Chantal Van den bussche

3/26/2026

Foundations of Programming: Python

Assignment08

GitHub Link: https://github.com/chantevan/IntroToProg-Python-Mod08

Employee Ratings Application Documentation

Introduction

This document covers what I did for Assignment08 of my Foundations of Programming: Python class. For brevity, I will not cover topics that I have covered in previous assignment documentations such as what classes are, the Separation of Concerns, etc. Instead, I will focus on new learnings that I have applied within my Employee Ratings application, an application that gathers and stores input data for employee first name, last name, review date, and review rating in a JSON file. This includes code modules and unit testing.

Code Modules

Python code modules are Python script files that get used by other code files. Code modules are typically composed of classes and/or functions. For my Assignment08, I created three modules (data_classes.py, presentation_classes.py, processing_classes.py) to store my data classes and values, I/O class, and FileProcessor class. I also created a main module for the Main Body of logic of my code, and also created three test modules to test my classes and methods.

Import Command

In order for a code module to get used by another code file, it must get imported into the code file. Just like using the "import json" command to import the json module into a code file, you can import your specific code module into your code file.

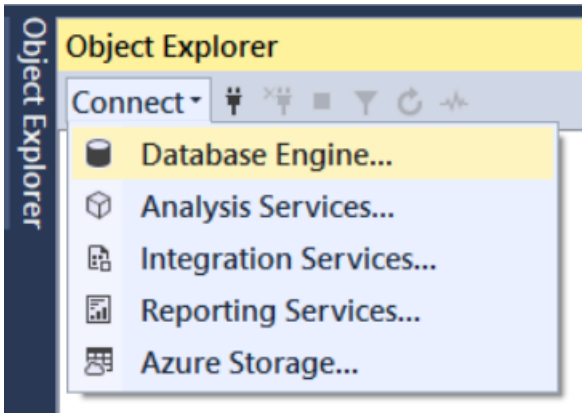

Figure 2: Connecting to a Database Engine

Import Aliases

Import aliases (also known as “import renaming” or “import as”) allows you to rename a module when importing it. This can prevent you from having to type a long module code name over and over in your code. In Example 1, I demonstrate how I imported several modules into my main module and renamed them by a shorter alias.

Example 1:

```
try:
   if __name__ == "__main__":
       import processing_classes as proc
       import presentation_classes as pres
       import data_classes as data
```

Main Module

Imported modules can be linked together in a long chain of modules. Their end destination will be to a main module which is the starting point of your program. Because all of your imported modules are linked together with an end destination of the main module, when it comes time to start and run your application, you would just run the main module from your terminal window. (If your program only has one file, that file is automatically considered the main module).

Unit Testing

Unit testing is important to ensure the quality and success of code. It puts each unit of code under a battery of tests to make sure that they all function successfully in isolation.

Test Harness

A test harness is a code module file that contains functions that tests the classes and methods of another code module. For my Assignment08, I created three test harnesses: test_data_classes.py (tests the classes and methods in data_classes.py code module), test_presentation_classes.py (tests the class and methods in presentation_classes.py), and test_processing_classes.py (tests the class and methods in presentation_classes.py code module).

Assert Statement

The assert statement checks if a given expression is true or not. If the expression is not True, then the assert statement will raise an AssertionError exception. In Example 2, I used the assert statement to check that the employee’s first name was equal to “David”.

Example 2:

```
employee = Employee("David", "Nguyen", "2025-03-25", 5)
self.assertEqual(employee.first_name, "David")
```

unittest Framework

Python's unittest framework automates the testing of individual units of code to ensure the code runs successfully and behaves as expected. To use Python's unittest framework, you must import the unittest module into your code. In Example 3, I imported the unittest module which then allows my TestPerson class to inherit from unittest.TestCase. This then allowed me to use assertion methods to check my code was working as intended.

```
import unittest
from data_classes import Person, Employee


class TestPerson(unittest.TestCase):
    '''
    A class representing test person data

    ChangeLog:
    Chantal Van den bussche, 3/25/2025, Created class
    '''

    def test_person_init_(self): # Test the constructor
        """
        This method tests the constructor of Person class

        :return:

        ChangeLog:
        Chantal Van den bussche, 3/25/2025, Created method
        Chantal Van den bussche, 3/26/2025, Edited method and added docstring
        """
        person = Person("Chantal", "Vandenbussche")
        self.assertEqual(person.first_name, "Chantal")
        self.assertEqual(person.last_name, "Vandenbussche")
```

Summary

The Employee Ratings application is composed of several modules. These modules include the files data_classes.py, presentation_classes.py, and processing_classes.py. For each of these modules, I also created a test harness to exercise my classes and methods. In each test harness, I imported the unittest module so that I could use assertion methods to verify my code was working as intended. The data_classes.py, presentation_classes.py, and processing_classes.py modules were then imported into the main module and renamed into shorter aliases so that I could easily refer to them later.