Hogan Orikasa

December 13, 2021

Foundations of Programming: Python

Assignment09

<https://github.com/horikasa/IntroToProg-Python-Mod09>

# **Modules and Main Script**

## Introduction

This week’s assignment covered the use of Modules and a Main script. We learn what modules are, how they are created, and how to implement them in our code. Our script creates three modules with which we pull into a main script. These types of scripting methods create more compartmentalization of Classes/Methods and also allow any other scripts to utilize them in an ad hoc environment.

## Modules

Simply put, Modules are standalone scripts that contain Classes and methods. We use Modules to pull in or “import” specific Classes and Methods into our scripts. This reduces our workload by allowing us to reuse code that is available and working. In our script, we use three modules called DataClasses, ProcessingClasses and IOClasses. These are all separate python scripts containing Classes and Methods that we call in our main script.

Importing Modules

Importing Modules, or more specifically the Classes and methods that we use from them, can be done in a few different ways. The first being to “import” the entire Module by itself. This is the most basic way, but you will need to append the Module name to every method call.

*import <module aka script name>*

*import ProcessingClasses*

To use a Class in the Module, you need to prepend the already called import to the Class name with a period. While this works, you will need to go one step further to utilize the Method attached to the Class.

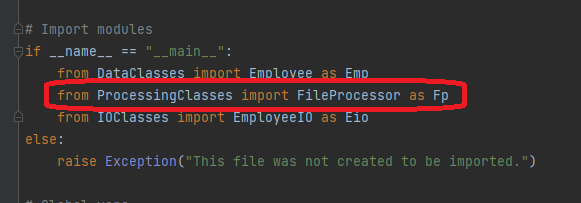
*Module.***Class.**Method()

*ProcessingClasses*.**FileProcessor**.save\_data\_to\_file()

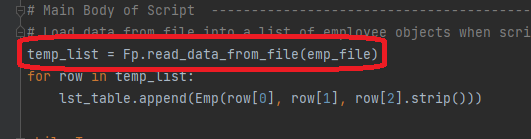
To simplify this process, you can drill down further on your import, using the “from” call in conjunction with “import”. This allows you to pick a Class within the Module as your starting point. When you use this call, it allows you to simplify your initial Method call by removing the Module name and jumping straight into the Class. This prevents potentially a lot of additional typing and can be simplified even further with the use of an “alias”. The alias acts like a declared variable for the Module.Class.Method call. We use the alias throughout this week’s script, and it does a great job of reducing clutter in the code.

*from <module> import <class> as <alias>*

*from ProcessingClasses import FileProcessor as Fp*



When calling this Class in the script, we use the simple alias “Fp”. If we had only imported the Module itself, the simple two letter call would be “ProcessingClasses.FileProcessor”. So you can see the alias really simplifies the usage of Modules.



## Main Script

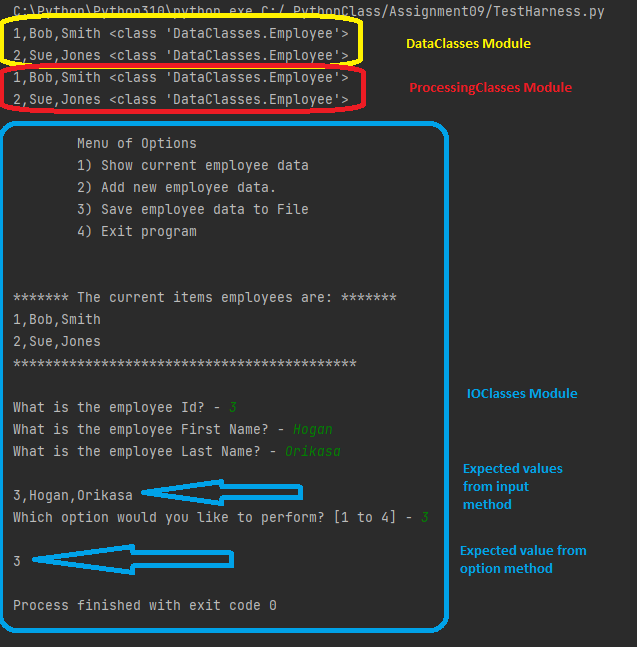
In our assignment, we use a Main script to pull all our separate Modules into one place. This provides a clean script that does not require us to write functions but use pre-existing Classes/Methods. We also have better control over the data, as we always create new objects from these methods. Our main script imports the required Modules/Methods, using aliases shown above. We then declare a few variables and create a temp list, while calling an imported Class/Method. Finally, we use a while loop to iterate through all our required actions, using only imported calls to Classes and Methods in other scripts. We do not do any real data processing in the Main script and that makes it extremely clean and easy to follow. All the data processing is handled in the imported Modules, so dependencies are isolated to each external script.

## Testing

This week, we also briefly touched on testing code. This ensures the result of our methods/functions are what we expect. In our TestHarness.py script, we code in calls to our Classes, to test their functionality. In the first section, we import our Modules as we do in our Main script. We then test our DataClasses Module by passing assigned arguments into the Employee Class as objP1 and objP2. We create a table from these objects and print each. Knowing the outcome of each Class is essential to accurately testing them. For our Employee Class, we expect to see a printed list of the Employee ID, First Name and Last Name, followed by the type which should be the object of our Class.

Testing the Processing Module is the similar outcome, in that we expect to see the Employee ID, First Name and Last Name, followed by the object type.

Finally, we test the IOClasses Module. This being an input/output Class, we expect to preview the results of running that module. First would be the input function that prompts the user for information. Then finally the selection process when asking for the user to select an option between 1-4.



## Summary

This week’s assignment was only slightly more involved than our previous. We created Modules, which hold themed Classes and Methods together. This allowed us to cleanly separate our Classes and Methods, into compartmentalized grouping, further simplifying future troubleshooting and usage of these Methods. We imported these newly created Modules into a Main script, so we could use their specific Classes and Methods. The use of a Main script allows us to trim out the unwanted code clutter and we were left with a script that calls imported methods, instead of a standard script that does the data processing within itself. This may seem like a round about way of completing a script, but it builds the foundation for complex coding. The testing we performed also shows that we have a way to validate our code.