita por detrás (derecha) $cant dígitos.
- `quitaPorDelante(int $num, int $cant): int` → le quita por delante (izquierda) $cant dígitos.

7.0.4 login.php

Vamos a simular un formulario de acceso:

`login.php`: el formulario de entrada, que solicita el usuario y contraseña. `compruebaLogin.php`: recibe los datos y comprueba si son correctos (los usuarios se guardan en un array asociativo) pasando el control mediante el uso de include a:

ok.php: El usuario introducido es correcto

ko.php: El usuario es incorrecto. Informar si ambos están mal o solo la contraseña. Volver a mostrar el formulario de acceso.

7.1 Funciones Predefinidas

7.1.1 fraseImpares.php:

Lee una frase y devuelve una nueva con solo los caracteres de las posiciones impares.

7.1.2 analizador.php:

A partir de una frase con palabras sólo separadas por espacios, devolver:

- Letras totales y cantidad de palabras
- Una línea por cada palabra indicando su tamaño

`Nota`: no se puede usar str_word_count

7.1.3 analizadorWC.php:

Investiga que hace la función str_word_count, y vuelve a hacer el ejercicio.

7.1.4 cani.php:

`EsCrIbE uNa FuNcIóN qUe TrAnSfOrMe UnA cAdEnA eN cAnI.`

7.1.5 palindromo.php:

Escribe una función que devuelva un booleano indicando si una palabra es palíndroma (se lee igual de izquierda a derecha que de derecha a izquierda, por ejemplo, `"ligar es ser agil"`).

7.1.6 CasasRuralesTelefonos.php

Crea un programa llamado `CasasRuralesTelefonos.php` que cargue los datos de este archivo CSV de casas rurales de la provincia de Castellón. Queremos quedarnos con el *id*, *localidad*, *nombre* y *telefono* de las casas rurales que tengan un *teléfono definido*, descartando el resto. El programa debe mostrar por pantalla el listado final procesado, y cuántas casas rurales *se han descartado* por tener datos nulos.