x Dismiss

Join the Stack Overflow Community

Stack Overflow is a community of 7.2 million programmers, just like you, helping each other. Join them; it only takes a minute:

Sign up

String literal with triple quotes in function definitions

I am following the Python tutorial and at some point they talk about how the 1st statement of a function can be a String Literal. As far as the example goes, this String Literal seems to be done with three `"` s, giving in the example

```
"""Print a Fibonacci series up to n."""
```

According to this documentation, this would be used mainly to create some kind of automatically produced documentation.

So I am wondering if someone here could explain to me what are these string literals exactly?

python string literals

edited Apr 1 at 17:48
vaultah
21.7k 8 48 78

asked May 31 '12 at 19:52
Estarius
471 3 8 20

With all these answers, I realized that one of my main issue with this was the comparison of normal strings with string literal. If I understood well, the major difference is that normal string are associated to a variable while string literal are more "floating" ? – Estarius May 31 '12 at 20:05

1 google.ca/search?q=define%3A"string+literal"; Seriously, Google is much more powerful than people give it credit for. – Karl Knechtel May 31 '12 at 23:32

Actually, *any* statement in a function can be a string literal, or a standalone integer, for that matter (2 or 2834329 can be considered an "integer literal"). It is syntactically correct; it just won't be doing anything. As an example of a different convention, if you are programming in Groovy (instead of Python), the value of the last statement will be returned as the return value; so there people sometimes end a function with just e.g. `1` on a separate line. – osa Feb 8 '15 at 22:02

String Literal: Well `string` is some text and `literal` maybe as in "take it literally", e.g. don't interpret this as code. – Dennis Kuypers Jan 27 at 9:47

6 Answers

What you're talking about (I think) are called docstrings (Thanks Boud for the link).

```
def foo():
    """This function does absolutely nothing"""
    pass
```

Now, if you type `help(foo)` from the interpreter, you'll get to see the string that I put in the function. You can also access that string by `foo.__doc__`

Of course, string literals are just that -- literal strings.

```
a = "This is a string literal"  #the string on the right side is a string literal, "a" is
a string variable.
```

or

```
foo("I'm passing this string literal to a function")
```

They can be defined in a bunch of ways:

```
'single quotes'
"double quotes"
""" triple-double quotes """  #This can contain line breaks!
```

or even

```
#This can contain line breaks too!  See?
''' triple-single
    quotes '''
```

edited Apr 2 '16 at 2:42

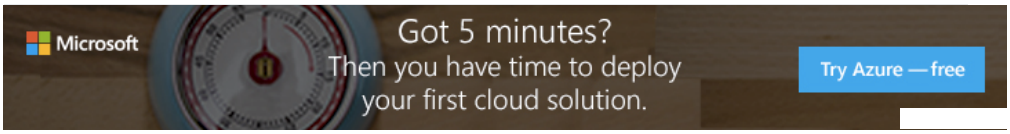
answered May 31 '12 at 19:56

mgilson

171k 27 294 405

add this: docs.python.org/tutorial/controlflow.html#documentation-strings – Boud May 31 '12 at 20:02

@Boud -- Thanks for the link. I've added it. – mgilson May 31 '12 at 20:06



Just to note: this is a somewhat basic answer. Even if you know much of what I say, I bet some pretty basic explanations can make the answer a good, more generic reference.

Strings, expressions and literals

In a program, we have to represent various *types* of data. One *type* of data are integer numbers; other type are floating pint numbers. A very important type of data is text, a sequence of letters, numbers and other characters. This type is usually called *string*.

A value of some type can be yielded by various ways. For example, the Python expression below yields the value 4 and put it in a variable. The value was yielded by the *expression* `2+2`:

```
i = 2+2
```

Given the expression above, the *expression* below yields the same value 4, but now from the *variable*:

```
i
```

Below, I generated a value by an *expression*, and retrieved it by a *variable*.

Languages, however, should provide a syntax to yield basic values directly. For example, the `2` in the expression above retrieves the value 2. Those expressions which yields basic values directly are called *literals*. Both expressions `2+2` and `4` yield the same value, 4, but the second expression yields it directly, if you know what I mean, so it is a literal.

String literals and multiline strings

A *string literal*, in this way, is a literal which yields a string. In Python, those literals are marked by many ways. Two ways are to put a single or double quote at either the beginning or the end of the literal:

```
"A string literal"
```

```
'Another string literal'
```

Other ways are to put three single or double quotes in the same positions. In this case, the literall can span through multple lines:

```
"""A single line string literal"""
```

```
"""A multiline
string literal"""
```

```
'''Another multiline
string literal'''
```

Note that the method of generating a string literal does not change its value. A single-quoted string is equal to a double-quoted string with the same characters, and a three-quote string is equal to a one-quote string with the same content:

```
"A single line string literal" == 'A single line string literal'

"""A single line string literal""" == "A single line string literal"

"A multiline\nstring literal" == """A multiline
string literal""" # \n is the character that represents a new line
```

Docstrings and why should they be string literals

What the documentation is saying is that you can put a string literal just after the method declaration and this literal will be used as documentation - what we use to call a *docstring*. It does not matter if you use single- or double-quoted strings, or one- or three-quote strings either. Consider the functions below:

```
def f1(value):
    "Doc for f1"
    return value + 1

def f2(value):
    """Doc for f2"""
return value + 2
```

Now, declare them in your python console and call `help(f1)` and `help(f2)` . Note that the declaration of the string does not matter. OTOH, you cannot use other expressions, such as variables or operations over strings, for generating your documentation. So the strings at the first line of the functions below *are no docstring*:

```
mydoc = "This is doc"
def f3(value):
     mydoc
     return value+3

 def f4(value):
     "This is no documentation " + "because it is concatenated"
     return value+4
```

It should be a literal because the compiler is prepared to manage it as documentation. However, the compiler is not prepared to manage variables, complex expressions etc. as documentation, so it will ignore them.

Why use triple quote strings as docstrings?

Although any form of string literal can be used in docstrings, you may consider that documentation usually includes very long texts, with multiple lines and paragraphs. Well, since it includes various lines, one is well advised to use the literal forms which accept multiple lines, right? This is the reason why triple-quote strings are the preferred (but not mandatory) way of writing docstrings.

A marginal note

Actually, you can put a string literal in any place of a Python function:

```
def flying_literals(param):
   "Oh, see, a string literal!"
   param += 2
   "Oh, see, ANOTHER string literal!"
   return param
   "the above literal is irrelevant, but this one can be still MORE IRRELEVANT"
```

However, only the literal a the first line makes some difference (being the documentation). The other ones are like no-op operations.

edited Apr 24 at 11:44

answered May 31 '12 at 20:25

brandizzi

15.8k 4 66 118

Sorry for such comment, but this is one of the most comprehensive and complete answers to this topic. The marginal note was what I was looking for. Thanks! – Dawid Ferenczy Jan 26 at 13:44

A string literal is simply a string given literally in the source code. Whether it is a docstring or another string does not matter. See the Python language documentation section on string literals for all the details, but you probably don't need these details now.

A few examples:

```
"abc"
'Guido'
r"""Norwegian Blue"""
```

answered May 31 '12 at 19:56

Sven Marnach
254k 48 626 624

A string literal is a string in one of the many quoting options, that is not assigned to a variable.

So,

```
"String" # string literal
'string' # string literal
"""
  Multiline
  String
  Literal
"""
foo = "string variable"
```

When you have a string literal immediately after a `def` block, it becomes part of the documentation for that method, and is called a *docstring*

```
def foo(hello):
    """This is part of the documentation for foo"""
```

This is how you would use it:

```
>>> def foo(hello):
...     """This is the docstring"""
...     pass
...
>>> foo.__doc__
'This is the docstring'
```

answered May 31 '12 at 19:57

Burhan Khalid
98k 10 108 165

In python there are several ways to divide strings to multiple lines. Strings literals is one of them, for example:

```
s = """Hello,
    world"""
print(s)
>>> Hello,
>>>     world #Notice, that spaces used in the literal are kept.
```

But as you noticed correctly, string literals are usually there for the in-line documentation

```
class MyClass(object):
    """This is my class it does this and that.

       It has some cool features and I may tell you about them below.
    """

    def my_method(self):
        """This is a brief description of my method."""

    def important_method(self):
        """Because this method is important, I'm going to tell you
           a lot about it. For example...
        """
```

Before you ask, a good way to split strings on multiple lines, is the holy Python parenthesis:

```
s = ('This is a very very long string. '
     'I have to split it to multiple lines. '
     'Whoa! It works!')
print(s)
>>> This is a very very long string. I have to split it to multiple lines. Whoa! It works!
```

You may need this to follow the PEP-8, which states that "Thou shalt not exceed 80 characters per line".

Happy Python hacking!

answered May 31 '12 at 20:05

Zaur Nasibov
12.2k 8 30 63

1 +1 for your example of automagic string concatenation. – mgilson May 31 '12 at 20:11

They are strings like any other strings with pairs of `'`, `"`, `'''` or `"""` around them.
The recommended form is the triple double quote:

```
def some_function(s):
    """this is documentation for some_function"""
    print(s)
```

edited May 31 '12 at 20:02

Martijn Pieters ♦
559k 87 1653 1709

answered May 31 '12 at 19:57

Anthon
16.8k 15 54 96