Style Guide, Unit Testing, Packaging, Notes

Chuks Okoli

08 November, 2023

Python Style Guide Summary

To be consistent in coding, use:

- Use four spaces instead of tabs for indentation
- Use blank lines to seperate functions and classes in code
- Use spaces around operators and commas
- Use functions for blocks of code
- Name functions and files using lowercase with underscores
- Name python packages without underscore e.g. "mypackage"
- Name classes using CamelCase e.g. class DataViz
- Name constants by capitalizing all words and sepeerate words with underscore e.g. "MAX_FILE_UPLOAD_SIZE = 3.0"
- Use static code analysis to evaluate code against predefined guidelines

```
# Run in terminal
pip3 install pylint==2.11.1

# Create file for code analysis. Save as "static_code_analysis.py"
"""Module provides a function to add two numbers"""
def add(number1, number2):
    """
    Returns the sum of two numbers

    Args:
        number1 (int): The first number
        number2 (int): The second number

    Returns:
        int: The sum of number1 and number2
    """
    return number1 + number2

NUM1 = 4
NUM2 = 5
TOTAL = add(NUM1, NUM2)
print(f"The sum of {NUM1} and {NUM2} is {TOTAL}")

# Create a sample python file for static code analysis
pylint static_code_analysis.py

# Pylint goes through every line of code and gives you a list all the non-compliant
 ↪  lines.
```

```
# with a compliance score. Correct the mistakes identified by pylint.
```

Unit Testing

Unit testing is a method to validate if units of code are operating as designed. A unit is a smaller, testable part of an application. To develop unit tests, you will use the unittest library, which is an installed Python module providing a framework containing test functionality. Unit testing is a method to validate if code units are operating as designed. You must test every unit before integration with the final codebase.

```
# Create mymodule.py
def square(number):
    """
    This function returns the square of a given number
    """
    return number ** 2

def double(number):
    """
    This function returns twice the value of a given number
    """
    return number * 2

# Create test_mymodule.py
import unittest

from mymodule import square, double

class TestSquare(unittest.TestCase):
    def test1(self):
        self.assertEqual(square(2), 4) # test when 2 is given as input the output is 4.
        self.assertEqual(square(3.0), 9.0)  # test when 3.0 is given as input the output
         ↪  is 9.0.
        self.assertNotEqual(square(-3), -9)  # test when -3 is given as input the output
         ↪  is not -9.


class TestDouble(unittest.TestCase):
    def test1(self):
        self.assertEqual(double(2), 4) # test when 2 is given as input the output is 4.
        self.assertEqual(double(-3.1), -6.2) # test when -3.1 is given as input the
         ↪  output is -6.2.
        self.assertEqual(double(0), 0) # test when 0 is given as input the output is 0.

unittest.main()
```

Packaging

A Python module is a .py file containing Python definitions, statements, functions, and classes. You can import a module to other scripts and notebooks. For example, consider a module named module.py that has two functions. The first function is def square, which squares the input and returns the result as an output. The second function is def doubler, which doubles the input and returns the result as an output. If the function is in the same directory, you can import it and use the functions in that module.

```python
# math_module.py
def square(number):
   return number ** 2

def doubler(number):
    return number * 2

# Call the math_module
from math_module import square, doubler
print("4^2=", square(4))
print("2*4=", doubler(4))
```

A package is a collection of python modules into a dictionary with a "___init___.py" file, which distinguishes it from just a dictionary of python scripts. To create a package "myproject" in parent_directory with two modules, math_module.py and stats_module.py, we add an "___init___.py" file.

myproject
myproject/___init___.py
myproject/math_module.py
myproject/stats_module.py

A library is a collection of packages or it can be a single package. Examples include numpy, pytorch, and Pandas. Note that the terms package and library are often used interchangeably. Therefore, numpy, pytorch, and Pandas are also referred to as packages.

```python
# Creating a Package "myproject"

# Within "myproject", create module "math_module.py"
def square(number):
   return number ** 2

def doubler(number):
    return number * 2

# Within "myproject", create module "stats_module.py"
def mean(numbers):
    return sum(numbers) / len(numbers)

# Within "myproject", create init file "__init__.py"
from . import math_module
from . import stats_module

# Verify the package. If the command runs without errors, package is successfully loaded.
import myproject

# Test the package
myproject.math_module.square(2)

# Use the package in other scripts if the package folder is in the same directory
parent_directory\test.py

"""Create test.py in parent directory"""
from myproject.math_module import square, doubler
from myproject.stats_module import mean
```

```
print("4^2=", square(4))
print("2*4=", doubler(4))

"""Run using python test.py"""
```

Packaging Summary

- To create a package:
  - Create a folder with the package name
  - Create an empty __init__.py file
  - Create the required modules
  - In the __init__.py file, add code to reference the modules needed in the package
- You can verify the package via the bash terminal in a Python shell

To verify the package: First, open a bash terminal. Make sure the directory is the same as the folder where your package is located. Open the python interpreter by running the command "python" in the shell. Using the python prompt type import followed by the project name, for example import myproject. If the command runs without errors, it is an indication that the package is successfully loaded.