Python Docstrings

GeeksforGeeks

When it comes to writing clean, well-documented code, Python developers have a secret weapon at their disposal – docstrings. Docstrings, short for documentation strings, are vital in conveying the purpose and functionality of Python functions, modules, and classes.

What are the docstrings in Python?

Python documentation strings (or docstrings) provide a convenient way of associating documentation with Python modules, functions, classes, and methods. It's specified in source code that is used, like a comment, to document a specific segment of code. Unlike conventional source code comments, the docstring should describe what the function does, not how.

Declaring Docstrings: The docstrings are declared using "'triple single quotes"' or """ triple double quotes """ just below the class, method, or function declaration. All functions should have a docstring.

Accessing Docstrings: The docstrings can be accessed using the __doc__ method of the object or using the help function. The below examples demonstrate how to declare and access a docstring.

What should a docstring look like?

The doc string line should begin with a capital letter and end with a period.

The first line should be a short description.

If there are more lines in the documentation string, the second line should be blank, visually separating the summary from the rest of the description.

The following lines should be one or more paragraphs describing the object's calling conventions, side effects, etc.

Docstrings in Python

Below are the topics that we will cover in this article:

Triple-Quoted Strings

Google Style Docstrings

Numpydoc Style Docstrings

One-line Docstrings

Multi-line Docstrings

Indentation in Docstrings

Docstrings in Classes

Difference between Python comments and docstrings

Triple-Quoted Strings

This is the most common docstring format in Python. It involves using triple quotes (either single or double) to enclose the documentation text. Triple-quoted strings can span multiple lines and are often placed immediately below the function, class, or module definition

Example 1: Using triple single quotes

Python3

Python3

```
def my_function():
    return None
print("Using __doc__:")
print(my_function.__doc__)
print("Using help:")
help(my_function)
```

Output:

```
Using __doc__:
Demonstrates triple double quotes
   docstrings and does nothing really.
Using help:
Help on function my_function in module __main__:
my_function()
   Demonstrates triple double quotes
   docstrings and does nothing really.
```

Example 2: Using triple-double quotes

Python3

Python3

```
def my_function():
    ' ''Demonstrates triple double quotes docstrings and does nothing
```

```
really' ' '
    return None
print("Using __doc__:")
print(my_function.__doc__)
print("Using help:")
help(my_function)
```

Output:

Using __doc__:
Demonstrates triple double quotes docstrings and does nothing really.
Using help:
Help on function my_function in module __main__:
my_function()
    Demonstrates triple double quotes
    docstrings and does nothing really.

Google Style Docstrings

Google style docstrings follow a specific format and are inspired by Google's documentation style guide. They provide a structured way to document Python code, including parameters, return values, and descriptions.

Python3

Python3

```
def multiply_numbers(a, b):
    return a * b
print(multiply_numbers(3,5))
```

Numpydoc Style Docstrings

Numpydoc-style docstrings are widely used in the scientific and data analysis community, particularly for documenting functions and classes related to numerical computations and data manipulation. It is an extension of Google-style docstrings, with some additional conventions for documenting parameters and return values.

Python3

Python3

```
def divide_numbers(a, b):
    if b == 0:
        raise ValueError("Division by zero is not allowed.")
    return a / b
print(divide_numbers(3,6))
```

One-line Docstrings

As the name suggests, one-line docstrings fit in one line. They are used in obvious cases. The closing quotes are on the same line as the opening quotes. This looks better for one-liners. For example:

Python3

Python3

```
def power(a, b):
    ' ''Returns arg1 raised to power arg2.' ' '
    return a**b
print(power.__doc__)
```

Output:

Returns arg1 raised to power arg2.

Multi-line Docstrings

Multi-line docstrings consist of a summary line just like a one-line docstring, followed by a blank line, followed by a more elaborate description. The summary line may be on the same line as the opening quotes or on the next line. The example below shows a multi-line docstring.

Python3

Python3

```
def add_numbers(a, b):
    return a + b
print(add_numbers(3, 5))
```

Output:

8

Indentation in Docstrings

The entire docstring is indented the same as the quotes in its first line. Docstring processing tools will strip a uniform amount of indentation from the second and further lines of the docstring, equal to the minimum indentation of all non-blank lines after the first line. Any indentation in the first line of the docstring (i.e., up to the first new line) is insignificant and removed. Relative indentation of later lines in the docstring is retained.

Python3

Python3

```
class Employee:
    def __init__(self, name, age, department, salary):
        self.name = name
        self.age = age
        self.department = department
        self.salary = salary
    def promote(self, raise_amount):
        self.salary += raise_amount
        return f"{self.name} has been promoted! New salary:
${self.salary:.2f}"
    def retire(self):
        return f"{self.name} has retired. Thank you for your service!"
help(Employee)
```

Output:

```
class Employee
 |  A class representing an employee.
 |
 |  Attributes:
 |      name (str): The name of the employee.
 |      age (int): The age of the employee.
 |      department (str): The department the employee works in.
 |      salary (float): The salary of the employee.
 |
 |  Methods defined here:
 |
 |  __init__(self, name, age, department, salary)
```

```
|     Initializes an Employee object.
|
|     Parameters:
|         name (str): The name of the employee.
|         age (int): The age of the employee.
|         department (str): The department the employee works in.
|         salary (float): The salary of the employee.
|
| promote(self, raise_amount)
|     Promotes the employee and increases their salary.
|
|     Parameters:
|         raise_amount (float): The raise amount to increase the salary.
|
|     Returns:
|         str: A message indicating the promotion and new salary.
|
| retire(self)
|     Marks the employee as retired.
|
|     Returns:
|         str: A message indicating the retirement status.
```

Docstrings in Classes

Let us take an example to show how to write docstrings for a class and the method '**help'** is used to access the docstring.

Python3

Python3

```
class ComplexNumber:
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag
    def add(self, num):
        re = self.real + num.real
```

```
        im = self.imag + num.imag
        return ComplexNumber(re, im)
help(ComplexNumber)
help(ComplexNumber.add)
```

Output:

```
Help on class ComplexNumber in module __main__:
class ComplexNumber(builtins.objects)
 | This is a class for mathematical operations on complex numbers.
 |
 | Attributes:
 |       real (int): The real part of complex number.
 |       imag (int): The imaginary part of complex number.
 |
 | Methods defined here:
 |
 | __init__(self, real, imag)
 |     The constructor for ComplexNumber class.
 |
 |     Parameters:
 |     real (int): The real part of complex number.
 |     imag (int): The imaginary part of complex number.
 |
 | add(self, num)
 |     The function to add two Complex Numbers.
 |
 |     Parameters:
 |           num (ComplexNumber): The complex number to be added.
 |
 |     Returns:
 |           ComplexNumber: A complex number which contains the sum.
Help on method add in module __main__:
add(self, num) unbound __main__.ComplexNumber method
    The function to add two Complex Numbers.

    Parameters:
          num (ComplexNumber): The complex number to be added.

    Returns:
```

ComplexNumber: A complex number which contains the sum.

Difference between Python comments, String, and Docstrings

String: In Python, a string is a sequence of characters enclosed within single quotes (‘ ‘) or double quotes (” “). Strings are used to represent text data and can contain letters, numbers, symbols, and whitespace. They are one of the fundamental data types in Python and can be manipulated using various string methods.

You all must have got an idea about Python docstrings but have you ever wondered what is the difference between Python comments and docstrings? Let’s have a look at them.

They are useful information that the developers provide to make the reader understand the source code. It explains the logic or a part of it used in the code. It is written using

#

symbols.

Example:

Python3

Python3

```
print("GFG")

name = "Abhishek Shakya"
```

Output:

GFG

Whereas Python Docstrings as mentioned above provides a convenient way of associating documentation with Python modules, functions, classes, and methods.

Last Updated : 07 Aug, 2023

Like Article

Save Article