DOCSTRINGS

1. Docstrings

00:00 - 00:18

Hi. My name is Shayne Miel. You've probably spent a lot of time using functions that someone else wrote. In this course, you'll learn how to write functions that others can use. Docstrings are a Python best practice that will make your code much easier to use, read, and maintain.

2. A complex function

00:18 - 00:31

Look at this split_and_stack() function. If you wanted to understand what the function does, what the arguments are supposed to be, and what it returns, you would have to spend some time deciphering the code.

3. A complex function with a docstring

00:31 - 00:44

With a docstring though, it is much easier to tell what the expected inputs and outputs should be, as well as what the function does. This makes it easier for you and other engineers to use your code in the future.

4. Anatomy of a docstring

00:44 - 01:15

A docstring is a string written as the first line of a function. Because docstrings usually span multiple lines, they are enclosed in triple quotes, Python's way of writing multi-line strings. Every docstring has some (although usually not all) of these five key pieces of information: what the function does, what the arguments are, what the return value or values should be, info about any errors raised, and anything else you'd like to say about the function.

5. Docstring formats

01:15 - 01:30

Consistent style makes a project easier to read, and the Python community has evolved several standards for how to format your docstrings. Google-style and Numpydoc are the most popular formats, so we'll focus on those.

6. Google Style - description

01:30 - 01:48

In Google style, the docstring starts with a concise description of what the function does. This should be in imperative language. For instance: "Split the data frame and stack the columns" instead of "This function will split the data frame and stack the columns".

7. Google style - arguments

01:48 - 02:15

Next comes the "Args" section where you list each argument name, followed by its expected type in parentheses, and then what its role is in the function. If you need extra space, you can break to the next line and indent as I've done here. If an argument has a default value, mark it as "optional" when describing the type. If the function does not take any parameters, feel free to leave this section out.

8. Google style - return value(s)

02:15 - 02:33

The next section is the "Returns" section, where you list the expected type or types of what gets returned. You can also provide some comment about what gets returned, but often the name of the function and the description will make this clear. Additional lines should not be indented.

9. Google-style - errors raised and extra notes

02:33 - 02:46

Finally, if your function intentionally raises any errors, you should add a "Raises" section. You can also include any additional notes or examples of usage in free form text at the end.

10. Numpydoc

02:46 - 03:06

The Numpydoc format is very similar and is the most common format in the scientific Python community. Personally, I think it looks better than the Google style. It takes up more vertical space though, so this course will either use Google-style or leave out the docstrings entirely to keep the examples compact and legible.

11. Retrieving docstrings

03:06 - 03:46

Sometimes it is useful for your code to access the contents of your function's docstring. Every function in Python comes with a __doc__ attribute that holds this information. Notice that the __doc__ attribute contains the raw docstring, including any tabs or spaces that were added to make the words line up visually. To get a cleaner version, with those leading spaces removed, you can use the getdoc() function from the inspect module. The inspect module contains a lot of useful methods for gathering information about functions.

12. Let's practice!

03:46 - 03:52

Now it's your turn to practice writing and retrieving docstrings.