crc32 »
« convert_uuencode

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

Change language: Spanish

Submit a Pull Request Report a Bug

# count_chars

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

count_chars — Devuelve información sobre los caracteres usados en una cadena

## Descripción ¶

**count_chars**(string `$string`, int `$mode` = 0): mixed

Cuenta el número de apariciones de cada byte-value (0..255) en `string` y lo devuelve de varias maneras.

## Parámetros ¶

`string`

La cadena a examinar

`mode`

Ver los valores a retornar.

## Valores devueltos ¶

Dependiendo de `mode` **count_chars()** devuelve uno de los siguientes:

- 0 - un array con el byte-value como clave y la frecuencia de cada uno como valor.
- 1 - Como el 0, pero listando únicamente los byte-values con frecuencia superior a cero.
- 2 - Como el 0, pero listando únicamente los byte-values con frecuencia igual a 0.
- 3 - devuelve una cadena que contiene todos los caracteres únicos.
- 4 - devuelve una cadena que contiene todos los caracteres no utilizados.

## Ejemplos ¶

### Ejemplo #1 count_chars() example

```
<?php
$data = "Two Ts and one F.";

foreach (count_chars($data, 1) as $i => $val) {
   echo "Se ha encontrado $val instancia (s) de \"" , chr($i) , "\" en la cadena.\n";
}
?>
```

El resultado del ejemplo sería:

```
Hubo 4  instancia(s) de " " en la cadena.
Hubo 1  instancia(s) de "." en la cadena.
Hubo 1  instancia(s) de "F" en la cadena.
Hubo 2  instancia(s) de "T" en la cadena.
Hubo 1  instancia(s) de "a" en la cadena.
Hubo 1  instancia(s) de "d" en la cadena.
Hubo 1  instancia(s) de "e" en la cadena.
Hubo 2  instancia(s) de "n" en la cadena.
Hubo 2  instancia(s) de "o" en la cadena.
Hubo 1  instancia(s) de "s" en la cadena.
Hubo 1  instancia(s) de "w" en la cadena.
```

## Ver también ¶

- strpos() - Encuentra la posición de la primera ocurrencia de un substring en un string
- substr_count() - Cuenta el número de apariciones del substring

+ add a note

## User Contributed Notes 11 notes

up
down
19
***marcus33cz*** ¶
**10 years ago**

If you have problems using count_chars with a multibyte string, you can change the page encoding. Alternatively, you can also use this mb_count_chars version of the function. Basically it is mode "1" of the original function.

```
<?php
/**
* Counts character occurences in a multibyte string
* @param string $input UTF-8 data
* @return array associative array of characters.
*/
function mb_count_chars($input) {
    $l = mb_strlen($input, 'UTF-8');
    $unique = array();
    for($i = 0; $i < $l; $i++) {
        $char = mb_substr($input, $i, 1, 'UTF-8');
        if(!array_key_exists($char, $unique))
            $unique[$char] = 0;
        $unique[$char]++;
    }
    return $unique;
}

$input = "Let's try some Greek letters: ααααεειιμμψψ, Russian: ЙЙЫЫЩН, Czech: ěščřžýáíé";
print_r( mb_count_chars($input) );
//returns: Array ( [L] => 1 [e] => 7 [t] => 4 ['] => 1 [s] => 5 [ ] => 9 [r] => 3 [y] => 1 [o] =>
1 [m] => 1 [G] => 1 [k] => 1 [l] => 1 [:] => 3 [α] => 5 [E] => 1 [ε] => 1 [I] => 1 [ι] => 1 [M] =>
1 [μ] => 1 [Ψ] => 1 [ψ] => 1 [,] => 2 [R] => 1 [u] => 1 [i] => 1 [a] => 1 [n] => 1 [Й] => 2 [Ы] =>
2 [Щ] => 1 [H] => 1 [C] => 1 [z] => 1 [c] => 1 [h] => 1 [ě] => 1 [š] => 1 [č] => 1 [ř] => 1 [ž] =>
1 [ý] => 1 [á] => 1 [í] => 1 [é] => 1 )
?>
```

up

down
1
***Andrey G ¶***
**2 years ago**
Checking that two strings are anagram:

```
<?php

function isAnagram($string1, $string2)
{
  return count_chars($string1, 1) === count_chars($string2, 1);
}

isAnagram('act', 'cat'); // true

?>
```

up
down
2
***Eric Pecoraro ¶***
**17 years ago**

```
<?php

// Require (n) unique characters in a string
// Modification of a function below which ads some flexibility in how many unique characters are
required in a given string.

$pass = '123456' ; // true
$pass = '111222' ; // false

req_unique($pass,3);

function req_unique($string,$unique=3) {
    if ( count(count_chars($string,1)) < $unique) {
        echo 'false';
    }else{
        echo 'true';
    }
}

?>
```

up
down
1
***seb at synchrocide dot net ¶***
**18 years ago**
After much trial and error trying to create a function that finds the number of unique characters in a string I same across count_chars() - my 20+ lines of useless code were wiped for this:

```
<?
function unichar($string) {
$two= strtolower(str_replace(' ', '', $string));
$res = count(count_chars($two, 1));
return $res;
}

/* examples :: */
```

```
echo unichar("bob"); // 2
echo unichar("Invisibility"); //8
echo unichar("The quick brown fox slyly jumped over the lazy dog"); //26

?>
```

I have no idea where this could be used, but it's quite fun

up
down
-1

***pzb at novell dot com ¶***

**15 years ago**

This function is great for input validation.  I frequently need to check that all characters in a string are 7-bit ASCII (and not null).  This is the fastest function I have found yet:

```
<?php
function is7bit($string) {
    // empty strings are 7-bit clean
    if (!strlen($string)) {
        return true;
    }
    // count_chars returns the characters in ascending octet order
    $str = count_chars($str, 3);
    // Check for null character
    if (!ord($str[0])) {
        return false;
    }
    // Check for 8-bit character
    if (ord($str[strlen($str)-1]) & 128) {
        return false;
    }
    return true;
}
?>
```

up
down
-2

***Anonymous ¶***

**6 years ago**

count_chars for multibyte supported.

```
<?php

function mb_count_chars ($string, $mode = 0) {

    $result =  array_fill(0, 256, 0);

    for ($i = 0, $size = mb_strlen($string); $i < $size; $i++) {
        $char = mb_substr($string, $i, 1);
        if (strlen($char) > 1) {
            continue;
        }

        $code = ord($char);
        if ($code >= 0 && $code <= 255) {
            $result[$code]++;
        }
    }
```

```
    switch ($mode) {
        case 1: // same as 0 but only byte-values with a frequency greater than zero are listed.
            foreach ($result as $key => $value) {
                if ($value == 0) {
                    unset($result[$key]);
                }
            }
            break;
        case 2: // same as 0 but only byte-values with a frequency equal to zero are listed.
            foreach ($result as $key => $value) {
                if ($value > 0) {
                    unset($result[$key]);
                }
            }
            break;
        case 3: // a string containing all unique characters is returned.
            $buildString = '';
            foreach ($result as $key => $value) {
                if ($value > 0) {
                    $buildString .= chr($key);
                }
            }
            return $buildString;
        case 4: // a string containing all not used characters is returned.
            $buildString = '';
            foreach ($result as $key => $value) {
                if ($value == 0) {
                    $buildString .= chr($key);
                }
            }
            return $buildString;
    }

    // change key names...
    foreach ($result as $key => $value) {
        $result[chr($key)] = $value;
        unset($result[$key]);
    }

    return $result;

}
?>
```

up
down
-2

***phpC2007 ¶***

**14 years ago**

Here's a function to count number of strings in a string. It can be used as a simple utf8-enabled count_chars (but limited to a single mode)...

```
<?php
function utf8_count_strings($stringChar)
{
    $num = -1;
    $lenStringChar = strlen($stringChar);
```

```
    for ($lastPosition = 0;
        $lastPosition !== false;
        $lastPosition = strpos($textSnippet, $stringChar, $lastPosition + $lenStringChar))
    {
        $num++;
    }

    return $num;
}
?>
```

up
down
-4
***qeremy [atta] gmail [dotta] com ¶***
**9 years ago**

```
Another approach to counting unicode chars.

<?php
function count_chars_unicode($str, $x = false) {
    $tmp = preg_split('//u', $str, -1, PREG_SPLIT_NO_EMPTY);
    foreach ($tmp as $c) {
        $chr[$c] = isset($chr[$c]) ? $chr[$c] + 1 : 1;
    }
    return is_bool($x)
        ? ($x ? $chr : count($chr))
        : $chr[$x];
}

$str = "şeker şeker yâriiiiiiiiiimmmmm";
print_r(count_chars_unicode($str, 'â'));    // frequency of "â"
print_r(count_chars_unicode($str));         // count of uniq chars
print_r(count_chars_unicode($str, true));   // all chars with own frequency
?>

Outputs;
1
9
Array
(
    [ş] => 2
    [e] => 4
    [k] => 2
    [r] => 3
    [ ] => 2
    [y] => 1
    [â] => 1
    [i] => 10
    [m] => 5
)
```

up
down
-6
***maotin at hongkong dot com ¶***
**21 years ago**

```
Here are some more experiments on this relatively new and extremely handy function.

<?php
$string = 'I have never seen ANYTHING like that before! My number is "4670-9394".';
```

```
foreach(count_chars($string, 1) as $chr => $hit)
echo 'The character '.chr(34).chr($chr).chr(34).' has appeared in this string '.$hit.' times.
<BR>';

#The result looks like
#The character " " has appeared in this string 11 times.

echo count_chars($string,3);
#The output is '!"-.034679AGHIMNTYabefhiklmnorstuvy'

echo strlen($string).' is not the same as '.strlen(count_chars($string, 3));

#This shows that '70 is not the same as 36'
?>
```

As we can see above:

1)If you cares only about what is in the string, use count_chars($string, 1) and it will return an (associative?) array of what shows up only.

2) Either I misunderstood what the manul actually said, or it does not work the way it described: count_chars($strting, 3) actually returned a string of what characters are in the string, not a string of their byte-values (which is great because a string of numbers would be much harder to handle);

3)This is a short version of password checking: get the original string's length, then compare with the length of the string returned by count_chars($string,3).

```
<?php
$length_of_string = strlen($string);
$num_of_chars = strlen(count_chars($string, 3));

$diff = ($length_of_string - $num_of_chars);

if ($diff)
echo 'At least one character has been used more than once.';
else
echo 'All character have been used only once.';
?>
```

Note that since $num_of_chars gives no information about the actual number of occurance, we cannot go any further by the same rationale and say when $diff =2 then 2 characters showed up twice; it might be 1 character showd up 3 times, we have no way to tell (a good tolerance level setter, though). You have to get the array and check the values if you want to have more control.

4) Final trick: now we have a primitive way to count the number of words in a string! (or do we have a fuction for that already?)

up
down
-5
***mlong at mlong dot org*** ¶
**20 years ago**

```
// Usefulness of the two functions

<?php
$string="aaabbc";
```

```
// You just want to count the letter a
$acount=substr_count($string,"a");

// You want to count both letter a and letter b
$counts=count_chars($string,0);
$acount=$counts[ord("a")];
$bcount=$counts[ord("b")];
?>
```

up
down
-14
***apinpratap at gmail dot com*** ¶
**15 years ago**

```
this code can find each characters count

<?php
$enter = 0;
$data = strtolower ($inputString);
foreach (count_chars ($data, 1) as $i => $val)
    {
        if ($enter == 1)
        {
            $enter = 0;
            continue;
        }
        if (chr ($i) == "\n")
        {
            echo "There are $val instance(s) of \" Enter \" in the string.\n";
            $enter = 1;
        }
        else
        {
            echo "There are $val instance(s) of \"" , chr ($i) , "\" in the string.\n";
        }
    }
?>
```

+ add a note

- 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