David Goldberg

May 015, 2024

Foundations of Programming: Python

Assignment 05

Link to Github: https://github.com/david-goldberg26/IntroToProg-Python-Mod05.git

Dictionaries & Error Handling

Introduction

During module 05 of this course, we learned some extremely important topics that will help us organize our data an easier and more readable format, we learned how to work with exceptions to help deviate from errors when running our script, and we learned a new type of file that allows us to easily write to and read from, called a JSON file. These were used in the assignment to prevent errors, organize data more efficiently, and reduce the amount of code when reading and writing from a file. On top of that we learned a new tool, Git, that allows you to track changes you make to your project so you can go back to earlier versions if you need. Is especially useful if you are working with other people on one project, so they can access the code as well.

Using Dictionaries

Dictionaries are very similar to multidimensional lists; except they store data in key-value pairs where each value or variable is associated with a key. What makes these keys important is that they provide a unique identifier for the values stored to it. They provide more efficient organization and faster lookups. A simple example of a dictionary is shown below.

```
1   student_row1: dict[str, str] = {'first_name':'vic', 'last_name':'vu', 'course':'python 100'}
2   student_row2: dict[str, str]  = {'first_name':'david', 'last_name':'goldberg', 'course':'pytho
3   students: list[dict[str, str]] = [student_row1, student_row2]
```

Figure 1: Dictionary Example

As seen in the example, a dictionary data structure contains strings inside a list. Looking at lines 1 and 2, the strings ‘first_name’, ‘last_name’, and ‘course’ are the keys of the dictionary which can be called later in the script and will populate that with what is associated with the key. If ‘first_name‘ is called in an f’string for example, it will print what it’s associated to, which is vic and david. The same goes with ‘last_name’ and ‘course’. An example can better show my explanation.

```
5   for row in students:
6       print(f"{row['first_name']} {row['last_name']} is registered for {row['course']}")
```

```
vic vu is registered for python 100
david goldberg is registered for python 100
```

Figure 2: Dictionary Output

When the f'string is being looped through each row, it will take the keys from the dictionaries and input what it was associated with. The output is shown at the bottom of figure 2. Another important usage of dictionaries is reading and writing data from a file. Dictionaries allow you to efficiently organize data without having to store everything in large lists. The example below shows how to create a dictionary from inputting a student's first and last name as well as their course.

```
# Read from the file
students = []
file_obj = open(FILE_NAME, 'r')
for line in file_obj:                # Read lines in the file
    parts = line.strip().split(',')            # Strip empty characters and split by ','
    student_first_name = parts[0]
    student_last_name = parts[1]
    student_course_name = parts[2]
    row = {'first_name': student_first_name, 'last_name': student_last_name, 'course_name': student_course_name}
    students.append(row)
```

Figure 3: Reading data into a dictionary

In the example above the data from the file is being stored in a list (parts = [ ]) and then manipulated by its indexes to extract the first name, last name, and course name and then associate it to the keys in the dictionary. Once the loop runs for every row, each dictionary will be appended to a list of dictionaries. Since this process seems slightly lengthy, there is a way to shorten the amount of code written dramatically by using .json files.

JSON Files

A JSON file is a text file that contains data formatted according to its syntax. JSON or JavaScript Object Notation consists of keys where the keys can be strings and values and the strings associated to the keys can be numbers, strings, and Booleans. What makes these types of files very useful is that they can contain an existing dictionary, which allows it to be easily read and written using the built in 'json' library import. An example of a JSON file is shown below.

```
50  [
51      {'first_name':'vic', 'last_name':'vu', 'IsRegistered': True, 'GPA': 3.7}
52      {'first_name':'Sue', 'last_name':'Jones', 'IsRegistered': False, 'GPA': 3.1}
53  ]
```

Figure 4: JSON File Example

This looks nothing different than what we've seen before. It is simply a list of dictionaries that include keys with associated numbers, strings, and Booleans. The great part about JSON files is that it can be read from the file with one simple line, shown below.

```
file = open(FILE_NAME, 'r')
students = json.load(file)
print(students)
```

Figure 5: Reading in JSON File

This process requires one line to read in a list of dictionaries, and the great part about this is that the format is unchanged. So from this command, 'students', ends up being a list of dictionaries. Writing to a file is similar, instead of using a .load command, you can use a dump command which will essentially "dump" all of the dictionaries into your JSON file. This is shown below.

```
file = open(FILE_NAME, 'w')
json.dump(students, file)
file.close()
```

Figure 5: Writing to JSON File

Very similar to the example before, writing to a JSON file requires one line of code, which basically dumps the information from 'students' into the file. Overall, JSON files are extremely efficient with transferring data from system to system and allow for quick reading/writing from our Python scripts.

Exceptions

As we move forward with expanding our Python knowledge, it is important to use exceptions. Exceptions in Python are unexpected events or errors that occur during the program execution, which usually disrupts the flow of the program. To prevent these interruptions, exceptions provide a way to handle errors and prevent the program from crashing. It can tell the user that a specific error is occurring, but will not crash your code. For example, if you want to read in an existing file in your directory and the file does not exist yet, Python will crash the code and output FileNotFoundError. This is where an exception can be implemented since we know the possible error. This example is shown below.

```
91  try:
92      with open("non_existent_file.txt", "r") as file:
93          content = file.read()
94          print(content)
95
96  except FileNotFoundError:
97      print("Error: The file does not exist.")
98
99  finally:
100     print("This code will always execute")
```

Figure 6: Exception Example

In the example, we see that a file is trying to be opened, but if the file doesn't exist it will produce and error. But since we have the 'except' line with the FileNotFoundError, Python will not stop the code, but instead warn the user the file does not exist and continue to proceed. If the 'finally' block is used after the 'except', it will contain the code that is always executed, regardless of an exception that occurred in the try block. Overall, this is an extremely useful tool that allows fluidity through our code. It will also be used in the module 05 assignment which will be explained in the next section of this paper.

Writing the Script

Now that we covered the essentials that were taught in module 5, it is time to explain the code for the assignment. As mentioned in the last module, the structure and output of the code is very similar, except this assignment involves dictionaries, structured error handling, and the use of .json files. The first part of the script is to define the constants and variables that will be used.

```
10  # Imports
11  import json     # Importing the json
12
13
14
15  # Define the Data Constants
16  MENU: str = '''
17  ---- Course Registration Program ----
18    Select from the following menu:
19      1. Register a Student for a Course.
20      2. Show current data.
21      3. Save data to a file.
22      4. Exit the program.
23  ----------------------------------------
24  '''
25  FILE_NAME: str = "Enrollments.json"
26
27  # Define Variables
28  student_first_name: str = ''  # Holds the fi
29  student_last_name: str = ''  # Holds the las
30  course_name: str = ''  # Holds the name of a
31  student_data: list = []  # one row of studer
32  students: list = []  # a table of student da
33  csv_data: str = ''  # Holds combined string
34  file = None  # Holds a reference to an opene
35  menu_choice: str = ''  # Hold the choice mad
36  student_data: dict = []     # Dictionary tha
37  students: list = []     # list that contains
```

Figure 7: Constants and Variables

The only difference between this and last weeks constants and variables is that there is a dictionary being defined on line 36 and json being imported on line 11. Importing json will allow us to use and manipulate .json files in the script. Next, we want to read in existing data that may exist in the JSON file which is shown below.

```
39  try:
40      # read through file to input previous entries
41      file = open(FILE_NAME, 'r')     # open the file in read mode
42      students = json.load(file)      # load the previous dictionaries from
43      print("\n")
44      print(students)
45      print("\n")
46      file.close()
47  except FileNotFoundError as e:      # Structured error handling if the fil
48      print(e)
49      print('''
50            This file does not exist or not previous data has been inputted
51            ''')        # Should I also open the file
52  finally:
53      print("\t"'This code will execute whether you have the file or not')
```

Figure 8: Reading from JSON File

Compared to last week, reading from a JSON file requires less code, which makes this more efficient. It is also used in a structured error seen in line 39, which will try the code until 47. If the file does no exist it will not crash the script, but instead just output to the user that there was no file to begin with. The 'finally' on line 52 will print the message whether there was an error or not. This will allow for more fluidity when reading the model. The exception used is FileNotFoundError which, originally, would crash the script if there was no previous file saved. But now it will still throw the error to the user and will continue on with the script.

The next section is the large while loop which was used in the previous module, except now when the student chooses option one from the menu, it will have to take their inputs and create a dictionary with them. This is shown below.

```
56      while (True):
57
58          # Present the menu of choices
59          print(MENU)
60          menu_choice = input("What would you like to do: ") # Will ask to choose from the menu
61
62          # Input user data
63          if menu_choice == "1":
64              try:
65                  student_first_name = input("Enter the student's first name: ")      # input first name
66                  student_last_name = input("Enter the student's last name: ")        # input last name
67                  if not student_first_name.isalpha() or not student_last_name.isalpha:       # check if the first n
68                      raise ValueError('''
69                                       inputted names can only use letters
70                                       ''')       # Will output a Value Error
71              except ValueError as e:     # Structured error handling if the input is non-alphabetic
72                  print(e)
73                  print('''
74                        enter a valid name that only uses the alphabet
75                        ''')
76                  continue
77              course_name = input("Please enter the name of the course: ")        # input course name
78              student_data = {"first_name":student_first_name, "last_name":student_last_name, "course":course_name}
79              students.append(student_data)       # append each dictionary at a list
80              print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")      # print
```

Figure 9: Writing to a Dictionary in Option 1

Once the student inputs their name and course the code will check if they input a value, which will output an error. The try/except block, on lines 71-76, will prevent the code from crashing and warn the student to only input letters in their name. It will then force them to re-input their first and last name. Then student_data is introduced, which is the dictionary we are going to write the inputs to. The dictionary's keys are 'first_name', 'last_name', and 'course' which will be used from now on in the script to call the student's inputs. Finally, the dictionary will get appended to the list called students, on line 79. Students could be a multidimensional list depending on whether there were previous entries.

Next, once the student selects option 2, it will show the data to them, shown below.

```
87          # Present the current data
88          elif menu_choice == "2":
89
90              # Process the data to create and display a custom message
91              print("-"*50)
92              for entry in students:      # loop through each entry in the dictionary
93                  print(f"Student {entry["first_name"]} {entry["last_name"]} is enrolled in {entry["course"]}")
94                  print("-"*50)
```

Figure 10: Printing from a Dictionary

If I would just print the dictionary out to the student, it would not be easy to understand and would be disorganized. So instead, I will loop through each entry and print the key in each entry. Since the key stays the same through every entry, this allows for the student's names and course name to be displayed. This is done from lines 92 – 93. When the student selects option 3, the script will write the list of lists (students) into the JSON file. This is shown below.

```
97          # Save the data to a file
98          elif menu_choice == "3":
99              try:
100                 file = open(FILE_NAME, 'w')     # open a file to wri
101                 json.dump(students, file)       # write the dictiona
102                 file.close()        # close the file
103             except Exception as e:      # Structured error handling
104                 print(e, e.__doc__)
105                 print(f"An unexpected error occurred: {e}")
106             finally:
107                 if file.closed == False:
108                     file.close()        # close/save file
```

Figure 11: Writing to JSON File

Using a JSON file allows for fewer lines to write to it. Line 101 is the only command you need to write to a JSON file, which makes life much easier. I also used structured error handling for this case as well, just in the case of any error. Once the student selects 3, the dictionaries will be written to the JSON file and then the student can select 4 to exit out of the program!

Summary

Overall, we learned a few extremely useful topics in this module that we can apply to our code. These topics make our code run more fluid and allow us to store data more efficiently. Dictionaries allow us to store data to keys which can be used throughout the code, allowing for more efficient data storage. JSON files allow us to store dictionaries inside them and simplify the task to write/read from those files as well. Finally, we learned how to use structured error handling which allows us to spot errors in our code but prevent the code from crashing while it's running.