Rex Sarabia

23 August 2022

IT FDN 110 B

Assignment07

GitHub repo: <https://github.com/rsar-uw/IntroToProg-Python-Mod07> (External)

GitHub page: <https://rsar-uw.github.io/IntroToProg-Python-Mod07/> (External)

Python Script: VIP Birthdays

Table of Contents

[2 Introduction 2](#_Toc112269629)

[3 My system information 2](#_Toc112269630)

[3.1 Operating system (OS) 3](#_Toc112269631)

[3.2 Console 3](#_Toc112269632)

[3.3 Shell 3](#_Toc112269633)

[3.4 Python 4](#_Toc112269634)

[3.5 Integrated Development Environment (IDE) 4](#_Toc112269635)

[3.6 Directory / File path 4](#_Toc112269636)

[4 Module assignment 5](#_Toc112269637)

[4.1 Requirements 5](#_Toc112269638)

[4.1.1 Out of scope 5](#_Toc112269639)

[4.1.2 Lessons learned 5](#_Toc112269640)

[4.2 Design 6](#_Toc112269641)

[4.2.1 Standard elements 6](#_Toc112269642)

[4.2.1.1 Code style 6](#_Toc112269643)

[4.2.1.2 Script header 6](#_Toc112269647)

[4.2.1.3 Saving files 7](#_Toc112269657)

[4.2.2 Program description 7](#_Toc112269658)

[4.2.3 Data structure 7](#_Toc112269659)

[4.2.4 Program architecture 8](#_Toc112269660)

[4.2.5 User journey flows 9](#_Toc112269661)

[4.2.5.1 Open file, Display data 9](#_Toc112269822)

[4.2.5.2 Menu 12](#_Toc112269823)

[4.2.5.3 Add data 13](#_Toc112269824)

[4.2.5.4 Save data 17](#_Toc112269825)

[4.2.5.5 Quit program 18](#_Toc112269826)

[4.2.6 Proposed solution 20](#_Toc112269867)

[4.3 Test 24](#_Toc112269868)

[4.3.1 Procedure 24](#_Toc112269869)

[4.3.2 Results 30](#_Toc112269870)

[4.4 Execution 30](#_Toc112269871)

[4.4.1 Terminal 31](#_Toc112269872)

[4.4.2 Results 36](#_Toc112269873)

[5 Summary 37](#_Toc112269874)

[6 References 37](#_Toc112269875)

[6.1 Schema 37](#_Toc112269876)

[6.1.1 Books 37](#_Toc112269877)

[6.1.2 Websites 37](#_Toc112269878)

[6.2 Sources 37](#_Toc112269879)

[6.2.1 Exception handling 38](#_Toc112269880)

[6.2.2 Pickling 38](#_Toc112269881)

# Introduction

I have concerns about the time requirements for this week’s assignment. In addition to the ‘normal’ assignment requirements, there is also an additional aspect of self-learning that will be needed. I think that one value I have found from the references provided to us so far is the “short cut” in time needed to “wade through” the vast amount of information available. There is a great quantity and wide spectrum in quality of content “in the wild” after all: *The net is vast and infinite.*

Additionally, there have been unforeseen competing priorities that may manifest this weekend. So, to crash my own schedule (which, admittedly is less than ideal), my approach will be to learn this module’s lessons by starting with the assignment. Although I acknowledge that there are simpler (and still effective) ways to demonstrate one’s knowledge of this week’s content, I would like to try where practice, to both: simulate real-world time pressures / workloads and build upon the preceding weeks’ lessons.

So far, it’s still been fun though.

[Table of Contents]

# My system information

I do not expect to include this information in every assignment. The purpose of including this information is to serve as a reference documentation for future needs (e.g., troubleshooting).

Where applicable, the information below will be updated after significant system component version updates, and the addition of new or replacement of existing system components (i.e., deltas from this information will be recorded).

[Table of Contents]

## Operating system (OS)

From the Apple menu, top left corner of screen:

 > About This Mac

**macOS Monterey version 12.5.1** (Figure 1)



Figure 1. Screen capture of macOS version

[Table of Contents]

## Console

**Terminal version 2.12.7 (445)**

*For more information, refer to A01-RSar.docx – Section 3.2.*

[Table of Contents]

## Shell

**zsh version 5.9**

*For more information, refer to A01-RSar.docx – Section 3.3.*

[Table of Contents]

## Python

**Python version 3.10.5**

*For more information, refer to A01-RSar.docx – Section 3.4.*

[Table of Contents]

## Integrated Development Environment (IDE)

Per instruction of Randal Root, using PyCharm Community Edition (CE) as default IDE for Module03. “Step 5.2 Create a new Project n PyCharm” (Randall, R. \_ Assignment03\_instructions.docx, Self-published, 2019).

**PyCharm version 2022.1.4 (Community Edition)**

*For more information, refer to A06-RSar.docx – Section 3.5.*

[Table of Contents]

## Directory / File path

Open Terminal

Navigate to Assignment07 working folder.

Enter the following command (see Figure 2):

|  |
| --- |
| cd documents/\_pythonclass/module07/a07rsarabia |

Figure . Command in Terminal for navigating to assignment directory

This folder and its contents will be compressed into .zip file and submitted for Assignment06.

Enter the following command in Terminal to return the directory path (Figure 3):

|  |
| --- |
| pwd |

Figure . Command in Terminal to return directory path

Enter the following command in Terminal to return the directory content (see Figure 4):

|  |
| --- |
| ls -la |

Figure . Command in Terminal to return directory contents (visible and invisible files)

**/Users/rex/Documents/\_PythonClass/Module07/A07RSarabia** (Figure 5)

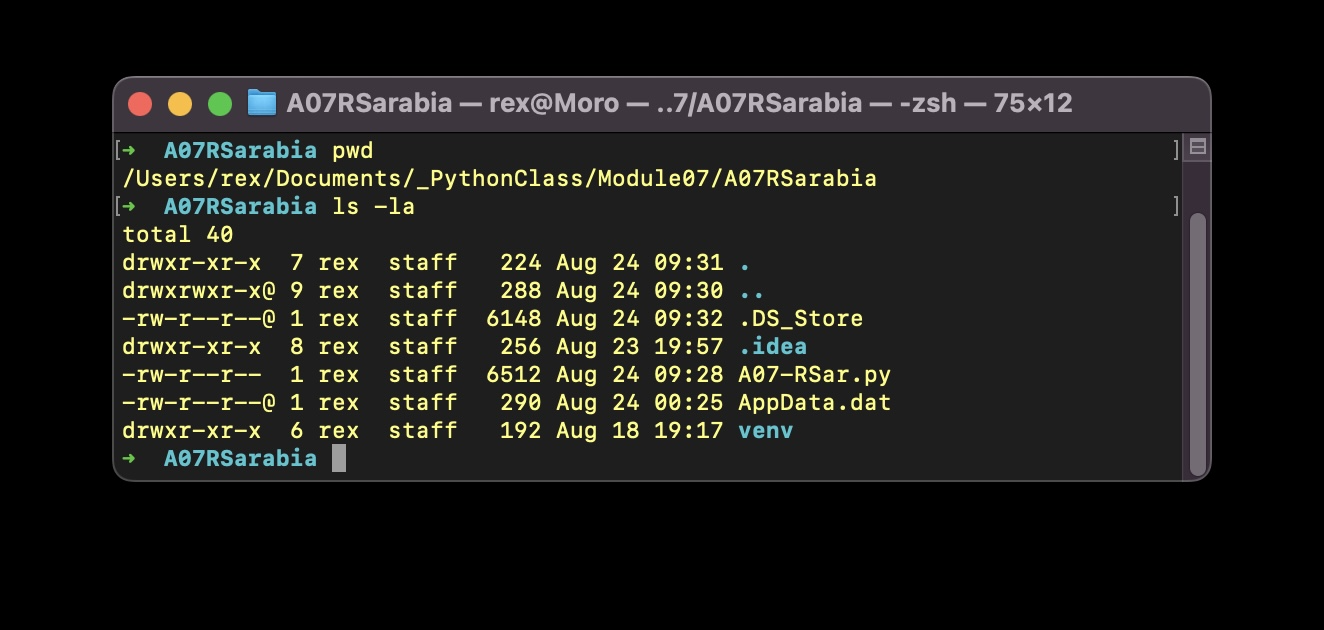


Figure . Screen capture of assignment directory and contents

[Table of Contents]

# Module assignment

## Requirements

“Pickling + Exception handling" (Randall R., Assignment07\_instructions.docx, Self-published, 2022).

Use case demos:

1. Open file (file extension: .dat (binary))
2. Add data
3. Save (write data to binary file)
4. Quit

[Table of Contents]

### Out of scope

For the purposes of this document, documentation is limited to “Apply your knowledge” (Step 5) and “Document your knowledge” (Step 6) of the assignment. Additional assignment tasks related to “Post your Files to GitHub” (Step 7) through “Perform Peer Review” (Step 11) are not included in this write-up.

[Table of Contents]

### Lessons learned

As weeks progress and assignments are graded, I thought it would be helpful to keep a log of comments received from prior assignments (Figure 6). The simplest benefit is to serve as a log of lessons learned along the way and more practically, as immediate reminders to not repeat the same mistakes. *Steel sharpens steel.*

| **Assignment filename** | **Comment** | **Penalties** |
| --- | --- | --- |
| A01-RSar.docx | end the file with  print(‘(Press Enter to End Program)’) #Conclusion line  input() | -2 |
| A02-RSar.docx | any figure# in a caption below a images/code snippet is referenced by figure # in the text written up above | -1 |
| A03-RSar.docx | Filename should be: HomeInventory.py | -2 |
| A04-RSar.docx | the save/exit option didn't ask me if i wanted to save to the file  Note: see Figure 5 - Assignment04\_instructions.docx | -2 |
| A05-RSar.docx | Needed to post a link to your GitHub site on the assignment textbox | -1 |
| A06-RSar.docx | for the zip file, only include the document file and the python file | -1 |

Figure . Table of review comments from prior assignments

[Table of Contents]

## Design

This section includes the relevant components in the Python code that compose my proposed solution to the assignment.

[Table of Contents]

### Standard elements

#### Code style

By default, PyCharm will automatically perform checks of code style for PEP 8 (JetBrains, <https://www.jetbrains.com/help/pycharm/tutorial-code-quality-assistance-tips-and-tricks.html#df2e3bcf>, 2022) (External site). I made a conscious decision to follow the Problems/Solutions PyCharm identified, which in some cases conflict with the code style taught in class / prior assignments – honestly, it makes reading for errors in PyCharm a lot easier without having to mess with config settings of the feature.

For more information about PEP 8 code style guide: <https://peps.python.org/pep-0008/>

[Table of Contents]

#### Script header

In the script header (top of the script file), the following information must be recorded in the following format (Figure 7):

|  |
| --- |
| # ------------------------------- #  # Title: Assignment##  # Dev: RSar  # Desc: short description of script  # ChangeLog: (date,name,change)  # yyyy/mm/dd, Dev, Created File  # ------------------------------- # |

Figure . Source code for header

[Table of Contents]



#### Saving files

“PyCharm automatically saves changes that you make in your files. Saving is triggered by various events, such as compiling, running, debugging, performing version control operations, closing a file or a project, or quitting the IDE.” (JetBrains, <https://www>.jetbrains.com/help/pycharm/saving-and-reverting-changes.html, 2022) (External site)

Per assignment requirements (Section 4.1), script filename and data output filename as follows:

* Script filename: **A07-RSar.py**
* Data output filename: **AppData.dat**
* Directory / file path: Figure 5

[Table of Contents]

### Program description

Since the primary objective of this assignment is to demonstrate the requirements, I am building a “dummy” program as a base reference. The base reference program will be a birthday record keeper: “VIP Birthdays”.

The program will record: Name, Relationship (a.k.a. circle), and Date of Birth (a.k.a., dob).

### Data structure

I have found it helpful to know before coding what is the expected data structure and data file output (Figure 8). As the output is going to a binary file (.dat) – where the raw data in the file is not human readable – I’m not going to output as a comma-separated values (CSV) file. Since viewing the data requires the program to do, I’ll leave the extract-transform-load (ETL) workload to the program.

|  |
| --- |
| Name: alphanumeric, free-text    Circle: mutually exclusive, pre-defined groups: Family | Friend | Business | Other    DOB: yyyy-m-d |

Figure . Data structure

Due to increased documentation requirements (e.g., publishing to GitHub webpage), I will deprioritize development to the minimum code required to demonstrate use cases specified (Section 4.1).

Where possible leverage existing code from prior assignment: **A06-RSar.py**

Since there is no base code to start from for this assignment, I am using the template I had created in the prior assignment to structure the code for each module (Figure 9).

|  |
| --- |
| # ------------------------------- #  # Title: Assignment07  # Dev: RSar  # ChangeLog: (date,name,change)  # 2022/mm/dd, RSar, Created module to complete Assignment  # ------------------------------- #  # Data ----------------------------------------------------------- #  # Declare variables and constants  # Processing ---------------------------------------------------- #  # Presentation (Input/Output) ----------------------------------- #  # Main Body of Script ------------------------------------------- #  # Step 1 - When the program starts, Load file.  # Step 2 - Display a menu of choices to the user  # Step 3 Show current data  # Step 4 - Process user's menu choice |

Figure . Module template

[Table of Contents]

### Program architecture

Following lessons from Module06 (Randall R., \_Mod6PythonProgrammingNotes.docx, Self-published, 2019), efforts will be made to follow principles of abstraction and separation of concerns.

I have found it helpful to plan out in advance (at least at a high-level), how to organize requirements into features, and the order in which to develop components, integrate, and then iterate further.

Module: Add

1. Manually create default output file
2. Write data to default output file
3. Verify data is being written (i.e., open default output file in text editor, note before / after changes)
4. Input functions to allow user to enter data
5. Exception catching

Module: Load

1. Open default output file
2. Read raw data from default output file
3. Display data from default output file
4. Exception catching

Module: Quit

1. Exit program
2. Check if changes to data have been saved before quitting
3. Input functions to allow user to choose to quit without saving or return to menu

### User journey flows

In contrast to Section 4.2.4, understanding how the user is expected to navigate through the program further aids development.

Flow #0 – Core: User opens program > Program opens default file > Program loads data from default file into program memory > Program displays current data to user in human-readable format > Program displays menu of options / functions to user: Add data, Save data, Quit program

Flow #1 – Standard: User adds data > User saves data > User quits program

Flow #2 – Changes made + No save: User adds data > User quits program (without saving)

> Quit program w/o change  
> Return user to menu

Flow #3 – No change + Save: User saves data (without making changes)

> Return user to menu







#### Open file, Display data

*Requirement 1: Allow program to open default output file.*



*Requirement 2: Allow program to display current data to user in human-readable format.*

The program begins by importing the pickle module: import pickle, which is needed for using the pickle function to ‘load’ the data from the file (Figure 10).

vip\_lst = read\_data\_from\_file(working\_file\_str=default\_file\_str)

The program calls the function read\_data\_from\_file() where the string value (filename) assigned to the default\_file\_str variable is passed to the parameter working\_file\_str. The vip\_lst list-variable is assigned to the output of the function.

The function read\_data\_from\_file() opens the filename string value passed to the working\_file\_str parameter in read mode for binary files ('rb’) and assigns the file object to file\_obj variable. The file\_obj variable is used as an input for the pickle function and .load method. The result of the function is to output a list with dictionary data type rows which is assigned to the list\_of\_rows variable.

The overall structure of the read\_data\_from\_file function follows this logic:

1. Try to open the default\_file\_str as working\_file\_str.
2. If file is not found, then return custom error message: “ERROR: File not found.”
3. If file is found, then try to extract data from file to list\_of\_rows variable with pickle.load() function and method.
4. EOFError = End of file error
5. Return list\_of\_rows variable with data from file.

The output\_current\_vip\_in\_list function uses the output of the read\_data\_from\_file function list\_of\_rows variable to pull each row value for Name and Birthday keys. At this point of the program, code is same as prior assignment for unpacking data from a list object with dictionary list as rows.

|  |
| --- |
| import pickle  # Data ----------------------------------------------------------- #  # Declare variables and constants  default\_file\_str = 'AppData.dat'  # Processing ---------------------------------------------------- #  def read\_data\_from\_file(working\_file\_str):  try:  file\_obj = open(working\_file\_str**,** 'rb')  except FileNotFoundError:  print('\nERROR: File not found.')  else:  try:  list\_of\_rows = pickle.load(file\_obj)  except EOFError:  pass  else:  file\_obj.close()  return list\_of\_rows  # Presentation (Input/Output) ----------------------------------- #  def output\_current\_vip\_in\_list(list\_of\_rows):  print('''\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \tCurrent VIPs \t--------------------------------- \tName \t\tBirthday \t\t\t(yyyy-m-d) \t---------------------------------''')  for row in list\_of\_rows:  print('\n\t' + row["Name"] +  '\t\t' + row["Birthday"]**,** end='')   print('\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')  # Main Body of Script ------------------------------------------- #  # Step 1 - When the program starts, Load file.  vip\_lst = read\_data\_from\_file(working\_file\_str=default\_file\_str)  # Step 2 - Display a menu of choices to the user  while True:  # Step 3 Show current data  print(''' ===================================== DISPLAY DATA =====================================''')   # Step 3 Show current data  output\_current\_vip\_in\_list(list\_of\_rows=vip\_lst)  # Step 4 – Process user’s menu choice  if choice\_str.strip() == '1': # Add a new Task  continue  elif choice\_str.strip() == '2': # Save Data to File  continue # to show the menu  elif choice\_str.strip() == '3': # Exit Program  break # exit Menu loop |

Figure . Program code for opening specified file and displaying data

To avoid “TypeError: 'NoneType' object is not iterable” which occurs when the data file is empty – there is no data to load – I preloaded the default data with dummy data. If I had more time, I would build a try-except or if statement to mitigate scenario.

[Table of Contents]

#### Menu

*Requirement 3: Allow user to select option from menu.*

To simplify coding, the program functions are reduced to Add, Save, and Exit. This code is based directly on menu from prior assignment (Figure 11).

|  |
| --- |
| # Data ----------------------------------------------------------- #  # Declare variables and constants  # Processing ---------------------------------------------------- #  # Presentation (Input/Output) ----------------------------------- #  def output\_menu():  print(''' ===================================== MAIN MENU =====================================  \t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \tOptions \t--------------------------------- \t1 - Add a new VIP \t2 - Save data to file  \t3 - Exit program \t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ''')  def input\_menu\_choice():  choice = str(input('Select option [1 to 3]: \t\t\t| ')).strip()  return choice  # Main Body of Script ------------------------------------------- #  # Step 1 - When the program starts, Load file.  # Step 2 - Display a menu of choices to the user  output\_menu()  choice\_str = input\_menu\_choice()  # Step 3 Show current data  # Step 4 - Process user's menu choice |

Figure . Source code for program to display menu options to user and prompt user for instruction

[Table of Contents]

#### Add data

*Requirement 4: Allow user to add data.*

The program collects allows the user to input three data elements for each VIP record: Name, Relationship, and Date of birth (Figure 12). I defined a function for each data element to simplify the code since each element requires unique “data validation” contingencies.

The input\_vip\_name function requires the user to input a name that must be longer than 1 character. The input by the user is assigned the name variable which becomes the output value of the function: return name.

The input\_vip\_circle function captures the relationship of the VIP to the user – data which may be helpful in future version. There are 4 categories of relationship: Family, Friend, Business, and Other. The user is required to pick one of the four. If an option is not in the range of 1 through 4 then custom error message: “ERROR: Invalid option selected. Choose number from list.” is triggered. Additionally, if the user inputs a non-numeric character (e.g., letter) then the try-except statement will trigger another custom error message: “ERROR: Non-numeric value entered. Choose number from list.” The input by the user is assigned the circle variable which becomes the output value of the function: return circle.

The input\_vip\_dob function requires the datetime module: import datetime. The try-except statement will catch occurrences where the value inputted by the user is not a valid date or if the date is not in the correct yyyy-m-d format: datetime.datetime.strptime(dob, '%Y-%m-%d'). If the user inputs an invalid date or incorrectly formatted date, then customer error message is triggered: “ERROR: Invalid date. Date should be in yyyy-m-d format.” The input by the user is assigned the dob variable which becomes the output value of the function: return dob.

The row\_dic variable in the add\_data\_to\_list function takes the outputs of the input\_vip\_name, input\_vip\_circle, input\_vip\_dob functions and creates a dictionary list record with and organized to the following keys, respectively: Name, Circle, Birthday. This row\_dic dictionary list is appended to the list\_of\_rows list similar prior assignment. In addition to returning the list\_of\_rows list, this function also returns zero integer.

In the vip\_lst, check\_save\_flag[0] = add\_data\_to\_list() statement the list\_of\_rows list is assigned to the global vip\_lst list. Additionally, zero integer is passed to the check\_save\_flag global array. If the value in the zero index of the check\_save\_flag global array is referenced later if the user chooses to quit the program without saving.

|  |
| --- |
| import datetime  # Data ----------------------------------------------------------- #  # Declare variables and constants  check\_save\_flag = [**1**] # If = 1, then data saved/no changes, if = 0, changes \  vip\_lst = []  # Processing ---------------------------------------------------- #  # Presentation (Input/Output) ----------------------------------- #  def input\_vip\_name():  while True:  name = str(input('Enter name: \t\t\t\t\t\t| '))  if len(name) < **1**:  print("\nERROR: Name cannot be blank.")  else:  return name  def input\_vip\_circle():  print(''' \t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \tRelationship \t--------------------------------- \t1 - Family \t2 - Friend \t3 - Business \t4 - Other \t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*''')  circle = None  while circle not in range(**1, 5**):  try:  circle = int(input('\nSpecify relationship [1-4]: \t\t|'))  if circle in range(**1, 5**):  return circle  else:  print('\nERROR: Invalid option selected. Choose number '  'from list.')  except ValueError:  print('\nERROR: Non-numeric value entered. Choose number from list.')  def input\_vip\_dob():  while True:  dob = str(input('Enter birthday (yyyy-m-d): \t\t\t| '))  try:  datetime.datetime.strptime(dob**,** '%Y-%m-%d')  except ValueError:  print('\nERROR: Invalid date. Date should be in yyyy-m-d '  'format.')  else:  return dob  def output\_vip\_added():  print('\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'  '\n\tNew VIP added'  '\n\t---------------------------------'  '\n\tName: \t\t\t' + \_name +  '\n\tRelationship: \t' + c +  '\n\tBirthday: \t\t' + \_dob +  '\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')  def add\_data\_to\_list(name**,** circle**,** dob**,** list\_of\_rows):  row\_dic = {"Name": str(name).strip()**,** "Circle": str(circle).strip()**,** "Birthday": str(dob).strip()}  list\_of\_rows.append(row\_dic)  return list\_of\_rows**, 0**  # Main Body of Script ------------------------------------------- #  # Step 1 - When the program starts, Load file.  # Step 2 - Display a menu of choices to the user  while True:  # Step 3 Show current data  # Step 4 - Process user's menu choice  if choice\_str.strip() == '1': # Add a new Task  print(''' ===================================== DATA ENTRY: Add new VIP ===================================== ''')  \_name = input\_vip\_name()   \_circle = input\_vip\_circle()  if \_circle == **1**:  c = 'Family'  elif \_circle == **2**:  c = 'Friend'  elif \_circle == **3**:  c = 'Business'  elif \_circle == **4**:  c = 'Other'   \_dob = input\_vip\_dob()   output\_vip\_added()   vip\_lst**,** check\_save\_flag[**0**] = add\_data\_to\_list(name=\_name**,** circle=\_circle**,** dob=\_dob**,** list\_of\_rows=vip\_lst)  continue |

Figure . Source code for user to add data

[Table of Contents]

#### Save data

*Requirement 5: Allow user to save data.*

The program begins by importing the pickle module: import pickle, which is needed for using the pickle function to ‘dump’ the data from the program into the file (Figure 13).

When the user chooses option 2 (save data) from the menu, the program calls the write\_data\_to\_file statement. If the value in the zero index of the check\_save\_flag array is zero, then the data has changed and has not been saved, so the program will proceed to save the file. If the check\_save\_flag array is one, then the write\_data\_to\_file statement is not called and the program notifies the user with custom message: “ALERT: No changes detected.”

Inside the write\_data\_to\_file statement, the default file specified is opened in wb mode because the program is writing data to a binary file. The pickle.dump(list\_of\_rows, file\_obj) pickle statement, takes the vip\_lst list of dictionaries values passed via the list\_of\_rows list array and dumps the object data into the file\_obj which is passed via the working\_file\_str parameter. Lastly, the check\_save\_flag is changed to 1 since the data has been saved and the program notifies the user: “Data saved.”

|  |
| --- |
| import pickle  # Data ----------------------------------------------------------- #  # Declare variables and constants  default\_file\_str = 'AppData.dat'  check\_save\_flag = [**1**] # If = 1, then data saved/no changes, if = 0, changes \ # not saved  vip\_lst = []  # Processing ---------------------------------------------------- #  def write\_data\_to\_file(working\_file\_str**,** list\_of\_rows):  file\_obj = open(working\_file\_str**,** 'wb')  pickle.dump(list\_of\_rows**,** file\_obj)  check\_save\_flag[**0**] = **1** print('Data saved.')  file\_obj.close()  # Presentation (Input/Output) ----------------------------------- #  # Main Body of Script ------------------------------------------- #  # Step 1 - When the program starts, Load file.  # Step 2 - Display a menu of choices to the user  while True:  # Step 3 Show current data  # Step 4 - Process user's menu choice  elif choice\_str == '2': # Save Data to File  if check\_save\_flag[**0**] == **0**:  write\_data\_to\_file(working\_file\_str=default\_file\_str**,** list\_of\_rows=vip\_lst)  else:  print('\nALERT: No changes detected.')  continue # to show the menu |

Figure . Source code for program to write data to file

[Table of Contents]

#### Quit program

*Requirement 6: Allow user to quit the program.*

*Requirement 7: Allow program to prompt user to save if unsaved changes are detected.*

When the user chooses to quit the program, the program will refer to the check\_save\_flag array value at the zero-index location (Figure 14). If the value is 0, then the “Add” function had been executed by the user and the new data has not been saved. The program prompts the user with custom message: “WARNING: You have unsaved changes. If you quit, your changes will not be saved.” I got a bit lazy at this point of the program and crammed all of the logic into this block with the exception of the output\_exit\_program function for displaying the exit greeting and user-input interrupt since this code is used more than once. Similar to the prior assignment, when the user quits the program, the break statement is executed where the while True loop is exited.

|  |
| --- |
| # Data ----------------------------------------------------------- #  # Declare variables and constants  check\_save\_flag = [**1**] # If = 1, then data saved/no changes, if = 0, changes not saved  # Processing ---------------------------------------------------- #  # Presentation (Input/Output) ----------------------------------- #  def output\_exit\_program():  print("\n\tByeeee!")  input("\n[Press ENTER key to quit.]")  # Main Body of Script ------------------------------------------- #  # Step 1 - When the program starts, Load file.  # Step 2 - Display a menu of choices to the user  while True:  # Step 3 Show current data  # Step 4 - Process user's menu choice  elif choice\_str == '3': # Exit Program  if check\_save\_flag[**0**] == **0**:  check\_save = str(input(''' WARNING: You have unsaved changes.  If you quit, your changes will not be saved.   Are you sure you want to quit? (Y/N): \t| '''))  if check\_save.lower() == 'n':  continue   elif check\_save.lower() == 'y':  output\_exit\_program()  break   elif check\_save\_flag[**0**] == **1**:  output\_exit\_program()  break # exit Menu loop |

Figure .

[Table of Contents]



### Proposed solution

The following source code is my program for Assignment07 (Figure 18).

|  |
| --- |
| # ------------------------------- # # Title: Assignment07 # Dev: RSar # Desc: Start # ChangeLog: (date,name,change) # 2022/08/23, RSar, Created module to complete Assignment # ------------------------------- # import pickle import datetime  # Data ----------------------------------------------------------- # # Declare variables and constants program\_title\_str = 'VIP Birthdays v1.0' default\_file\_str = 'AppData.dat' check\_save\_flag = [**1**] # If = 1, then data saved/no changes, if = 0, changes \ # not saved   # Processing ---------------------------------------------------- #  def read\_data\_from\_file(working\_file\_str):  try:  file\_obj = open(working\_file\_str**,** 'rb')  except FileNotFoundError:  print('\nERROR: File not found.')  else:  try:  list\_of\_rows = pickle.load(file\_obj)  except EOFError:  pass  else:  file\_obj.close()  return list\_of\_rows  def add\_data\_to\_list(name**,** circle**,** dob**,** list\_of\_rows):  row\_dic = {"Name": str(name).strip()**,** "Circle": str(circle).strip()**,** "Birthday": str(dob).strip()}  list\_of\_rows.append(row\_dic)  return list\_of\_rows**, 0** def write\_data\_to\_file(working\_file\_str**,** list\_of\_rows):  file\_obj = open(working\_file\_str**,** 'wb')  pickle.dump(list\_of\_rows**,** file\_obj)  check\_save\_flag[**0**] = **1** print('Data saved.')  file\_obj.close()   # Presentation (Input/Output) ----------------------------------- #  def input\_menu\_choice():  choice = str(input('Select option [1 to 3]: \t\t\t| ')).strip()  return choice   def input\_vip\_name():  while True:  name = str(input('Enter name: \t\t\t\t\t\t| '))  if len(name) < **1**:  print("\nERROR: Name cannot be blank.")  else:  return name   def input\_vip\_circle():  print(''' \t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \tRelationship \t--------------------------------- \t1 - Family \t2 - Friend \t3 - Business \t4 - Other \t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*''')  circle = None  while circle not in range(**1, 5**):  try:  circle = int(input('\nSpecify relationship [1-4]: \t\t| '))  if circle in range(**1, 5**):  return circle  else:  print('\nERROR: Invalid option selected. Choose number '  'from list.')  except ValueError:  print('\nERROR: Non-numeric value entered. Choose number from '  'list.')   def input\_vip\_dob():  while True:  dob = str(input('Enter birthday (yyyy-m-d): \t\t\t| '))  try:  datetime.datetime.strptime(dob**,** '%Y-%m-%d')  except ValueError:  print('\nERROR: Invalid date. Date should be in yyyy-m-d '  'format.')  else:  return dob   def output\_menu():  print(''' ===================================== MAIN MENU =====================================  \t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \tOptions \t--------------------------------- \t1 - Add a new VIP \t2 - Save data to file  \t3 - Exit program \t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ''')   def output\_current\_vip\_in\_list(list\_of\_rows):  print('''\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \tCurrent VIPs \t--------------------------------- \tName \t\tBirthday \t\t\t(yyyy-m-d) \t---------------------------------''')  for row in list\_of\_rows:  print('\n\t' + row["Name"] +  '\t\t' + row["Birthday"]**,** end='')   print('\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')   def output\_vip\_added():  print('\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'  '\n\tNew VIP added'  '\n\t---------------------------------'  '\n\tName: \t\t\t' + \_name +  '\n\tRelationship: \t' + c +  '\n\tBirthday: \t\t' + \_dob +  '\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')   def output\_exit\_program():  print("\n\tByeeee!")  input("\n[Press ENTER key to quit.]")   # Main Body of Script ------------------------------------------- # print('\n\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\'  '\\\\\\\\\\\\\\\\' +  '\nWelcome to ' + program\_title\_str + '!' +  '\n/////////////////////////////////////') # Display program name  print('\n\tOpening file: ' + default\_file\_str + '...') # Step 1 - When the program starts, Load file.  vip\_lst = read\_data\_from\_file(working\_file\_str=default\_file\_str)  # Step 2 - Display a menu of choices to the user while True:   print(''' ===================================== DISPLAY DATA =====================================''')   # Step 3 Show current data  output\_current\_vip\_in\_list(list\_of\_rows=vip\_lst)  output\_menu()  choice\_str = input\_menu\_choice()   # Step 4 - Process user's menu choice  if choice\_str.strip() == '1': # Add a new Task  print(''' ===================================== DATA ENTRY: Add new VIP ===================================== ''')  \_name = input\_vip\_name()   \_circle = input\_vip\_circle()  if \_circle == **1**:  c = 'Family'  elif \_circle == **2**:  c = 'Friend'  elif \_circle == **3**:  c = 'Business'  elif \_circle == **4**:  c = 'Other'   \_dob = input\_vip\_dob()   output\_vip\_added()   vip\_lst**,** check\_save\_flag[**0**] = add\_data\_to\_list(name=\_name**,** circle=\_circle**,** dob=\_dob**,** list\_of\_rows=vip\_lst)  continue   elif choice\_str.strip() == '2': # Save Data to File  if check\_save\_flag[**0**] == **0**:  write\_data\_to\_file(working\_file\_str=default\_file\_str**,** list\_of\_rows=vip\_lst)  else:  print('\nALERT: No changes detected.')  continue # to show the menu   elif choice\_str.strip() == '3': # Exit Program  if check\_save\_flag[**0**] == **0**:  check\_save = str(input(''' WARNING: You have unsaved changes.  If you quit, your changes will not be saved.   Are you sure you want to quit? (Y/N): \t| '''))  if check\_save.lower() == 'n':  continue   elif check\_save.lower() == 'y':  output\_exit\_program()  break   elif check\_save\_flag[**0**] == **1**:  output\_exit\_program()  break # exit Menu loop |

Figure . Source code for my proposed solution to Assignment07

[Table of Contents]

## Test

### Procedure



For the purpose of this assignment, testing is performed in PyCharm IDE.

Open PyCharm

PyCharm > File > Open > directory path (Section 3.6) > A07-RSar.py

PyCharm > Run > “A07-RSar”

For this assignment and based on the limitations placed on the inputs of the user, I intend to limit my test cases to expected errors and a few valid input types (Figure 16).

| **Test flow ID** | **Test description** | **Actual output** | **Result** |
| --- | --- | --- | --- |
| 1 | Start program | Start program / Open data file and display contents | Pass |
| 2 | Happy flow: Add data, Quit without saving, Save, Reopen file | Add data  *Name: “Name7”, Circle: “Business”, DOB: “1900-7-7”*    Quit without saving    Save file    Reopen file | Pass |
| 3 | Error flow: File not found | Change default\_file\_str to “BadAppData.dat” | Pass |
| 4 | Error flow: Add bad VIP data | Add data  *Circle: “5”*    *DOB: “1900-13-13”*  *DOB: “8-8-1900”* |  |

Figure 16. Summary of tests performed and results in PyCharm IDE

[Table of Contents]

### Results

Results from test cases all passed as the actual result matched what I was expecting for each of the value combinations input.

[Table of Contents]

## Execution

For the purpose of this assignment, execution is done via the Terminal which simulates executing in Production environment (PROD) as running the program in PyCharm IDE simulates testing environment (TEST).

[Table of Contents]

### Terminal

Open Terminal

Enter the following command (Figure 17):

|  |
| --- |
| python3 [file path][file name] |

Figure . Command for executing script in Terminal

Where:

File path: Figure 5

File name: Section 4.2.1.3

Re-perform test procedures (see Figure 18). For this assignment, the data inputted into default data file has been erased and so the file is null.

| **Test flow ID** | **Test description** | **Actual output** | **Result** |
| --- | --- | --- | --- |
| 1 | Start program | Start program / Open data file and display contents | Pass |
| 2 | Happy flow: Add data, Quit without saving, Save, Reopen file | Add data  *Name: “Name7”, Circle: “Business”, DOB: “1900-7-7”*    Quit without saving    Save file    Reopen file | Pass |
| 3 | Error flow: File not found | Change default\_file\_str to “BadAppData.dat” | Pass |
| 4 | Error flow: Add bad VIP data | Add data  *Circle: “5”,*    *DOB: “1900-13-13”*  *DOB: “8-8-1900”* |  |

Figure . Summary of tests performed and results in Terminal

[Table of Contents]

### Results

Using the same input values per Section 4.3.1, produced the same results as testing for both error messages of invalid input as well as correctly formatted and stored outputs of valid user inputs.

[Table of Contents]

# Summary

Similar to preceding assignments, I thought this week’s assignment was a significant step up in complexity and challenge. Despite the program functionality being stripped down from the prior assignment, I can confirm the feeling “60% of code is exception handling”. At least, for me, it sure felt that way. Unforeseen (and highly-irregular) opportunities over this past weekend may have also effectively compressed my available time for this assignment. I think the code structures and frameworks that were provided as part of prior assignments were helpful. One challenging (and personally, frustrating) aspect of this assignment was the reliance on self-research to learn the level of coding needed to adapt prior assignments’ code. I will surely appreciate the “assignment answers” example for this module’s lessons for future reference.

[Table of Contents]

# References

## Schema

### Books

“Quoted text“ (Author Last name Author First name initial., Title of book, Publisher, Year published)

### Websites

“Quoted text“ (Website name, URL, yearAccessed) (External site)

[Table of Contents]

## Sources

*Note: It is assumed that knowledge builds, therefore, duplicate sources already included in prior assignments have been removed – unless it has been directly referenced within this assignment.*

Sarabia R., A01-RSar.docx, Self-published, 2022

Sarabia R., A02-RSar.docx, Self-published, 2022

Sarabia R., A03-RSar.docx, Self-published, 2022

Sarabia R., A04-RSar.docx, Self-published, 2022

Sarabia R., A05-RSar.docx, Self-published, 2022

Sarabia R., A06-RSar.docx, Self-published, 2022

JetBrains, <https://www.jetbrains.com/help/pycharm/saving-and-reverting-changes.html>, 2022 (External site)

JetBrains, <https://www.jetbrains.com/help/pycharm/tutorial-code-quality-assistance-tips-and-tricks.html#df2e3bcf>, 2022 (External site)

Randall R., \_Mod6PythonProgrammingNotes.docx, Self-published, 2019

Randall, R. \_ Assignment07\_instructions.docx, Self-published, 2019

Behind the Name, <https://www.behindthename.com/random/random.php?gender=both&number=1&sets=5&surname=&usage_eng=1>, 2022 (External site): Random name generator

### Exception handling

Corey Schafter – YouTube, <https://www.youtube.com/watch?v=NIWwJbo-9_8>, 2022 (External site): Python Tutorial: Using Try/Except Blocks for Error Handling

Notes: Clear, very concise instruction, Logical progression through content

Socratica – YouTube, <https://www.youtube.com/watch?v=nlCKrKGHSSk&t=10s>, 2022 (External site): Exceptions in Python||Python Tutorial||Learn Python Programming

Notes: Fun, relevant, “more elaborate” example (opening .dat file)

Stack Overflow, <https://stackoverflow.com/questions/2244270/get-a-try-statement-to-loop-around-until-correct-value-obtained>, 2022 (External site): Get a Try statement to loop around until correct value obtained

Stack Overflow, <https://stackoverflow.com/questions/2083987/how-to-retry-after-exception>, 2022 (External site): How to retry after exception?

TutorialKart, <https://www.tutorialkart.com/python/python-range/python-if-in-range>, 2022 (External site): Python – if in Range, if not in Range

Notes: Explicitly searched for solution to loop input\_vip\_data\_circle()

Stack Overflow, <https://stackoverflow.com/questions/16870663/how-do-i-validate-a-date-string-format-in-python>, 2022 (External site): How do I validate a date string format in python?

Notes: Explicitly searched for solution to validate input\_vip\_data\_dob()

### Pickling

DelfSatck, <https://www.delftstack.com/howto/python/python-read-pickle/>, 2022 (External site): Read a Pickle File Using Python

Notes: Explicitly searched for solution on how to read all data from binary file.

[Table of Contents]