sscanf »
« soundex

- Manual de PHP
- Referencia de funciones
- Procesamiento de texto
- Strings
- Funciones de strings

Change language: Spanish

Submit a Pull Request Report a Bug

# sprintf

(PHP 4, PHP 5, PHP 7, PHP 8)

sprintf — Devuelve un string formateado

## Descripción ¶

**sprintf**(string `$format`, mixed `$args` = ?, mixed `$...` = ?): string

Devuelve un string producido según el string de formateo dado por `format`.

## Parámetros ¶

`format`

El string de formateo está compuesto de cero o más directivas: caracteres ordinarios (excluyendo `%`) que son copiados directamente al resultado, y *especificaciones de conversión*, donde cada una de las cuales da lugar a extraer su propio parámetro. Esto se aplica tanto para **sprintf()** como para printf().

Cada especificación de conversión consiste en un signo de porcentaje (`%`), seguido por uno o más de estos elementos, en orden:

1. Un *especificador de signo* opcional que fuerza a usar un signo (- o +) en un número. Por defecto, sólo el signo - se utiliza en un número si es negativo. Esta especificación fuerza números positivos para que también tengan adjunto el signo + (agregado en PHP 4.3.0).
2. Un *especificador de relleno* opcional que indica qué carácter se utiliza para rellenar el resultado hasta el tamaño justo del string. Este puede ser un carácter de espacio o un `0` (el carácter cero). El valor por defecto es rellenar con espacios. Un carácter de relleno alternativo se puede especificar prefijándolo con una comilla simple (`'`). Ver los ejemplos más adelante.
3. Un *especificador de alineación* opcional que indica si el resultado debe ser alineado a la izquierda o a la derecha. El valor por defecto es justificado a la derecha, un carácter - lo justificará a la izquierda.
4. Un número opcional, un *especificador de ancho* que indica de cuántos caracteres (mínimo) resultará esta conversión.
5. Un *especificador de precisión* opcional en la forma de un punto (`.`) seguido de un string opcional de dígitos decimales que indica cuántos dígitos decimales deben mostrarse para los números de punto flotante. Cuando se utiliza este especificador con un string, actúa como un punto de corte, estableciendo un límite máximo de caracteres al string. Además, el carácter para empleado cuando se rellena un número podría especificarse opcionalmente entre el punto y el dígito.

6. Un *especificador de tipo* que indica con qué tipo deben ser tratados los datos del argumento. Los tipos posibles son:

- `%` - un carácter de porcentaje literal. No se requiere argumento.
- `b` - el argumento es tratado como un valor de tipo integer y presentado como un número binario.
- `c` - el argumento es tratado como un valor de tipo integer y presentado como el carácter con ese valor ASCII.
- `d` - el argumento es tratado como un valor de tipo integer y presentado como un número decimal (con signo).
- `e` - el argumento es tratado con notación científica (e.g. 1.2e+2). El especificador de precisión indica el número de dígitos después del punto decimal a partir de PHP 5.2.1. En versiones anteriores, se tomó como el número de dígitos significativos (menos uno).
- `E` - como `%e` pero utiliza la letra mayúscula (e.g. 1.2E+2).
- `f` - el argumento es tratado como un valor de tipo float y presentado como un número de punto flotante (considerando la configuración regional).
- `F` - el argumento es tratado como un valor de tipo float y presentado como un número de punto flotante (no considerando la configuración regional). Disponible desde PHP 4.3.10 y PHP 5.0.3.
- `g` - lo mismo que `%e` y `%f`.
- `G` - lo mismo que `%E` y `%f`.
- `o` - el argumento es tratado como un valor de tipo integer y presentado como un número octal.
- `s` - el argumento es tratado y presentado como un string.
- `u` - el argumento es tratado como un valor de tipo integer y presentado como un número decimal sin signo.
- `x` - el argumento es tratado como un valor de tipo integer y presentado como un número hexadecimal (con las letras en minúsculas).
- `X` - el argumento es tratado como un valor de tipo integer y presentado como un número hexadecimal (con las letras en mayúsculas).

Las variables serán forzadas por el especificador a un tipo adecuado:

**Manejo de tipos**

| Tipo | Especificador |
|---|---|
| string | s |
| integer | d, u, c, o, x, X, b |
| double | g, G, e, E, f, F |

**Advertencia**

Intentar usar una combinación de especificadores de string y ancho con conjuntos de caracteres que requieran más de un byte por carácter podría tener resultados inesperados.

El string de formato soporta la numeración/intercambio de argumentos. Aquí está un ejemplo:

**Ejemplo #1 Intercambio de argumentos**

```
<?php
$num = 5;
$ubicación = 'árbol';

$formato = 'Hay %d monos en el %s';
echo sprintf($formato, $num, $ubicación);
?>
```

Esto producirá " Hay 5 monos en el árbol". Pero imaginemos que estamos creando un string de formato en un fichero aparte, generalmente por que nos gustaría internacionalizarlo, reescribiéndolo así:

**Ejemplo #2 Intercambio de argumentos**

```php
<?php
$formato = 'El %s contiene %d monos';
echo sprintf($formato, $num, $ubicación);
?>
```

Ahora tenemos un problema. El orden de los marcadores de posición en el string de formato no coincide con el orden de los argumentos en el código. Nos gustaría dejar el código tal cual y simplemente indicar en el string de formato a cuáles argumentos de los marcadores de posición se refieren. Tendríamos que escribir el string de formato de esta forma:

**Ejemplo #3 Intercambio de argumentos**

```php
<?php
$formato = 'El %2$s contiene %1$d monos';
echo sprintf($formato, $num, $ubicación);
?>
```

Un beneficio adicional es que se pueden repetir los marcadores de posición sin agregar más argumentos en el código. Por ejemplo:

**Ejemplo #4 Intercambio de argumentos**

```php
<?php
$formato = 'El %2$s contiene %1$d monos.
            Es un bonito %2$s con %1$d monos.';
echo sprintf($formato, $num, $ubicación);
?>
```

Cuando se utiliza el intercambio de argumentos, el *especificador de posición* `n$` debe ir inmediatamente después del signo de porcentaje (`%`), antes de cualquier otro especificador, tal como se muestra en el ejemplo siguiente.

**Ejemplo #5 Especificar un carácter de relleno**

```php
<?php
echo sprintf("%'.9d\n", 123);
echo sprintf("%'.09d\n", 123);
?>
```

El resultado del ejemplo sería:

```
......123
000000123
```

**Ejemplo #6 Especificador de posición con otros especificadores**

```php
<?php
$formato = 'El %2$s contiene %1$04d monos';
echo sprintf($formato, $num, $ubicación);
?>
```

El resultado del ejemplo sería:

```
El árbol contiene 0005 monos
```

> **Nota**:
>
> Tratar de utilizar marcadores de posición mayores que **`PHP_INT_MAX`** provocará que **sprintf()** genere mensajes de advertencia.

**Advertencia**

El especificador de tipo `c` ignora el relleno y el ancho

```
args
...
```

## Valores devueltos ¶

Devuelve un string producido de acuerdo con el string de formato `format`.

## Ejemplos ¶

### Ejemplo #7 printf(): ejemplos varios

```php
<?php
$n =  43951789;
$u = -43951789;
$c = 65; // ASCII 65 es 'A'

// observar el doble %%, esto muestra un carácter '%' literal
printf("%%b = '%b'\n", $n); // representación binaria
printf("%%c = '%c'\n", $c); // muestra el carácter ascii, igual que la función chr()
printf("%%d = '%d'\n", $n); // representación estándar de un entero
printf("%%e = '%e'\n", $n); // notación científica
printf("%%u = '%u'\n", $n); // representación sin signo de un entero positivo
printf("%%u = '%u'\n", $u); // representación sin signo de un entero negativo
printf("%%f = '%f'\n", $n); // representación de punto flotante
printf("%%o = '%o'\n", $n); // representación octal
printf("%%s = '%s'\n", $n); // representación en una cadena
printf("%%x = '%x'\n", $n); // representación hexadecimal (minúsculas)
printf("%%X = '%X'\n", $n); // representación hexadecimal (mayúsculas)

printf("%%+d = '%+d'\n", $n); // especificador de signo sobre un entero positivo
printf("%%+d = '%+d'\n", $u); // especificador de signo sobre un entero negativo
?>
```

El resultado del ejemplo sería:

```
%b = '10100111101010011010101101'
%c = 'A'
%d = '43951789'
%e = '4.39518e+7'
%u = '43951789'
%u = '4251015507'
%f = '43951789.000000'
%o = '247523255'
%s = '43951789'
%x = '29ea6ad'
%X = '29EA6AD'
%+d = '+43951789'
%+d = '-43951789'
```

### Ejemplo #8 printf(): especificadores de string

```php
<?php
$s = 'mono';
$t = 'muchos monos';

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

```
printf("[%10.10s]\n", $t); // justificación a la izquierda pero con un corte a los 10 caracteres
?>
```

El resultado del ejemplo sería:

```
[mono]
[      mono]
[mono      ]
[000000mono]
[######mono]
[muchos mon]
```

### Ejemplo #9 sprintf(): valores de tipo integer rellenados con ceros

```
<?php
$fecha_iso = sprintf("%04d-%02d-%02d", $año, $mes, $día);
?>
```

### Ejemplo #10 sprintf(): formato de moneda

```
<?php
$dinero1 = 68.75;
$dinero2 = 54.35;
$dinero = $dinero1 + $dinero2;
// echo $dinero producirá "123.1";
$formateado = sprintf("%01.2f", $dinero);
// echo $formateado producirá "123.10"
?>
```

### Ejemplo #11 sprintf(): notación científica

```
<?php
$número = 362525200;

echo sprintf("%.3e", $número); // produce 3.625e+8
?>
```

## Ver también ¶

- printf() - Imprimir una cadena con formato
- sscanf() - Interpreta un string de entrada de acuerdo con un formato
- fscanf() - Analiza la entrada desde un archivo de acuerdo a un formato
- vsprintf() - Devuelve una cadena con formato
- number_format() - Formatear un número con los millares agrupados
- date() - Dar formato a la fecha/hora local

+ add a note

## User Contributed Notes 36 notes

up
down
90
***remy dot damour at -please-no-spam-laposte dot net*** ¶
**13 years ago**

```
With printf() and sprintf() functions, escape character is not backslash '\' but rather '%'.

Ie. to print '%' character you need to escape it with itself:
<?php
```

```
printf('%%%s%%', 'koko'); #output: '%koko%'
?>
```

up
down
65
***Alex R. Gibbs ¶***
**9 years ago**

1. A plus sign ('+') means put a '+' before positive numbers while a minus sign ('-') means left justify. The documentation incorrectly states that they are interchangeable. They produce unique results that can be combined:

```
<?php
echo sprintf ("|%+4d|%+4d|\n",   1, -1);
echo sprintf ("|%-4d|%-4d|\n",   1, -1);
echo sprintf ("|%+-4d|%+-4d|\n", 1, -1);
?>
```

outputs:

```
|  +1|  -1|
|1   |-1  |
|+1  |-1  |
```

2. Padding with a '0' is different than padding with other characters. Zeros will only be added at the front of a number, after any sign. Other characters will be added before the sign, or after the number:

```
<?php
echo sprintf ("|%04d|\n",   -2);
echo sprintf ("|%':4d|\n",  -2);
echo sprintf ("|%-':4d|\n", -2);

// Specifying both "-" and "0" creates a conflict with unexpected results:
echo sprintf ("|%-04d|\n",  -2);

// Padding with other digits behaves like other non-zero characters:
echo sprintf ("|%-'14d|\n", -2);
echo sprintf ("|%-'04d|\n", -2);
?>
```

outputs:

```
|-002|
|::-2|
|-2::|
|-2  |
|-211|
|-2  |
```

up
down
16
***timo dot frenay at gmail dot com ¶***
**11 years ago**

Here is how to print a floating point number with 16 significant digits regardless of magnitude:

```
<?php
    $result = sprintf(sprintf('%%.%dF', max(15 - floor(log10($value)), 0)), $value);
?>
```

This works more reliably than doing something like sprintf('%.15F', $value) as the latter may cut off significant digits for very small numbers, or prints bogus digits (meaning extra digits beyond what can reliably be represented in a floating point number) for very large numbers.

up
down
21

***kontakt at myseosolution dot de*** ¶
**7 years ago**

There are already some comments on using sprintf to force leading leading zeros but the examples only include integers. I needed leading zeros on floating point numbers and was surprised that it didn't work as expected.

Example:

```
<?php
sprintf('%02d', 1);
?>
```

This will result in 01. However, trying the same for a float with precision doesn't work:

```
<?php
sprintf('%02.2f', 1);
?>
```

Yields 1.00.

This threw me a little off. To get the desired result, one needs to add the precision (2) and the length of the decimal seperator "." (1). So the correct pattern would be

```
<?php
sprintf('%05.2f', 1);
?>
```

Output: 01.00

Please see http://stackoverflow.com/a/28739819/413531 for a more detailed explanation.

up
down
6

***Anderson*** ¶
**2 years ago**

The old "monkey" example which helped me a lot has sadly disappeared.

I'll Re-post it in comment as a memory.

```
<?php
$n =  43951789;
$u = -43951789;
$c = 65; // ASCII 65 is 'A'

// notice the double %%, this prints a literal '%' character
printf("%%b = '%b'\n", $n); // binary representation
printf("%%c = '%c'\n", $c); // print the ascii character, same as chr() function
printf("%%d = '%d'\n", $n); // standard integer representation
printf("%%e = '%e'\n", $n); // scientific notation
printf("%%u = '%u'\n", $n); // unsigned integer representation of a positive integer
printf("%%u = '%u'\n", $u); // unsigned integer representation of a negative integer
printf("%%f = '%f'\n", $n); // floating point representation
```

```
printf("%%o = '%o'\n", $n); // octal representation
printf("%%s = '%s'\n", $n); // string representation
printf("%%x = '%x'\n", $n); // hexadecimal representation (lower-case)
printf("%%X = '%X'\n", $n); // hexadecimal representation (upper-case)

printf("%%+d = '%+d'\n", $n); // sign specifier on a positive integer
printf("%%+d = '%+d'\n", $u); // sign specifier on a negative integer

/*
%b = '10100111101010011010101101'
%c = 'A'
%d = '43951789'
%e = '4.395179e+7'
%u = '43951789'
%u = '18446744073665599827'
%f = '43951789.000000'
%o = '247523255'
%s = '43951789'
%x = '29ea6ad'
%X = '29EA6AD'
%+d = '+43951789'
%+d = '-43951789'
*/

$s = 'monkey';
$t = 'many monkeys';

printf("[%s]\n",      $s); // standard string output
printf("[%10s]\n",    $s); // right-justification with spaces
printf("[%-10s]\n",   $s); // left-justification with spaces
printf("[%010s]\n",   $s); // zero-padding works on strings too
printf("[%'#10s]\n",  $s); // use the custom padding character '#'
printf("[%10.10s]\n", $t); // left-justification but with a cutoff of 10 characters

/*
[monkey]
[    monkey]
[monkey    ]
[0000monkey]
[####monkey]
[many monke]
*/
?>
```

up
down

10

***dwieeb at gmail dot com ¶***

**12 years ago**

If you use the default padding specifier (a space) and then print it to HTML, you will notice that HTML does not display the multiple spaces correctly. This is because any sequence of white-space is treated as a single space.

To overcome this, I wrote a simple function that replaces all the spaces in the string returned by sprintf() with the character entity reference " " to achieve non-breaking space in strings returned by sprintf()

```
<?php
//Here is the function:
```

```
function sprintf_nbsp() {
   $args = func_get_args();
   return str_replace(' ', ' ', vsprintf(array_shift($args), array_values($args)));
}

//Usage (exactly like sprintf):
$format = 'The %d monkeys are attacking the [%10s]!';
$str = sprintf_nbsp($format, 15, 'zoo');
echo $str;
?>

The above example will output:
The 15 monkeys are attacking the [       zoo]!

<?php
//The variation that prints the string instead of returning it:
function printf_nbsp() {
   $args = func_get_args();
   echo str_replace(' ', ' ', vsprintf(array_shift($args), array_values($args)));
}
?>
```

up
down
3

***Anonymous ¶***

**5 years ago**

```
Be cafeful while trying to refactor longer strings with repeated placeholders like

    sprintf("Hi %s. Your name is %s", $name, $name);

to use argument numbering:

   sprintf("Hi %1$s. Your name is %1$s", $name);

This will nuke you at **runtime**, because of `$s` thing being handled as variable. If you got no
$s for substitution, notice will be thrown.

The solution is to use single quotes to prevent variable substitution in string:

   sprintf('Hi %1$s. Your name is %1$s', $name);

If you need variable substitution, then you'd need to split your string to keep it in single
quotes:

   sprintf("Hi " . '%1$s' . ". Your {$variable} is " . '%1$s', $name);
```

up
down
10

***viktor at textalk dot com ¶***

**13 years ago**

```
A more complete and working version of mb_sprintf and mb_vsprintf. It should work with any "ASCII
preserving" encoding such as UTF-8 and all the ISO-8859 charsets. It handles sign, padding,
alignment, width and precision. Argument swapping is not handled.

<?php
if (!function_exists('mb_sprintf')) {
  function mb_sprintf($format) {
      $argv = func_get_args() ;
```

```
        array_shift($argv) ;
        return mb_vsprintf($format, $argv) ;
    }
  }
  if (!function_exists('mb_vsprintf')) {
    /**
     * Works with all encodings in format and arguments.
     * Supported: Sign, padding, alignment, width and precision.
     * Not supported: Argument swapping.
     */
    function mb_vsprintf($format, $argv, $encoding=null) {
        if (is_null($encoding))
            $encoding = mb_internal_encoding();

        // Use UTF-8 in the format so we can use the u flag in preg_split
        $format = mb_convert_encoding($format, 'UTF-8', $encoding);

        $newformat = ""; // build a new format in UTF-8
        $newargv = array(); // unhandled args in unchanged encoding

        while ($format !== "") {

          // Split the format in two parts: $pre and $post by the first %-directive
          // We get also the matched groups
          list ($pre, $sign, $filler, $align, $size, $precision, $type, $post) =
              preg_split("!\%(\+?)('.|[0 ]|)(-?)([1-9][0-9]*|)(\.[1-9][0-9]*|)([%a-zA-Z])!u",
                         $format, 2, PREG_SPLIT_DELIM_CAPTURE) ;

          $newformat .= mb_convert_encoding($pre, $encoding, 'UTF-8');

          if ($type == '') {
            // didn't match. do nothing. this is the last iteration.
          }
          elseif ($type == '%') {
            // an escaped %
            $newformat .= '%%';
          }
          elseif ($type == 's') {
            $arg = array_shift($argv);
            $arg = mb_convert_encoding($arg, 'UTF-8', $encoding);
            $padding_pre = '';
            $padding_post = '';

            // truncate $arg
            if ($precision !== '') {
              $precision = intval(substr($precision,1));
              if ($precision > 0 && mb_strlen($arg,$encoding) > $precision)
                $arg = mb_substr($precision,0,$precision,$encoding);
            }

            // define padding
            if ($size > 0) {
              $arglen = mb_strlen($arg, $encoding);
              if ($arglen < $size) {
                if($filler==='')
                    $filler = ' ';
                if ($align == '-')
                    $padding_post = str_repeat($filler, $size - $arglen);
```

```
                else
                    $padding_pre = str_repeat($filler, $size - $arglen);
              }
            }

            // escape % and pass it forward
            $newformat .= $padding_pre . str_replace('%', '%%', $arg) . $padding_post;
          }
          else {
            // another type, pass forward
            $newformat .= "%$sign$filler$align$size$precision$type";
            $newargv[] = array_shift($argv);
          }
          $format = strval($post);
        }
        // Convert new format back from UTF-8 to the original encoding
        $newformat = mb_convert_encoding($newformat, $encoding, 'UTF-8');
        return vsprintf($newformat, $newargv);
    }
}
?>
```

up
down
8
***no dot email dot address at example dot com ¶***
**20 years ago**

Using argument swapping in sprintf() with gettext: Let's say you've written the following script:

```
<?php
$var = sprintf(gettext("The %2\$s contains %1\$d monkeys"), 2, "cage");
?>
```

Now you run xgettext in order to generate a .po file. The .po file will then look like this:

```
#: file.php:9
#, ycp-format
msgid "The %2\\$s contains %1\\$d monkeys"
msgstr ""
```

Notice how an extra backslash has been added by xgettext.

Once you've translated the string, you must remove all backslashes from the ID string as well as the translation, so the po file will look like this:

```
#: file.php:9
#, ycp-format
msgid "The %2$s contains %1$d monkeys"
msgstr "Der er %1$d aber i %2$s"
```

Now run msgfmt to generate the .mo file, restart Apache to remove the gettext cache if necessary, and you're off.

up
down
7
***splogamurugan at gmail dot com ¶***
**13 years ago**

```
$format = 'There are %1$d monkeys in the %s and %s ';
printf($format, 100, 'Chennai', 'Bangalore');
```

Expecting to output
"There are 100 monkeys in the Chennai and bangalore"

But, this will output
"There are 100 monkeys in the 100 and Chennai"

Because, the second and Third specifiers takes 1rst and 2nd arguments. Because it is not assigned with any arguments.

up
down
9

***php at sharpdreams dot com ¶***

**18 years ago**

Note that when using the argument swapping, you MUST number every argument, otherwise sprintf gets confused. This only happens if you use number arguments first, then switch to a non-numbered, and then back to a numbered one.

```
<?php
$sql = sprintf( "select * from %1\$s left join %2\$s on( %1\$s.id = %2\$s.midpoint ) where
%1\$s.name like '%%%s%%' and %2\$s.tagname is not null", "table1", "table2", "bob" );
// Wont work:
// Sprintf will complain about not enough arguments.
$sql = sprintf( "select * from %1\$s left join %2\$s on( %1\$s.id = %2\$s.midpoint ) where
%1\$s.name like '%%%3\$s%%' and %2\$s.tagname is not null", "table1", "table2", "bob" );
// Will work: note the %3\$s
?>
```

up
down
9

***jfgrissom at gmail dot com ¶***

**13 years ago**

I had a nightmare trying to find the two's complement of a 32 bit number.

I got this from http://www.webmasterworld.com/forum88/13334.htm (credit where credit is due... =P )

Quote: ...find out the 2's complement of any number, which is -(pow(2, n) - N) where n is the number of bits and N is the number for which to find out its 2's complement.

This worked magic for me... previously I was trying to use

```
sprintf ("%b",$32BitDecimal);
```

But it always returned 10000000000000000000000000000000 when the $32BitDecimal value got above 2,000,000,000.

This -(pow(2, n) - N)
Worked remarkably well and was very accurate.

Hope this helps someone fighting with two's complement in PHP.

up
down
4

***abiltcliffe at bigfoot.com ¶***

**20 years ago**

To jrust at rustyparts.com, note that if you're using a double-quoted string and *don't* escape the dollar sign with a backslash, $s and $d will be interpreted as variable references. The

backslash isn't part of the format specifier itself but you do need to include it when you write the format string (unless you use single quotes).

up
down
9

***Pacogliss ¶***

**17 years ago**

Just a reminder for beginners : example 6 'printf("[%10s]\n",    $s);' only works (that is, shows out the spaces) if you put the html '<pre></pre>' tags ( head-scraping time saver ;-).

up
down
7

***nate at frickenate dot com ¶***

**13 years ago**

Here's a clean, working version of functions to allow using named arguments instead of numeric ones. ex: instead of sprintf('%1$s', 'Joe');, we can use sprintf('%name$s', array('name' => 'Joe'));. I've provided 2 different versions: the first uses the php-like syntax (ex: %name$s), while the second uses the python syntax (ex: %(name)s).

```
<?php

/**
* version of sprintf for cases where named arguments are desired (php syntax)
*
* with sprintf: sprintf('second: %2$s ; first: %1$s', '1st', '2nd');
*
* with sprintfn: sprintfn('second: %second$s ; first: %first$s', array(
*  'first' => '1st',
*  'second'=> '2nd'
* ));
*
* @param string $format sprintf format string, with any number of named arguments
* @param array $args array of [ 'arg_name' => 'arg value', ... ] replacements to be made
* @return string|false result of sprintf call, or bool false on error
*/
function sprintfn ($format, array $args = array()) {
    // map of argument names to their corresponding sprintf numeric argument value
    $arg_nums = array_slice(array_flip(array_keys(array(0 => 0) + $args)), 1);

    // find the next named argument. each search starts at the end of the previous replacement.
    for ($pos = 0; preg_match('/(?<=%)([a-zA-Z_]\w*)(?=\$)/', $format, $match,
PREG_OFFSET_CAPTURE, $pos);) {
        $arg_pos = $match[0][1];
        $arg_len = strlen($match[0][0]);
        $arg_key = $match[1][0];

        // programmer did not supply a value for the named argument found in the format string
        if (! array_key_exists($arg_key, $arg_nums)) {
            user_error("sprintfn(): Missing argument '${arg_key}'", E_USER_WARNING);
            return false;
        }

        // replace the named argument with the corresponding numeric one
        $format = substr_replace($format, $replace = $arg_nums[$arg_key], $arg_pos, $arg_len);
        $pos = $arg_pos + strlen($replace); // skip to end of replacement for next iteration
    }

    return vsprintf($format, array_values($args));
```

```
}

/**
* version of sprintf for cases where named arguments are desired (python syntax)
*
* with sprintf: sprintf('second: %2$s ; first: %1$s', '1st', '2nd');
*
* with sprintfn: sprintfn('second: %(second)s ; first: %(first)s', array(
*  'first' => '1st',
*  'second'=> '2nd'
* ));
*
* @param string $format sprintf format string, with any number of named arguments
* @param array $args array of [ 'arg_name' => 'arg value', ... ] replacements to be made
* @return string|false result of sprintf call, or bool false on error
*/
function sprintfn ($format, array $args = array()) {
    // map of argument names to their corresponding sprintf numeric argument value
    $arg_nums = array_slice(array_flip(array_keys(array(0 => 0) + $args)), 1);

    // find the next named argument. each search starts at the end of the previous replacement.
    for ($pos = 0; preg_match('/(?<=%)\(([a-zA-Z_]\w*)\)/', $format, $match, PREG_OFFSET_CAPTURE,
$pos);) {
        $arg_pos = $match[0][1];
        $arg_len = strlen($match[0][0]);
        $arg_key = $match[1][0];

        // programmer did not supply a value for the named argument found in the format string
        if (! array_key_exists($arg_key, $arg_nums)) {
            user_error("sprintfn(): Missing argument '${arg_key}'", E_USER_WARNING);
            return false;
        }

        // replace the named argument with the corresponding numeric one
        $format = substr_replace($format, $replace = $arg_nums[$arg_key] . '$', $arg_pos,
$arg_len);
        $pos = $arg_pos + strlen($replace); // skip to end of replacement for next iteration
    }

    return vsprintf($format, array_values($args));
}

?>
```

up
down
6
***Hayley Watson ¶***
**10 years ago**
If you use argument numbering, then format specifications with the same number get the same argument; this can save repeating the argument in the function call.

```
<?php

$pattern = '%1$s %1$\'#10s %1$s!';

printf($pattern, "badgers");
?>
```

up

down
5
***john at jbwalker dot com ¶***
**8 years ago**
I couldn't find what should be a WARNING in the documentation above, that if you have more specifiers than variables to match them sprintf returns NOTHING. This fact, IMHO, should also be noted under return values.
up
down
3
***ian dot w dot davis at gmail dot com ¶***
**17 years ago**
Just to elaborate on downright's point about different meanings for %f, it appears the behavior changed significantly as of 4.3.7, rather than just being different on different platforms. Previously, the width specifier gave the number of characters allowed BEFORE the decimal. Now, the width specifier gives the TOTAL number of characters. (This is in line with the semantics of printf() in other languages.) See bugs #28633 and #29286 for more details.
up
down
2
***php at mikeboers dot com ¶***
**14 years ago**
And continuing on the same theme of a key-based sprintf...

I'm roughly (I can see a couple cases where it comes out wierd) copying the syntax of Python's string formatting with a dictionary. The improvement over the several past attempts is that this one still respects all of the formating options, as you can see in my example.

And the error handling is really crappy (just an echo). I just threw this together so do with it what you will. =]

```
<?php

function sprintf_array($string, $array)
{
    $keys    = array_keys($array);
    $keysmap = array_flip($keys);
    $values  = array_values($array);

    while (preg_match('/%\(([a-zA-Z0-9_ -]+)\)/', $string, $m))
    {
        if (!isset($keysmap[$m[1]]))
        {
            echo "No key $m[1]\n";
            return false;
        }

        $string = str_replace($m[0], '%' . ($keysmap[$m[1]] + 1) . '$', $string);
    }

    array_unshift($values, $string);
    var_dump($values);
    return call_user_func_array('sprintf', $values);
}

echo sprintf_array('4 digit padded number: %(num)04d ', array('num' => 42));

?>
```

Cheers!
up
down
3
***carmageddon at gmail dot com ¶***
**11 years ago**
If you want to convert a decimal (integer) number into constant length binary number in lets say 9 bits, use this:

```
$binary = sprintf('%08b', $number );
```

for example:

```
<?php
$bin = sprintf('%08b',511 );
echo $bin."\n";
?>
```

would output 111111111
And 2 would output 00000010

I know the leading zeros are useful to me, perhaps they are to someone else too.
up
down
2
***nmmm at nmmm dot nu ¶***
**7 years ago**
php printf and sprintf not seems to support star "*" formatting.

here is an example:

```
printf("%*d\n",3,5);
```

this will print just "d" instead of "<two spaces>5"
up
down
2
***hdimac at gmail dot com ¶***
**8 years ago**
In the examples, is being shown printf, but it should say sprintf, which is the function being explained... just a simple edition mistake.
up
down
1
***Nathan Alan ¶***
**5 years ago**
Just wanted to add that to get the remaining text from the string, you need to add the following as a variable in your scanf

```
%[ -~]
```

Example:

```
sscanf($sql, "[%d,%d]%[ -~]", $sheet_id, $column, $remaining_sql);
```
up
down
2
***krzysiek dot 333 at gmail dot com - zryty dot hekko dot pl ¶***

**11 years ago**

Encoding and decoding IP adress to format: 1A2B3C4D (mysql column: char(8) )

```
<?php
function encode_ip($dotquad_ip)
{
    $ip_sep = explode('.', $dotquad_ip);
    return sprintf('%02x%02x%02x%02x', $ip_sep[0], $ip_sep[1], $ip_sep[2], $ip_sep[3]);
}

function decode_ip($int_ip)
{
    $hexipbang = explode('.', chunk_split($int_ip, 2, '.'));
    return hexdec($hexipbang[0]). '.' . hexdec($hexipbang[1]) . '.' . hexdec($hexipbang[2]) . '.'
. hexdec($hexipbang[3]);
}
?>
```

up
down
2

***jrpozo at conclase dot net ¶***

**17 years ago**

Be careful if you use the %f modifier to round decimal numbers as it (starting from 4.3.10) will no longer produce a float number if you set certain locales, so you can't accumulate the result. For example:

```
setlocale(LC_ALL, 'es_ES');
echo(sprintf("%.2f", 13.332) + sprintf("%.2f", 14.446))

gives 27 instead of 27.78, so use %F instead.
```

up
down
1

***geertdd at gmail dot com ¶***

**12 years ago**

Note that when using a sign specifier, the number zero is considered positive and a "+" sign will be prepended to it.

```
<?php
printf('%+d', 0); // +0
?>
```

up
down
1

***Astone ¶***

**13 years ago**

When you're using Google translator, you have to 'escape' the 'conversion specifications' by putting  around them.

Like this:

```
<?php

function getGoogleTranslation($sString, $bEscapeParams = true)
{
    // "escape" sprintf paramerters
    if ($bEscapeParams)
    {
```

```
        $sPatern = '/(?:%%|%(?:[0-9]+\$)?[+-]?(?:[ 0]|\'.)?-?[0-9]*(?:\.[0-9]+)?[bcdeufFosxX])/';

        $sEscapeString = '<span class="notranslate">$0</span>';
        $sString = preg_replace($sPatern, $sEscapeString, $sString);
    }

    // Compose data array (English to Dutch)
    $aData = array(
        'v'            => '1.0',
        'q'            => $sString,
        'langpair'    => 'en|nl',
    );

    // Initialize connection
    $rService = curl_init();

    // Connection settings
    curl_setopt($rService, CURLOPT_URL,
'http://ajax.googleapis.com/ajax/services/language/translate');
    curl_setopt($rService, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($rService, CURLOPT_POSTFIELDS, $aData);

    // Execute request
    $sResponse = curl_exec($rService);

    // Close connection
    curl_close($rService);

    // Extract text from JSON response
    $oResponse = json_decode($sResponse);
    if (isset($oResponse->responseData->translatedText))
    {
        $sTranslation = $oResponse->responseData->translatedText;
    }
    else
    {
        // If some error occured, use the original string
        $sTranslation = $sString;
    }

    // Replace "notranslate" tags
    if ($bEscapeParams)
    {
        $sEscapePatern = '/<span class="notranslate">([^<]*)<\/span>/';
        $sTranslation = preg_replace($sEscapePatern, '$1', $sTranslation);
    }

    // Return result
    return $sTranslation;
}

?>
```

Thanks to MelTraX for defining the RegExp!

up
down
1
***ignat dot scheglovskiy at gmail dot com ¶***

**10 years ago**

Here is an example how alignment, padding and precision specifier can be used to print formatted list of items:

```
<?php

$out = "The Books\n";
$books = array("Book 1", "Book 2", "Book 3");
$pages = array("123 pages ", "234 pages", "345 pages");
for ($i = 0; $i < count($books); $i++) {
    $out .= sprintf("%'.-20s%'.7.4s\n", $books[$i], $pages[$i]);
}
echo $out;

// Outputs:
//
// The Books
// Book 1..............123
// Book 2..............234
// Book 3..............345
?>
```

up
down
1

***scott dot gardner at mac dot com ¶***

**14 years ago**

In the last example of Example#6, there is an error regarding the output.

```
printf("[%10.10s]\n", $t); // left-justification but with a cutoff of 10 characters
```

This outputs right-justified.

In order to output left-justified:

```
printf("[%-10.10s]\n", $t);
```

up
down
1

***John Walker ¶***

**13 years ago**

To add to other notes below about floating point problems, I noted that %f and %F will apparently output a maximum precision of 6 as a default so you have to specify 1.15f (eg) if you need more.

In my case, the input (from MySQL) was a string with 15 digits of precision that was displayed with 6. Likely what happens is that the rounding occurs in the conversion to a float before it is displayed. Displaying it as 1.15f (or in my case, %s) shows the correct number.

up
down
1

***2838132019 at qq dot com ¶***

**1 year ago**

```
echo sprintf("%.2f", "123456789012345.82");
// result: 123456789012345.81

echo sprintf("%.2f", "123456789012345.85");
// result: 123456789012345.84

echo sprintf("%.2f", "123456789012345.87");
```

```
//result:  123456789012345.88

echo sprintf("%.2f", "123456789012345.820");
//result: 123456789012345.81

echo sprintf("%.2f", "123456789012345.821");
//result: 123456789012345.83

echo sprintf("%.2f", "123456789012345.828");
//result: 123456789012345.83

echo sprintf("%.2f", "123456789012345.8209");
//result : 123456789012345.83

echo sprintf("%.2f", "1234567890123456.82");
//result: 1234567890123456.75

echo sprintf("%.2f", "123456789012345.82002");
//result: 123456789012345.81

echo sprintf("%.2f", "123456789012345.820001");
//result: 123456789012345.81

echo sprintf("%.2f", "123456789012345.820101");
//result: 123456789012345.81

echo sprintf("%.2f", "123456789012345.820201");
//result: 123456789012345.81

echo sprintf("%.2f", "123456789012345.820301");
//result: 123456789012345.81

echo sprintf("%.2f", "123456789012345.820401");
//result: 123456789012345.83
```

up
down
1

***Andrew dot Wright at spamsux dot atnf dot csiro dot au ¶***

**20 years ago**

An error in my last example:

```
$b = sprintf("%30.s", $a);
```

will only add enough spaces before $a to pad the spaces + strlen($a) to 30 places.

My method of centering fixed text in a 72 character width space is:

```
$a = "Some string here";
$lwidth = 36; // 72/2
$b = sprintf("%".($lwidth + round(strlen($a)/2)).".s", $a);
```

up
down
0

***Anonymous ¶***

**8 months ago**

If the format string is enclosed in double-quotes (""), you need to escape the dollar sign after argnum with a backslash character (\), like this %1\$s, so that the PHP doesn't try to interpret them as variable. Using a backslash like this is called an escape sequence.

```
<?php
```

```
// Sample string
$number = 499;
$format = "The number without decimal points: %1\$d, and the number with two decimal points:
%1\$.2f";

// Formatting and print the string
printf($format, $number);
?>
```

up
down
0

***ivan at php dot net*** ¶
**8 years ago**

There is a minor issue in a code of mb_vsprintf function from viktor at textalk dot com.

In "truncate $arg" section the following line:

```
  $arg = mb_substr($precision,0,$precision,$encoding);
```

needs to be replaced with:

```
  $arg = mb_substr($arg,0,$precision,$encoding);
```

up
down
-1

***Sam Bull*** ¶
**7 years ago**

Fix for sprintfn function for named arguments (http://php.net/manual/en/function.sprintf.php#94608):

Change the first line from:

```
  $arg_nums = array_slice(array_flip(array_keys(array(0 => 0) + $args)), 1);
```

to:

```
  $arg_nums = array_keys($args);
  array_unshift($arg_nums, 0);
  $arg_nums = array_flip(array_slice($arg_nums, 1, NULL, true));
```

up
down
-1

***Mirek Z...*** ¶
**2 years ago**

I've performed a simple speed test. sprintf against PHP string concatenation operator. Test was performed on PHP 7.3 for 1 million interations.

I run this several times and what I've noted that string concatenation took about 2.9 seconds, sprintf took 4.3 seconds.
I was thinking about what is faster, what is better to do when we're going to format our string (for example, the message to the user or for log purposes) containing some variables values. Is it better to concatenate string with variables using operator (dot ".") or to use sprintf. The answer is: when you do not plan to implement any multilanguage mechanisms and feel good with hardcoding some texts, the "dot" is almost 1.5 times faster!

Here's the code:

```
echo 'Start' . PHP_EOL;
$vS_text = 'some text';
$vS = '';
$vf = microtime(true);
for ($vI = 0; $vI < 1000000; $vI++) {
    $vS = 'Start ' . $vI . ' ' . $vS_text . ' ' . $vf . ' end';
}
```

```
$vf = microtime(true) - $vf;
echo 'Concat:' . $vf . PHP_EOL;
$vS = '';
$vf = microtime(true);
for ($vI = 0; $vI < 1000000; $vI++) {
    $vS = sprintf('Start %d %s %f end', $vI, $vS_text, $vf);
}
$vf = microtime(true) - $vf;
echo 'Spritf:' . $vf . PHP_EOL;
```

+ add a note

- Funciones de strings
  - addcslashes
  - addslashes
  - bin2hex
  - chop
  - chr
  - chunk_split
  - convert_uudecode
  - convert_uuencode
  - count_chars
  - crc32
  - crypt
  - echo
  - explode
  - fprintf
  - get_html_translation_table
  - hebrev
  - hex2bin
  - html_entity_decode
  - htmlentities
  - htmlspecialchars_decode
  - htmlspecialchars
  - implode
  - join
  - lcfirst
  - levenshtein
  - localeconv
  - ltrim
  - md5_file
  - md5
  - metaphone
  - money_format
  - nl_langinfo
  - nl2br
  - number_format
  - ord
  - parse_str
  - print
  - printf
  - quoted_printable_decode
  - quoted_printable_encode
  - quotemeta
  - rtrim
  - setlocale
  - sha1_file
  - sha1
  - similar_text

- soundex
- sprintf
- sscanf
- str_contains
- str_ends_with
- str_getcsv
- str_ireplace
- str_pad
- str_repeat
- str_replace
- str_rot13
- str_shuffle
- str_split
- str_starts_with
- str_word_count
- strcasecmp
- strchr
- strcmp
- strcoll
- strcspn
- strip_tags
- stripcslashes
- stripos
- stripslashes
- stristr
- strlen
- strnatcasecmp
- strnatcmp
- strncasecmp
- strncmp
- strpbrk
- strpos
- strrchr
- strrev
- strripos
- strrpos
- strspn
- strstr
- strtok
- strtolower
- strtoupper
- strtr
- substr_compare
- substr_count
- substr_replace
- substr
- trim
- ucfirst
- ucwords
- utf8_decode
- utf8_encode
- vfprintf
- vprintf
- vsprintf
- wordwrap

- Deprecated
  - convert_cyr_string
  - hebrevc