Adam Savage

November 20, 2024

IT FDN 110 A

Assignment 06

https://github.com/adsav54/IntroToProg-Python-Mod06

Functions and Structured Coding

Assignment06

Introduction

In this module we learned how to create functions and organize them into classes, how to make functions generalizable using arguments, and how to return data from functions. The result is the ability to make modular, organized, and easy to read and debug code.

Functions in Python

Functions are a way of structuring code for modularity, organization, and reuse. At the most basic, it is simply to indent-group code under the function's name (***Figure 1)***. This allows the function's code to be used repeatedly without repeating the code's text in the script, promoting readability and reducing errors. More useful is the ability to use functions as modular IO operations; that is, to pass in data, operate on it, and return derivations of it. To pass in, one writes the function with parameters in parentheses following the function name. These parameters are what the user can supply to the function to allow operations on different data with each call and/or using different values for operations. The user-supplied objects, values, etc, are called arguments and they are assigned to temporary function-local variables when the function is called. Instead, the code author can assign default values to these parameters with "=". In addition to modularity, assigning these arguments to local temporary variables also promotes data fidelity, making it easier to keep input data distinct from output data.

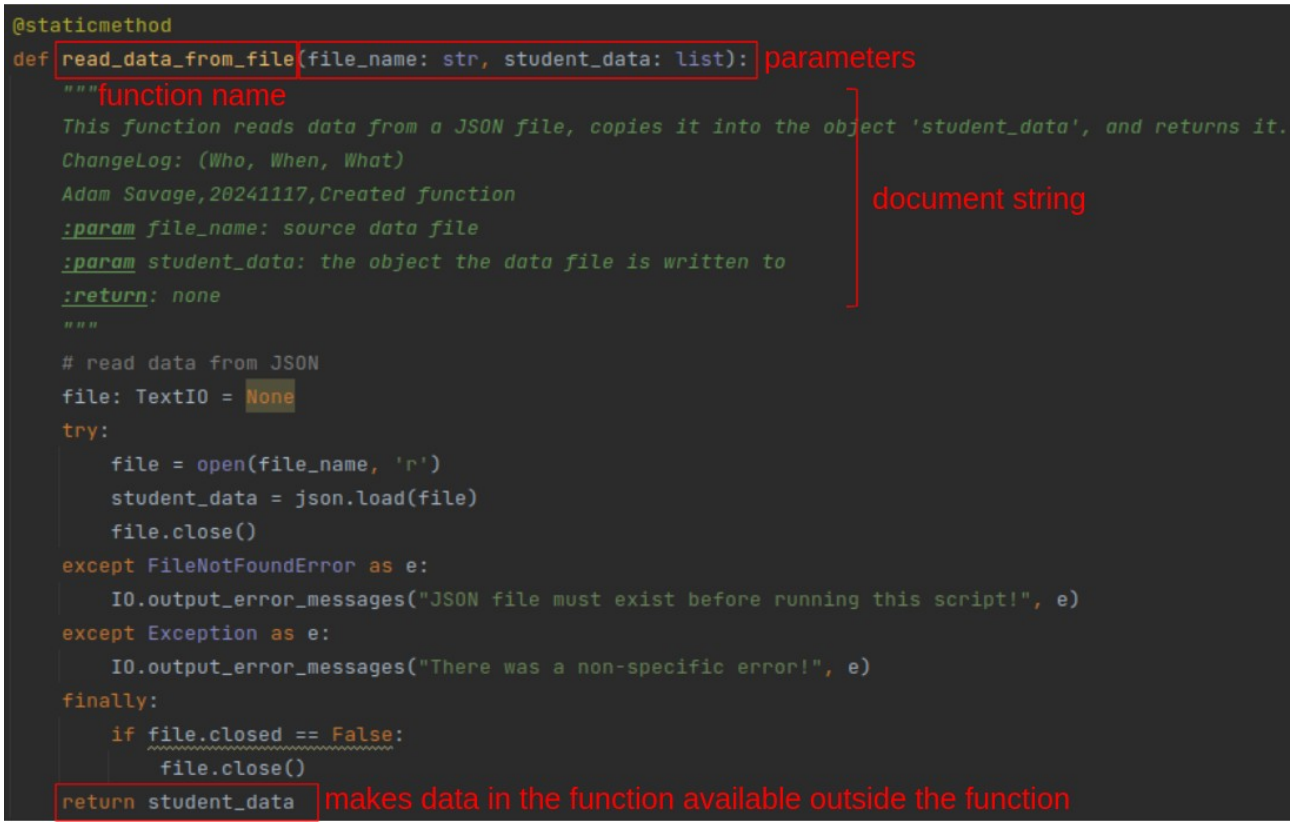

Figure 1: An example function with some annotations.

A key concept with regard to function variables is global vs local variables. Global variables are accessible throughout the code and any changes to a global variable is lasting through the program operation. To access a global variable within a function, that global variable must for 'brought into' the function with "*global [function_name]*". Local variables can only be used within the local environment of the function. This means changes to a variable within a local function cannot be accessed outside that function. It also means that a local variable can have the same name as global variable, with the value accessible on calling the variable determined by where in the code (inside or outside the function) that call is made.

The output of functions can be returned to the user with "*return*". This return data can be provided to the user (eg, via "*print()*") or stored in it's own variable. In this way, starting data can be daisy-chained to final data/results through a series of functions that operate on the data, pass the result to a new variable, which itself can then be input to another function in the chain.

Classes in Python

Classes in Python are a way of organizing code into different functional grouping. As opposed to functions, which themselves organize code into a functional unit, classes organize functions (=functionalities) into groups to conceptually organize the code. As is frequently implemented, functions do one task while classes are a collection of related tasks. The use of classes is purely organizational, promoting modularity and what's known as Separation of Concerns. Separation of Concerns in coding is the idea of organizing code conceptually and as written in the script into three domains: Presentation (what the user experiences), Logic (the core functions of the script), and Data Storage (data management). This rational organization makes it easier to modularize and maintain code, to update it, and to easily share across projects. To invoke a function that resides within a

Class, one must preface the function with the Class name, as “[Class_name].[Function_name]”. This logically resembles the way in which operations from imported packages are invoked, as packages (at least in one iteration) are a collection of one or more classes containing one or more functions each.

The Assignment06 script

In Assignment06, we are to reformat the student enrollment code to be organized by functions and classes. Each function handles a particular operation or set of operations formerly embedded in the code in a linear fashion. These functions are then called in the code, in some instances passing the appropriate variables to functions as arguments and some instances receiving data back from the function and assigning it to a variable. The end result is easily readable and modular code. This approach reduces the number of declared variables at the start of the code from 10 to 4, making it easier to keep track of what is being done with the data. The use of the function allow the *output_error_message* code to be used 5 times with only a single line each, and the *output_student_courses* code is also used more than once. Ultimately, the code for menu selections and their operations is reduced from 53 lines to 14 lines (***Figure 2***).

```
# Core program
while (True):

    # Present the menu of choices
    IO.output_menu(MENU)

    # Solicit a menu selection from the user
    menu_choice = IO.input_menu_choice()

    # Receive enrollment input from the user
    if menu_choice == "1":  # This will not work if it is an integer!
        students = IO.input_student_data(student_data=students)

    # Present the current data to the user
    elif menu_choice == "2":
        IO.output_student_courses(student_data=students)

    # Save the data to a JSON file
    elif menu_choice == "3":
        FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
        print()
        print("These data written to file:")
        IO.output_student_courses(student_data=students)

    # Stop the loop
    elif menu_choice == "4":
        break  # out of the loop
```

Figure 2: Core operations of the program are handled by 14 lines of code.

Summary

In Assignment06 we shifting our learning from basic skills to professionalizing our code. Implementing functions, and the organizational structure, is an important step to writing flexible, generalizable, and manageable code.