

Name: Varun Reddy Param

Date: Nov 12th 2024

Course: Foundations Of Programming: Python

Assignment: (Assignment 05) - Create a Python program that uses constants, variables, and print statements to display a student's registration for a Python course, incorporating data processing using dictionaries and exception handling.

GitHub: <https://github.com/varunreddyparam/IntroToProg-Python-Mod05>

.

Learning usage on dictionaries, Json files, and exception handling

Introduction:

In this assignment, I worked on creating a Python program for a student registration system that demonstrates concepts such as JSON file handling, dictionaries, lists of dictionaries, exception handling, and error raising. This document explains the process and the concepts I learned while working through the assignment.

Declaring Data Constants

- **Menu Constant:** Displays the menu options for the user to choose from.
- **File Name Constant:** Defines the file name for saving student enrollments.
- Constants like MENU and FILE_NAME are used to store values that do not change during the execution of the program.

```

9
10 # Define the Data Constants
11 MENU: str = ''
12 ---- Course Registration Program ----
13     Select from the following menu:
14         1. Register a Student for a Course.
15         2. Show current data.
16         3. Save data to a file.
17         4. Exit the program.
18     -----
19     '''
20
21 # Define the Data Constants
22 FILE_NAME: str = "Enrollments.json"

```

Figure 1: Declaring Constants with data Types

Declaring Variables in Python

Store data that changes as the program runs.

- *student_first_name*, *student_last_name*, and *course_name* are used to store the student's information.
- *csv_data* is used to store formatted data as a comma-separated string
- *file* is used for read and write data to Json file
- *menu_choice* stores the user's choice from the menu
- *student_data* is a dictionary that holds data for a single student, and *students* is a list of dictionaries, each representing an individual student's record.

```

24 # Define the Data Variables and constants
25 student_first_name: str = '' # Holds the first name of a student entered by the user.
26 student_last_name: str = '' # Holds the last name of a student entered by the user.
27 course_name: str = '' # Holds the name of a course entered by the user.
28 student_data: dict = {} # one row of student data
29 students: list = [] # a table of student data
30 csv_data: str = '' # Holds combined string data separated by a comma.
31 file = None # Holds a reference to an opened file.
32 menu_choice: str # Hold the choice made by the user.
33

```

Figure 2: Declaring Variables

Collections: Using Lists and Dictionaries to Store Data

The program uses both single-dimensional and two-dimensional lists to manage student enrollment data effectively:

- **Single-Dimensional Dictionary (*student_data*)** Holds the information for a single student, including first name, last name, and course name.
- **Two-Dimensional List (students)**, A collection of all *student_data* entries, making it a list of dictionaries where each element represents an individual student's record.

Properties of Lists

- **Ordered:** Lists maintain the order of items, meaning that items are stored and retrieved in the same sequence.
- **Mutable:** Lists can be modified after their creation. This allows adding, updating, or deleting elements easily.
- **Appending Data:** The program uses `students.append(student_data)` to add new student records to the list. This method ensures that new data is added to the end of the list.

Using lists as collections helps in easily managing multiple records and performing operations such as adding, displaying, and saving data.

Properties of Dictionary

- **Key-Value Pair Structure:** Dictionaries store data in key-value pairs, making it easy to access values using their corresponding keys. For example, *student_data* has keys like `FirstName`, `LastName`, and `CourseName` which map to the student's details.
- **Unordered:** Unlike lists, dictionaries do not maintain the order of items. The data is stored in a way that allows for fast access, regardless of the order in which items were added.
- **Mutable:** Dictionaries are mutable, meaning that they can be modified after their creation. This allows for adding, updating, or deleting key-value pairs as needed.
- **Adding and Updating Data:** The program uses statements like `student_data['FirstName'] = student_first_name` to add or update data in the dictionary. This flexibility makes dictionaries ideal for managing structured data.

Json File handling:

The program uses JSON to save student records and reload them when the program restarts.

- The program reads data from *Enrollments.json* at startup to populate the students list.
- The *json.load(file)* function is used to parse the JSON file into a list of dictionaries. Error handling is implemented to manage scenarios where the file might not exist.

```
34 # ----- #
35 # When the program starts,
36 # the contents of the "Enrollments.json" are automatically read into a two-dimensional list table (a list of dictionary rows).
37 # (Tip: Make sure to put some starting data into the file or you will get an error!)
38 # ----- #
39
40 # Read all the data from Json file and load it to a collection
41 try:
42     file = open(FILE_NAME,"r")
43     students = json.load(file)
44 except FileNotFoundError as e:
45     print("file must exist before running this script!\n")
46     print("-- Technical Error Message -- ")
47     print(e, e.__doc__, type(e), sep='\n')
48 except Exception as e:
49     print("There was a non-specific error!\n")
50     print("-- Technical Error Message -- ")
51     print(e, e.__doc__, type(e), sep='\n')
52 finally:
53     if file.closed == False:
54         file.close()
```

Figure 3: reading Json file to students List

Writing Data to JSON:

- The *json.dump(students, file)* function is used to save the students list to *Enrollments.json* in JSON format. The file is then properly closed to ensure all data is saved and the file is no longer in use.

```
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

Figure 4: Writing Data to Json

Exception Handling and errors

Error handling is important because it helps make programs stable and easy to use. It makes sure that the program can deal with unexpected problems without crashing.

- Error handling is used to prevent program crashes when unforeseen errors occur, allowing the program to continue running or terminate gracefully.

- It provides meaningful feedback to the user, helping them understand what went wrong and how to fix it.
- It also helps developers identify and debug issues effectively.

File Handling Errors:

- When working with files, there is always a possibility that the file might not exist, be corrupted, or be inaccessible. In such cases, a *FileNotFoundError* or other *IOError* might be raised.
- In this program, when trying to read from *Enrollments.json*, if the file does not exist, a *FileNotFoundError* is caught, and an appropriate message is displayed to the user.

```

40 # Read all the data from Json file and load it to a collection
41
42 try:
43     file = open(FILE_NAME,"r")
44     students = json.load(file)
45 except FileNotFoundError as e:
46     print("file must exist before running this script!\n")
47     print("-- Technical Error Message -- ")
48     print(e, e.__doc__, type(e), sep='\n')
49 except Exception as e:
50     print("There was a non-specific error!\n")
51     print("-- Technical Error Message -- ")
52     print(e, e.__doc__, type(e), sep='\n')
53 finally:
54     if file.closed == False:
55         file.close()

```

Figure 5 File Handling error and generic exception

Generic Exception Handling:

- In situations where an unexpected error occurs that is not specifically handled (e.g., a value error, permission error, etc.), a generic `except Exception as e` block can be used to catch all other exceptions.
- This makes that the program provides a generic error message instead of crashing without explanation.

Input Validation:

- When capturing user input, it's important to ensure that the data is valid. For example, an empty first name or last name is not acceptable. To enforce this, a *ValueError* is raised if the input is invalid.
- Raising custom errors helps maintain data integrity and ensures that only correct and expected data is processed.
- Gained some insights around type of exceptions we can use (<https://docs.python.org/3.13/library/exceptions.html#ValueError>)
- The program prompts the user to enter the student's first name, last name, and the course name. These are stored in the respective variables.
- Collecting and storing this data allows for later use in displaying and saving the registration details.

```
except ValueError as e:
    print(f"Error: {e}")
    student_last_name = input('Please enter a valid Last Name: ')
```

Figure 6 Input validation value error

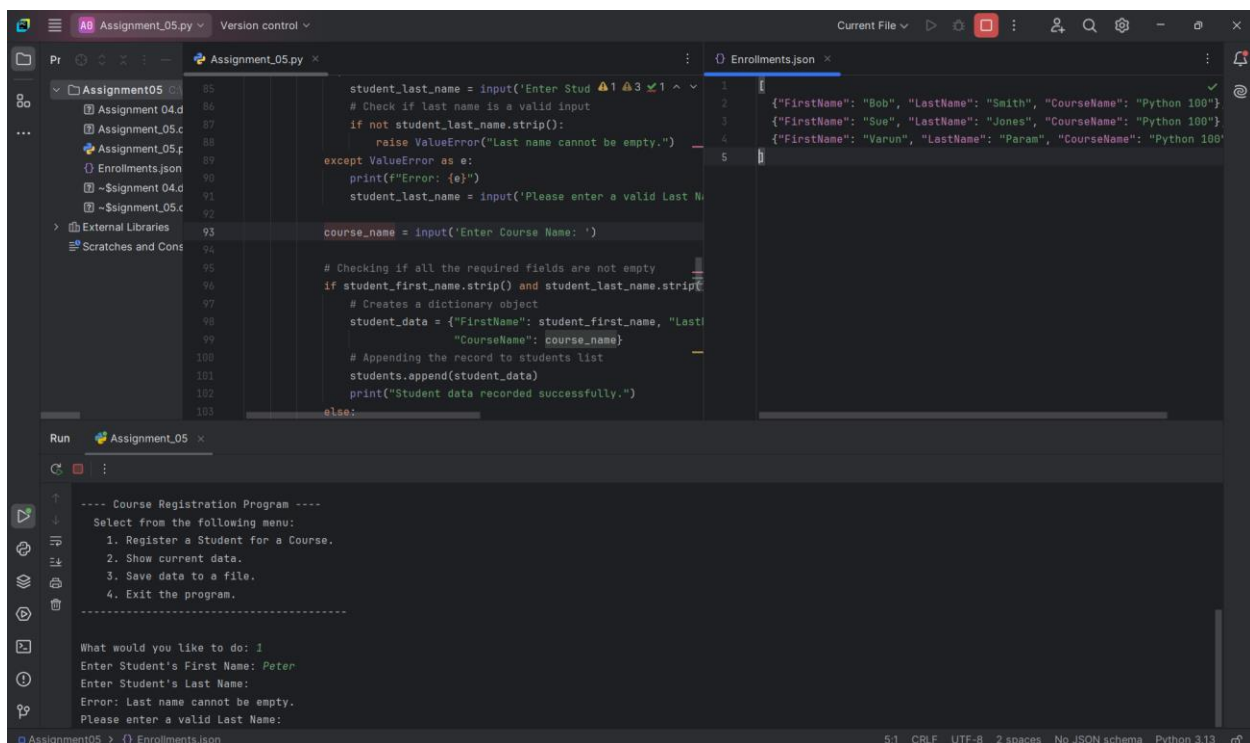
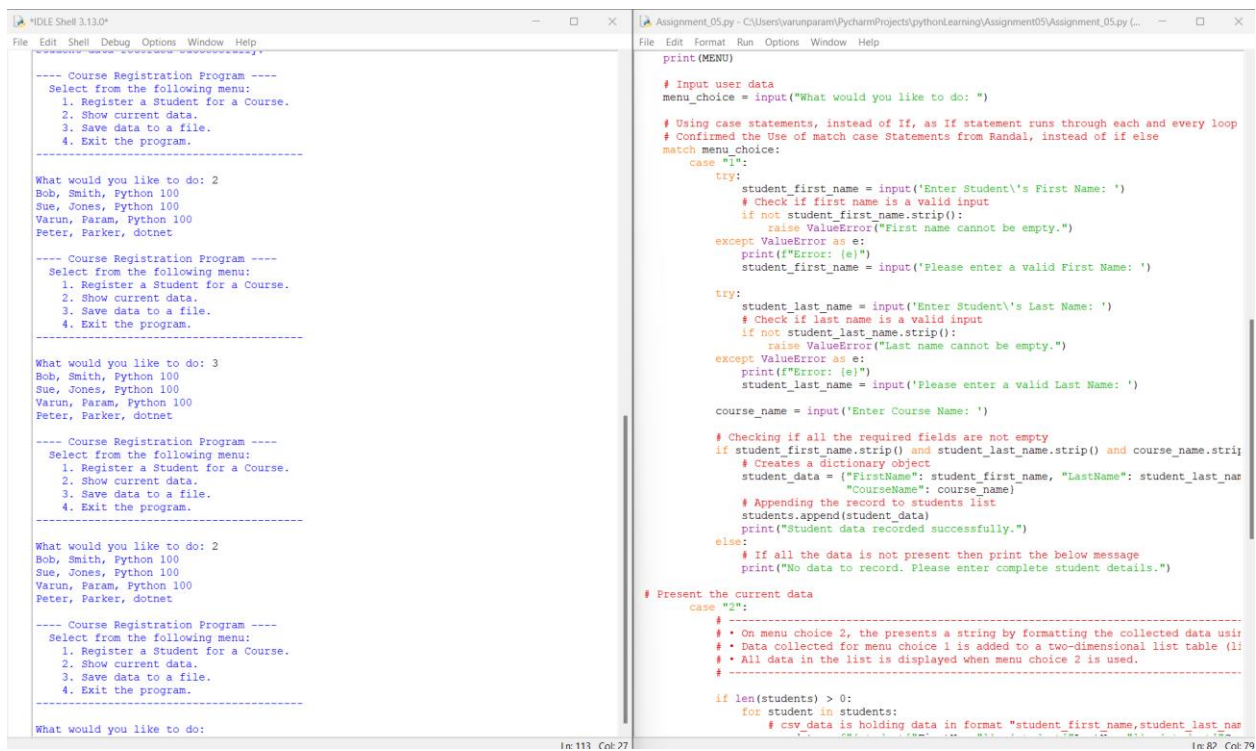


Figure 7: Py charm IDE Execution



```
----- Course Registration Program -----
Select from the following menu:
1. Register a Student for a Course.
2. Show current data.
3. Save data to a file.
4. Exit the program.

-----
What would you like to do: 2
Bob, Smith, Python 100
Sue, Jones, Python 100
Varun, Param, Python 100
Peter, Parker, dotnet

----- Course Registration Program -----
Select from the following menu:
1. Register a Student for a Course.
2. Show current data.
3. Save data to a file.
4. Exit the program.

-----
What would you like to do: 3
Bob, Smith, Python 100
Sue, Jones, Python 100
Varun, Param, Python 100
Peter, Parker, dotnet

----- Course Registration Program -----
Select from the following menu:
1. Register a Student for a Course.
2. Show current data.
3. Save data to a file.
4. Exit the program.

-----
What would you like to do: 2
Bob, Smith, Python 100
Sue, Jones, Python 100
Varun, Param, Python 100
Peter, Parker, dotnet

----- Course Registration Program -----
Select from the following menu:
1. Register a Student for a Course.
2. Show current data.
3. Save data to a file.
4. Exit the program.

-----
What would you like to do:
```

```
print(MENU)

# Input user data
menu_choice = input("What would you like to do: ")

# Using case statements, instead of If, as If statement runs through each and every loop
# Confirmed the Use of match case Statements from Randal, instead of if else
match menu_choice:
    case "1":
        try:
            student_first_name = input('Enter Student\'s First Name: ')
            # Check if first name is a valid input
            if not student_first_name.strip():
                raise ValueError("First name cannot be empty.")
            except ValueError as e:
                print(f"Error: {e}")
                student_first_name = input('Please enter a valid First Name: ')

        try:
            student_last_name = input('Enter Student\'s Last Name: ')
            # Check if last name is a valid input
            if not student_last_name.strip():
                raise ValueError("Last name cannot be empty.")
            except ValueError as e:
                print(f"Error: {e}")
                student_last_name = input('Please enter a valid Last Name: ')

        course_name = input('Enter Course Name: ')

        # Checking if all the required fields are not empty
        if student_first_name.strip() and student_last_name.strip() and course_name.strip():
            # Creates a dictionary object
            student_data = {"FirstName": student_first_name, "LastName": student_last_name,
                           "CourseName": course_name}

            # Appending the record to students list
            students.append(student_data)
            print("Student data recorded successfully.")
        else:
            # If all the data is not present then print the below message
            print("No data to record. Please enter complete student details.")

    # Present the current data
    case "2":
        # -----
        # • On menu choice 2, the presents a string by formatting the collected data using
        # • Data collected for menu choice 1 is added to a two-dimensional list table (l)
        # • All data in the list is displayed when menu choice 2 is used.
        # -----

        if len(students) > 0:
            for student in students:
                # csv_data is holding data in format "student_first_name,student_last_name,course_name"
                csv_data = f"{student['FirstName']},{student['LastName']},{student['CourseName']}"
                print(csv_data)
```

Figure 8: Executed using Idle

Summary

In this assignment, I learned about using constants, variables, conditional logic (*if-else*), different types of loops (*while* and *for*), match-case statements, and file handling in Python. We created a menu-driven program that allows users to register a student, display the current registration data from a JSON file, and save the data back to the JSON file. I also learned how to use error handling to manage unexpected issues and keep the program running smoothly.