FUNCIONES PARA CADENAS PHP

Lo que viene entre corchetes significa que es opcional, no es necesario ponerlo al llamar a las funciones.

Cuando una variable se pase como referencia (&) significa que se modifica su valor en el transcurso de la función.

Se puede acceder a las letras de un string como si fueran posiciones de un array.

La cadena PHP_EOL equivale al salto de línea.

int **strcmp**(string $str1, string $str2)	Devuelve 0 si son iguales, < 0 si el primer string es menor que el segundo, y > 0 si es mayor. La función **strcasecmp** es igual pero no sensible a mayúsculas y minúsculas.
int **strnatcmp**(string $str1, string $str2)	Compara igualmente dos strings pero siguiendo el orden natural, "10" es mayor que "2".
int **strlen**(string $str)	Devuelve la longitud de una cadena
string **substr**(string $str, int $start [, int $length])	Devuelve un nuevo string empezando en la posición start, y desde ahí tantas letras como diga length
array **explode**(string $delimeter, string $string [, int $limit])	Divide un string en un array cada elemento obtenido separando la cadena por un delimitador. Se puede decir cuántos elementos generar, un límite.
string **implode**([string $delimeter,] array $pieces)	Genera un string con los elementos de un array y los separa con un delimitador opcional.
array **str_split**(string $string [, int $split_length = 1])	Divide un string en un array, cada elemento es una cadena de longitud split_length
string **trim**(string $str)	Elimina espacios en blancos u otros caracteres del principio y final de la cadena. **ltrim** del principio, **rtrim** del final.
mixed **count_chars**(string $string)	Devuelve un array con los caracteres usados en un string, indicando para cada uno el número de apariciones.
mixed **str_word_count**(string $string [, int $format = 0 [, string $charlist]])	Cuenta las palabras de un string. Dependiendo del valor pasado en format tenemos si: 0 devuelve el número de palabras encontradas, 1 devuelve un array con las palabras encontradas, 2 devuelve array asociativo con cada palabra y la clave es la posición en el array (pero en caracteres).
int **substr_count**(string $haystack, string $needle [, int $offset = 0 [, int $length]])	Cuenta el número de veces que aparece la subcadena needle en el array haystack. Se puede pasar una zona donde busque, offset y length.

string **strstr**(string $haystack, mixed $needle)	Devuelve la parte del string haystack desde la primera aparición de needle hasta el final. **stristr** es igual pero no sensible a mayúsculas y minúsculas.
mixed **strpos**(string $haystack, mixed $needle)	Devuelve la posición de la primera ocurrencia de la cadena needle en la cadena haystack. **stripos** no tiene en cuenta mayúsculas y min.
string **lcfirst**(string $str)	Devuelve un string con el primer carácter de $str en minúscula.
string **ucfirst**(string $str)	Devuelve un string con el primer carácter de $str en mayúscula.
string **ucwords**(string $str [, string $delimeters = " \t\r\n\f\v"])	Convierte a mayúsculas el primer carácter de cada palabra de una cadena *$str.* Los delimitadores se usan para indicar con qué separamos palabras, por defecto, todos esos.
string **strtolower**(string $string)	Devuelve una cadena con sus caracteres en minúsculas. **strtoupper** igual pero a mayúsculas.
string **str_repeat**(string $input, int $multiplier)	Repite un string tantas veces como indique.
string **strip_tags**(string $str [, string $allowable_tags])	Retira los caracteres null del string, y las etiquetas HTML. Opcional le pasamos las etiquetas que no queremos que quite.
mixed **str_replace**(mixed $search, mixed $replace, mixed $subject [, int $contador])	Devuelve un string o un array con todas las apariciones de *$search* en $subject reemplazadas con el valor dado de $replace. En $contador muestra las veces que sustituye.
int **printf**(string $format [, mixed $args [, mixed $...]])	Imprime una cadena con formato.
string **sprintf**(string $format [, mixed $args [, mixed $...]])	Devuelve un string formateado pero no lo imprime.
string **vsprintf**(string $format, array $args)	Igual que la anterior pero los argumentos a imprimir se le pasan en un array.
int **vprintf**(string $format, array $args)	Igual que la anterior, pero no lo devuelve, lo imprime.
int **fprintf**(resource $handle, string $format [, mixed $args [, mixed $...]])	Escribe un string formateado pero dentro de un recurso: archivo, ...
mixed **fscanf**(resource $handle, string $format [, mixed &$...])	Coge el input de un archivo *$handle* y lo interpreta en función del formato *$format*.

FORMATO DE CADENAS CON FUNCIONES PRINTF

El **string de formateo** está compuesto de cero o más **caracteres** (excluyendo %), que se copian al resultado, junto con **especificaciones de conversión** que permiten concretar la salida de los argumentos. Esto se aplica para **sprintf()** y **printf()**.

Cada especificación de conversión se constituye por un **signo de porcentaje %** seguido de un elemento (los 5 primeros puntos son opcionales):

1. Especificador de **signo** que fuerza a usar un **signo en un número** (+ ó -).
2. Especificador de **relleno** que indica que **carácter** se utiliza para **rellenar el resultado** hasta el tamaño justo del string, con ceros o con espacios. El valor por defecto es rellenar con espacios, si se pone un cero, se rellenará con ceros. Un carácter de relleno alternativo se puede especificar prefijándolo con una comilla simple (').
3. Especificador de **alineación** que indica si el resultado ha de alinearse a la izquierda o derecha. Por defecto es a la derecha, si incluye (-) se alineará a la izquierda.
4. Especificación de un **número** opcional de **ancho mínimo de caracteres** como resultado de la conversión.
5. Especificación de precisión en forma de **punto (.)** seguido de un **número** que indica cuántos dígitos decimales deben mostrarse para los números *float*. Si se utiliza con un string, actúa como punto de corte, estableciendo un **número máximo de caracteres el string**.
6. **Especificador de tipo** (obligatorio) que indica el tipo con el que han de ser tratados los datos del argumento:

Tipo	**Valor**
%	**% literal**, no requiere argumento
b	Argumento de tipo integer y presentado como **número binario**
c	Argumento de tipo integer y presentado como el **carácter** con ese **valor ASCII**
d	Argumento de tipo integer y presentado como un **número decimal**
e	Argumento tratado con **notación científica** (ej: 1.2e+2). El especificador de precisión indica el número de dígitos después del punto decimal
E	Como **%e** pero utiliza la letra mayúscula (ej: 1.2E+2)
f	Argumento tratado como valor de tipo float y presentado como un **float** (considerando la configuración regional)
F	Lo mismo que el anterior pero sin considerar la configuración regional
o	Argumento tratado como valor de tipo integer y presentado como un **número octal**
s	Argumento tratado y presentado como un **string**
u	Argumento tratado como valor integer y presentado como **número decimal sin signo**
x	Argumento tratado como valor de tipo integer y presentado como **número hexadecimal** con letras en minúscula
X	Lo mismo que x pero con las letras en mayúscula

EJEMPLOS

//Números

```
$a =  19905;
$b = -19905;
$c = 68; // ASCII 68 es 'D'
echo "Binario: " . sprintf("%%b = '%b'\n", $a) . "<br>"; // Binario:%b = '100110111000001'
echo "ASCII: " . sprintf("%%c = '%c'\n", $c) . "<br>"; // ASCII:%c = 'D'
echo "Integer: " . sprintf("%%d = '%d'\n", $a) . "<br>"; // Integer:%d = '19905'
echo "Notación científica: " . sprintf("%%e = '%e'\n", $a) . "<br>"; // Notación científica: %e = '1.990500e+4'
echo "Integer positivo: " . sprintf("%%u = '%u'\n", $a) . "<br>"; // Integer positivo: %u = '19905'
echo "Integer negativo: " . sprintf("%%u = '%u'\n", $b) . "<br>"; // Integer negativo: %u = '18446744073709531711'
echo "Float: " . sprintf("%%f = '%f'\n", $a) . "<br>"; // Float: %f = '19905.000000'
echo "Octal: " . sprintf("%%o = '%o'\n", $a) . "<br>"; // Octal: %o = '46701'
echo "Cadena: " . sprintf("%%s = '%s'\n", $a) . "<br>"; // Cadena: %s = '19905'
echo "Hexadecimal(minúsculas): " . sprintf("%%x = '%x'\n", $a) . "<br>"; // Hexadecimal(minúsculas): %x = '4dc1'
echo "Signo sobre entero negativo: " . sprintf("%%+d = '%+d'\n", $b);// Signo sobre entero negativo: %+d = '-19905'
```

//Cadenas

```
$a = 'moto';
$b = 'muchas motos';
printf("[%s]\n",      $a); // salida estándar de string
printf("[%10s]\n",    $a); // justificación a la derecha con espacios
printf("[%-10s]\n",   $a); // justificación a la izquierda con espacios
printf("[%010s]\n",   $a); // rellenado con ceros también funciona con strings
printf("[%'#10s]\n",  $a); // utiliza el carácter de relleno personalizado '#'
printf("[%.10s]\n", $b); // corte a los 10 caracteres
```

//Ficheros

Ejemplo leer archivo:

```
$handle = fopen("usuarios.txt", "r");
while ($userinfo = fscanf($handle, "%s\t%s\t%s\n"))
{
   list ($name, $profession, $countrycode) =
$userinfo;
   //... hacer algo con los valores
}
fclose($handle);
```

Ejemplo escribir archivo:

```
if (!($archivo = fopen('hola.log', 'w')))
   return;
$dia = 28;
$mes = 9;
$año = 2015;
$texto = "Texto ejemplo a fecha %02d/%02d/%4d";
fprintf($archivo, $texto, $dia, $mes, $año);
```