Daniel Esporma

November 23rd, 2021

IT FDN 110 A Foundations Of Programming: Python

Assignment06

<https://github.com/desporma/IntroToProg-Python-Mod06>

Functions

# Introduction

Functions allow one or more statements to be grouped and can be an effective way to manage complex code in an organized fashion. Functions are controlled by parameters which can be defined by an author and can accept arguments to modify the function’s behavior. Functions can also output return values that can be fed into later functions as arguments. In this assignment, functions are used to organize the code into discreet sections and enable repeated use of the same code for different purposes.

# Methodology

The script is broken into four main sections: Data, Processing, Presentation, and Main Body of Script. These four sections break apart the code into organized pieces, but refer to one another in order to execute the program’s tasks. The Data section declares variables and constants, the Processing section is home to the Processor class of functions, the Presentation section is home to the IO class of functions, and the Main Body of Script determines what happens following user interaction. In order to best explain the script, it is easier to work backward and explain how each user option works within the script.

## Program Start

When the program starts, data is loaded from a text file. This is done from line 111 from the Main Body of Script section (Figure 1a), where the read\_data\_from\_file function is given the arguments strFileName and lstTable for its file\_name and list\_of\_rows parameters (Figure 1b). Within the function, lstTable is cleared and the data within ToDoFile.txt (strFileName = “ToDoFile.txt” in the Data section) is read, the first value of each line in ToDoFile.txt is funneled into the “Task” dictionary key, the second value of each line in ToDoFile.txt is funneled into the “Priority” dictionary key, lstTable is appended with this data, and the ToDoFile.txt is closed.

# Step 1 - When the program starts, Load data from ToDoFile.txt.  
Processor.read\_data\_from\_file(strFileName, lstTable) # read file data

**Figure 1a.** A Processor class function is called from within the Main Body of Script section.

def read\_data\_from\_file(file\_name, list\_of\_rows):  
 list\_of\_rows.clear() # clear current data  
 file = open(file\_name, "r")  
 for line in file:  
 lineelement = line.split(",")  
 row = {"Task": lineelement[0].strip(), "Priority": lineelement[1].strip()}  
 list\_of\_rows.append(row)  
 file.close()  
 return list\_of\_rows

**Figure 1b.** The read\_data\_from\_file function.

After the text file is read, the script calls three IO class functions from the Main Body of Script section (Figure 2b) to present the user with table data, a menu of options, and a string requesting user input (Figure 2a).

The table data contains current tasks on the text file, which is called by the print\_current\_Tasks\_in\_list Process class function found in the Presentation section (Figure 2c). The function contains a for-loop that prints values for each Task and Priority key from each row of a table list named lstTable. The menu of options is then called by the print\_menu\_Tasks function (Figure 2d), which prints a text string.

Lastly, the input\_menu\_choice function is called and queries the user to enter an integer between 1 and 5 to select an option (Figure 2e). The user’s response defines the value for the “choice” variable, which the function outputs as a return variable. The return variable for the input\_menu\_choice function then defines the value for the strChoice variable as written in line 118 in the Main Body of Script section (Figure 2a). This governs the user interaction of the script to follow.

\*\*\*\*\*\*\* The current Tasks ToDo are: \*\*\*\*\*\*\*

Eat (5)

Drink (3)

Bathe (10)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Menu of Options

1) Add a new Task

2) Remove an existing Task

3) Save Data to File

4) Reload Data from File

5) Exit Program

Which option would you like to perform? [1 to 5] -

**Figure 2a.** The script greets the user with current tasks, a menu of options, and a query to select an option from the menu when viewed via a terminal.

while(True):  
 # Step 3 Show current data  
 IO.print\_current\_Tasks\_in\_list(lstTable) # Show current data in the list/table  
 IO.print\_menu\_Tasks() # Shows menu  
 strChoice = IO.input\_menu\_choice() # Get menu option

**Figure 2b.** From within the Main Body of Script section, the script calls the print\_current\_Tasks\_in\_list, print\_menu\_Tasks, and input\_menu\_choice functions which are IO class functions found in the Presentation section.

def print\_current\_Tasks\_in\_list(list\_of\_rows):  
 print("\*\*\*\*\*\*\* The current Tasks ToDo are: \*\*\*\*\*\*\*")  
 for row in list\_of\_rows:  
 print(row["Task"] + " (" + row["Priority"] + ")")  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 print() # Add an extra line for looks

**Figure 2c.** The print\_current\_Tasks\_in\_list function found in the Presentation section.

def print\_menu\_Tasks():  
 print('''  
 Menu of Options  
 1) Add a new Task  
 2) Remove an existing Task  
 3) Save Data to File   
 4) Reload Data from File  
 5) Exit Program  
 ''')  
 print() # Add an extra line for looks

**Figure 2d.** The print\_menu\_Tasks function found in the Presentation section.

def input\_menu\_choice():  
 choice = str(input("Which option would you like to perform? [1 to 5] - ")).strip()  
 print() # Add an extra line for looks  
 return choice

**Figure 2e.** The input\_menu\_choice function found in the Presentation section.

## Adding a New Task

If the user enters “1” when queried, the script will start running code from line 121 found at the top of Figure 3a to add a new task to lstTable. The input\_new\_task\_and\_priority IO class function is called and the user is queried to enter a task and a priority via input() functions (Figure 3b).

These values define the return variables of the function and are fed back to line 122. The task and priority variables are then fed into the add\_data\_to\_list Processor class function as the first two arguments. The lstTable variable is also input as an argument for the “list\_of\_rows” parameter (Figures 3b-3c).

The add\_data\_to\_list function uses the values for task and priority to define the “Task” and “Priority” dictionary keys for dicRow and appends the dictionary to list\_of\_rows (Figure 3c). Finally, the input\_press\_to\_continue function is called (Figure 3a) and prints an instruction to the user to press the Enter key to continue in the program (Figure 3d).

if strChoice.strip() == '1': # Add a new Task  
 task, priority = IO.input\_new\_task\_and\_priority()  
 Processor.add\_data\_to\_list(task, priority, lstTable)  
 IO.input\_press\_to\_continue(strStatus)  
 continue # to show the menu

**Figure 3a.** From within the Main Body of Script section, the script calls the input\_new\_task\_and\_priority, add\_data\_to\_list, and input\_press\_to\_continue functions.

def input\_new\_task\_and\_priority():  
 task = str(input("Please enter a task: ")).strip()  
 priority = str(input("Please enter a priority: ")).strip()  
 return task, priority

**Figure 3b.** The input\_new\_task\_and\_priority function found in the Presentation section.

def add\_data\_to\_list(task, priority, list\_of\_rows):  
 dicRow = {"Task": str(task), "Priority": str(priority)}  
 list\_of\_rows.append(dicRow)  
 return list\_of\_rows

**Figure 3c.** The add\_data\_to\_list function found in the Processing section.

def input\_press\_to\_continue(optional\_message=''):  
 print(optional\_message)  
 input('Press the [Enter] key to continue.')

**Figure 3d.** The input\_press\_to\_continue function found in the Presentation section.

## Removing an Existing Task

If the user enters “2” when queried, the script will start running code from line 127 found at the top of Figure 4a to remove an existing task from lstTable. The input\_task\_to\_remove IO class function is called and the user is queried to enter a task via an input() function (Figure 4b). This value defines the return variable of the function and is fed back to line 128 to define the “task” variable (Figure 4a).

In line 129, the task variable is fed into the remove\_data\_from\_list Processor class function as the first argument and the lstTable variable is fed as the second argument to define the list\_of\_rows parameter (Figures 4b-4c). The remove\_data\_from\_list function contains a for-loop to read through each row of the list\_of\_rows. Within the for-loop, the value of task is transformed into lower-case to compare to values of the “Task” key in the list\_of\_rows. Where the values are equal, the row is removed by using the remove() function. The list\_of\_rows is returned and the script resumes at line 130.

Finally, the script runs the Input\_press\_to\_continue IO function again as seen in Figure 3d with the strStatus variable as the argument for the optional\_message parameter. The strStatus variable is defined as an empty string (“”), so the function only prints an instruction to the user to press the Enter key to continue in the program (Figure 3d).

elif strChoice == '2': # Remove an existing Task  
 task = IO.input\_task\_to\_remove()  
 Processor.remove\_data\_from\_list(task, lstTable)  
 IO.input\_press\_to\_continue(strStatus)  
 continue # to show the menu

**Figure 4a.** From within the Main Body of Script section, the script calls the input\_task\_to\_remove, remove\_data\_from\_list, and input\_press\_to\_continue functions.

def input\_task\_to\_remove():  
 task = str(input("Which task would you like to remove?: ")).strip()  
 return task

**Figure 4b.** The input\_task\_to\_remove function found in the Presentation section.

def remove\_data\_from\_list(task, list\_of\_rows):  
 for row in list\_of\_rows:  
 if row["Task"].lower() == task.lower():  
 list\_of\_rows.remove(row)  
 return list\_of\_rows

**Figure 4c.** The remove\_data\_from\_list function found in the Processing section.

## Save Data to File

If the user enters “3” when queried, the script will start running code from line 133 found at the top of Figure 5a to save data to the text file ToDoFile.txt. The input\_yes\_no\_choice IO class function is fed the string “Save this data to file? (y/n) – “ which defines the “message” parameter (Figure 5b) of the function. The user’s data is fed into an input() function, made lower-case and without leading or trailing spaces, and becomes the value of the return variable for the function. The value of the return variable defines strChoice in line 134 (Figure 5a).

If the user enters a “y” when prompted by the input\_yes\_no\_choice function, the write\_data\_to\_file Processor class function is called with strFileName and lstTable as values for the file\_name and list\_of\_rows parameters (Figure 5a, Figure 5c). The write\_data\_to\_file function contains a for-loop to read through each row of list\_of\_rows. Within the for-loop, each row of the list\_of\_rows is written to the file using the write() command. The file is closed after the for-loop completes writing. The script then runs the input\_press\_to\_continue IO function (Figure 5d). If the user does not enter a “y” when prompted by the input\_yes\_no\_choice function, the input\_press\_to\_continue function prints “Save Cancelled!” as the optional message.

elif strChoice == '3': # Save Data to File  
 strChoice = IO.input\_yes\_no\_choice("Save this data to file? (y/n) - ")  
 if strChoice.lower() == "y":  
 Processor.write\_data\_to\_file(strFileName, lstTable)  
 IO.input\_press\_to\_continue(strStatus)  
 else:  
 IO.input\_press\_to\_continue("Save Cancelled!")  
 continue # to show the menu

**Figure 5a.** From within the Main Body of Script section, the script calls the input\_yes\_no\_choice, write\_data\_to\_file, and input\_press\_to\_continue functions.

def input\_yes\_no\_choice(message):  
 return str(input(message)).strip().lower()

**Figure 5b.** The input\_yes\_no\_choice function found in the Presentation section.

def write\_data\_to\_file(file\_name, list\_of\_rows):  
 file = open(file\_name, "w")  
 for row in list\_of\_rows:  
 file.write(row["Task"] + "," + row["Priority"] + "\n")  
 file.close()  
 return list\_of\_rows

**Figure 5c.** The write\_data\_to\_file function found in the Processing section.

## Reload Data from File

If the user enters “4” when queried, the script will start running code from line 142 found at the top of Figure 6a to reload data from the text file ToDoFile.txt. The script calls the input\_yes\_no\_choice IO class function with “Are you sure you want to reload data from file? (y/n)” as the message parameter. The return variable from the function defines the strChoice variable.

If the user enters a “y” when prompted by the input\_yes\_no\_choice function, the script calls the read\_data\_from\_file Processor class function with strFileName and lstTable as arguments for the file\_name and list\_of\_rows parameters (Figure 6a, Figure 6b).

The read\_data\_from\_file function clears list\_of\_rows using the clear() function (Figure 6b). The text file is then opened using the open() function, the mode parameter set to “r” for read. Within a for-loop, each line of the file is split with a comma delimiter. The first element of the line is the value for the “Task” key and the second element is the value for the “Priority” key into a dictionary. The dictionary is then appended to list\_of\_rows and the file is closed.

The input\_press\_to\_continue function will then give the user instructions to press Enter to continue. If the user does not enter a “y” when prompted by the input\_yes\_no\_choice function, the input\_press\_to\_continue command is called by the script with the string “File Reload Cancelled” as an argument for the message parameter.

elif strChoice == '4': # Reload Data from File  
 print("Warning: Unsaved Data Will Be Lost!")  
 strChoice = IO.input\_yes\_no\_choice("Are you sure you want to reload data from file? (y/n) - ")  
 if strChoice.lower() == 'y':  
 Processor.read\_data\_from\_file(strFileName, lstTable)  
 IO.input\_press\_to\_continue(strStatus)  
 else:  
 IO.input\_press\_to\_continue("File Reload Cancelled!")  
 continue # to show the menu

**Figure 6a.** From within the Main Body of Script section, the script calls the input\_yes\_no\_choice, read\_data\_from\_file, and input\_press\_to\_continue functions.

def read\_data\_from\_file(file\_name, list\_of\_rows):  
 list\_of\_rows.clear() # clear current data  
 file = open(file\_name, "r")  
 for line in file:  
 lineelement = line.split(",")  
 row = {"Task": lineelement[0].strip(), "Priority": lineelement[1].strip()}  
 list\_of\_rows.append(row)  
 file.close()  
 return list\_of\_rows

**Figure 6b.** The read\_data\_from\_file function found in the Processing section.

## Exit Program

If the user enters “5” when queried, the script will start running code from line 152 found at the top of Figure 7 to exit the program. This is done by a simple print() function and a break.

elif strChoice == '5': # Exit Program  
 print("Goodbye!")  
 break # and Exit

**Figure 7.** From within the Main Body of Script section, the script prints a “Goodbye!” message to the user and exits the program.

# Summary

Functions can greatly improve the organization and efficiency of a script. When calling the same function in different sections of the same script, using different arguments can enable the function to perform multiple operations or play into multiple workflows. Additionally, the ability to call functions from other sections of a script enables the functions to be organized by purpose and lead to cleaner, more readable code.