Theory: Docstrings

23 minutes 0 / 5 problems solved

Skip this topic Start practicing

150 users solved this topic. Latest completion was about 4 hours ago.

By now, you know why it is useful to supply your code with ***comments***. With their help, you can explain the inner logic of your code and clarify some non-obvious steps. Such notes are often intended for other maintainers and developers of your program. However, for those who will just use the functionality without getting into implementation details, you may want to provide a general description of what objects in your program are used for (or even for yourself to quickly remind you of the old objects' functionality). So, here you may appeal to writing a ***docstring***, the main unit of documenting in Python.

In this topic, we will observe what is considered documentation in Python. Moreover, docstrings have several Python Enhancement Proposals dedicated to them, here we will sum up the code documenting conventions of PEP 257 and PEP 287.

§1. What is a docstring?

Docstring (*documentation string*) is a string literal. It is written as the first statement in the definition of a module, a class, a method, a function, etc., and briefly describes its behavior and how you can use it, for one, what parameters you should pass to the function.

Let's take one example right away! Python has the built-in module **statistics**. It contains a number of functions for calculating data statistics, and all of them are documented. For example, below, we show the source code for the function `median()` , and everything within `"""triple double-quotes"""` is actually the docstring describing the function.

```
def median(data):
    """Return the median (middle value) of numeric data.

    When the number of data points is odd, return the middle data point.
    When the number of data points is even, the median is interpolated by
    taking the average of the two middle values:

    >>> median([1, 3, 5])
    3

    >>> median([1, 3, 5, 7])

    4.0



    """

    data = sorted(data)

    n = len(data)

    if n == 0:

        raise StatisticsError("no median for empty data")

    if n%2 == 1:

        return data[n//2]

    else:

        i = n//2

        return (data[i - 1] + data[i])/2
```

Current topic:

Docstrings

Topic depends on:

✓ Else statement Stage 1 15
✓ Create module Stage 1
✓ Methods Stage 1 3

Table of contents:

↑ Docstrings
§1. What is a docstring?
§2. Types of docstrings
§3. One-line docstrings for functions and methods
§4. Multi-line docstrings for functions and methods
§5. Docstrings for classes and modules
§6. The help() function
§7. Conclusion
Feedback & Comments

The docstring here contains a description of what the function does and its expected behavior towards values passed to it, accompanied by usage examples. Note that *triple double-quotes* are the conventional punctuation signs to indicate a docstring in Python, and the annotation should start with a capital letter and end with a period, as recommended by PEP 257. What is more, just like a comment, each line in a docstring should be no longer than 72 characters.

Importantly, we can access docstrings without reading the source code itself: either by calling the `help()` function on the object or by using the `__doc__` attribute:

```
import statistics

statistics.median.__doc__  # the same as: help(statistics.median)
# Return the median (middle value) of numeric data.
#
#    When the number of data points is odd, return the middle data point.
#    When the number of data points is even, the median is interpolated by
#    taking the average of the two middle values:
#

#    >>> median([1, 3, 5])

#    3

#    >>> median([1, 3, 5, 7])

#    4.0
```

When using `help()`, you can access a docstring for the object even without importing it. To do so, you just need to pass its name in quotes: for example, `help('statistics.median')`.

Now that you know what documentation strings generally are, let's move further and learn more about them.

§2. Types of docstrings

The two main types of documentation strings in Python are *one-liners* and *multi-liners*. In the previous example, the docstring for the function `median()` is a multi-line docstring; however, the very first line of it can be regarded as a one-liner itself:

```
# Return the median (middle value) of numeric data.
```

One-line docstrings are a sort of quick summary for your object, it is the minimum description it should have. Ideally, they are easy to understand for any person who uses something in your program for the first time. Generally, it is better to provide a multi-line description, but in some obvious cases that don't require further explanation, one-liners are acceptable.

So, multi-line docstrings should contain a more detailed description. For example, the docstring in `median` included the synopsis of the function's behavior and two use cases. Overall, the structure of multi-line docstrings can be summed up as:

- A brief *one-line* description of the object's purpose;
- A more elaborate explanation of the functionality, for instance, a list of classes a module has or usage examples.

Now, let's learn to write docstrings ourselves!

§3. One-line docstrings for functions and methods

First, let's create a small example with a one-line string. Below, we declare the function `count_factorial()` and specify what it does in triple double-quotes right after the declaration:

```python
def count_factorial(num):
    """Return the factorial of the number."""
    if num == 0:
        return 1
    else:
        return num * count_factorial(num - 1)
```

In accordance with PEP 257, you should follow the next conventions for docstrings for functions and methods:

- The opening and the closing quotes should be on the same line.
- There should be *no* empty strings either before or after the docstring.
- Your description should be in imperative style: hence, we write """*Return the factorial.*""" or """*Return the number.*""" instead of """*Returns the number.*""" or """*It returns the number.*""".
- The description ***shouldn't*** be just a scheme that repeats the object's parameters and return values, eg. `"""count_factorial(num) -> int."""`.

As in the example with `median()`, we can access the annotation via `__doc__`.

```python
print(count_factorial.__doc__)
# Return the factorial of the number.
```

If you have backslashes in your docstring, you should also wrap it with the `r` prefix, for example, as in `r"""A \new example with \triple double-quotes."""`. Otherwise, the combination of a backslash and a letter will be handled as an escape sequence.

§4. Multi-line docstrings for functions and methods

Now, let's create a bit more elaborate description of the function's behavior. Following the general structure above, we start by leaving the first line unchanged: it continues to be the main summary of our function. Next, the multi-line docstring for a method or a function should include information about the arguments, return values, and other points concerning it. Thus, in the example below, we indicate what the argument `num` and its data type should be, what value the function returns, and what this return value denotes:

```python
def count_factorial(num):
    """Return the factorial of the number.

    Arguments:
    num -- an integer to count the factorial of.
    Return values:
    The integer factorial of the number.
    """
    if num == 0:
        return 1
    else:
        return num * count_factorial(num-1)
```

As for style conventions, it is worth noting the following three things:

1. The summary is separated from the detailed description using a single blank line.
2. The docstring in the example starts right after the triple double-quotes. However, it is also possible to specify them on the next line after the opening quotes:

   ```python
   def count_factorial(num):
         """
         Return the factorial of the number.

         The rest of the doctsring.
         """
   ```

3. The detailed description starts at the same position as the first quote of the first docstring line: there's no indent.

§5. Docstrings for classes and modules

Now, we will turn to class and module docstrings. PEP 257 proposes the following conventions:

- *Module docstrings* should first also provide a brief one-line description. After that, it is recommended to specify all classes, methods, functions, or any other of the module's objects.
- In *class docstrings*, apart from the general purpose of the class, you should indicate the information about methods, instance variables, attributes, and so forth. Nevertheless, all these individual objects should still have their own docstrings, with more thorough information given.

To practice this, let's create an example summing up both cases. Below, we briefly visualize the docstrings of the class `Person`.

```python
# information.py module
"""The functionality for manipulating the user-related information."""


class Person:
    """The creation of the Person object and the related functionality."""

    def __init__(self, name, surname, birthdate):
        """The initializer for the class.

        Arguments:
        name -- a string representing the person's name.
        surname -- a string representing the person's surname.
        birthdate -- a string representing the person's birthdate.
        """
        self.name = name
        self.surname = surname
        self.birthdate = birthdate

    def calculate_age(self):
        """Return the current age of the person."""
        # the body of the method
```

First of all, note that the class constructor should be documented in the `__init__` method. Also, according to PEP 257, we should insert a blank line after a docstring for a class. It is done to dissociate the class documentation and the first method.

In the given example, we don't give a comprehensive annotation for the module and for the class, even though it is recommended. Why so? Imagine that you list *all* the objects that the `information.py` module contains in its docstring, and then, in its turn, *all* the objects the class `Person` contains in *its* docstring. In such a case, your annotation may get redundant: for example, you would have to repeat what the `calculate_age()` function does three times: in the module's annotation, in the class' one, and, finally, in the function's annotation. This is done so that just by looking at the docstring for the object we can get a deeper understanding of what it contains. However, when we use some documentation generating tools or the `help()` function and call them on the class, we are likely to get the synopsis of all its methods even without specifying them in the class docstring.

§6. The help() function

Perhaps, you dealt with the `help()` function before: it is used to access the documentation of the object. If you type this command without any arguments, an interactive help utility will start. To get the documentation of a particular module, class, method, etc, you simply need to pass it as the argument to this function. Take a look at the example below — there we learn the documentation for the class `Person`, defined in the *information.py* module.

```
help(Person)
# Help on class Person in module __main__:
#
# class Person(builtins.object)
#  |  The creation of the Person object and the related functionality.
#  |
#  |  Methods defined here:
#  |
#  |  __init__(self, name, surname, birthdate)
#  |      The initializer for the class.
#  |
#  |      Arguments:
#  |      name -- a string representing the person's name.
#  |      surname -- a string representing the person's surname.
#  |      birthdate -- a string representing the person's birthdate.
#  |
#  |  calculate_age(self)
#  |      Return the current age of the person.
```

As you can see, here we obtain the docstrings not only for the class itself but also for all of its objects.

§7. Conclusion

So far, in this topic, we overviewed the general recommendations concerning the style of writing docstrings in Python with respect to PEP 257. Though it is the prevalent style of designing docstrings, there are some other style guides: for example, you can check out Google-format docstrings.

Generally, docstrings can be used to describe the behavior of modules, classes, functions, and so forth. Python uses triple double-quotes `"""..."""` for them. The minimum description you are recommended to supply your code with is one-line docstrings. However, you can use multi-line docstrings to describe the features of your objects more thoroughly.

You can use the built-in function `help()` to access the documentation of the object. To do so, you just need to pass it as the argument to the function. Alternatively, you can call the `.__doc__` method on the object.

Now, let's practice your new knowledge!

Report a typo

20 users liked this theory. 0 didn't like it. **What about you?**

😍 🙂 😐 🙁 😡

Start practicing

This content was created about 2 months ago and updated 2 days ago. Share your feedback below in comments to help us improve it!

Comments (0) Hints (0) Useful links (0) Show discussion