

Name: Stephen Jenkins

Date: August 21st, 2023

Course: FDN 110 A: Foundations of Programming – Python

Assignment: Asgmt-06

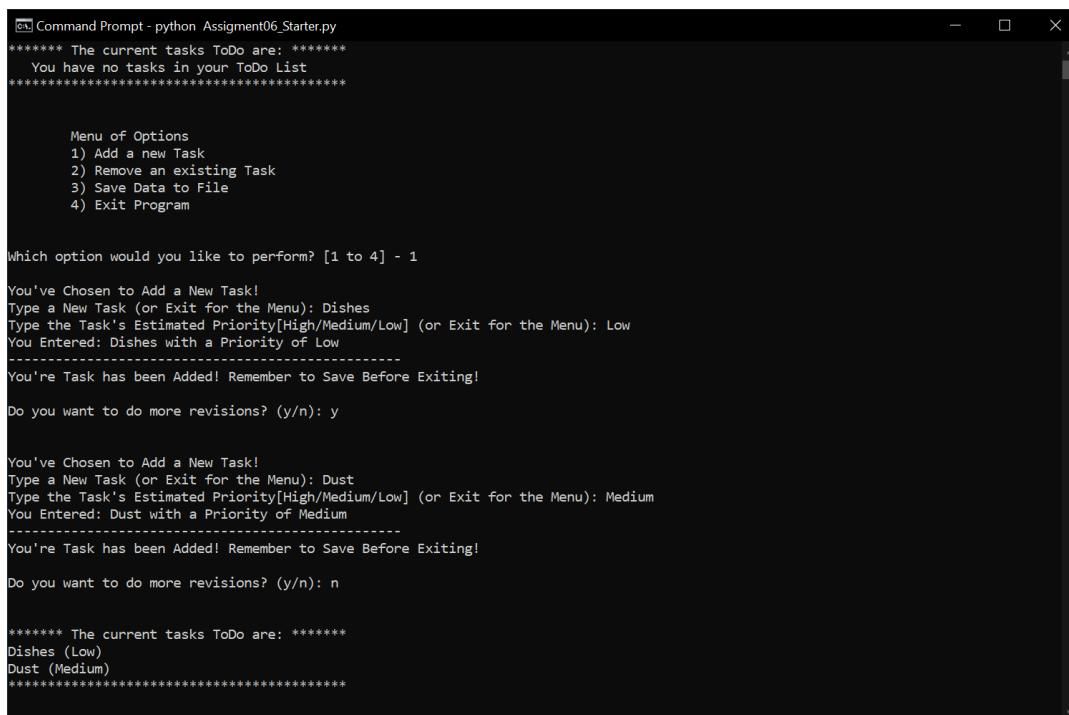
GitHub Link: <https://github.com/swjenk87/IntroToProg-Python-Mod06>

The To Do List – Version 02

Introduction

Module Six of the “Foundations of Programming – Python” class was centered on the concept of encapsulating code to increase its usability, readability, and the overall organization of scripts. Through the lessons and additional materials concepts like, “parameters”, “arguments”, and returning of values were covered at length. This was to ensure students were able to fully grasp how to process and transfer data between the main body and the associated functions.

The assignment involved revisiting the base code from Module Five, where students were tasked with collecting information from users to assemble a “ToDo” list, complete with tasks, and associated priorities. This week, however, the long, monolithic code was chunked, and associated with individual functions, complete with traveling values and variables that allowed interconnectivity. Finally, students revisited GitHub to not only store their assignments for review but create their first webpage. These pages are used by a large portion of the tech industry to track and document their code. After week six assignment documents will be stored within these webpages, following the industry trend.



```
Command Prompt - python Assignment06_Starter.py
***** The current tasks ToDo are: *****
You have no tasks in your ToDo List
*****

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4] - 1

You've Chosen to Add a New Task!
Type a New Task (or Exit for the Menu): Dishes
Type the Task's Estimated Priority[High/Medium/Low] (or Exit for the Menu): Low
You Entered: Dishes with a Priority of Low
-----
You're Task has been Added! Remember to Save Before Exiting!

Do you want to do more revisions? (y/n): y

You've Chosen to Add a New Task!
Type a New Task (or Exit for the Menu): Dust
Type the Task's Estimated Priority[High/Medium/Low] (or Exit for the Menu): Medium
You Entered: Dust with a Priority of Medium
-----
You're Task has been Added! Remember to Save Before Exiting!

Do you want to do more revisions? (y/n): n

***** The current tasks ToDo are: *****
Dishes (Low)
Dust (Medium)
*****
```

Figure 1: The desired result of this assignment

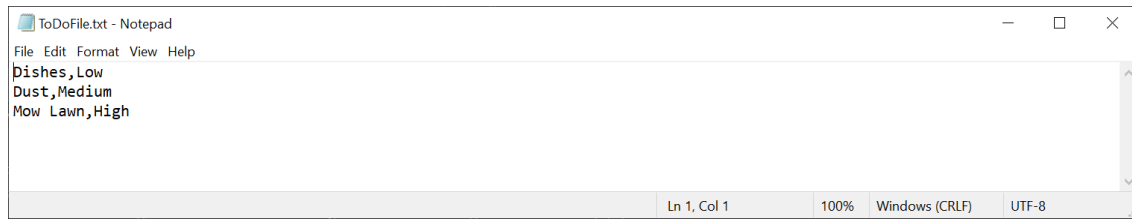


Figure 2: Data written to text file

NOTE: Not every combination has been QA'ed. If used in order, as the assignment dictated, the functionality works as intended.

Required Resources and Tools

As has been covered in the previous weeks there were new tools and resources taught that provided additional assistance in creating and executing the weekly assignment. Some of the new concepts are covered in more detail below, with examples from the final, written assignment. Topics covered in the previous weeks will not be included, to focus on the new content from Module 06.

Functions

Functions are a method of encapsulating blocks of code, to be utilized later. In the previous assignments students used one long body of code, separated only by comments, and various functionality. With the addition of functions these snippets of code can be compartmentalized, and then “called”, or run in the main portion of the python script.

A function is first defined, using the “def” keyword, then named, and finally some block of code is added. Later, invoking the name of the function, that snippet can be run by the computer. It will run the script, once it hits this function “call” it jumps back up to where the Function was defined, runs that portion, then when completed, runs from where it left off in the main body.

```
1 usage
@staticmethod
def input_task_to_remove():
    """ Gets the task name to be removed from the list

    :return: (string) with task
    """
```

Figure 3: A defined function and its descriptor, or docstring

Parameters

Within a function’s definition resides parentheses, inside of which variables or values can be added. These are snippets of information that are being passed into the function, to process.

Where the function is called, down in the main body of the script, arguments, which are defined below, are entered and these translate, via variables, or by their location in the parentheses, to parameters up in the function definition. This is, again, a way of passing information into the function to be processed.

```

@staticmethod
def write_data_to_file(file_name, list_of_rows):
    """ Writes data from a list of dictionary rows to a File

    :param file_name: (string) with name of file:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """

```




Figure 4: A function definition with parameters

Arguments

At the most basic definition an argument is a value that is accepted by or into a function. When a call is made, data can be transferred into the function, via association to a parameter. It is important to remember that a lot of programmers use the terms argument and parameter interchangeably or refer to both as arguments. Arguments are in the function call and parameters are in the function definition.

There are two types of arguments that will be seen in Python, “positional”, which means the order the arguments are passed in affects which parameter they are associated with in the definition, and “named” which set them as variables, associating them directly, versus by their location in the list.

```

elif choice_str == '3': # Save Data to File
    table_lst = Processor .write_data_to_file(file_name=file_name_str, list_of_rows=table_lst)
    print("Data Saved!")
    continue # to show the menu

```




Figure 5: Function call with named arguments

Returning Values

Unless a global variable is being used, which will not be covered in this document, the only way to read data back out of a function, to be used in other locations, is via “returning” the value. Essentially a variable, or tuple of variables can be processed out of the function, via the “return” keyword.

Down in the main body of the script the user can set the function call to a variable, saving out the result to be used in further processing. This makes the resultant value available in other areas of the script.

```

while(remove_user_choice != False):
    task = IO.input_task_to_remove()
    table_lst = Processor.remove_data_from_list(task=task, list_of_rows=table_lst)

    # Determine if the user wishes to continue revising the list
    remove_user_choice = IO.continue_task_revisions()

```

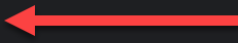


Figure 6: Function’s result being saved to a global variable

GitHub Webpages

Students used GitHub repositories to store their work in the previous module. This time around they will be hosting a single page website from within their repository via GitHub Pages. This service is a way of those in the industry to host code, and documentation that is associated with their code repositories. In the future students will be using these pages to host their project documents, but for this week's module it is a simple set of links.

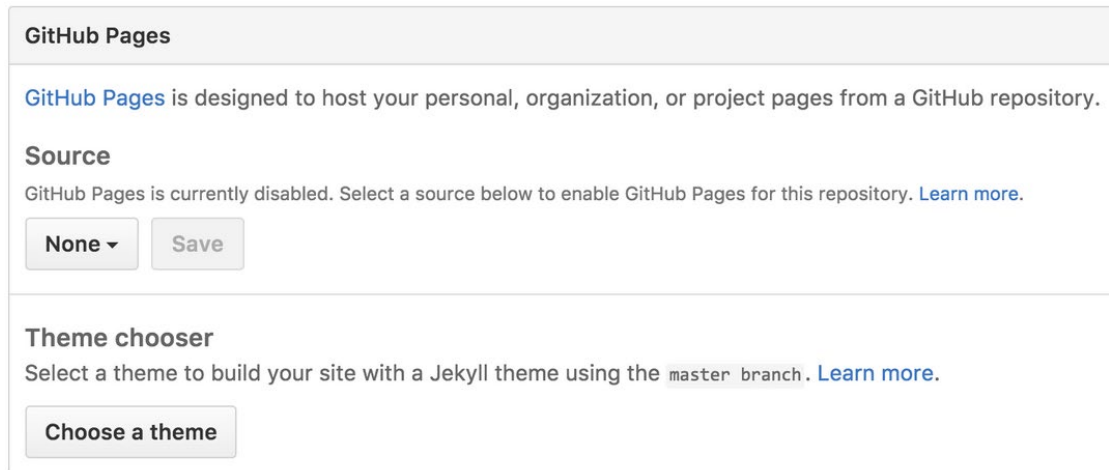
The image shows the GitHub Pages settings for a repository. At the top, there's a header "GitHub Pages". Below it, a description states: "GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository." The "Source" section indicates that GitHub Pages is currently disabled and provides a "None" dropdown menu and a "Save" button. The "Theme chooser" section instructs the user to select a theme to build their site with a Jekyll theme using the "master" branch, with a "Choose a theme" button.

Figure 7: GitHub Pages Settings

Breaking Down the Application

The new concepts that were utilized in this week's module have been covered with enough detail to get a foundational understanding of how this week's assignment was assembled. The next section will break down the actual code, as it was written, explain the layout, functions that were created and called, and how it was all assembled in the main body of the script.

The Header

The header will be added to each week's assignment as it is a best practice in programming to be detail oriented and provide as much information to those accessing the code as possible. This is a collection of fundamental information about the script, its creation and what it does.

- Information included with the Header
 - Title of the Script
 - A brief description of the functionality
 - A change log that includes the creation and any revisions to the script
 - *Note: An additional space was added after the number sign to prevent an underlining format error within PyCharm. Though it was not causing any fatal errors it was improperly setup in previous assignments and has been adjusted from this point on.*

```
# ----- #
# Title: Assignment 06
# Description: Working with functions in a class,
#             When the program starts, load each "row" of data
#             in "ToDoToDoList.txt" into a python Dictionary.
#             Add the each dictionary "row" to a python list "table"
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# SJenkins,2023-08-19,Initial testing, adding functionality
# SJenkins,2023-08-21,Completed remaining coding to complete Assignment 06
# ----- #
```

Figure 8: Header, as shipped with the “What is in Your House” script

Script Segments

There are three main sections to be covered, regarding this week’s assignment; added variables, function definitions, and the Main Body where the processes kick off and the functions are called.

Due to the complex nature of this week’s assignment not every line will be covered, with segments written by the professor, Randal Root, excluded. Only student additions will be covered, ensuring the week’s concepts have been fully grasped.

Additional Variables

The assignment starter file included several useful variables that were already being utilized. To fulfil parts of the assignment, requiring new code additional variables may have been required, these are defined below.

String Warning

This variable assigns a string, “Warning: Your To Do List Doesn’t Exist!” to a variable to be used in the try/except block.

- (Line 22) `str_warning = "Warning: Your To Do List Doesn't Exist!"` #
Create a warning message and check if it is being used
 - Set the warning string message.

Functions

The “To Do List- Version 02” script consists of ten functions that are used to do everything from gather user input and store that input into a table, to writing that input to an external text file. Each function is detailed below, breaking down its name, brief description of its purpose, and a line-by-line evaluation. Each function was originally authored by Randal Root, and the only portions covered will be those functions and lines added to complete the assignment.

Read Data from File - After “TODO: Add Code Here”

`read_data_from_file(file_name, list_of_rows)` – Looks for an existing text file that may contain previous ToDo tasks. If one is found it reads each line and saves the entries in a list. If a text file is not found it creates one and alerts the user that it has been created and is ready for editing.

- (Line 39) `try:`
 - Set up an Error Handling stack, attempt the next set of statements.
- (Line 40) `list_of_rows.clear()`
 - Removes any pre-existing data stored within “list_of_rows”
- (Line 41) `file = open(file_name, "r")`
 - Attempt to open the “ToDoList.txt” document and read it.
- (Line 42) `for line in file:`
 - If the file exists it will attempt to run a series of statements each row at a time.
- (Line 43) `task, priority = line.split(",")`
 - Splits each line into two variables “task” and “priority”.
- (Line 44) `row = {"Task": task.strip(), "Priority": priority.strip()}`
 - Stores each list row in a dictionary with the Keys of “Task” and “Priority”.
- (Line 45) `list_of_rows.append(row)`
 - Appends each Dictionary row to the main List Table.
- (Line 46) `file.close()`
 - Closes the opened document.
- (Line 48) `except:`
 - If the “Try” is unsuccessful will execute the following statements.
- (Line 49) `print(str_warning)`
 - Prints the stored warning, “Warning: Your To Do List Doesn't Exist!”
- (Line 50) `print("Creating One for You Now!\n")`
 - Let’s the user know a new file will be created for them.
- (Line 51) `file = open(file_name, "w")`
 - Create a text document named “ToDoList.txt” and write to the file.
- (Line 52) `file.close()`
 - Immediately close the document, readying it for the rest of the application.
- (Line 53) `print("Your To Do List is Created and Ready to Modify!")`
 - Let’s the user know that a file has been created.

Add Data to List - After “TODO: Add Code Here”

`add_data_to_list(task, priority, list_of_rows)` – Takes user input, covered in another section, and saves the dictionary entries to the main table list. The function utilizes three parameters that will need to be passed in, to process this code block; “task”, “priority”, and “list_of_rows”.

- (Line 68) `print("You Entered:", task, "with a Priority of", priority)`
 - Print the enter values from the user, back to the user.
- (Line 69) `print("-" * 50)`
 - Prints a line-based divider.
- (Line 70) `list_of_rows.append(row)`
 - Append an entry to the Table.

- (Line 71) `print("You're Task has been Added! Remember to Save Before Exiting!\n")`
 - Informs the user their task entry has been saved, but reminds them to save before exiting.

Add Data to List - After "TODO: Add Code Here"

`remove_data_from_list(task, list_of_rows)` – Takes user input, a task, and processes the removal of the dictionary entry from the table list. It will search line-by-line for each instance, and if the key/value pair is found remove them from the list.

- (Line 86) `bln_item_removed = False`
 - Establish a Boolean variable.
- (Line 87) `for row in list_of_rows:`
 - Read every row within the table.
- (Line 88) `rm_task, rm_priority = dict(row).values()`
 - Sets the dictionary values to the variables "rm_task" and "rm_priority".
- (Line 89) `if(rm_task == task):`
 - If the entered task matches an entry found in the table.
- (Line 90) `list_of_rows.remove(row)`
 - If found remove the entire entry.
- (Line 91) `bln_item_removed = True`
 - Set the Boolean to True.
- (Line 94) `if bln_item_removed == True:`
 - If an item was removed and Boolean is True let the user know the entry was deleted.
- (Line 95) `print("The task was removed.")`
 - Message letting the user know the entry was removed successfully.
- (Line 96) `else:`
 - If the user enters an item not in the list, let the user know the task doesn't exist.
- (Line 97) `print("I'm sorry, but I could not find that task.")`
 - Print to the user that the task doesn't exist.

Write Data to File - After "TODO: Add Code Here"

`write_data_to_file(file_name, list_of_rows)` – Once the data has been entered, and if the user chooses to do so, the table information can be written to an external text file. It will write out each row from the table that was created in the previous function and save the document.

- (Line 111) `disp_data = input("Do you wish to see your list before saving?: ")`
 - Ask the user they would like to see their data before writing to file.
- (Line 112) `if(disp_data.lower() == "y"):`
 - Check if the user answered yes.
- (Line 113) `print("Currently Your Tasks Are: \n")`
 - If they selected yes, display the message "Currently your tasks are:".
- (Line 114) `for row in list_of_rows:`
 - Performs a set of statements on each row in the table.

- (Line 115) `print(row["Task"] + " (" + row["Priority"] + ")")`
○ Print each row back to the user.
- (Line 116) `print("-" * 50)`
○ Prints a line-based divider.
- (Line 119) `file = open(file_name, "w")`
○ Opens the "ToDoList.txt" file, sets it to write, and stores it as file.
- (Line 120) `for row in list_of_rows:`
○ For each item in the table store it as the variable "row".
- (Line 121) `file.write(row["Task"] + "," + row["Priority"] + "\n")`
○ Write each Task and Priority to the document row by row.
- (Line 122) `file.close()`
○ Closes the document after use.
- (Line 123) `input("Your tasks have been saved to your To Do List! Hit [Enter] to go back to the main menu.")`
○ Prompts the user to hit [Enter] to exit back to the main menu.

Output Current Tasks in List – Added Additional Code

`output_current_tasks_in_list(list_of_rows)` – This operation will display any items current saved in the table list to the user, before displaying the menu.

- (Line 167) `if(list_of_rows != []):`
○ Checks if the List is empty and if it isn't will display each task as originally written.
- (Line 170) `else:`
○ If the list is empty perform another set of tasks.
- (Line 171) `print("\tYou have no tasks in your ToDo List")`
○ If the list is empty will display "You have no tasks in your ToDo List" to the user.

Continue Task Revisions – New Function

`continue_task_revisions()` – This function provides a set of reusable statements for checking, via yes or no answers, if the user wants to keep adding or removing content.

- (Line 176) `def continue_task_revisions():`
○ Names and defines the custom function.
- (Line 183) `if (str_con_data.lower() == "y"):`
○ Checks if the user provided answer was yes.
- (Line 184) `print("\n")`
○ Prints a new line.
- (Line 185) `yes_no_choice = True`
○ If the answer was yes set the Boolean to True.
- (Line 187) `elif (str_con_data.lower() == "n"):`
○ Otherwise check if the user entered no.
- (Line 188) `print("\n")`
○ Prints a new line.
- (Line 189) `yes_no_choice = False`
○ If the answer was no sets the Boolean to False.

- (Line 191) `return yes_no_choice`
 - Returns the result of `yes_no_choice`.

Input New Task and Priority – After “TODO: Add Code Here”

`input_new_task_and_priority()` – Asks the user for input regarding a new task name and the priority for that task. Then, returns the results to be used down in the main body of the script.

- (Line 201) `print("You've Chosen to Add a New Task!")`
 - Confirms to the user that they have chosen to enter a new task.
- (Line 204) `str_task = input("Type a New Task (or Exit for the Menu): ")`
 - Asks the user for a new task name and stores it to the variable “`str_task`”. The user may also enter exit to stop and return to the main menu.
- (Line 207) `# if(str_task.lower() == "exit"):`
 - Commented Out (revised in version 03 - Check if user entered exit for the task input.
- (Line 208) `# return`
 - Commented Out (revised in version 03 - Return to main.
- (Line 211) `str_priority = input("Type the Task's Estimated Priority[High/Medium/Low] (or Exit for the Menu): ")`
 - Asks the user for a priority for the new task and stores it to the variable “`str_priority`”. The user may also enter exit to stop and return to the main menu.
- (Line 214) `# if(str_priority.lower() == "exit"):`
 - Commented Out (revised in version 03 - Check if user entered exit for the priority input.
- (Line 215) `# return`
 - Commented Out (revised in version 03 - Return to main.
- (Line 217) `return str_task, str_priority`
 - Returns the two values to the main script body for use.

Input Task to Remove – After “TODO: Add Code Here”

`input_task_to_remove()` – Gets a task name from the user, to remove the task and associated priority from the table list.

- (Line 227) `print("Follow the instructions below to remove a To Do List item!")`
 - Tells the user to follow the instructions to remove a task.
- (Line 230) `str_remove_task = input("Which Task Would you Like to Remove?: ")`
 - Asks the user which task they would like to remove.
- (Line 232) `return str_remove_task`
 - Return the task name to the main body.

Main Body

The main body is where the actual script originates, as Functions are stored into memory until they are called. At this juncture the script begins to fire off commands, starting the input sequence that will be interacted with by the user.

Components that make up the main body and are not originally in the template provided for the assignment. Each line described was a new addition.

- (Line 251) `add_user_choice = True`
 - Ensures that if the option “1” is selected that the user choice Boolean is set back to True.
- (Line 254) `while(add_user_choice != False):`
 - While loop that checks the Boolean from the user choice variable.
- (Line 259) `add_user_choice = IO.continue_task_revisions()`
 - Sets the results of the “continue_task_revisions” function to the variable “add_user_choice”.
- (Line 265) `remove_user_choice = True`
 - Ensures that if the option “2” is selected that the user choice Boolean is set back to True.
- (Line 268) `while(remove_user_choice != False):`
 - While loop that checks the Boolean from the user choice variable.
- (Line 273) `remove_user_choice = IO.continue_task_revisions()`
 - Sets the results of the “continue_task_revisions” function to the variable “remove_user_choice”.

Summary

This week’s assignment helped cement best practices when it comes to encapsulating parts of scripts to better organize code, but also increase reusability. Without functions scripts would be large, monolithic novels that are hard to understand and debug. By building upon the previous week’s assignment, students can see how certain tools and practices will help streamline their coding in the future. On top of the functions, arguments and parameters allow data to traverse these encapsulations, meaning that any result can be shared where and when needed more easily. The result is a much more organized end product entitled “To Do List Versions 02”.