How do I escape curly-brace ({}) characters characters in a string while using .format?

Asked 13 years, 9 months ago Modified 3 months ago Viewed 1.2m times

2653

Non-working example:

```
print(" \{ Hello \} {0} ".format(42))
```

Desired output:

```
 {Hello} 42
```

python string format string-formatting curly-braces

Share

Improve this question

Follow

edited Sep 12 at 15:26

bfontaine

19.7k ● 13 ● 78 ● 120

asked Mar 29, 2011 at 0:04

Schitti

27.8k ● 8 ● 25 ● 21

1 See also: stackoverflow.com/questions/35574349 – dreftymac Dec 6, 2016 at 5:11

33 For those who want to avoid doubling braces (`{{ }}`), use `string.Template` . There you substitute identifiers of the form `$foo` (handy for generating LaTeX code). – 0 _ Jan 11, 2018 at 5:51

3 For those who want to avoid doubling braces, and who are not averse to adding another dependency to their Python projects, there is also Jinja2 which definitively solves this problem, by allowing user-defined custom placeholder delimiter syntax. – dreftymac Jan 15, 2020 at 20:07

24 Answers

Sorted by: Highest score (default)

3497

You need to double the `{{` and `}}`:

```
>>> x = " {{ Hello }} {0} "
>>> print(x.format(42))
' { Hello } 42 '
```

Here's the relevant part of the Python documentation for format string syntax:

> Format strings contain “replacement fields” surrounded by curly braces `{}`. Anything that is not contained in braces is considered literal text, which is copied unchanged to the output. If you need to include a brace character in the literal text, it can be escaped by doubling: `{{` and `}}`.

Share

Improve this answer

Follow

edited Jan 17, 2020 at 0:14

user3064538

answered Mar 29, 2011 at 0:08

Greg Hewgill

990k ● 191 ● 1.2k ● 1.3k

400 So if you want to print "{42}", you'd use `"{{{0}}}".format(42)` ! – hughes Jul 24, 2013 at 20:21

22 What about if you want a single curly brace? `"{ something { } {value}".format(42)` doesn't work. – AJP Oct 2, 2013 at 10:10

28 "{{".format() and "}}".format() print single curly braces. In your example: print "{{ something {{ }} {0}".format(42) will print "{ something { } 42". – Mark Visser Oct 18, 2013 at 21:19

10 @Imray: `{0}` refers to the first argument to `.format()`. You can print more than one value like `{0} {1} {2}` as long as you give the same number of arguments to `.format()`. See docs.python.org/library/string.html#format-examples for extensive examples. – Greg Hewgill Feb 21, 2014 at 1:30

2 @Mithril: You still need to *double* each curly brace. So you might need `f'/blah/blah/{{{{ strftime(...) }}}/blah'` or something. – Greg Hewgill Aug 28, 2019 at 10:06

297

Python 3.6+ (2017)

In the recent versions of Python one would use f-strings (see also PEP498).

With f-strings one should use double `{{` or `}}`

```
n = 42
print(f" {{Hello}} {n} ")
```

produces the desired

```
 {Hello} 42
```

If you need to resolve an expression in the brackets instead of using literal text you'll need three sets of brackets:

```
hello = "HELLO"
print(f"{{{hello.lower()}}}")
```

produces

```
{hello}
```

Share

Improve this answer

Follow

edited Aug 15, 2018 at 0:02

Bryan Bryce

1,379 ● 1 ● 17 ● 30

answered Dec 21, 2017 at 11:12

divenex

17.1k ● 10 ● 62 ● 58

2 From `my_greet = "HELLO"` you can get `{hello}` as output, using just 2 sets of brackets, with `print(f"{ {my_greet.lower()} }")`. Just leave a space between brackets. – yrnr Jul 20, 2020 at 7:07

1 This should now be the accepted answer if you are using Python3.6+ in the times of the rona. – Gwi7d31 Nov 20, 2020 at 18:27

1 @Gwi7d31 No, f-strings are not a replacement for `str.format()`. For example, this answer I wrote is not possible with f-strings since the template is coming from input, not source code. – wjandrea May 20, 2021 at 18:20

1 @wjandrea your link really doesn't pertain to the OPs question. The OP wants to keep curly-braces while you are removing them in your linked answer via .format() and your dictionary unpacking method. If you want to preserve {} in Python 3.6+ and you want to insert a value into a string, `this is the way`. That's the question at hand. I also never said f-strings are a replacement for .format(). You said that. – Gwi7d31 May 26, 2021 at 3:06

1 So basically just keep adding curly brackets until it works. Got it. – Patrick Sep 18, 2021 at 4:52

120

You escape it by doubling the braces.

Eg:

```
x = "{{ Hello }} {0}"
print(x.format(42))
```

Share

Improve this answer

Follow

edited Jan 17, 2020 at 0:14

user3064538

answered Mar 29, 2011 at 0:08

Kamil Kisiel

20.4k ● 12 ● 49 ● 57

104

You want to format a string with the character `{` or `}`

You just have to double them.

format `{` with `f'{{'` and `}` with `f'}}'`

So :

```
name = "bob"
print(f'Hello {name} ! I want to print }} and {{ or {{ }}')
```

Output :

> Hello bob ! I want to print } and { or { }

OR for the exact example :

```
number = 42
print(f'{{Hello}} {number}')
```

Will print :

> {Hello} 42

Finally :

```
number = 42
string = "bob"
print(f'{{Hello}} {{{number}}} {number} {{{string}}} {string} ')
```

> {Hello} {42} 42 {bob} bob

Share

Improve this answer

Follow

edited Feb 21, 2023 at 15:29

answered Sep 24, 2021 at 20:37

Nicoolasens

3,538 ● 2 ● 19 ● 22

This answer is just a duplicate. – Akaisteph7 Jul 19 at 13:43

The OP wrote this comment:

72

> I was trying to format a small JSON for some purposes, like this: `'{"all": false, "selected": "{}"}'.format(data)` to get something like `{"all": false, "selected": "1,2"}`

It's pretty common that the "escaping braces" issue comes up when dealing with JSON.

I suggest doing this:

```
import json
data = "1,2"
mydict = {"all": "false", "selected": data}
json.dumps(mydict)
```

It's cleaner than the alternative, which is:

```
'{{"all": false, "selected": "{}"}}'.format(data)
```

Using the `json` library is definitely preferable when the JSON string gets more complicated than the example.

Share

Improve this answer

Follow

edited Jun 7, 2019 at 15:41

Nick T

26.6k ● 14 ● 87 ● 127

answered Jun 6, 2016 at 18:59

twasbrillig

18.8k ● 9 ● 44 ● 69

1 Amen! It might seem like more work, but using libraries to do what libraries are supposed to do versus cutting corners...makes for better things. – Kaolin Fire Apr 10, 2018 at 19:12

1 But the order of the keys in a Python object isn't guaranteed... Still, the JSON library is guaranteed to serialise in a JSON way. – wizzwizz4 Jun 30, 2018 at 8:34

2 wizzwizz4: Good point. From Python 3.6 onward, dictionaries are insertion ordered, so it wouldn't be an issue. Versions of Python between 2.7 and 3.5 can use OrderedDict from the collections library. – twasbrillig Jul 1, 2018 at 18:47

1 The alternative is also terribly wrong if, e.g., `data = 'foo"'`, because the `"` in the value of `data` won't be properly escaped. – chepner Dec 31, 2020 at 21:42

1 If you're dealing with JSON, this answer is for you. It wins in terms of readability and maintainability - imagine dealing with complex JSON structures and a lot of double braces in it – panK Oct 16, 2021 at 21:00

41

Try this:

`x = "{{ Hello }} {0}"`

Share Improve this answer Follow

answered Mar 29, 2011 at 0:08

pajton

16.2k ● 8 ● 58 ● 65

33

Try doing this:

```
x = " {{ Hello }} {0} "
print x.format(42)
```

Share Improve this answer Follow

answered Mar 29, 2011 at 0:08

DNR

986 ● 6 ● 10

17

Although not any better, just for the reference, you can also do this:

```
>>> x = '{}Hello{} {}'
>>> print x.format('{','}',42)
{Hello} 42
```

It can be useful for example when someone wants to print `{argument}`. It is maybe more readable than `'{{{}}}'.format('argument')`

Note that you omit argument positions (e.g. `{}` instead of `{0}`) after Python 2.7

Share Improve this answer Follow

answered Nov 5, 2015 at 11:22

George Aprilis

1,200 ● 1 ● 9 ● 24

17

```
key = "FOOBAR"
print(f"hello {{{key}}}")
```

outputs

```
hello {FOOBAR}
```

In case someone wanted to print something inside curly brackets using fstrings.

Share Improve this answer Follow

answered Dec 23, 2021 at 17:49

defiant

17

f-strings (python 3)

You can **avoid** having to double the curly brackets by using f-strings **ONLY** for the parts of the string where you want the f-magic to apply, and use regular (dumb) strings for everything that is literal and might contain 'unsafe' special characters. Let python do the string joining for you simply by stacking multiple strings together.

```
number = 42
print("{The answer is} "
f"{number}"
" {thanks for all the fish}")

### OUTPUT:
{The answer is} 42 {thanks for all the fish}
```

> NOTE: Line breaks between the strings are *optional*. I have only added them for readability. You could as well write the code above as shown below:
>
> ⚠️ WARNING: *This might hurt your eyes or make you dizzy!*
>
> ```
> print("{The answer is} "f"{number}"" {thanks for all the fish}")
> ```

Share

Improve this answer

Follow

edited Aug 7 at 8:13

answered Jul 6, 2021 at 21:55

ccpizza

31.5k ● 23 ● 181 ● 182

3 Implicit string concatenation is discouraged. Guido copied it from C but the reason why it's needed there doesn't really apply to Python. - groups.google.com/g/python-ideas/c/jP1YtlyJqxs?pli=1 – ankit Aug 5, 2022 at 14:12

2 @ankit That's effectively a dead link now. – Donal Fellows Jul 31 at 13:33

10

If you need to keep two curly braces in the string, you need 5 curly braces on each side of the variable.

```
>>> myvar = 'test'
>>> "{{{{{0}}}}}".format(myvar)
'{{test}}'
```

Share Improve this answer Follow

answered Jun 24, 2019 at 19:55

Richard
2,605 ● 25 ● 27

4 For those using f-strings, use 4 curly braces on either side instead of 5 – TerryA Sep 19, 2019 at 7:11

1 @TerryA there isn't a difference in brace behavior between .format and f-strings. The code `a = 1; print('{{{{{a}}}}}'.format(a=a))` produces the same results as `a = 1; print(f'{{{{{a}}}}}')`. – Kerrick Staley Jun 10, 2021 at 19:07

7

If you are going to be doing this a lot, it might be good to define a utility function that will let you use arbitrary brace substitutes instead, like

```
def custom_format(string, brackets, *args, **kwargs):
    if len(brackets) != 2:
        raise ValueError('Expected two brackets. Got
{}.'.format(len(brackets)))
    padded = string.replace('{', '{{').replace('}', '}}')
    substituted = padded.replace(brackets[0], '{').replace(brackets[1], '}')
    formatted = substituted.format(*args, **kwargs)
    return formatted

>>> custom_format('{{[cmd]} process 1}', brackets='[]', cmd='firefox.exe')
'{{firefox.exe} process 1}'
```

Note that this will work either with brackets being a string of length 2 or an iterable of two strings (for multi-character delimiters).

Share
Improve this answer
Follow

edited Dec 1, 2016 at 21:32

answered Nov 29, 2016 at 23:43

tvt173
1,866 ● 21 ● 18

Thought about that also. Of course, that will work too and the algorithm is simpler. But, imagine you have a lot of text like this, and you just want to parameterize it here and there. Everytime you create an input string you wouldn't want to replace all those braces manually. You would just want to 'drop in' your parameterizations here and there. In this case, I think this method is both easier to think about and accomplish from a user perspective. I was inspired by linux's 'sed' command which has similar capabilities to arbitrarily choose your delimiter based on what is convenient. – tvt173 Dec 1, 2016 at 18:53

In short, I'd rather have the utility function be a little more complex than have it be a pain in the @$$ to use everytime. Please let me know if I misunderstood your proposition. – tvt173 Dec 1, 2016 at 18:56

I've gone ahead and added a short demo to my public.lab space github.com/dreftymac/public.lab/blob/master/topic/python/… – dreftymac Dec 1, 2016 at 20:56

5

I recently ran into this, because I wanted to inject strings into preformatted JSON. My solution was to create a helper method, like this:

```
def preformat(msg):
    """ allow {{key}} to be used for formatting in text
    that already uses curly braces.  First switch this into
    something else, replace curlies with double curlies, and then
    switch back to regular braces
    """
    msg = msg.replace('{{', '<<<').replace('}}', '>>>')
    msg = msg.replace('{', '{{').replace('}', '}}')
    msg = msg.replace('<<<', '{').replace('>>>', '}')
    return msg
```

You can then do something like:

```
formatted = preformat("""
    {
        "foo": "{{bar}}"
    }""").format(bar="gas")
```

Gets the job done if performance is not an issue.

Share Improve this answer Follow

answered Dec 17, 2019 at 9:37

Kristján Valur

157 ● 2 ● 2

Simple AND elegant to integrate into existing code with little modification required. Thanks! – Column01 May 12, 2020 at 16:22

of course, assuming your text never contained any `<<<` and `>>>` to begin with, otherwise those would get overwritten. best to use escape strategies for reliability! – axolotl Aug 25, 2021 at 15:39

What escape strategy do you suggest? Anyway, you know your templated text and can amend the magic strings in case you worry about clashes. – Kristján Valur Aug 27, 2021 at 12:38

This approach is highly flexible and is even more workable by just borrowing a convention from Perl. Simply 1) choose a sequence of characters that you know will not appear in the target text (eg `QQX`); 2) use that sequence as your variable placeholders `QQX<name>`; 3) do a regex replacement to convert your placeholders to the expected format (eg) `r'QQ<([^>]+)>','{$1}'`. **See also:** perl quotelike operator – dreftymac Mar 28, 2023 at 23:17

4

Reason is , `{}` is the syntax of `.format()` so in your case `.format()` doesn't recognize `{Hello}` so it threw an error.

you can override it by using double curly braces {{}},

```
x = " {{ Hello }} {0} "
```

or

try `%s` for text formatting,

```
x = " { Hello } %s"
print x%(42)
```

Share Improve this answer Follow

answered Jun 27, 2017 at 12:40

Mohideen bin Mohammed

20.1k ● 11 ● 118 ● 128

This is so simple. I always forget about percent-style formatting. It is still valid in all Python versions, and it makes my quick and dirty test with a JSON string that needs number formatting so much cleaner. – Matthew May 12, 2023 at 16:52

4

I am ridiculously late to this party. I am having success placing the brackets in the replacement element, like this:

```
print('{0} {1}'.format('{hello}', '{world}'))
```

which prints

```
{hello} {world}
```

Strictly speaking this is not what OP is asking, as s/he wants the braces in the format string, but this may help someone.

Share Improve this answer Follow

answered Dec 11, 2020 at 19:05

Puddles

130 ● 7

1 Even `print("{} {}".format("{hello}", "{world}"))` works – ankit Aug 5, 2022 at 13:36

2

Here is another solution for it. Why it's great:

1. Allows you to keep the curly brackets originally single.

2. No exceptions if a variable is not found.

It's just this simple function:

```
def format_string_safe(text:str, **kwargs):
    if text is None:
        return ''
    result = text
    try:
        for k in kwargs:
            result = result.replace(f'{{{k}}}', str(kwargs[k]))
        return result
    except Exception as ex:
        print_traceback(ex)
    return text
```

How to use:

```
format_string_safe('THis is my text with some {vars}.', vars=22)
```

Share Improve this answer Follow

answered May 10 at 16:15

mimic

5,207 ● 7 ● 62 ● 104

1

I used a double {{ }} to prevent fstring value injection,

for example, heres my Postgres UPDATE statement to update a integer array column that takes expression of {} to capture the array, ie:

ports = '{100,200,300}'

with fstrings its,

```
ports = [1,2,3]

query = f"""
   UPDATE table SET ports = '{{{ports}}}' WHERE id = 1
"""
```

the actual query statement will be,

```
UPDATE table SET ports = '{1,2,3}'
```

which is a valid postgres satement

Share Improve this answer Follow

answered Jun 15, 2021 at 2:14

perfecto25
842 ● 11 ● 13

0

I stumbled upon this problem when trying to print text, which I can copy paste into a Latex document. I extend on this answer and make use of named replacement fields:

Lets say you want to print out a product of mulitple variables with indices such as $A_{0042} * A_{3141} * A_{2718} * A_{0042}$, which in Latex would be `$A_{ 0042 }*A_{ 3141 }*A_{ 2718 }*A_{ 0042 }$` The following code does the job with named fields so that for many indices it stays readable:

```
idx_mapping = {'i1':42, 'i2':3141, 'i3':2178 }
print('$A_{{ {i1:04d} }} * A_{{ {i2:04d} }} * A_{{ {i3:04d} }} * A_{{ {i1:04d} }}$'.format(**idx_mapping))
```

Share

Improve this answer

Follow

edited Dec 19, 2019 at 9:39

answered Nov 19, 2019 at 18:56

v.tralala
1,685 ● 3 ● 21 ● 41

0

You can use a "quote wall" to separate the formatted string part from the regular string part.

From:

```
print(f"{Hello} {42}")
```

to

```
print("{Hello}"f" {42}")
```

A clearer example would be

```
string = 10
print(f"{string} {word}")
```

Output:

```
NameError: name 'word' is not defined
```

Now, add the quote wall like so:

```
string = 10
print(f"{string}"" {word}")
```

Output:

```
10 {word}
```

Share Improve this answer Follow

answered Dec 31, 2020 at 21:13

Red

27.5k ● 7 ● 42 ● 63

1 This looks more like concatenation, but nice thinking – Delrius Euphoria Jun 14, 2021 at 6:49

1 I would advise against this - it's using a feature of the language which is itself controversial and described by Guido as a mistake (implicit string concatenation) and using it in a way that is, itself, unusual and therefore confusing. Many people who hit this are going to struggle to work out what is going on. It's essentially just going f"{string}" + " {word}" which is simple and straightforward but doing so in a more confusing way. It reminds me of the fake 'getting the single element of a set operator' ,= as used in elem ,= {'single_element'} which works but just causes confusion! – Andrew McLeod Jun 25, 2021 at 14:31

0

If you want to print just one side of the curly brace:

```
a=3
print(f'{"{"}{a}')
>>> {3
```

Share Improve this answer Follow

answered Jul 25, 2021 at 15:35

DalyaG

3,097 ● 3 ● 20 ● 23

Unnecessary, doubling the `{` as explained in the top answer is still sufficient. So `f'{{{a}'`. – user202729 Jun 11, 2022 at 4:47

-1

If you want to **only** print one curly brace (for example `{`) you can use `{{`, and you can add more braces later in the string if you want. For example:

```
>>> f'{{ there is a curly brace on the left. Oh, and 1 + 1 is {1 + 1}'
'{ there is a curly brace on the left. Oh, and 1 + 1 is 2'
```

Share Improve this answer Follow

answered Mar 23, 2020 at 1:32

Parzibyte
1,588 ● 13 ● 21

-2

When you're just trying to interpolate code strings I'd suggest using jinja2 which is a full-featured template engine for Python, ie:

```
from jinja2 import Template

foo = Template('''
#include <stdio.h>

void main() {
    printf("hello universe number {{number}}");
}
''')

for i in range(2):
    print(foo.render(number=i))
```

So you won't be enforced to duplicate curly braces as the whole bunch of other answers suggest

Share Improve this answer Follow

answered Apr 2, 2020 at 11:00

BPL
9,833 ● 12 ● 64 ● 130

1 I agree that avoiding duplicating curly braces is a good thing — but rather than reach for jinja2 I'd just use python's own `string.Template` class, which is plenty powerful enough for this kind of thing. – gimboland Aug 3, 2021 at 16:06

-2

If you need curly braces within a f-string template that can be formatted, you need to output a string containing two curly braces within a set of curly braces for the f-string:

```
css_template = f"{{tag}} {'{'} margin: 0; padding: 0;{'}'}"
for_p = css_template.format(tag="p")
# 'p { margin: 0; padding: 0;}'
```

Share Improve this answer Follow

answered Jan 28, 2021 at 9:40

RunOrVeith
4,805 ● 5 ● 35 ● 51

1 this question was not about f-strings, and IMO combining f-strings and format in this way makes for pretty unreadable code – avigil Feb 25, 2021 at 17:07

It is the first result that comes up when you google how to put curly braces in python f-strings though, and yes I agree it's not pretty but sometimes you just need it. – RunOrVeith Feb 26,

2021 at 8:28

Or just parametrize the bracket itself? Probably very verbose.

-3

```
x = '{open_bracket}42{close_bracket}'.format(open_bracket='{',
close_bracket='}')
print(x)
# {42}
```

Share Improve this answer Follow

answered Apr 21, 2021 at 17:10

Mortz

4,869 ● 1 ● 22 ● 37

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.