Yuting Lin

Nov. 25, 2020

Introduction to Python Scripts

Assignment06

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

<https://filizlin.github.io/IntroToProg-Python-Mod06/>

# Using Functions to better organize the script

## Introduction

### Function

Functions are groups of methods that can be first defined (and loaded into the program) and called out later. It can help organize the code better into separation of concerns (separating the processing and presentation). Functions can be defined using def with the name of the function that is followed by a colon.

def function():

#### Documenting the Function

Notes can be added at the bottom of the “def function():” to describe the purpose of this function. This note is referred as *docstring*. It would include the purpose of the function, the parameter and the return value.

#### Parameters

Parameters are the “receiver” end of the function. It is a local variable that is only used in the function, and the receiving point in the function. It can be defined in the parenthesis:

def function(parameter1, parameter2):

And later the two variables will be referred as parameter1 and parameter2 in the function. Later when the function is used, the arguments can be passed into the parameter:

Argument1, argument 2 = def function(parameter1, parameter2):

When running the script, “argument1” will be pass into the function as “parameter1.” The arguments are variables that is global and used throughout the script.

In general, there will be no pre-fixes for the name of the parameter (best practice).

#### Arguments and Variables

Arguments are variables (that are more specific) that is passed into the function. It can be used multiple times in the script. It is usually defined when running the function (instead of when defining the function). At the beginning, the variables can be first declared (as None) and later used as arguments.

#### Return Values

The Return comment indicates the end of a function. It lists the variables that is being “returned” as a result of the function. For instance, if the parameters contain 2 numbers, the return value can be the sum of the 2 numbers.

#### Global and Local variables

Global variables are variables that are used throughout the entire script and is defined at the beginning of the script. Local variables are variables that are only used (and valid) inside a function. If the same variable is used in a function and also declared as a global variable, the variable will be shadowed in the function (and remain unchanged outside of the function).

#### Class of functions

Functions can be grouped into classes to apply another layer of organization. When calling a function from a class. Class can be declared as:

Class NameOfClass():

And when calling a function fro the class, it can be referred as:

Class.Function()

## A Python Script Example that uses Functions to organize the script

Here is an example of a python script. Note that there is a pseudo-code that is included at the beginning of the script to indicate the Developer, date, and logging the changes. This portion is not included in this document for it does not provide function that is executed by Python. The goal of this script is to provide the user with a list of options, and let the user decide which option to proceed with. The program will first look for text files that is already being created and contains information (Step #1), then provide the users with a menu of options as to what they would like to do with the file (Step #2). If there was no pre-existing file, the script will create one when the user starts to write information into the file (Step #4). The user can check what is in the file (Step #3) and remove selected data from the file (Step #5). The user can save the data (Step #6) and exit (Step #7) once they are ready. The script of the program is adopted from the Assignment05 script and it is organized using Functions. The main idea is to separate the processing and presenting layers into different sections, and later on call them out with arguments.

*# Data ---------------------------------------------------------------------- #  
# Declare variables and constants*strFileName = **"ToDoFile.txt"** *# The name of the data file*objFile = None *# An object that represents a file*dicRow = {} *# A row of data separated into elements of a dictionary {Task,Priority}*lstTable = [] *# A list that acts as a 'table' of rows*strChoice = **""** *# Captures the user option selection*strTask = **""** *# Captures the user task data*strPriority = **""** *# Captures the user priority data*strStatus = **""** *# Captures the status of an processing functions  
  
# Processing --------------------------------------------------------------- #*class Processor:  
 *""" Performs Processing tasks """* @staticmethod  
 def read\_data\_from\_file(file\_name, list\_of\_rows):  
 *""" Reads data from a file into 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  
 """* list\_of\_rows.clear() *# clear current data* file = open(file\_name, **"r"**)  
 for line in file:  
 task, priority = line.split(**","**)  
 row = {**"Task"**: task.strip(), **"Priority"**: priority.strip()}  
 list\_of\_rows.append(row)  
 file.close()  
 return list\_of\_rows, **'Success'** @staticmethod  
 def add\_data\_to\_list(task, priority, list\_of\_rows):  
 *"""Add the data input by user into the file as a dictionary pair  
 :param task: (string) the task to input  
 :param priority: (string) the priority to input  
 :return: a dictionary pair added to the table list  
 """* dicRow = {**"Task"**: str(task.strip()), **"Priority"** : str(priority.strip())}  
 list\_of\_rows.append(dicRow)  
 return list\_of\_rows, **'Success'** @staticmethod  
 def remove\_data\_from\_list(task, list\_of\_rows):  
 *"""remove the task row input by user  
 :param task: (string) the task to input  
 :return: a dictionary pair removed the table list  
 """* for row in list\_of\_rows:  
 if task.strip().lower() in row[**"Task"**].strip().lower(): *# try see if can do remove task and priority* list\_of\_rows.remove(row)  
 print(**"The task "** + strDelete + **" has been removed from the file."**)  
 else:  
 print(**"The task entered does not exist in this row."**)  
 return list\_of\_rows, **'Success'** @staticmethod  
 def write\_data\_to\_file(file\_name, list\_of\_rows):  
 *"""write the data to the file  
 :return: write the data to the file  
 """* objFile = open(**"file\_name.txt"**, **"w"**)  
 for row in list\_of\_rows:  
 objFile.write(row[**"Task"**] + **","** + row[**"Priority"**] + **"**\n**"**)  
 objFile.close()  
 return list\_of\_rows, **'Success'***# Presentation (Input/Output) -------------------------------------------- #*class IO:  
 *""" Performs Input and Output tasks """* @staticmethod  
 def print\_menu\_Tasks():  
 *""" Display a menu of choices to the user  
  
 :return: nothing  
 """* 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* @staticmethod  
 def input\_menu\_choice():  
 *""" Gets the menu choice from a user  
  
 :return: string  
 """* choice = str(input(**"Which option would you like to perform? [1 to 5] - "**)).strip()  
 print() *# Add an extra line for looks* return choice  
  
 @staticmethod  
 def print\_current\_Tasks\_in\_list(list\_of\_rows):  
 *""" Shows the current Tasks in the list of dictionaries rows  
  
 :param list\_of\_rows: (list) of rows you want to display  
 :return: nothing  
 """* 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* @staticmethod  
 def input\_yes\_no\_choice(message):  
 *""" Gets a yes or no choice from the user  
  
 :return: string  
 """* return str(input(message)).strip().lower()  
  
 @staticmethod  
 def input\_press\_to\_continue(optional\_message=**''**):  
 *""" Pause program and show a message before continuing  
  
 :param optional\_message: An optional message you want to display  
 :return: nothing  
 """* print(optional\_message)  
 input(**'Press the [Enter] key to continue.'**)  
  
 @staticmethod  
 def input\_new\_task\_and\_priority():  
 *""" request input taks and priority from user when choosing option 1  
 :param task: input requested from user  
 :param priority: input requested from user  
 :return: task and priority  
 """* task = input(**"Type in a Task: "**)  
 priority = input(**"Indicate its Priority (High/Medium/Low): "**)  
 print(**"Adding"**, str(task.strip()), str(priority.strip()), **"to Table"**, sep=**" "**)  
 return task, priority  
  
 @staticmethod  
 def input\_task\_to\_remove():  
 task = input(**"Which task would you like to delete from the To-Do list? "**)  
 return task  
  
*# Main Body of Script ------------------------------------------------------ #  
  
# Step 1 - When the program starts, Load data from ToDoFile.txt.*Processor.read\_data\_from\_file(strFileName, lstTable) *# read file data  
  
# Step 2 - Display a menu of choices to the user*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  
   
 # Step 4 - Process user's menu choice* if strChoice.strip() == **'1'**: *# Add a new Task* strTask, strPriority = IO.input\_new\_task\_and\_priority()  
 lstTable, strStatus = Processor.add\_data\_to\_list(strTask, strPriority, lstTable)  
 IO.input\_press\_to\_continue(strStatus)  
 continue *# to show the menu* elif strChoice == **'2'**: *# Remove an existing Task* strDelete = IO.input\_task\_to\_remove()  
 lstTable, strStatus = Processor.remove\_data\_from\_list(strDelete, lstTable)  
 IO.input\_press\_to\_continue(strStatus)  
 continue *# to show the menu* elif strChoice == **'3'**: *# Save Data to File* strChoice = IO.input\_yes\_no\_choice(**"Save this data to file? (y/n) - "**)  
 if strChoice.lower() == **"y"**:  
 lstTable, strStatus = 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* 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'**:  
 lstTable, strStatus = 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* elif strChoice == **'5'**: *# Exit Program* print(**"Goodbye!"**)  
 break *# and Exit*

### Separation of Concerns

The script is organized and noted by the different comments. The data is first declared, and in the processing layer, there are 2 classes of functions: the processor and the input/output. And lastly the presentation layer calls out the function

### Data

#### Declaring Variables

At first, all the global variables were declared, and these variables were later used as arguments to pass into the parameters of the functions that were defined.

### Processing

#### Function Class: Processor

This class of function were used as processors that does not require any input/output from the user. The script came from the Assignment05 script (and excluding the input/print commands). It includes reading a data from a file, adding/removing data from the file, and writing the data to the file.

#### Function Class: IO (Input/Output)

This class of function were used as processors focuses on receiving the user unput. Therefore, the input commands that were in Assignment05 were organized into functions. This includes printing the menu, asking for user input in menu choice, new tasks, task to remove, and y/n choice.

### Main Body of Script

The main body of the script provides the menu and the different options. In this section, the previous commands were replaced by functions with arguments at the beginning to declare the variable that is passed into the function.

Graphical user interface, text, application

Description automatically generated

**Figure 1.** A screenshot of running ToDoList.py in PyCharm with user input

A picture containing text

Description automatically generated

**Figure 2.** A screenshot of running ToDoLis.py in Terminal with user input

## Summary

Function are useful ways to organize the code and help the other developer better read the code and process it. It can help separate the layers based on different concerns and make the “reusing” of the function easier by calling them out multiple times.

## Citation

The resources used to complete this document includes the IT FDN 110 A Module 5 course material provided by Prof. Randle Root and the textbook: Python Programming for the Absolute Beginner, Third Edition, By Michael Dawson.