Justin Price

May 26, 2024

IT FDN 100 A

Assignment 07

GitHub: https://github.com/jwprice3 (external)

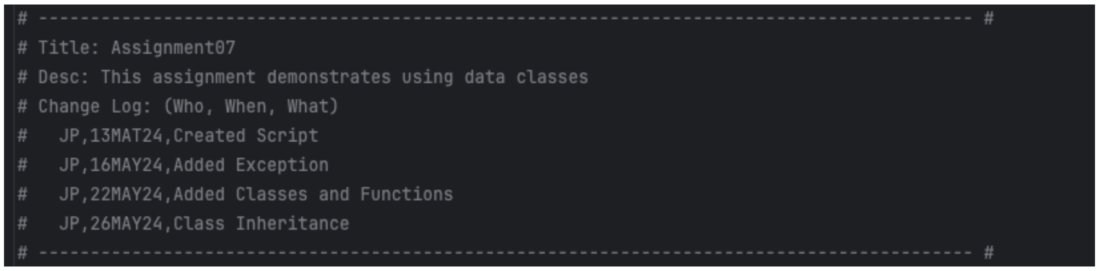
The Code of Requirements

Intro:

It has been almost a year since Def, Param, and Static had shown Hogwarts University that they formed the Classless Hallows. Since then, no student could register for a class because the school could not afford to maintain its overly excessive and expensive school grounds. Professor Root was angry as did not see the traces of the Dark Python Programming Arts sooner. Professor Justin was sure that there was a way to defeat the brothers, but he could not do it alone. He had to enlist help from others, but how? He paced his room for hours until he had an idea. He would create a script for students and faculty to decipher. If they could determine which class they belonged to, then they would be enlisted in Professor Roots' coding Army.

Body:

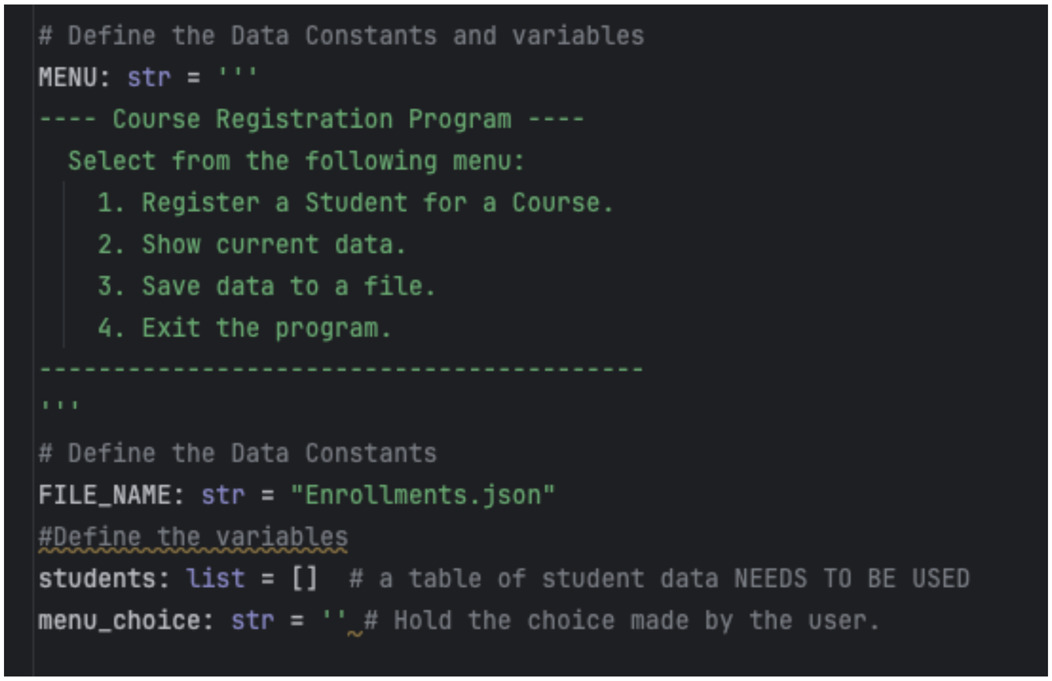
To mimic multiple entries to a script, I will implement the change log as one continuous script.



```
# ------------------------------------------------------------------------------------------ #
# Title: Assignment07
# Desc: This assignment demonstrates using data classes
# Change Log: (Who, When, What)
#   JP,13MAT24,Created Script
#   JP,16MAY24,Added Exception
#   JP,22MAY24,Added Classes and Functions
#   JP,26MAY24,Class Inheritance
# ------------------------------------------------------------------------------------------ #
```

Figure 1.1 Description of File Admin

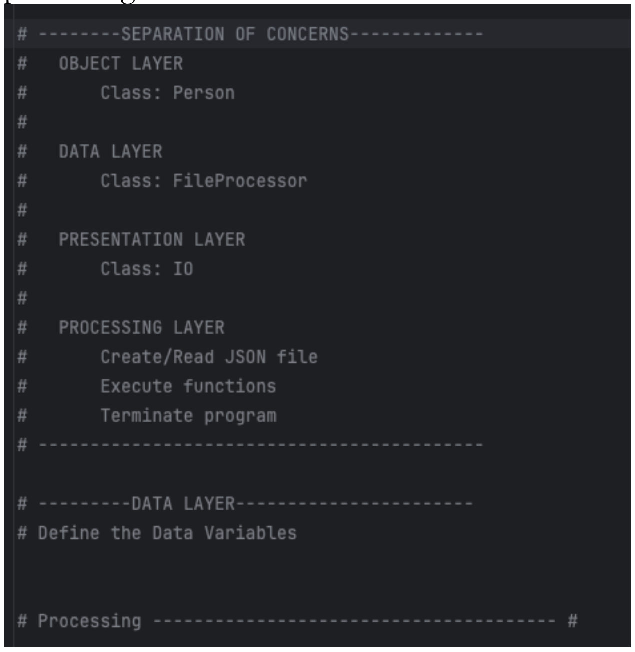
This version has a lot fewer variables since we are embedding them in the function calls.



```
# Define the Data Constants and variables
MENU: str = '''
---- 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.
-----------------------------------------
'''
# Define the Data Constants
FILE_NAME: str = "Enrollments.json"
#Define the variables
students: list = []  # a table of student data NEEDS TO BE USED
menu_choice: str = ''# Hold the choice made by the user.
```

Figure 2.1 Constants and Variables

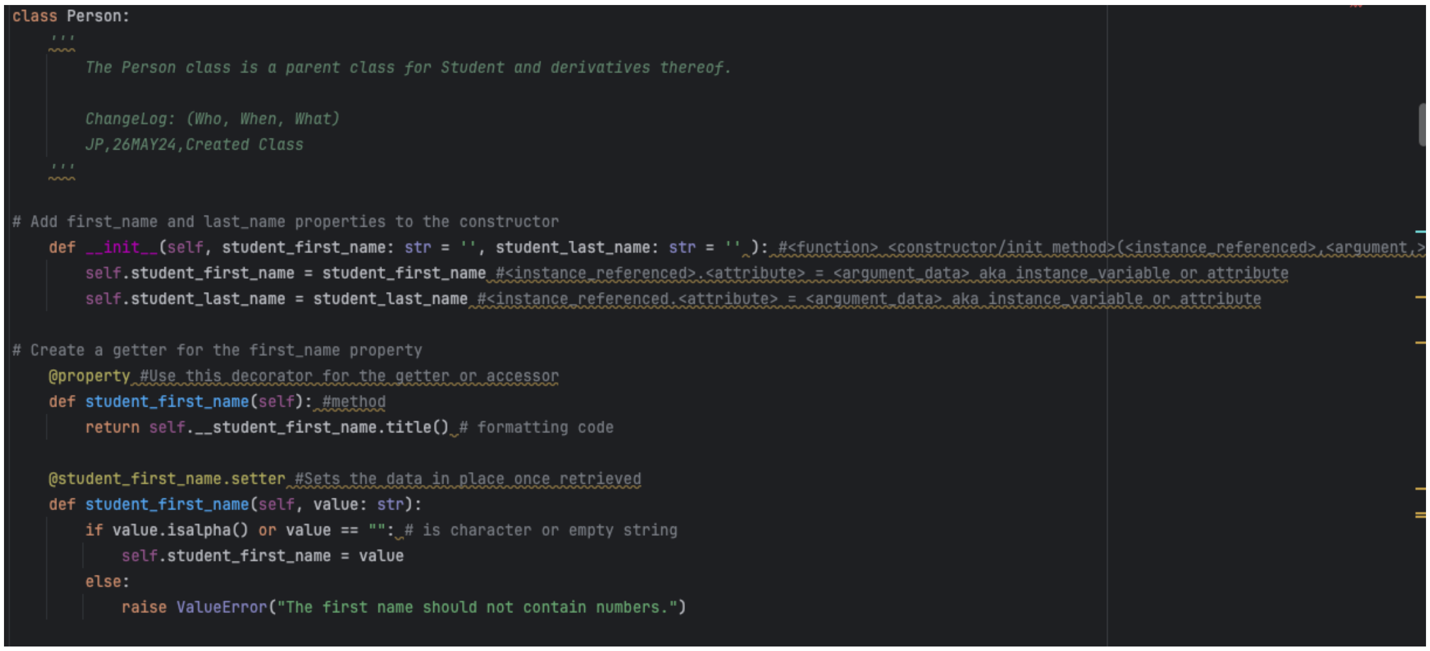
Separation of concerns is like a table of contents for the user's script; it should help prepare the reader for processing the information.



```
# --------SEPARATION OF CONCERNS------------
#   OBJECT LAYER
#       Class: Person
#
#   DATA LAYER
#       Class: FileProcessor
#
#   PRESENTATION LAYER
#       Class: IO
#
#   PROCESSING LAYER
#       Create/Read JSON file
#       Execute functions
#       Terminate program
# -------------------------------------------

# ---------DATA LAYER----------------------
# Define the Data Variables


# Processing --------------------------------------- #
```

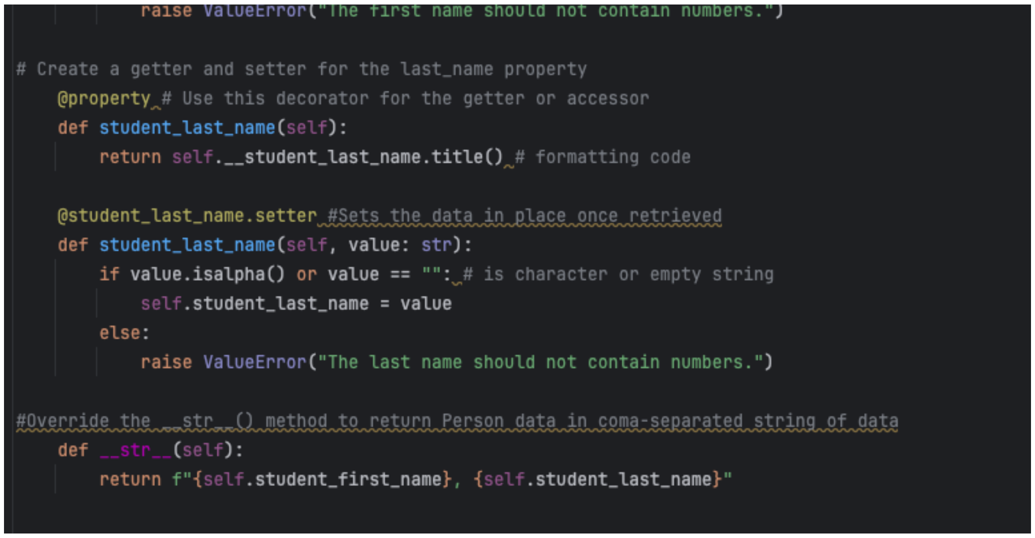
Figure 3.1 Separation of Concerns

I added the Person class as a parent class and the student class as a child class.



```python
class Person:
    '''
        The Person class is a parent class for Student and derivatives thereof.

        ChangeLog: (Who, When, What)
        JP,26MAY24,Created Class
    '''

# Add first_name and last_name properties to the constructor
    def __init__(self, student_first_name: str = '', student_last_name: str = '' ): #<function> <constructor/init method>(<instance_referenced>,<argument,>
        self.student_first_name = student_first_name #<instance_referenced>.<attribute> = <argument_data> aka instance_variable or attribute
        self.student_last_name = student_last_name #<instance_referenced.<attribute> = <argument_data> aka instance_variable or attribute

# Create a getter for the first_name property
    @property #Use this decorator for the getter or accessor
    def student_first_name(self): #method
        return self.__student_first_name.title() # formatting code

    @student_first_name.setter #Sets the data in place once retrieved
    def student_first_name(self, value: str):
        if value.isalpha() or value == "": # is character or empty string
            self.student_first_name = value
        else:
            raise ValueError("The first name should not contain numbers.")
```

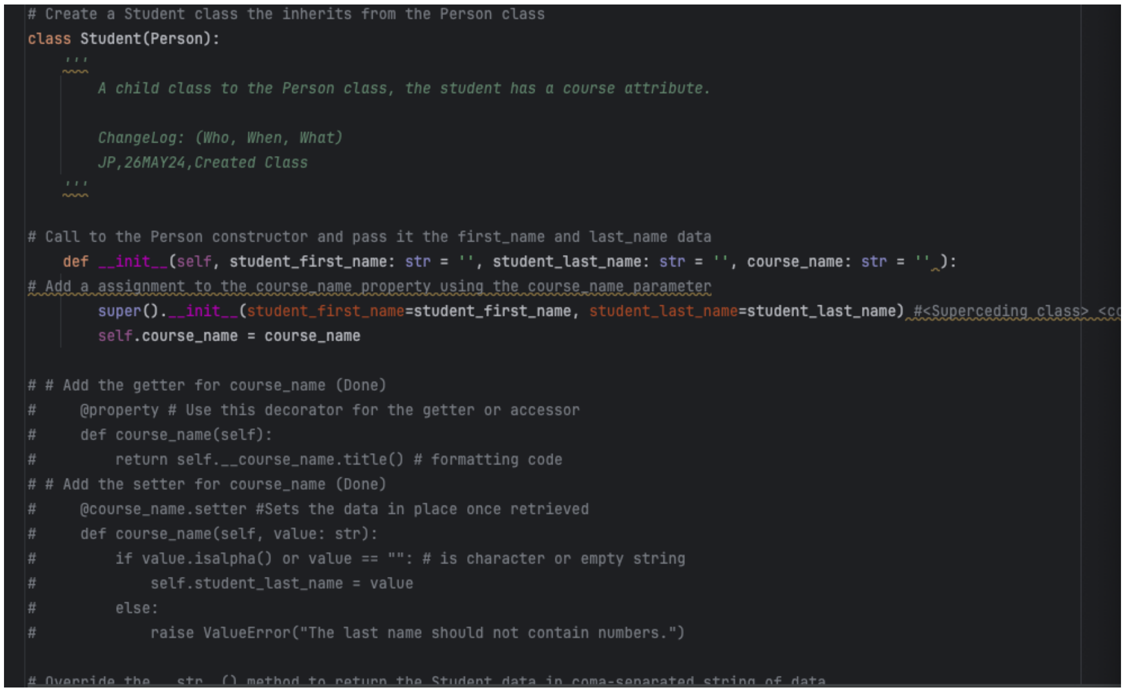
Figure 4.1 Person class



```python
            raise ValueError("The first name should not contain numbers.")

# Create a getter and setter for the last_name property
    @property # Use this decorator for the getter or accessor
    def student_last_name(self):
        return self.__student_last_name.title() # formatting code

    @student_last_name.setter #Sets the data in place once retrieved
    def student_last_name(self, value: str):
        if value.isalpha() or value == "": # is character or empty string
            self.student_last_name = value
        else:
            raise ValueError("The last name should not contain numbers.")

#Override the __str__() method to return Person data in coma-separated string of data
    def __str__(self):
        return f"{self.student_first_name}, {self.student_last_name}"
```

Figure 4.2 Person class

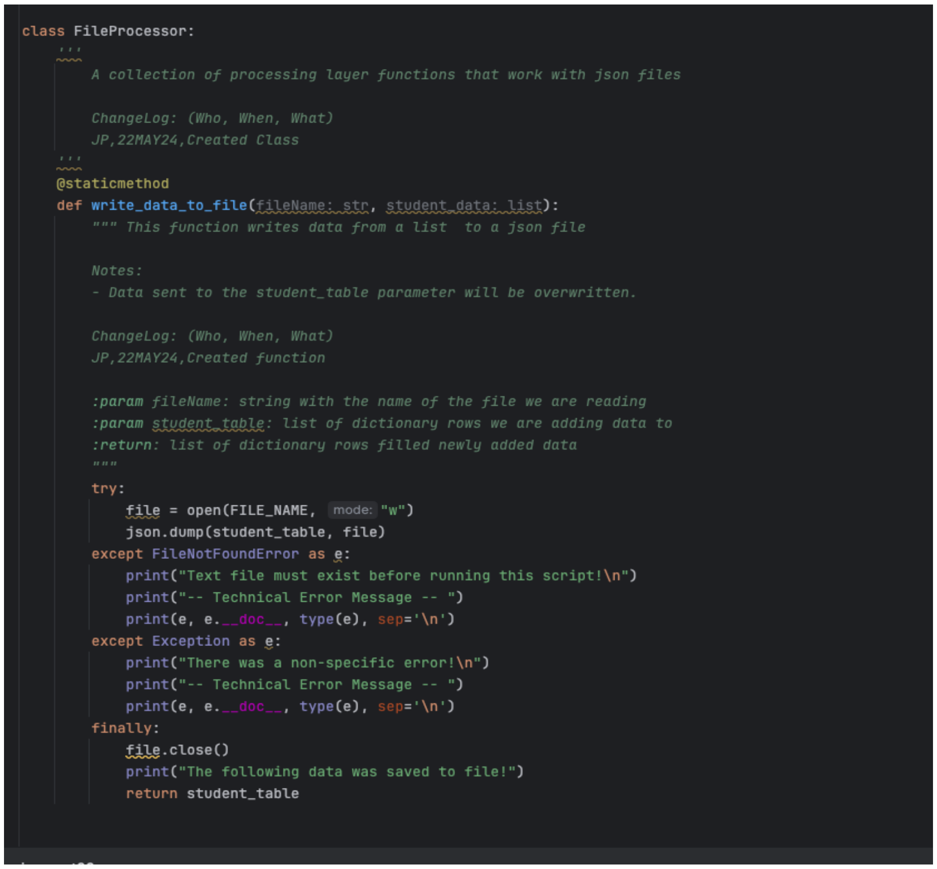
The student class inherited attributes from the parent class. To make the script cleaner I commented out the redundant code.



```python
# Create a Student class the inherits from the Person class
class Student(Person):
    '''
        A child class to the Person class, the student has a course attribute.

        ChangeLog: (Who, When, What)
        JP,26MAY24,Created Class
    '''

# Call to the Person constructor and pass it the first_name and last_name data
    def __init__(self, student_first_name: str = '', student_last_name: str = '', course_name: str = ''):
# Add a assignment to the course_name property using the course_name parameter
        super().__init__(student_first_name=student_first_name, student_last_name=student_last_name) #<Superceding class> <cc
        self.course_name = course_name

# # Add the getter for course_name (Done)
#     @property # Use this decorator for the getter or accessor
#     def course_name(self):
#         return self.__course_name.title() # formatting code
# # Add the setter for course_name (Done)
#     @course_name.setter #Sets the data in place once retrieved
#     def course_name(self, value: str):
#         if value.isalpha() or value == "": # is character or empty string
#             self.student_last_name = value
#         else:
#             raise ValueError("The last name should not contain numbers.")

# Override the __str__() method to return the Student data in coma-separated string of data
```

Figure 5.1 Student class

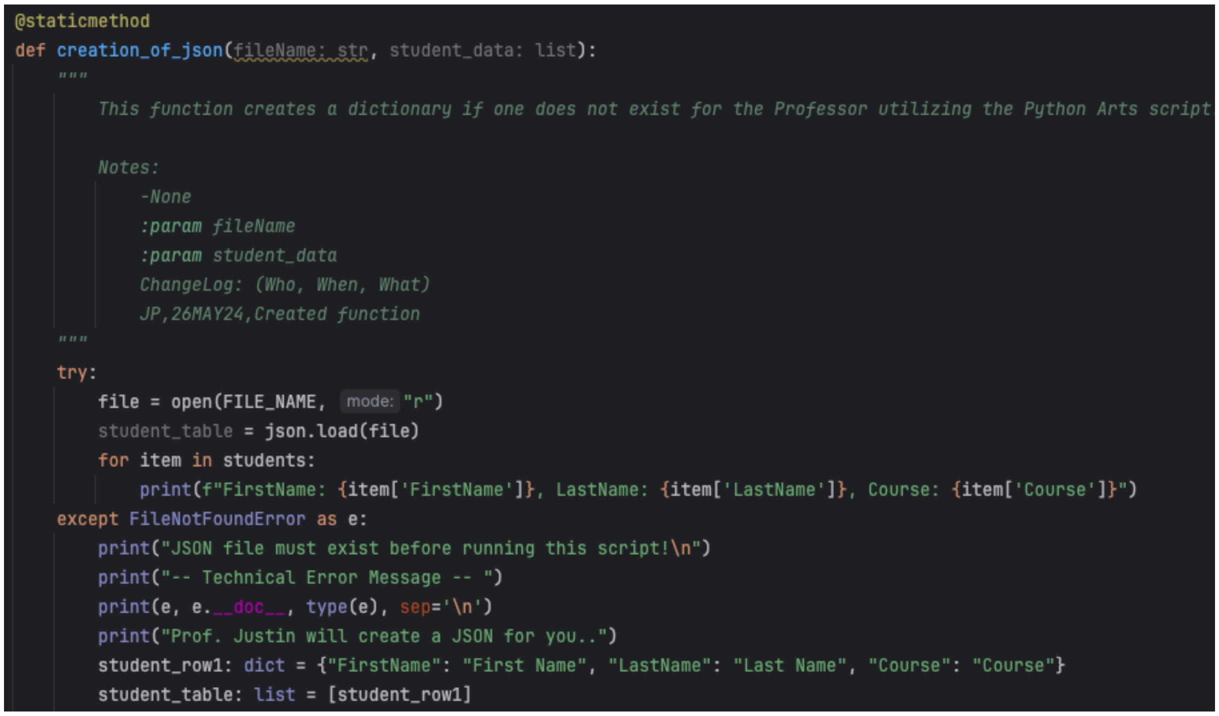
The FileProccessor class mostly remained the same, however I did add a function.



```python
class FileProcessor:
    '''
    A collection of processing layer functions that work with json files

    ChangeLog: (Who, When, What)
    JP,22MAY24,Created Class
    '''
    @staticmethod
    def write_data_to_file(fileName: str, student_data: list):
        """ This function writes data from a list  to a json file

        Notes:
        - Data sent to the student_table parameter will be overwritten.

        ChangeLog: (Who, When, What)
        JP,22MAY24,Created function

        :param fileName: string with the name of the file we are reading
        :param student_table: list of dictionary rows we are adding data to
        :return: list of dictionary rows filled newly added data
        """
        try:
            file = open(FILE_NAME,  mode: "w")
            json.dump(student_table, file)
        except FileNotFoundError as e:
            print("Text file must exist before running this script!\n")
            print("-- Technical Error Message -- ")
            print(e, e.__doc__, type(e), sep='\n')
        except Exception as e:
            print("There was a non-specific error!\n")
            print("-- Technical Error Message -- ")
            print(e, e.__doc__, type(e), sep='\n')
        finally:
            file.close()
            print("The following data was saved to file!")
            return student_table
```

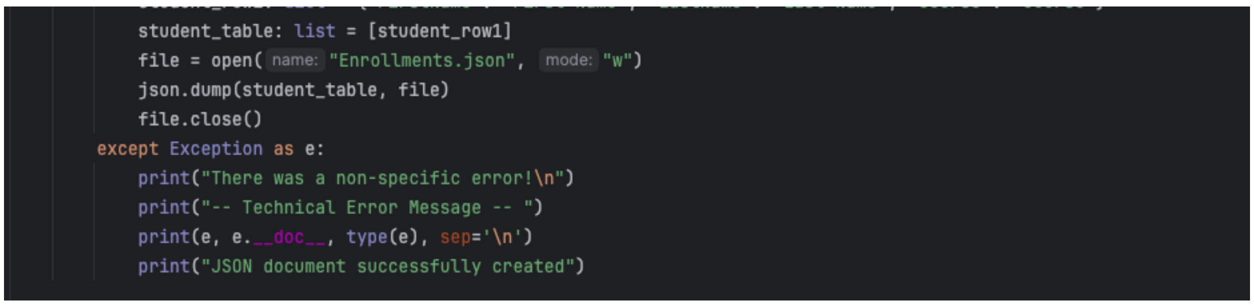
Figure 6.1 FileProcessor class

I created a function to create a JSON file since I completed Professor Root's Army initiation script on multiple workstations.



```python
@staticmethod
def creation_of_json(fileName: str, student_data: list):
    """
        This function creates a dictionary if one does not exist for the Professor utilizing the Python Arts script

        Notes:
            -None
            :param fileName
            :param student_data
            ChangeLog: (Who, When, What)
            JP,26MAY24,Created function
    """
    try:
        file = open(FILE_NAME,  mode: "r")
        student_table = json.load(file)
        for item in students:
            print(f"FirstName: {item['FirstName']}, LastName: {item['LastName']}, Course: {item['Course']}")
    except FileNotFoundError as e:
        print("JSON file must exist before running this script!\n")
        print("-- Technical Error Message -- ")
        print(e, e.__doc__, type(e), sep='\n')
        print("Prof. Justin will create a JSON for you..")
        student_row1: dict = {"FirstName": "First Name", "LastName": "Last Name", "Course": "Course"}
        student_table: list = [student_row1]
```

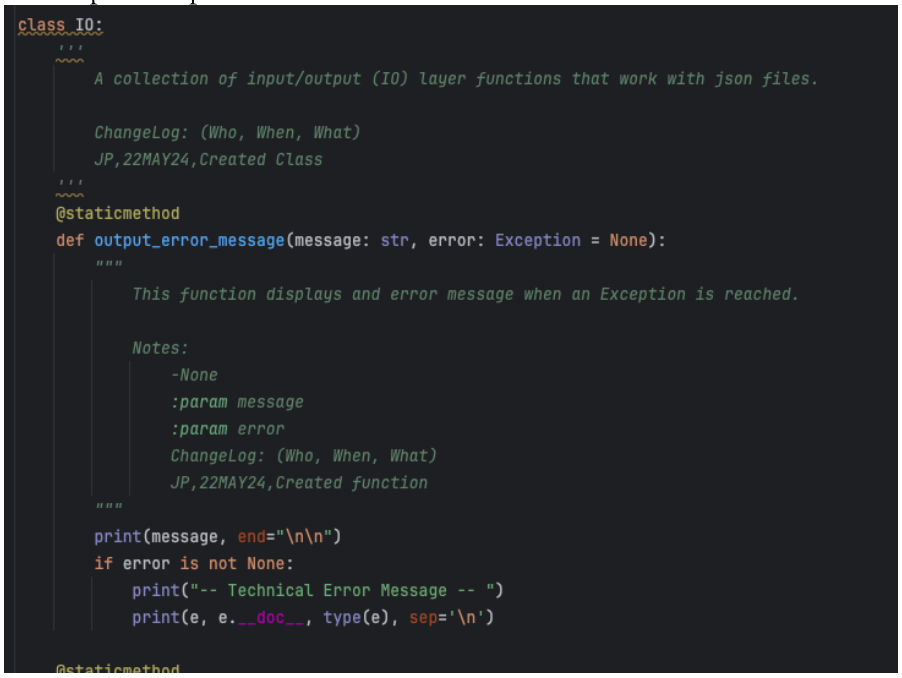
Figure 7.1 Creation of json function



```python
        student_table: list = [student_row1]
        file = open( name: "Enrollments.json",  mode: "w")
        json.dump(student_table, file)
        file.close()
    except Exception as e:
        print("There was a non-specific error!\n")
        print("-- Technical Error Message -- ")
        print(e, e.__doc__, type(e), sep='\n')
        print("JSON document successfully created")
```

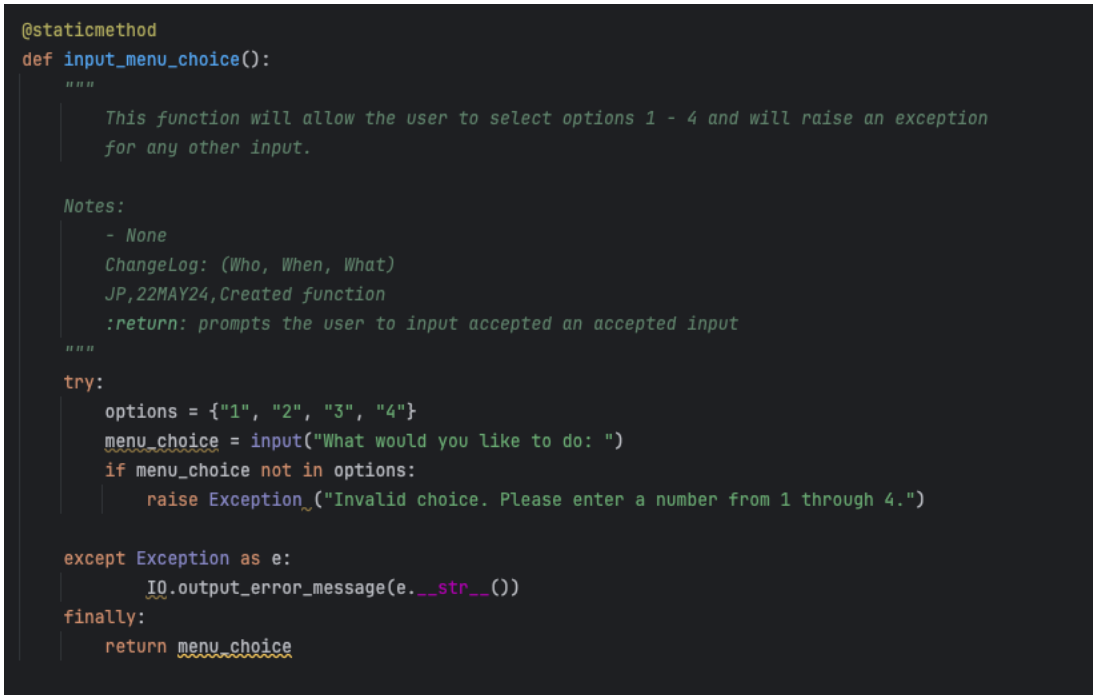
Figure 7.2 Creation of json function

The Input Output functions remained the same.



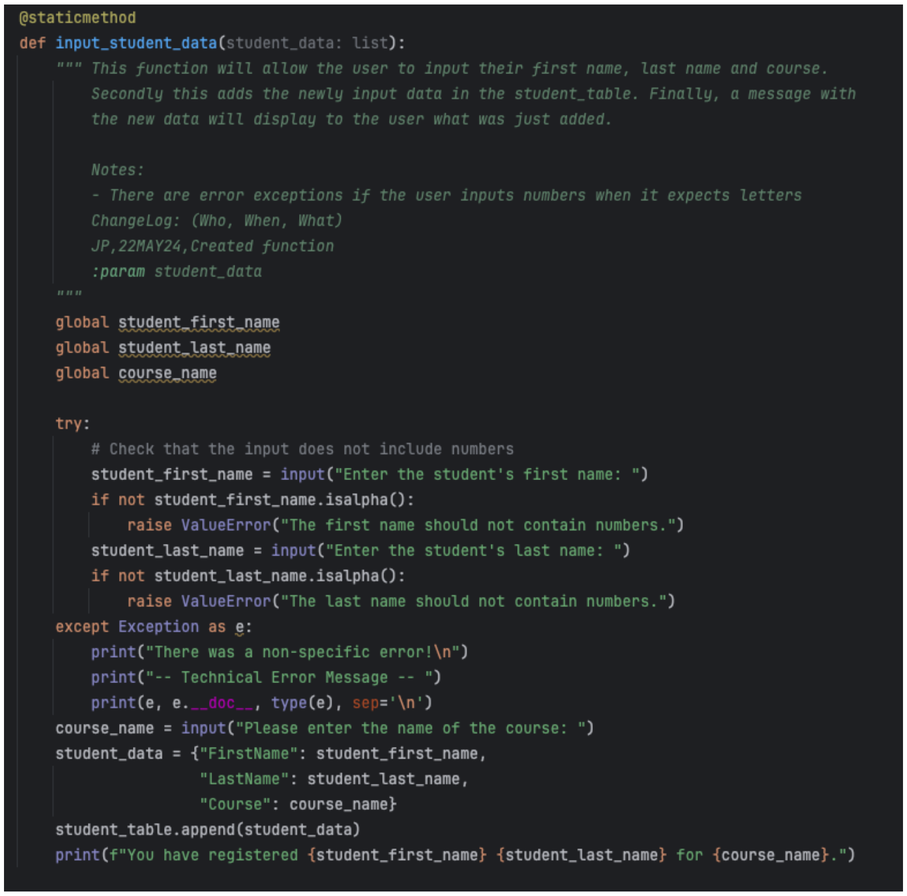
```
class IO:
    '''
        A collection of input/output (IO) layer functions that work with json files.

        ChangeLog: (Who, When, What)
        JP,22MAY24,Created Class
    '''
    @staticmethod
    def output_error_message(message: str, error: Exception = None):
        """
            This function displays and error message when an Exception is reached.

            Notes:
                -None
                :param message
                :param error
                ChangeLog: (Who, When, What)
                JP,22MAY24,Created function
        """
        print(message, end="\n\n")
        if error is not None:
            print("-- Technical Error Message -- ")
            print(e, e.__doc__, type(e), sep='\n')

    @staticmethod
```

Figure 8.1 IO class



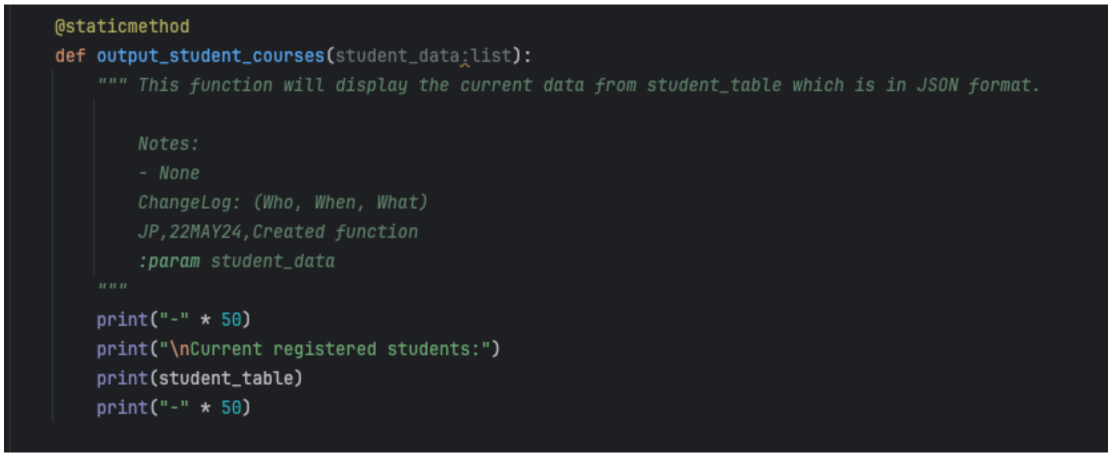
```
@staticmethod
def input_menu_choice():
    """
        This function will allow the user to select options 1 - 4 and will raise an exception
        for any other input.

    Notes:
        - None
        ChangeLog: (Who, When, What)
        JP,22MAY24,Created function
        :return: prompts the user to input accepted an accepted input
    """
    try:
        options = {"1", "2", "3", "4"}
        menu_choice = input("What would you like to do: ")
        if menu_choice not in options:
            raise Exception ("Invalid choice. Please enter a number from 1 through 4.")

    except Exception as e:
            IO.output_error_message(e.__str__())
    finally:
        return menu_choice
```

Figure 8.2 IO class



```python
@staticmethod
def input_student_data(student_data: list):
    """ This function will allow the user to input their first name, last name and course.
        Secondly this adds the newly input data in the student_table. Finally, a message with
        the new data will display to the user what was just added.

        Notes:
        - There are error exceptions if the user inputs numbers when it expects letters
        ChangeLog: (Who, When, What)
        JP,22MAY24,Created function
        :param student_data
    """
    global student_first_name
    global student_last_name
    global course_name

    try:
        # Check that the input does not include numbers
        student_first_name = input("Enter the student's first name: ")
        if not student_first_name.isalpha():
            raise ValueError("The first name should not contain numbers.")
        student_last_name = input("Enter the student's last name: ")
        if not student_last_name.isalpha():
            raise ValueError("The last name should not contain numbers.")
    except Exception as e:
        print("There was a non-specific error!\n")
        print("-- Technical Error Message -- ")
        print(e, e.__doc__, type(e), sep='\n')
    course_name = input("Please enter the name of the course: ")
    student_data = {"FirstName": student_first_name,
                    "LastName": student_last_name,
                    "Course": course_name}
    student_table.append(student_data)
    print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
```

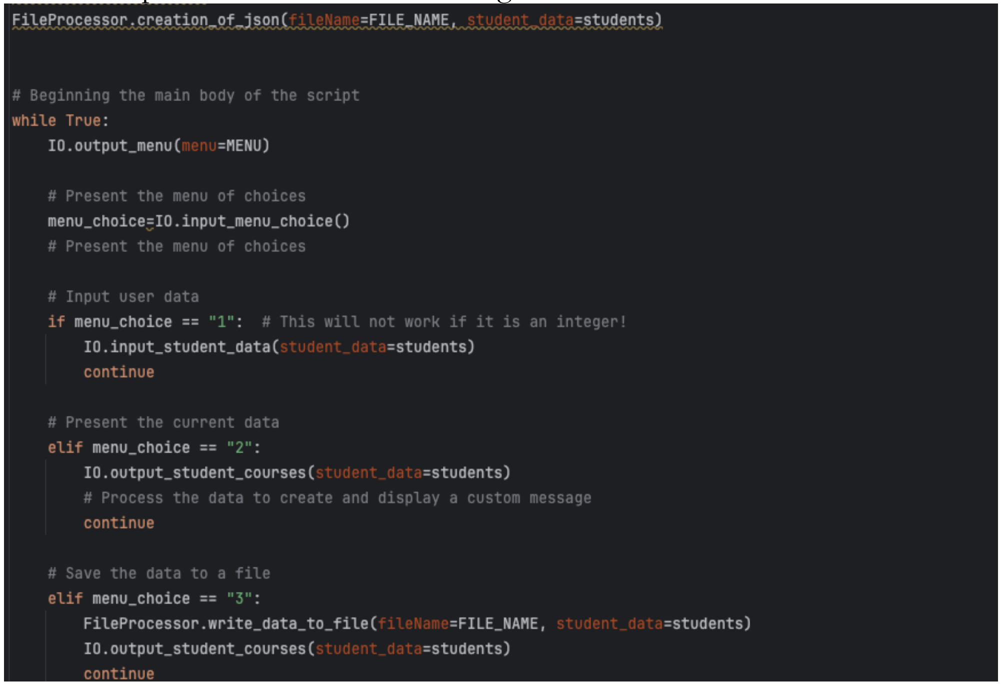
Figure 8.3 IO class



```python
    @staticmethod
    def output_student_courses(student_data:list):
        """ This function will display the current data from student_table which is in JSON format.

            Notes:
            - None
            ChangeLog: (Who, When, What)
            JP,22MAY24,Created function
            :param student_data
        """
        print("-" * 50)
        print("\nCurrent registered students:")
        print(student_table)
        print("-" * 50)
```

Figure 8.4 IO class

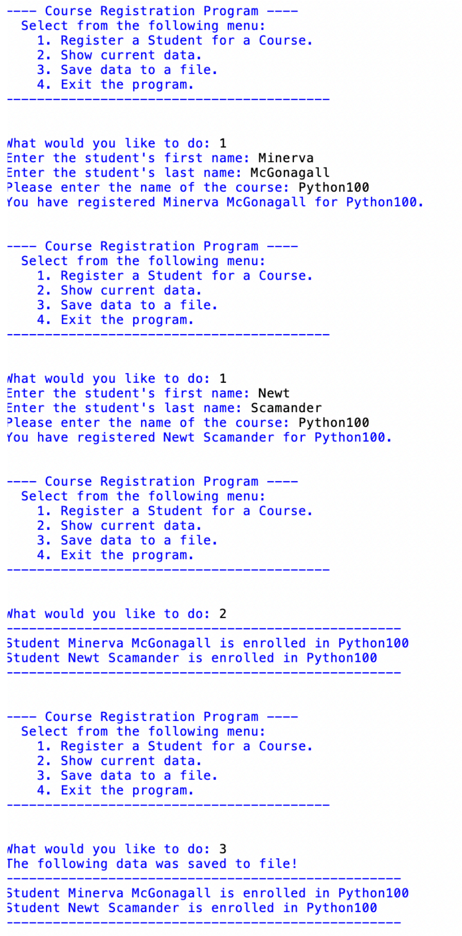
This is the portion where the functions get executed.



```
FileProcessor.creation_of_json(fileName=FILE_NAME, student_data=students)


# Beginning the main body of the script
while True:
    IO.output_menu(menu=MENU)

    # Present the menu of choices
    menu_choice=IO.input_menu_choice()
    # Present the menu of choices

    # Input user data
    if menu_choice == "1":  # This will not work if it is an integer!
        IO.input_student_data(student_data=students)
        continue

    # Present the current data
    elif menu_choice == "2":
        IO.output_student_courses(student_data=students)
        # Process the data to create and display a custom message
        continue

    # Save the data to a file
    elif menu_choice == "3":
        FileProcessor.write_data_to_file(fileName=FILE_NAME, student_data=students)
        IO.output_student_courses(student_data=students)
        continue
```

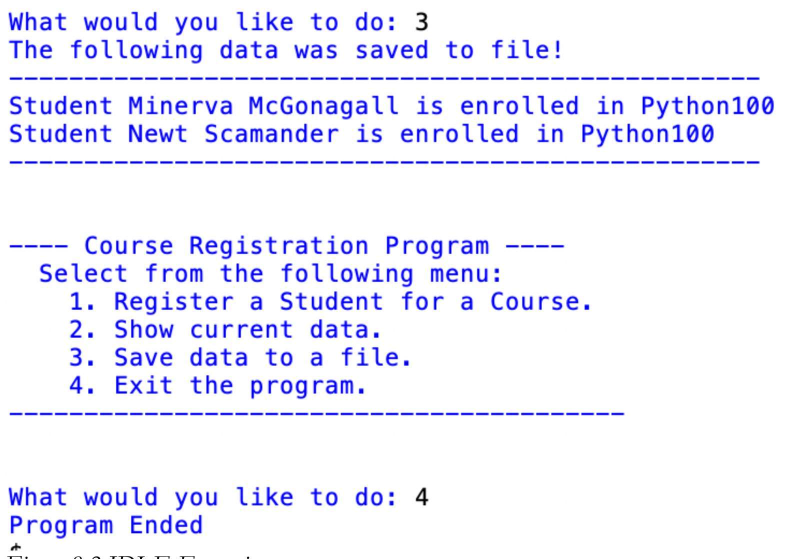
Figure 9.1 Execution of the funtions.

I also ran this in IDLE check my work.



```
---- 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: 1
Enter the student's first name: Minerva
Enter the student's last name: McGonagall
Please enter the name of the course: Python100
You have registered Minerva McGonagall for Python100.


---- 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: 1
Enter the student's first name: Newt
Enter the student's last name: Scamander
Please enter the name of the course: Python100
You have registered Newt Scamander for Python100.


---- 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
--------------------------------------------------
Student Minerva McGonagall is enrolled in Python100
Student Newt Scamander is enrolled in Python100
--------------------------------------------------


---- 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
The following data was saved to file!
--------------------------------------------------
Student Minerva McGonagall is enrolled in Python100
Student Newt Scamander is enrolled in Python100
--------------------------------------------------
```

Figure 10.1 IDLE Execution



```
What would you like to do: 3
The following data was saved to file!
--------------------------------------------------
Student Minerva McGonagall is enrolled in Python100
Student Newt Scamander is enrolled in Python100
--------------------------------------------------


---- 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: 4
Program Ended
```

Figure 8.2 IDLE Execution

Summary:

The answer to the script was that a student will always be a person, but a person will not always be a student. Professor Justin looked around the room and into the eyes of all the students and faculty. He cleared his throat and said, “Welcome to Professor Root’s Army”. (They all jumped in the air; music played for an epic freeze frame).