Kate Stevens

November 13, 2019

IT FDN 100 A

Assignment06

ToDoFile.py

Assignment06: Modify a script that manages a "ToDo list" by adding more functions to organize the code found in "[Assigment06\_Starter.py](https://canvas.uw.edu/courses/1342958/modules/items/9971459)" (external link).

# Intro

The focus of week six (IT FDN 100 A) was learning how to use functions and classes to organize code blocks for accessibility purposes (readability, portability and reusability). Other topics covered in this module are outlined below; further research on these topics can be found by visiting the links posted under [Module06:Functions & Classes](https://canvas.uw.edu/courses/1342958/modules) (external link) on the IT FDN 100 A course module page. To follow the notes directly related to the week six, Module06, study exercises go to [\_Mod6PythonProgrammingNotes.pdf](https://canvas.uw.edu/courses/1342958/files/59548208?module_item_id=9959133) (external link). To review the requirements outline for week six, Assignment06 go to [Assignment06.pdf](https://canvas.uw.edu/courses/1342958/files/59791634?module_item_id=9959134) (external link).

Functions

Parameters

Arguments

Difference between parameters and arguments

Return values

Differences between a global and a local variable

Using functions to organize code

Difference between a functions and classes

“Separations of Concerns" pattern

Debugging tools in PyCharm

GitHub webpages

Another highlight of week six, was learning the basic setup best practices behind using and creating GitHub webpages in order to properly present your files. To review these steps refer to pages 23-27 of [\_Mod6PythonProgrammingNotes.pdf](https://canvas.uw.edu/courses/1342958/files/59548208?module_item_id=9959133) (external link), or [Intro to Python Mod06 Video](https://www.youtube.com/watch?v=jiXmXhwgHp8&feature=youtu.be) (1:25:19, external link). My personal **GitHub Website** link for week six, Mod06 (part of Assignment06 requirements) can be found at <https://github.com/ksteve3/IntroToProg-Python-Mod06> (external link).

The following information describes the steps I took and the resources that I found helpful to write the script located in the "Assignment06\_ToDoFile.py" file submitted with this document.

Creating the Application

Assignment06 Directions: Short Description

Modify a script that manages a "ToDo list" by adding more functions to organize the code found in "[Assigment06\_Starter.py](https://canvas.uw.edu/courses/1342958/modules/items/9971459)" (external link). (For full assignment outline go to [Assignment06.pdf](https://canvas.uw.edu/courses/1342958/files/59791634?module_item_id=9959134) (external link)

I started my application by downloading and opening the "[Assigment06\_Starter.py](https://canvas.uw.edu/courses/1342958/modules/items/9971459)" in a txt file to go through and find (ctrl + F) the commented out flags by searching for “# ToDo:”, which indicated where code was to be added/modified to complete this assignment.

I located the flags, and determined that there were two class operators in which the functions would be created and organized under, i.e.

“FileProcessor” 🡪 written like 🡪 FileProcessor.<function\_name>()

“IO” (aka input/output) 🡪 written like 🡪 IO.<function\_name>()

By grouping the flagged code blocks in to either input/output type code or general processing type code I cross referencing my pre-grouped/pre-classified code blocks to the hinted placeholder statement following a “ToDo:” text flag indicators (highlighted in yellow).

“# ToDo: Place processing code in a new function” *(Figure1)*

“# ToDo: Place IO code in a new function” (Figure2)

***A screenshot of a cell phone

Description automatically generated***

## *Figure1(above): Fileprocessor. Class from* [Assignment06\_Starter.py](https://canvas.uw.edu/courses/1342958/files/59683877?module_item_id=9971459) *(external link)*

***A screenshot of a cell phone

Description automatically generated***

***Figure 1.2: This depicts the basic usage of print(‘ ‘) and x = input(‘ ‘) Python functions and how it fig***

***Figure2(above): IO. Class from*** [***Assignment06\_Starter.py***](https://canvas.uw.edu/courses/1342958/files/59683877?module_item_id=9971459) ***(external link)***

# …Continued

After identifying and classifying the four blocks of code, I then took each block according to the class and inserted it after the pre-added code blocks under the Processing headings below (Figure3, Figure4).

# Processing ------------------------------------------------------------- #

class FileProcessor:

***Figure3(above):*** *FileProcessor. class*

# Processing ------------------------------------------------------------- #

# Presentation (Input/Output) -------------------------------------------- #

class IO:

***Figure4(above):*** *IO. class*

The code blocks classified under FileProcessor followed a pattern as far as the definitions and the parameters go, as they are code that strictly deals with reading, writing, appending, and removing data to files only. (Figure5).

@staticmethod

def <**Function-Name**>(file\_name, list\_of\_rows):

"""

Desc - <**Function action**\* > data from a file of a list of dictionary rows

:param file\_name: (string) with name of file:

:param list\_of\_rows: (list) you want filled with file data:

:return: (list) of dictionary rows

"""

*\* reads, writes, appends, removes*

***Figure5: FileProcessor. Class***

Code that falls under Input / Output or IO. class, or # Presentation (Input/Output), are code blocks that include input/output statements or code intended to present data to the user through the program console (Figure6, highlighted in yellow). Any code that follows an input statement within the same code block will be represented as a function under the IO. class even if a read, write, append, remove action is included (Figure6, highlighted in green).

# Step 3.2.a - Ask user for new task and priority

# ToDo: Place IO code in a new function

strTask = str(input("What is the task? - ")).strip() # Get task from user

strPriority = str(input("What is the priority? [high|low] - ")).strip() #

Get priority from user

print() # Add an extra line for looks

# Step 3.2.b Add item to the List/Table

# ToDo: Place processing code in a new function

dicRow = {"Task": strTask, "Priority": strPriority} # Create a new

dictionary row

lstTable.append(dicRow) # Add the new row to the list/table

IO.ShowCurrentItemsInList(lstTable) # Show current data in the list/table

continue # to show the menu

***Figure6: IO. Class***

Final Script

Below are screenshots of the finished code represented as Functions & Classes in ToDoFile.py.

A screenshot of a social media post

Description automatically generated

***Figure7: Code in PyCharm***

Write list data to a file

***A screenshot of a social media post

Description automatically generatedFigure8(above): File.Processor class | Function Code to Write list data to a file***

***A picture containing screenshot

Description automatically generatedFigure9(above): File.Processor class | Finished Function Statement for Figure8 (Write list data to a file)***

# ------------------------------------------------------------------------------------------------------- #

Remove list data from a file

***A screenshot of a social media post

Description automatically generatedFigure10(above): File.Processor class | Function Code to Remove list data from a file A screenshot of a cell phone

Description automatically generated***

***Figure11(above): File.Processor class | Finished Function Statement for Figure10 (Remove list data from a file)***

# ------------------------------------------------------------------------------------------------------- #

User Data Input / User Data Output

### ***A screenshot of a social media post Description automatically generated***(Complex Functions-multiple functions in/out exchanges)

### ***Figure12(above): IO. class | Function codes for Get new user input data & Add new user input data to list***

A screenshot of a cell phone

Description automatically generated

### ***Figure13(above): IO. class | Finished Function Statements for Get new user input data & Add new user input data to list***

Summary

This python script shows how to successfully use basic Functions & Classes to make data more reusable and more organized. By using functions and classes to stay organized, it is much easier to debug and trouble shoot long script files with hundreds of lines of code.