Alexander Bryant

6/1/2020

Foundations of Programming

Assignment 07

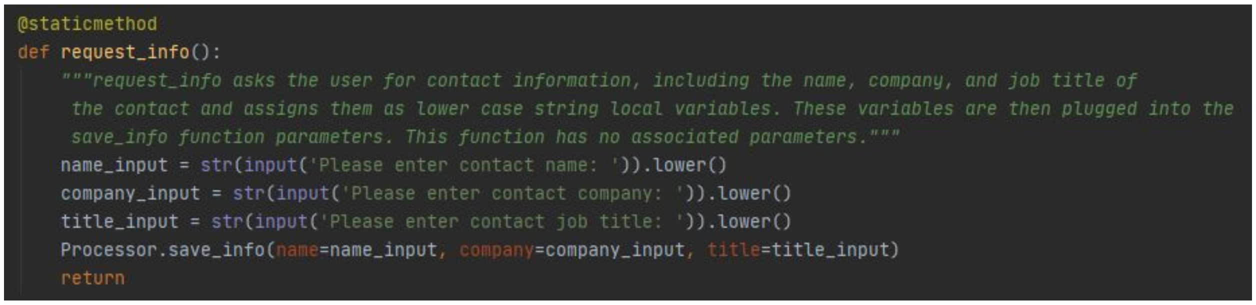
Pickling Contacts

Intro

This week’s module focused on both pickling functions and error handling within Python. This week’s assignment was to create a script that demonstrated the usage of pickling and error handling. I chose to write a script that creates, saves, and reviews contact information, with a few deliberate choices made to demonstrate error handling.

Process

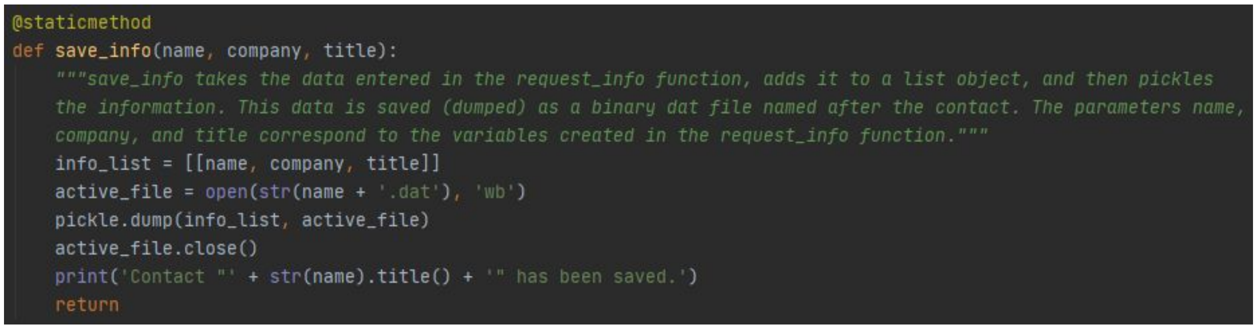
Keeping up with separation of concerns, this script is split into data, processing, and presentation sections. Within the data section, the pickle module is imported and the lone global variable is declared. The variable *menu_input* captures the user’s menu choice as an integer in the main body of the script. Next, the processing section contains the definitions for functions within the *Processor* class.



```
@staticmethod
def request_info():
    """request_info asks the user for contact information, including the name, company, and job title of
     the contact and assigns them as lower case string local variables. These variables are then plugged into the
     save_info function parameters. This function has no associated parameters."""
    name_input = str(input('Please enter contact name: ')).lower()
    company_input = str(input('Please enter contact company: ')).lower()
    title_input = str(input('Please enter contact job title: ')).lower()
    Processor.save_info(name=name_input, company=company_input, title=title_input)
    return
```

Figure 1: The request_info function requests contact info from the user.

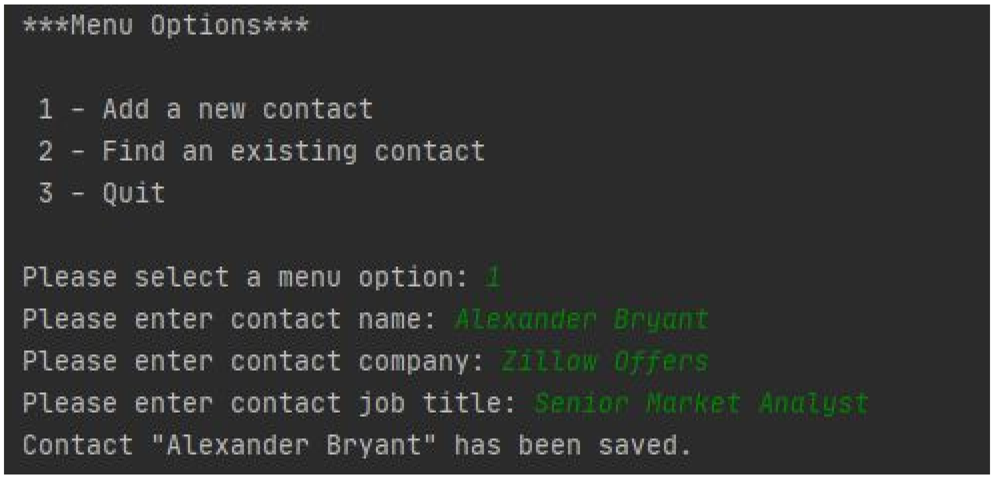
In ***Figure 1*** above, the *request_info* function is defined as a method within the *Processor* class. Three input statements request a contact name, company, and job title, which are interpreted as lower case strings and assigned to the local variables *name_input*, *company_input*, and *title_input*. The function then inserts the local variables as the keyword arguments *name*, *company*, and *title* into another function, *save_info*.



```
@staticmethod
def save_info(name, company, title):
    """save_info takes the data entered in the request_info function, adds it to a list object, and then pickles
    the information. This data is saved (dumped) as a binary dat file named after the contact. The parameters name,
    company, and title correspond to the variables created in the request_info function."""
    info_list = [[name, company, title]]
    active_file = open(str(name + '.dat'), 'wb')
    pickle.dump(info_list, active_file)
    active_file.close()
    print('Contact "' + str(name).title() + '" has been saved.')
    return
```

Figure 2: The save_info function saves data from the request_info function as a dat file.

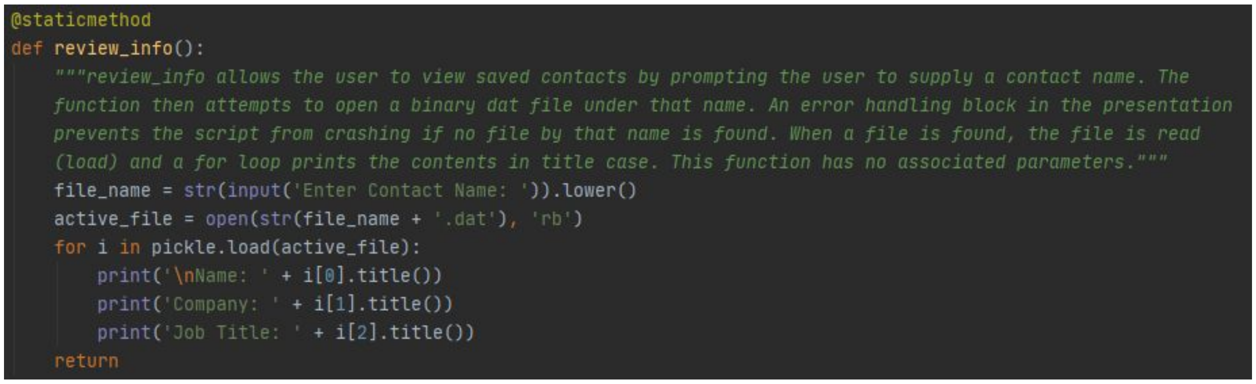
In ***Figure 2*** above, the *save_info* function is defined as a method within the *Processor* class. The parameters *name*, *company*, and *title* are supplied by the *request_info* function. The data is added to a list and a new binary file is created with a naming convention based on the *name* parameter. The pickle function then saves (dumps) the list to the new file, and then closes the file for later review. A confirmation message is printed for the user to let them know the contact has been saved.



```
***Menu Options***

 1 - Add a new contact
 2 - Find an existing contact
 3 - Quit

Please select a menu option: 1
Please enter contact name: Alexander Bryant
Please enter contact company: Zillow Offers
Please enter contact job title: Senior Market Analyst
Contact "Alexander Bryant" has been saved.
```

Figure 3: In PyCharm, the user adds a new contact's name, company, and title.

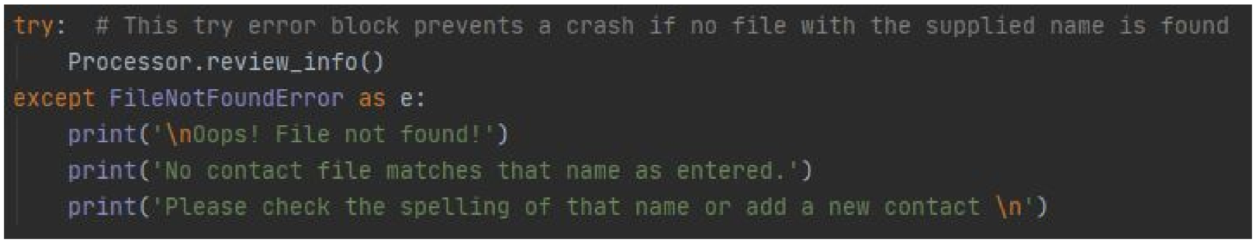
In ***Figure 3*** above, a PyCharm user selects menu option 1 and adds new contact information. Behind the scenes, the *request_info* function collects the contact information and passes it along to the *save_info* function, which creates a binary file based on the contact name.



```python
@staticmethod
def review_info():
    """review_info allows the user to view saved contacts by prompting the user to supply a contact name. The
    function then attempts to open a binary dat file under that name. An error handling block in the presentation
    prevents the script from crashing if no file by that name is found. When a file is found, the file is read
    (load) and a for loop prints the contents in title case. This function has no associated parameters."""
    file_name = str(input('Enter Contact Name: ')).lower()
    active_file = open(str(file_name + '.dat'), 'rb')
    for i in pickle.load(active_file):
        print('\nName: ' + i[0].title())
        print('Company: ' + i[1].title())
        print('Job Title: ' + i[2].title())
    return
```

Figure 4: The review_info function retrieves saved contact files.

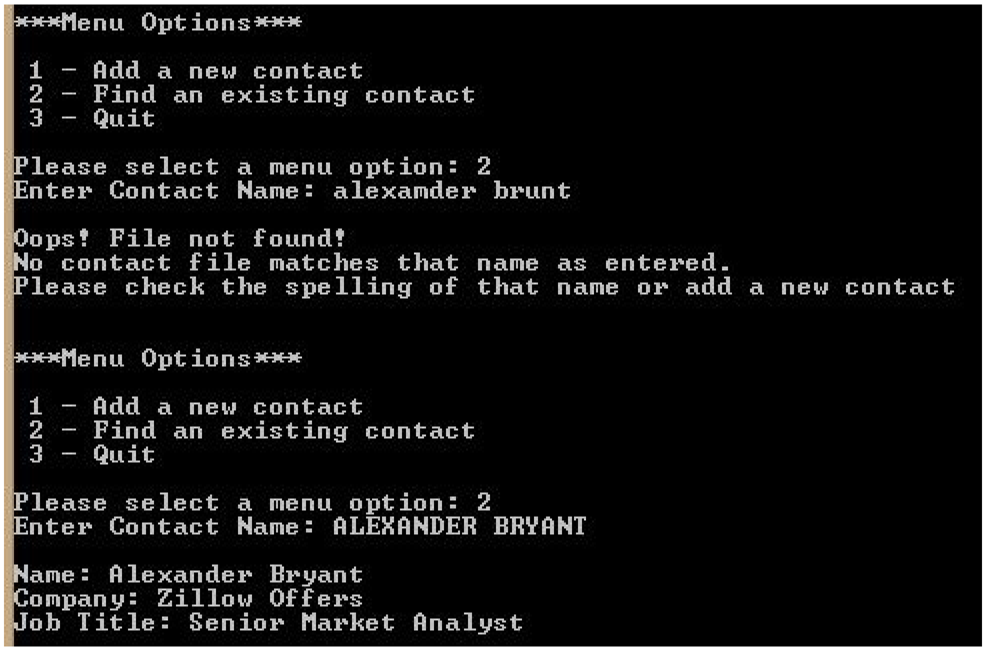
In ***Figure 4*** above, the *review_info* function is defined as a method within the *Processor* class. A prompt asks the user to supply a contact name which is interpreted as a lowercase string and assigned to the local variable *file_name*. The script then attempts to open a binary file based on the name provided by the user. An error handling block in the body of the script is activated if no file matching the supplied name is found.



```
try:  # This try error block prevents a crash if no file with the supplied name is found
    Processor.review_info()
except FileNotFoundError as e:
    print('\nOops! File not found!')
    print('No contact file matches that name as entered.')
    print('Please check the spelling of that name or add a new contact \n')
```

Figure 5: If no matching file is found by the review_info function an error message prints.

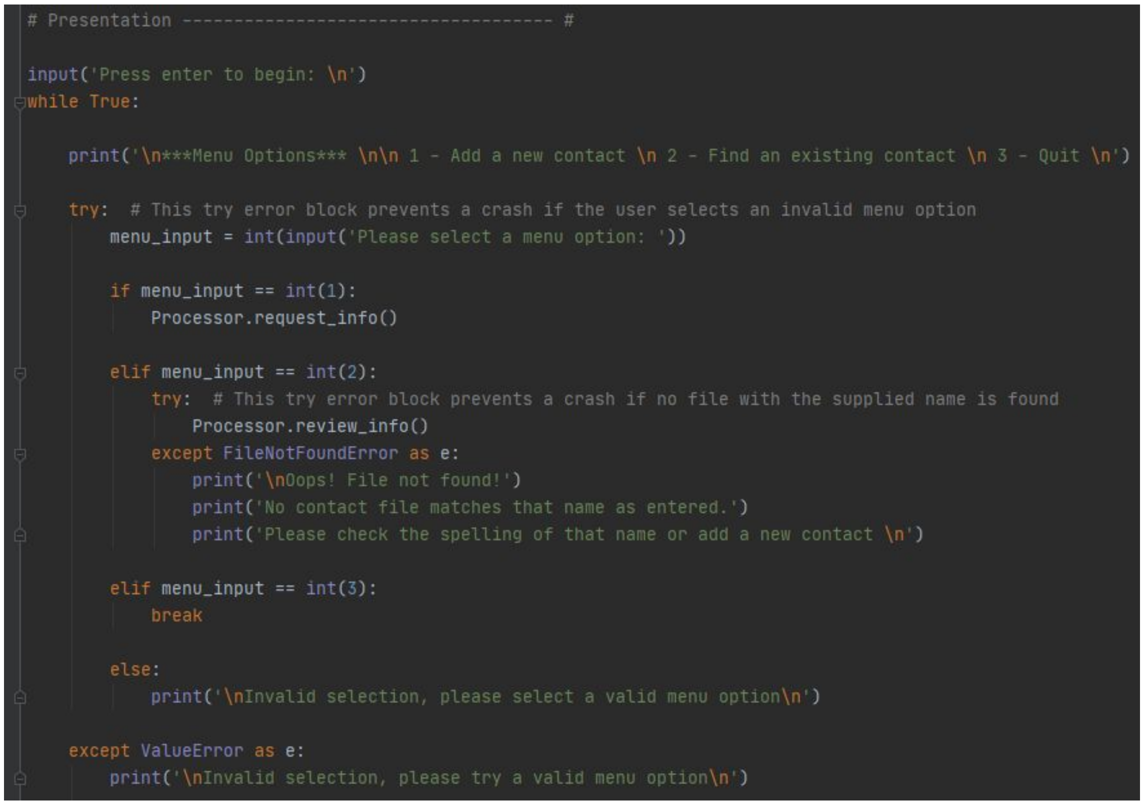
In ***Figure 5*** above, a try-except block prevents the code from crashing due to a *FileNotFoundError*. Because the *review_info* and *save_info* take user input as lowercase strings, case does not need to match a saved file, but characters and spacing does.



```
***Menu Options***

 1 - Add a new contact
 2 - Find an existing contact
 3 - Quit

Please select a menu option: 2
Enter Contact Name: alexamder brunt

Oops! File not found!
No contact file matches that name as entered.
Please check the spelling of that name or add a new contact


***Menu Options***

 1 - Add a new contact
 2 - Find an existing contact
 3 - Quit

Please select a menu option: 2
Enter Contact Name: ALEXANDER BRYANT

Name: Alexander Bryant
Company: Zillow Offers
Job Title: Senior Market Analyst
```

Figure 6: A command line user causes an error due to spelling mistakes

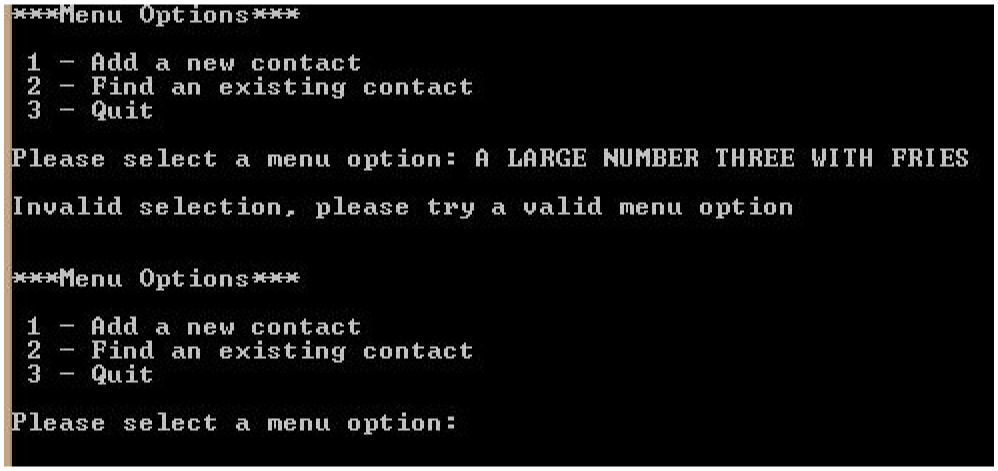
In ***Figure 6*** above, a command line user runs the program and looks for the contact saved earlier, but makes several spelling mistakes. The try-except block prevents the code from crashing due to an error and reminds the user to check the spelling. The user then tries again with the correct spelling, but this time in uppercase to reflect annoyance. The *review_info* function finds the correct file due to the correct characters and spacing, while ignoring the capitalization.



```python
# Presentation ---------------------------------- #

input('Press enter to begin: \n')
while True:

    print('\n***Menu Options*** \n\n 1 - Add a new contact \n 2 - Find an existing contact \n 3 - Quit \n')

    try:  # This try error block prevents a crash if the user selects an invalid menu option
        menu_input = int(input('Please select a menu option: '))

        if menu_input == int(1):
            Processor.request_info()

        elif menu_input == int(2):
            try:  # This try error block prevents a crash if no file with the supplied name is found
                Processor.review_info()
            except FileNotFoundError as e:
                print('\nOops! File not found!')
                print('No contact file matches that name as entered.')
                print('Please check the spelling of that name or add a new contact \n')

        elif menu_input == int(3):
            break

        else:
            print('\nInvalid selection, please select a valid menu option\n')

    except ValueError as e:
        print('\nInvalid selection, please try a valid menu option\n')
```

Figure 7: The presentation block houses the menu in a while loop and try-except block.

The body of the script is in the presentation block. After pressing enter to begin the program, a menu is presented inside of a while loop. A try-except block ensures that the program does not crash if the user selects an invalid selection. Typically I would specify menu options as strings and compare them against a list of appropriate options. However, I specified the variable *menu_input* as an integer to test catching another type of error, the *ValueError*. If the user submits an integer other than 1, 2, or 3, an else statement reminds the user to pick another option. But, if the user submits an entry that can't be interpreted as an integer, such as '1.0' or 'three', then the except statement catches the ValueError. Instead of crashing the program, the user is reminded to select a valid option and is returned to the menu. In ***Figure 8*** below, the command line user mistakes the program for a drive thru window and submits string characters that cannot be interpreted as an integer. Instead of crashing, the user is warned that this is an invalid selection and returned to the menu.



```
***Menu Options***

 1 - Add a new contact
 2 - Find an existing contact
 3 - Quit

Please select a menu option: A LARGE NUMBER THREE WITH FRIES

Invalid selection, please try a valid menu option


***Menu Options***

 1 - Add a new contact
 2 - Find an existing contact
 3 - Quit

Please select a menu option:
```

Figure 8: When menu_option is not an integer, a try-except block prevents a script crash.

Summary

Pickling presents an alternative method of writing and reading files within Python. While these small pieces of contact information do not take up very much space, binary files can compress much larger objects into smaller pieces of information. The most difficult portion of this project was devising a concept program to illustrate pickling and error handling. I chose to model the program after the modest Rolodex, knowing a name would allow the user to look up more information about that contact. This code would function more similar to a Rolodex or phonebook if an additional menu item would allow the user to review all of the names which had been added. For the sake of simplicity to focus on pickling and error handling, I omitted the phonebook feature. Another difficult aspect of this assignment was pre-empting the various types of errors that a user could make that would trigger the code to crash. More extensive testing would likely illustrate a use for try-except blocks in other locations within the script.