# 1 CONTENTS

# 2 INTRODUCTION

This report describes the python program in which we analyse if a person is a nerd or not by examining few aspects such as fandom score, hobbies score and number of sports items owned. The program allows the user to calculate ‘nerd skill level’. The higher the skill level, the more powerful the nerd. This level is calculated by using a special formula mentioned later in this report.

This assignment is divided into 3 tasks –

a. Implement a menu to allow the program to be used.
b. An equation which can be used to calculate the nerd score.
c. A printout that allows a summation of the results.

# 3 TASK 1 – IMPLEMENTION OF THE MENU

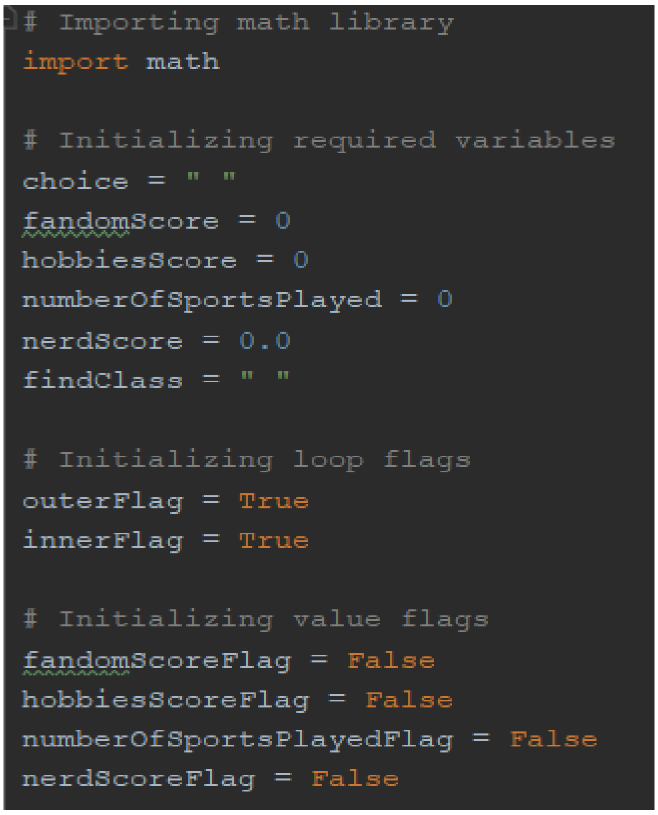
This menu is the main point of interaction between the user and the program. The menu list is as below:

1. Fandom Score
2. Hobbies Score
3. Number of Sports Played
4. Calculate Nerd Score
5. Display Nerd Class
6. Exit

Upon choosing either option from 1, 2 or 3, the user has to enter the appropriate value for corresponding score. Only then the program returns to the main menu so that the user can select another option. Based on the values from the first to third options in the menu, nerd score is calculated and displayed when user chooses option 4. Similarly, based on the value of the nerd score in option 4, nerd class is calculated and displayed when the user chooses option 5. The user can exit the menu by selecting option 6. All the inputs from the user are stored in variables and their validity is checked. The main file to run this task is **menu.py**.

## 3.1 IMPORTING REQUIRED LIBRARY AND INITIALIZING REQUIRED VARIABLES AND FLAGS

This program requires the square root function to calculate nerd score and hence we need to import math library for easy calculations. Also, we require few variables and flag to store some default values. Below is the screenshot of the code snippet which shows importing of the math library and initializing required variables and flags.



```python
# Importing math library
import math

# Initializing required variables
choice = " "
fandomScore = 0
hobbiesScore = 0
numberOfSportsPlayed = 0
nerdScore = 0.0
findClass = " "

# Initializing loop flags
outerFlag = True
innerFlag = True

# Initializing value flags
fandomScoreFlag = False
hobbiesScoreFlag = False
numberOfSportsPlayedFlag = False
nerdScoreFlag = False
```

Fig 1. Code for importing math library initializing the variables and flags

The variables initialized are as below:

1. choice:
   The option which the user chooses from the menu that is displayed (1-6).

2. fandomScore:
   Default value for the fandom score is set to 0 as it is an integer value.

3. hobbiesScore:
   Default monthly value for the hobbies score is set to 0 as it is an integer value.

4. numberOfSportsPlayed:
Default value for the number of sports played is set to 0 as it is an integer value.

5. nerdScore:
Default value for the nerd score is set to 0.0 as it will a floating value.

6. findClass:
Default value for the nerd class rating is set to blank (“ “).

The flags initialized are as below:

1. outerFlag:
Flag to set the outer loop for the menu iteration. Default value is set to True.

2. innerFlag:
Flag to set the inner loop for the menu options (1-6). Default value is set to True.

3. fandomScoreFlag:
Flag to check the validity of the value of the fandom score. Default value is set to False.

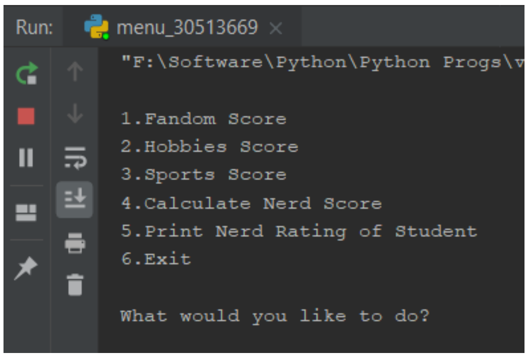
4. hobbiesScoreFlag:
Flag to check the validity of the value of the hobbies score. Default value is set to False.

5. numberOfSportsPlayedFlag:
Flag to check the validity of the value of the number of the sports played. Default value is set to False.

6. nerdScoreFlag:
Flag to check the validity of the value of the nerd score. Default value is set to False.

## 3.2 DISPLAYING THE MENU

The first step after importing the required library and initializing the required flags and variables is to display the menu from which the user can pick option. The program is designed in such a way that unless the user picks the options from the menu, it'll keep asking the user to enter correct input by displaying the message on the output screen (Implemented using while loop).

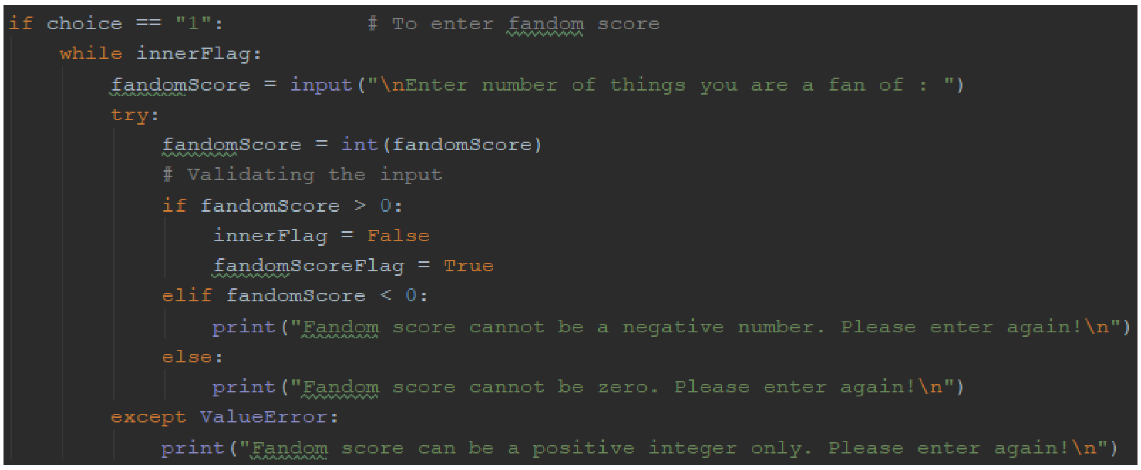


```
"F:\Software\Python\Python Progs\v

1.Fandom Score
2.Hobbies Score
3.Sports Score
4.Calculate Nerd Score
5.Print Nerd Rating of Student
6.Exit

What would you like to do?
```

Fig 2. Menu Output

## 3.3 FANDOM SCORE

The fandom score can be a positive integer. So, necessary checks are to be done before accepting the value and storing in the variable. The program checks if the entered input is zero, negative number or something other than positive integer. Below figure shows the checks and the error messages the program displays upon entering wrong inputs for the fandom score. If fandomScore value meets the requirements, fandomScoreFlag is set to True and innerFlag set to False. Then, the program takes the user to the main menu.

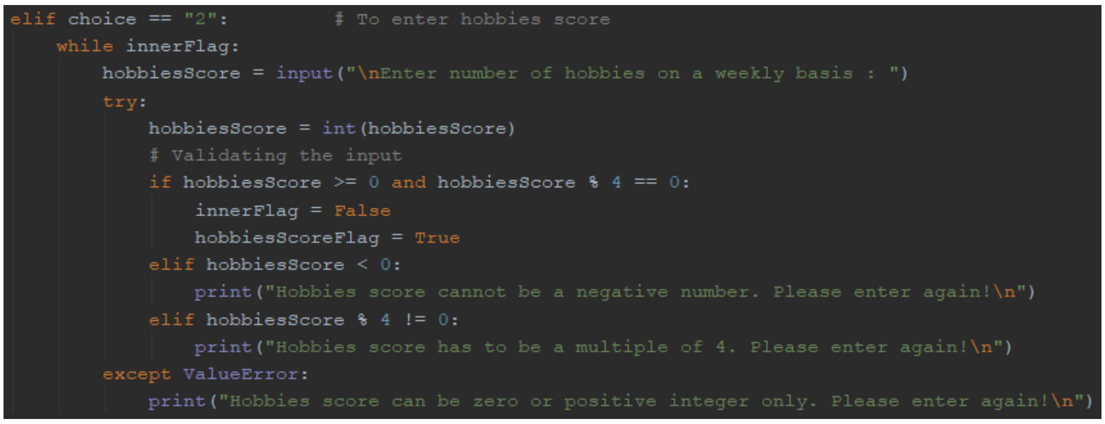


```
if choice == "1":              # To enter fandom score
    while innerFlag:
        fandomScore = input("\nEnter number of things you are a fan of : ")
        try:
            fandomScore = int(fandomScore)
            # Validating the input
            if fandomScore > 0:
                innerFlag = False
                fandomScoreFlag = True
            elif fandomScore < 0:
                print("Fandom score cannot be a negative number. Please enter again!\n")
            else:
                print("Fandom score cannot be zero. Please enter again!\n")
        except ValueError:
            print("Fandom score can be a positive integer only. Please enter again!\n")
```

Fig 3. Validations on fandom score

## 3.4 HOBBIES SCORE

The hobbies score can be zero or a positive integer multiple of 4. So, necessary checks are to be done before accepting the value and storing in the variable. The program checks if the entered input is negative number, non-multiple of 4 or something other than positive integer multiple of 4/zero. Below figure shows the checks and the error messages the program displays upon entering wrong inputs for the hobbies score. If hobbiesScore value meets the requirements, hobbiesScoreFlag is set to True and innerFlag set to False. Then, the program takes the user to the main menu.

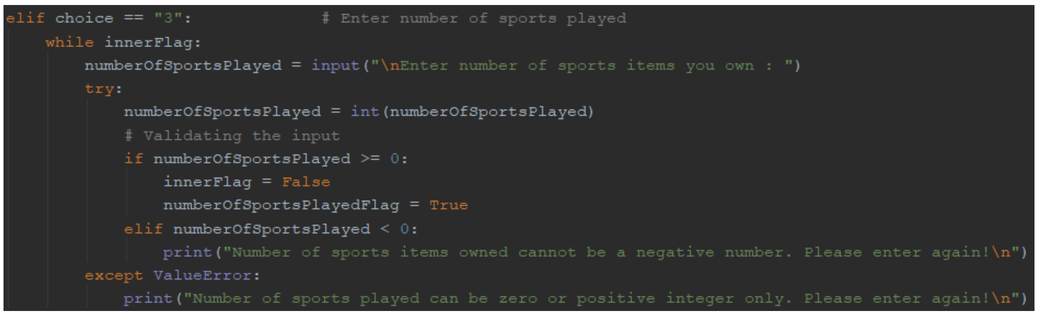


```
elif choice == "2":          # To enter hobbies score
    while innerFlag:
        hobbiesScore = input("\nEnter number of hobbies on a weekly basis : ")
        try:
            hobbiesScore = int(hobbiesScore)
            # Validating the input
            if hobbiesScore >= 0 and hobbiesScore % 4 == 0:
                innerFlag = False
                hobbiesScoreFlag = True
            elif hobbiesScore < 0:
                print("Hobbies score cannot be a negative number. Please enter again!\n")
            elif hobbiesScore % 4 != 0:
                print("Hobbies score has to be a multiple of 4. Please enter again!\n")
        except ValueError:
            print("Hobbies score can be zero or positive integer only. Please enter again!\n")
```

Fig 4. Validations on hobbies score

## 3.5 NUMBER OF SPORTS PLAYED

The number of sports played/sports score can be zero or a positive integer. The program checks if the entered input is negative number or something other than zero/positive number. Below figure shows the checks and the error messages the program displays upon entering wrong inputs for the number of sports played. If numberOfSportsPlayed value meets the requirements, numberOfSportsPlayedFlag is set to True and innerFlag set to False. Then, the program takes the user to the main menu.



```
elif choice == "3":              # Enter number of sports played
    while innerFlag:
        numberOfSportsPlayed = input("\nEnter number of sports items you own : ")
        try:
            numberOfSportsPlayed = int(numberOfSportsPlayed)
            # Validating the input
            if numberOfSportsPlayed >= 0:
                innerFlag = False
                numberOfSportsPlayedFlag = True
            elif numberOfSportsPlayed < 0:
                print("Number of sports items owned cannot be a negative number. Please enter again!\n")
        except ValueError:
            print("Number of sports played can be zero or positive integer only. Please enter again!\n")
```

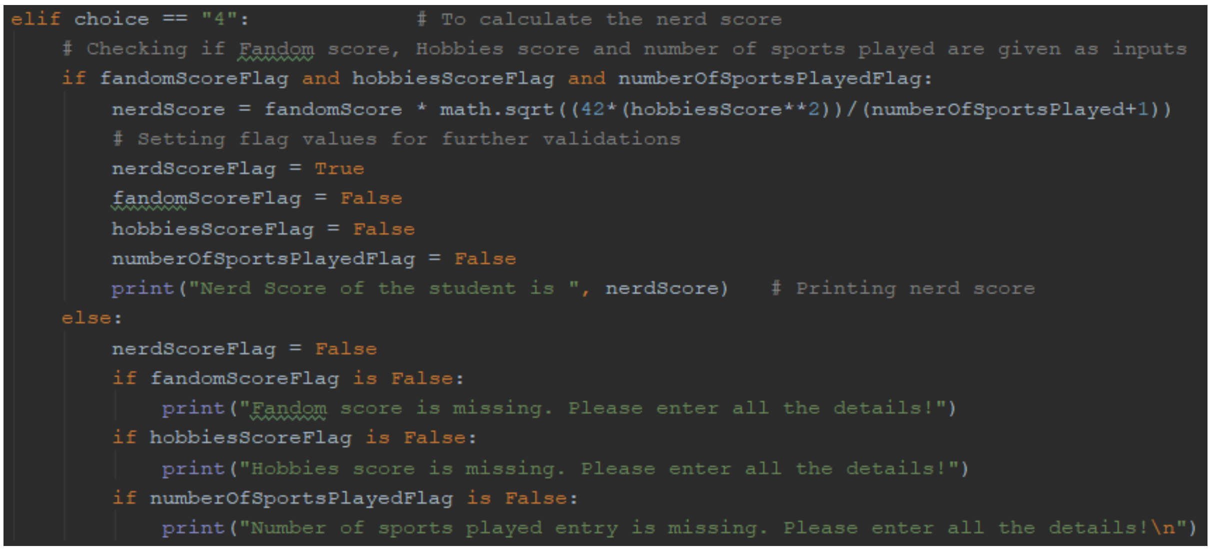
Fig 5. Validations on number of sports played

## 3.6 CALCULATE NERD SCORE

To calculate the nerd score, we need fandom score, hobbies score and number of sports played. So, the program validates those values first by checking the corresponding flag values, calculates the nerd score using the math library and displays the result. If nerdScore value meets the requirements, nerdScoreFlag is set to True and other score flags are set to false so that the user can't calculate nerd score again with the previously entered scores. If nerdScore value doesn't meets the requirements, nerdScoreFlag is set to false and the program displays missing score details. In such case, the user has to enter score values again and then calculate the nerd score. The same can be seen in Fig 6.

The special formula to calculate the nerd score is given by -

$$\textit{Nerd Score} = \textit{Fandom Score} \times ((42 \times (\textit{Hobbies Score})^2) / (\textit{Number of Sports Played} + 1))^{1/2}$$



```
elif choice == "4":              # To calculate the nerd score
    # Checking if Fandom score, Hobbies score and number of sports played are given as inputs
    if fandomScoreFlag and hobbiesScoreFlag and numberOfSportsPlayedFlag:
        nerdScore = fandomScore * math.sqrt((42*(hobbiesScore**2))/(numberOfSportsPlayed+1))
        # Setting flag values for further validations
        nerdScoreFlag = True
        fandomScoreFlag = False
        hobbiesScoreFlag = False
        numberOfSportsPlayedFlag = False
        print("Nerd Score of the student is ", nerdScore)   # Printing nerd score
    else:
        nerdScoreFlag = False
        if fandomScoreFlag is False:
            print("Fandom score is missing. Please enter all the details!")
        if hobbiesScoreFlag is False:
            print("Hobbies score is missing. Please enter all the details!")
        if numberOfSportsPlayedFlag is False:
            print("Number of sports played entry is missing. Please enter all the details!\n")
```

Fig 6. Validating, calculating and printing the nerd score

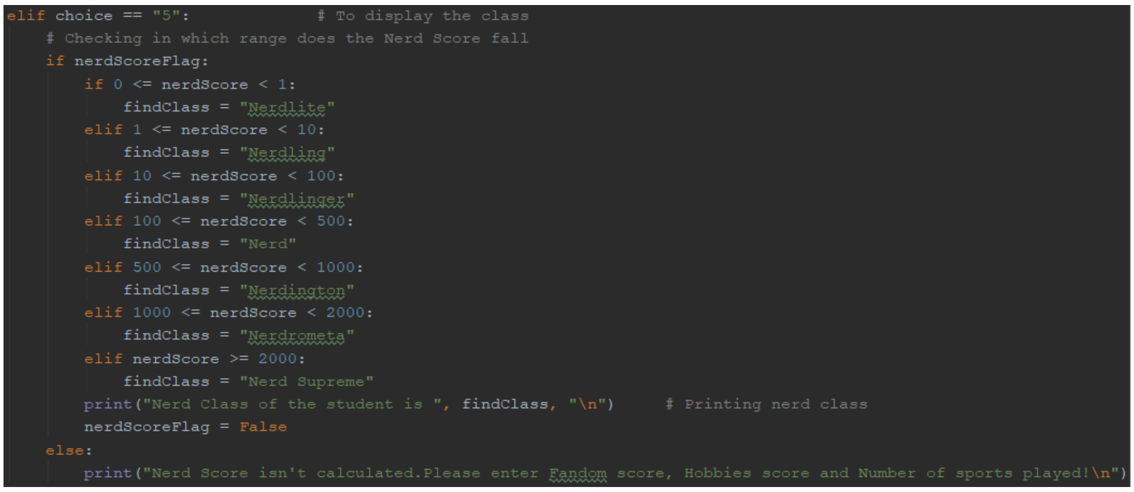
## 3.7 DISPLAY NERD CLASS

To find out the nerd class and display it, we need nerd score. Hence, the program validates the nerd score and displays the nerd class accordingly. For our reference, we have a table which defines various nerd classes based on the nerd score. If the nerd score is not calculated previously (nerdScoreFlag is False), the program asks the user to input fandom score, hobbies score, number of sports played and then calculate the nerd score.

| Score | Class | Position in the output list |
|---|---|---|
| 0 | Nerdlite | 0 |
| 1 | Nerdling | 1 |
| 10 | Nerdlinger | 2 |
| 100 | Nerd | 3 |
| 500 | Nerdington | 4 |
| 1000 | Nerdrometa | 5 |
| 2000 | Nerd Supreme | 6 |

Table 1. Nerd score thresholds and their associated Class.

Below is the code snippet:



```
elif choice == "5":             # To display the class
    # Checking in which range does the Nerd Score fall
    if nerdScoreFlag:
        if 0 <= nerdScore < 1:
            findClass = "Nerdlite"
        elif 1 <= nerdScore < 10:
            findClass = "Nerdling"
        elif 10 <= nerdScore < 100:
            findClass = "Nerdlinger"
        elif 100 <= nerdScore < 500:
            findClass = "Nerd"
        elif 500 <= nerdScore < 1000:
            findClass = "Nerdington"
        elif 1000 <= nerdScore < 2000:
            findClass = "Nerdrometa"
        elif nerdScore >= 2000:
            findClass = "Nerd Supreme"
        print("Nerd Class of the student is ", findClass, "\n")     # Printing nerd class
        nerdScoreFlag = False
    else:
        print("Nerd Score isn't calculated.Please enter Fandom score, Hobbies score and Number of sports played!\n")
```

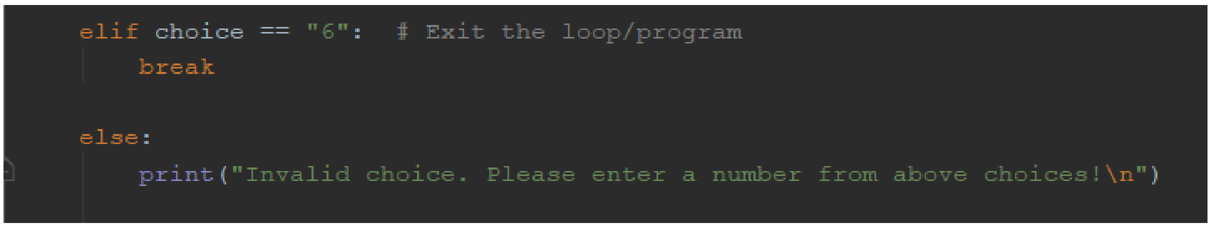
Fig 7. Validating the nerd score and displaying nerd rating

If all the requirements are met for this, the program displays the nerd class and sets the nerdScoreFlag to false so that the user cannot find the nerd class with the previously entered values. The user has to enter all the scores again, calculate the nerd score and then find the nerd class.

If any of the scores i.e fandom score, hobbies score and sports score is missing, the program doesn’t allow the user to calculate the nerd score and there by display the nerd class. The user can do so only upon entering valid inputs to the fandom, hobbies and sports scores.

## 3.8 EXITING FROM THE MENU

This brings us to the last stage of the menu where the user wants to do nothing but exit the program. Thus, by selecting exit option from the menu, user can quit the program. Throughout the program if user picks any other values apart from the menu options, the program prompts for user input until appropriate input is provided. Below is the code snippet.



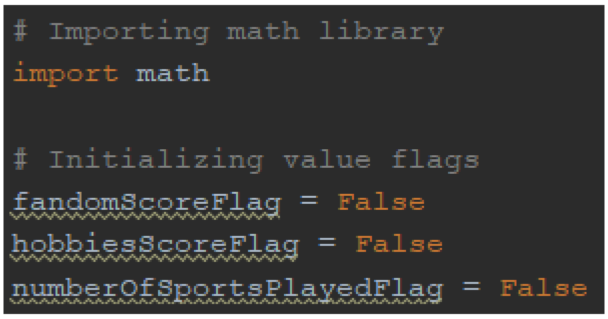
```
    elif choice == "6":  # Exit the loop/program
        break

    else:
        print("Invalid choice. Please enter a number from above choices!\n")
```

Fig 8. Exiting the program

# 4 TASK 2 – IMPLEMENT THE NERD SCORE

After obtaining the input values (fandom score, hobbies score and number of sports played) from the template provided, the program calculates the nerd score and displays it. All the inputs from the user are stored in variables and their validity is checked. The main file to run this task is **nerdScore.py**.

Similar to the task 1, the program imports math library and initializes few value flags to calculate the nerd score (using formula as mentioned above). Below is the screenshot.



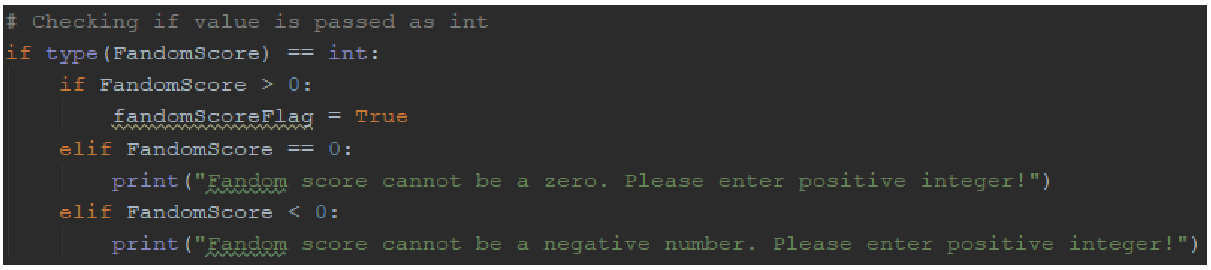
```
# Importing math library
import math

# Initializing value flags
fandomScoreFlag = False
hobbiesScoreFlag = False
numberOfSportsPlayedFlag = False
```

Fig 9. Importing math library and initializing value flags

## 4.1 VALIDATING FANDOM SCORE

Based on the task 1, fandom score can be a positive integer only. So, the program checks for the following:

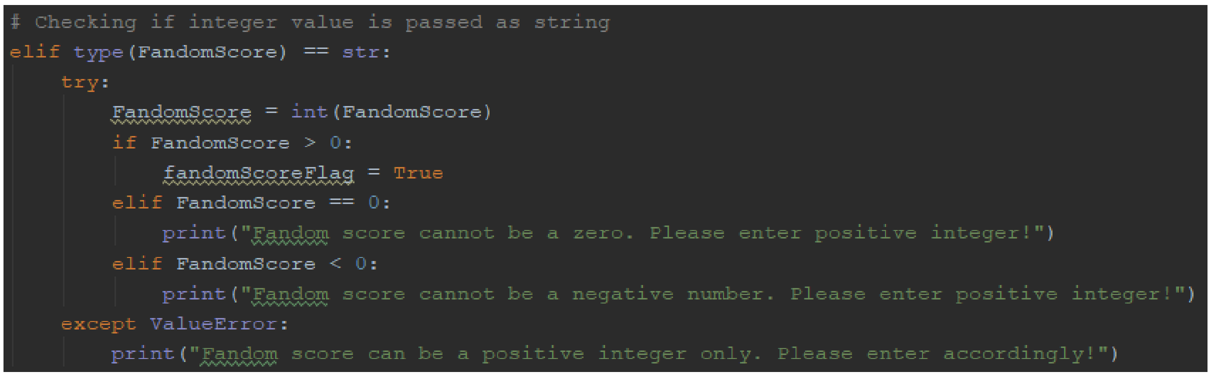
- If the input is passed as an integer:
  The program checks if the value is zero or a negative number, displays error messages accordingly and asks the user to input valid details. If the requirements are met, fandomScoreFlag is set to true. Below is the code snippet.



```python
# Checking if value is passed as int
if type(FandomScore) == int:
    if FandomScore > 0:
        fandomScoreFlag = True
    elif FandomScore == 0:
        print("Fandom score cannot be a zero. Please enter positive integer!")
    elif FandomScore < 0:
        print("Fandom score cannot be a negative number. Please enter positive integer!")
```

Fig 10. Validating fandom score when it is passed as integer type

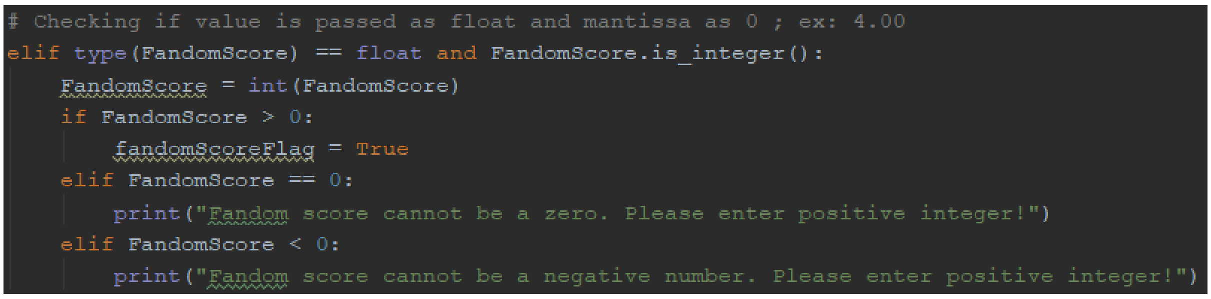
- If the input is passed as a string:
  The program checks if the value can be converted to integer format. If that is possible, it checks if the value is zero or negative number, displays error messages accordingly and asks the user to input valid details. If the value is a positive integer after conversion, fandomScoreFlag is set to true. If the string to integer conversion fails, the program prompts the user to enter correct input. Below is the code snippet.



```python
# Checking if integer value is passed as string
elif type(FandomScore) == str:
    try:
        FandomScore = int(FandomScore)
        if FandomScore > 0:
            fandomScoreFlag = True
        elif FandomScore == 0:
            print("Fandom score cannot be a zero. Please enter positive integer!")
        elif FandomScore < 0:
            print("Fandom score cannot be a negative number. Please enter positive integer!")
    except ValueError:
        print("Fandom score can be a positive integer only. Please enter accordingly!")
```

Fig 11. Validating fandom score when it is passed as string type

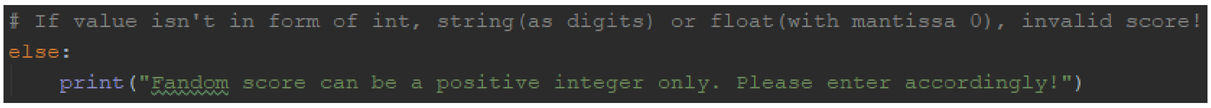
- If the input is passed as a float:
  The program checks if the value is float type and has mantissa as zero. For example – 5.00, 10.0 etc. If that is true, it checks if the value is zero or negative number, displays error messages accordingly and asks the user to input valid details. If the value is a positive integer, fandomScoreFlag is set to true.



```
# Checking if value is passed as float and mantissa as 0 ; ex: 4.00
elif type(FandomScore) == float and FandomScore.is_integer():
    FandomScore = int(FandomScore)
    if FandomScore > 0:
        fandomScoreFlag = True
    elif FandomScore == 0:
        print("Fandom score cannot be a zero. Please enter positive integer!")
    elif FandomScore < 0:
        print("Fandom score cannot be a negative number. Please enter positive integer!")
```

Fig 12. Validating fandom score when it is passed as float type

- If the input isn't in the form of integer, string (as digits) or float (with mantissa 0), the score is invalid. So, the program displays the message asking the user to enter valid inputs. Below is the screenshot.



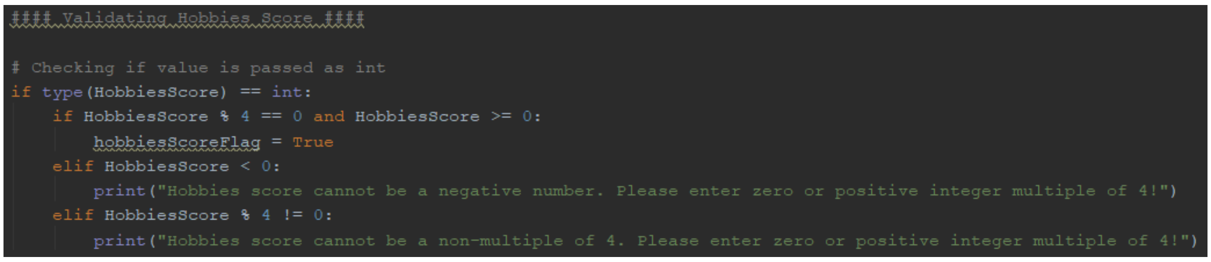
```
# If value isn't in form of int, string(as digits) or float(with mantissa 0), invalid score!
else:
    print("Fandom score can be a positive integer only. Please enter accordingly!")
```

Fig 13. Asking user to enter correct inputs when requirements aren't met

## 4.2 VALIDATING HOBBIES SCORE

Based on the task 1, hobbies score can be zero or positive integer multiple of 4 only. So, the program checks for the following:

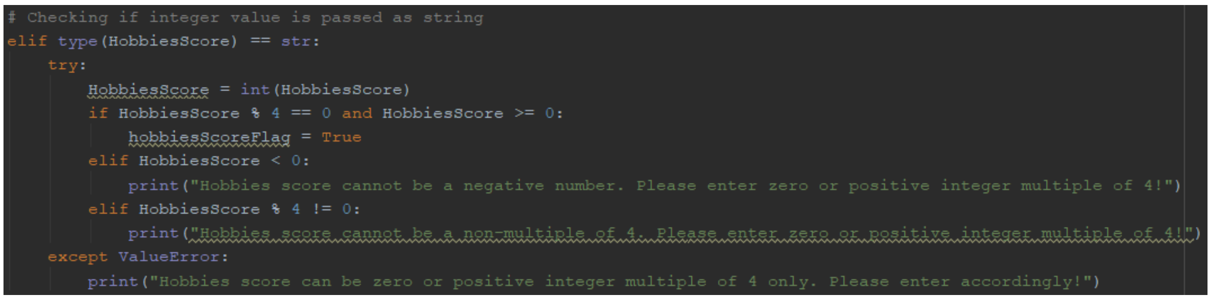
- If the input is passed as an integer:
  The program checks if the value is non-multiple of 4 or a negative number, displays error messages accordingly and asks the user to input valid details. If the requirements are met, hobbiesScoreFlag is set to true. Below is the code snippet.



```
#### Validating Hobbies Score ####

# Checking if value is passed as int
if type(HobbiesScore) == int:
    if HobbiesScore % 4 == 0 and HobbiesScore >= 0:
        hobbiesScoreFlag = True
    elif HobbiesScore < 0:
        print("Hobbies score cannot be a negative number. Please enter zero or positive integer multiple of 4!")
    elif HobbiesScore % 4 != 0:
        print("Hobbies score cannot be a non-multiple of 4. Please enter zero or positive integer multiple of 4!")
```

Fig 14. Validating hobbies score when it is passed as integer

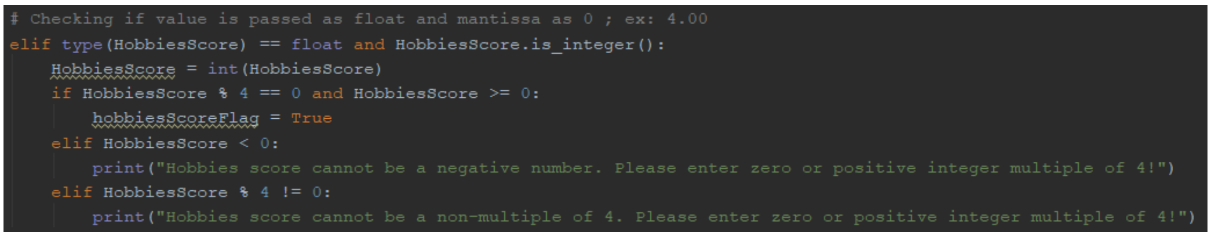
- If the input is passed as a string:
  The program checks if the value can be converted to integer format. If that is possible, it checks if the value is non-multiple of 4 or negative number, displays error messages accordingly and asks the user to input valid details. If the value is zero or a positive integer multiple of 4 after conversion, hobbiesScoreFlag is set to true. If the string to integer conversion fails, the program prompts the user to enter correct input. Below is the code snippet.



```
# Checking if integer value is passed as string
elif type(HobbiesScore) == str:
    try:
        HobbiesScore = int(HobbiesScore)
        if HobbiesScore % 4 == 0 and HobbiesScore >= 0:
            hobbiesScoreFlag = True
        elif HobbiesScore < 0:
            print("Hobbies score cannot be a negative number. Please enter zero or positive integer multiple of 4!")
        elif HobbiesScore % 4 != 0:
            print("Hobbies score cannot be a non-multiple of 4. Please enter zero or positive integer multiple of 4!")
    except ValueError:
        print("Hobbies score can be zero or positive integer multiple of 4 only. Please enter accordingly!")
```

Fig 15. Validating hobbies score when it is passed as string

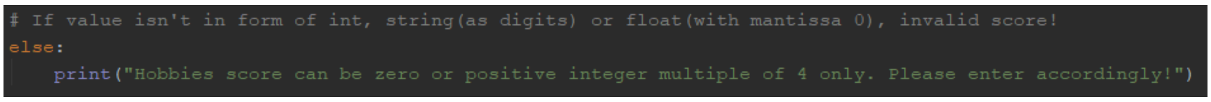
- If the input is passed as a float:
  The program checks if the value is float type and has mantissa as zero. For example – 5.00, 10.0 etc. If that is true, it checks if the value is non-multiple of 4 or negative number, displays error messages accordingly and asks the user to input valid details. If the value is zero or a positive integer multiple of 4, hobbiesScoreFlag is set to true.



```
# Checking if value is passed as float and mantissa as 0 ; ex: 4.00
elif type(HobbiesScore) == float and HobbiesScore.is_integer():
    HobbiesScore = int(HobbiesScore)
    if HobbiesScore % 4 == 0 and HobbiesScore >= 0:
        hobbiesScoreFlag = True
    elif HobbiesScore < 0:
        print("Hobbies score cannot be a negative number. Please enter zero or positive integer multiple of 4!")
    elif HobbiesScore % 4 != 0:
        print("Hobbies score cannot be a non-multiple of 4. Please enter zero or positive integer multiple of 4!")
```

Fig 16. Validating hobbies score when it is passed as float type

- If the input doesn’t isn’t in the form of integer, string (as digits) or float(with mantissa 0), the score is invalid. So, the program displays the message asking the user to enter valid inputs. Below is the screenshot.



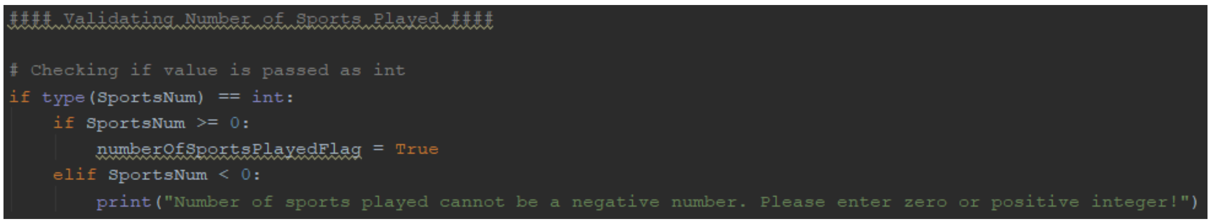
```
# If value isn't in form of int, string(as digits) or float(with mantissa 0), invalid score!
else:
    print("Hobbies score can be zero or positive integer multiple of 4 only. Please enter accordingly!")
```

Fig 17. Asking user to enter correct inputs when requirements aren’t met

## 4.3 VALIDATING SPORTS SCORE

Based on the task 1, sports score can be zero or a positive integer only. So, the program checks for the following:

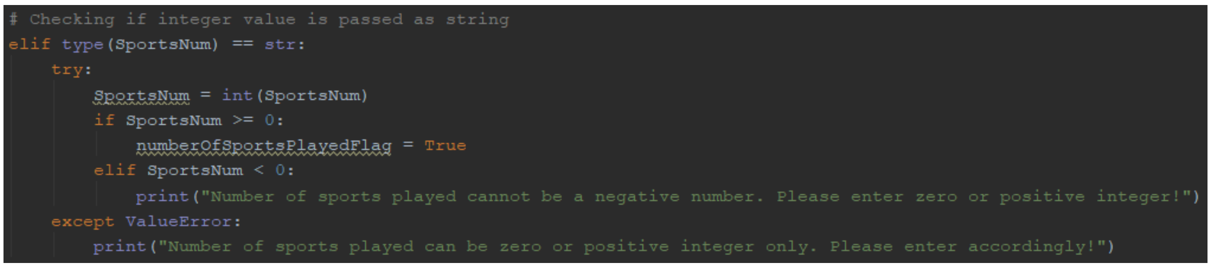
- If the input is passed as an integer:
  The program checks if the value is a negative number, displays error messages accordingly and asks the user to input valid details. If the requirements are met, numberOfSportsPlayedFlag is set to true. Below is the code snippet.



```
#### Validating Number of Sports Played ####

# Checking if value is passed as int
if type(SportsNum) == int:
    if SportsNum >= 0:
        numberOfSportsPlayedFlag = True
    elif SportsNum < 0:
        print("Number of sports played cannot be a negative number. Please enter zero or positive integer!")
```

Fig 18. Validating sports score when it is passed as integer type

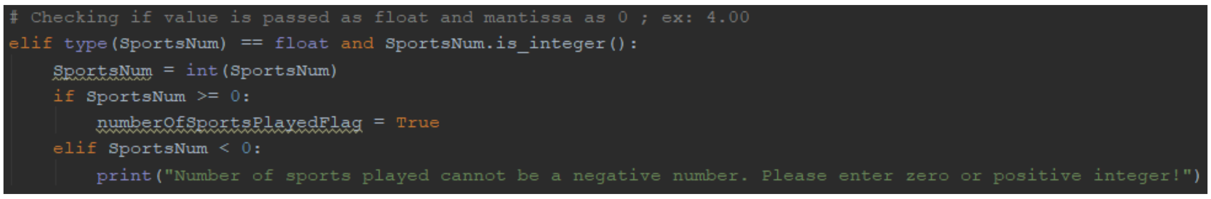
- If the input is passed as a string:
  The program checks if the value can be converted to integer format. If that is possible, it checks if the value is negative number, displays error messages accordingly and asks the user to input valid details. If the value is zero or a positive integer after conversion, numberOfSportsPlayedFlag is set to true. If the string to integer conversion fails, the program prompts the user to enter correct input. Below is the code snippet.



```
# Checking if integer value is passed as string
elif type(SportsNum) == str:
    try:
        SportsNum = int(SportsNum)
        if SportsNum >= 0:
            numberOfSportsPlayedFlag = True
        elif SportsNum < 0:
            print("Number of sports played cannot be a negative number. Please enter zero or positive integer!")
    except ValueError:
        print("Number of sports played can be zero or positive integer only. Please enter accordingly!")
```

Fig 19. Validating sports score when it is passed as string type

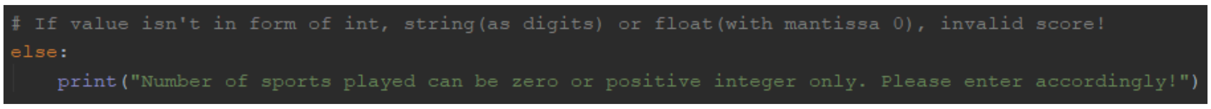
- If the input is passed as a float:
  The program checks if the value is float type and has mantissa as zero. For example – 5.00, 10.0 etc. If that is true, it checks if the value is a negative number, displays error messages accordingly and asks the user to input valid details. If the value is zero or a positive integer, numberOfSportsPlayedFlag is set to true.



```
# Checking if value is passed as float and mantissa as 0 ; ex: 4.00
elif type(SportsNum) == float and SportsNum.is_integer():
    SportsNum = int(SportsNum)
    if SportsNum >= 0:
        numberOfSportsPlayedFlag = True
    elif SportsNum < 0:
        print("Number of sports played cannot be a negative number. Please enter zero or positive integer!")
```

Fig 20. Validating sports score when it is passed as float type

- If the input isn't in the form of integer, string (as digits) or float (with mantissa 0), the score is invalid. So, the program displays the message asking the user to enter valid inputs. Below is the screenshot.



```
# If value isn't in form of int, string(as digits) or float(with mantissa 0), invalid score!
else:
    print("Number of sports played can be zero or positive integer only. Please enter accordingly!")
```

Fig 21. Asking user to enter correct inputs when requirements aren't met

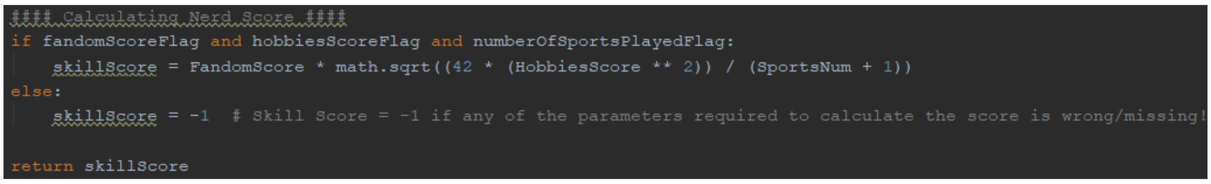
## 4.4 CALCULATING THE NERD SCORE

The template pushes few values as input to the function. The following results are expected.

| Fandom score | Hobbies score | Number of sports played | Nerd Score |
|---|---|---|---|
| 1 | 4 | 1 | 18.33030278 |
| 20 | 4 | 1 | 366.6060556 |
| 20 | 20 | 1 | 1833.030278 |
| 1 | -4 | 1 | -1 |
| 1 | 4 | 50 | 3.629940852 |

Table 2. Expected test results for nerd score

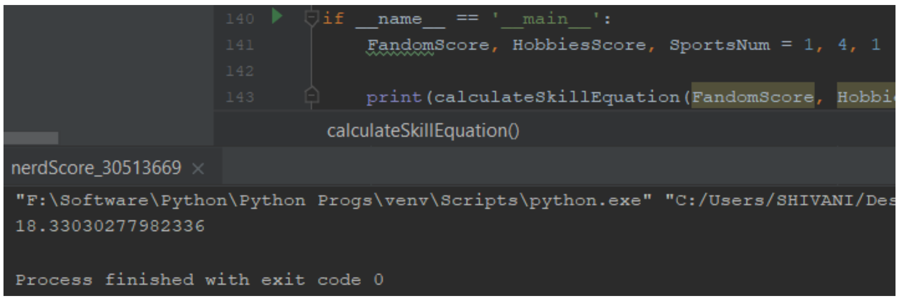
To calculate the nerd score, the program uses the formula mentioned in previous section. It validates the scores flags, calculates the nerd score and then returns the score. If the validation of the scores flags fails, the program returns **'-1'** to imply that the user has entered invalid or wrong inputs. Below is the code snippet.



```
#### Calculating Nerd Score ####
if fandomScoreFlag and hobbiesScoreFlag and numberOfSportsPlayedFlag:
    skillScore = FandomScore * math.sqrt((42 * (HobbiesScore ** 2)) / (SportsNum + 1))
else:
    skillScore = -1  # Skill Score = -1 if any of the parameters required to calculate the score is wrong/missing!

return skillScore
```

Fig 22. Calculate the nerd score and return the value

To test if the program is running correctly, let's try few cases. This is showed using the code snippets and screenshots of the output.



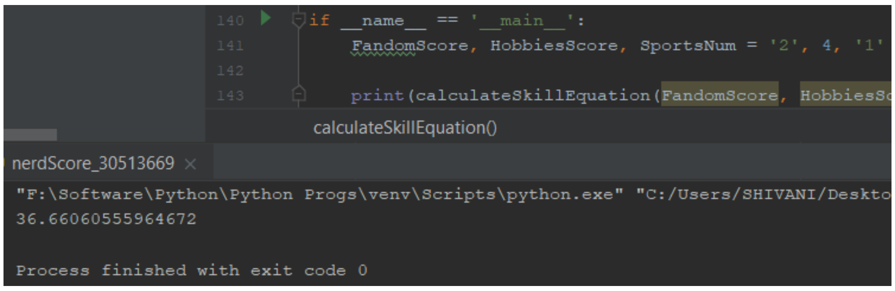
```
if __name__ == '__main__':
    FandomScore, HobbiesScore, SportsNum = 1, 4, 1

    print(calculateSkillEquation(FandomScore, Hobbi
```

calculateSkillEquation()

nerdScore_30513669

```
"F:\Software\Python\Python Progs\venv\Scripts\python.exe" "C:/Users/SHIVANI/Des
18.33030277982336

Process finished with exit code 0
```

Fig 23. Test result for inputs (1, 4, 1)



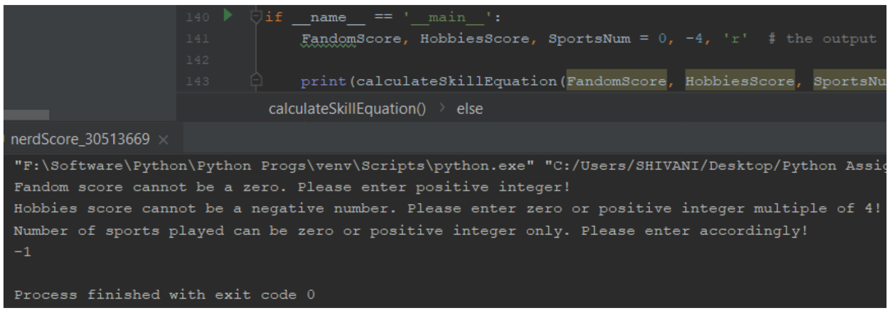
```
if __name__ == '__main__':
    FandomScore, HobbiesScore, SportsNum = '2', 4, '1'

    print(calculateSkillEquation(FandomScore, HobbiesSc
```

calculateSkillEquation()

nerdScore_30513669

```
"F:\Software\Python\Python Progs\venv\Scripts\python.exe" "C:/Users/SHIVANI/Deskto
36.66060555964672

Process finished with exit code 0
```

Fig 24. Test result for inputs ('2', 4, '1')



```
if __name__ == '__main__':
    FandomScore, HobbiesScore, SportsNum = 0, -4, 'r'  # the output

    print(calculateSkillEquation(FandomScore, HobbiesScore, SportsNu
```

calculateSkillEquation() › else

nerdScore_30513669

```
"F:\Software\Python\Python Progs\venv\Scripts\python.exe" "C:/Users/SHIVANI/Desktop/Python Assig
Fandom score cannot be a zero. Please enter positive integer!
Hobbies score cannot be a negative number. Please enter zero or positive integer multiple of 4!
Number of sports played can be zero or positive integer only. Please enter accordingly!
-1

Process finished with exit code 0
```

Fig 25. Test result for inputs (0, -4, 'r')

# 5 TASK 3 - FINDING NERD CLASS

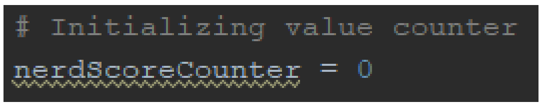
The nerd score that was calculated earlier in task 2 is of no use unless a class is assigned to a specific range of score. Once a score surpasses a class's higher threshold, then they are considered to be a part of the next class. The main aim of this task is that given several students' score in the form of a list, we have to output frequency of each nerd class in the form of a list defined in table 1 mentioned earlier. Each index of the output list represent different class as mentioned in table.

For example, given a list of nerd score [23, 76, 1300, 600], the program output should be a list [0, 0, 2, 0, 1, 1, 0]. This list signifies that there are:

- 0 Nerdlite
- 0 Nerdling
- 2 Nerdlinger
- 0 Nerd
- 1 Nerdington
- 1 Nerdrometa
- 0 Nerd Supreme

All the inputs from the user are validated and then the result is displayed in the form of a list. The main file to run this task is **findClass.py**.

Similar to the task 1 and 2, the program initializes a value flag and a value counter to validate the inputs and find the nerd rating frequency respectively. Default flag value is set to true and default counter value is set to 0. Below is the screenshot.



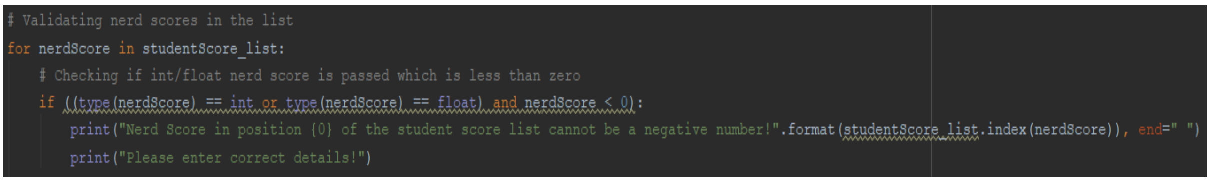
```
# Initializing value counter
nerdScoreCounter = 0
```

Fig 26. Initializing value counter

## 5.1 VALIDATING NERD SCORES IN THE LIST

Nerd scores in the list can be in the form of integer, float or string similar to the task 2. The program checks for the following:

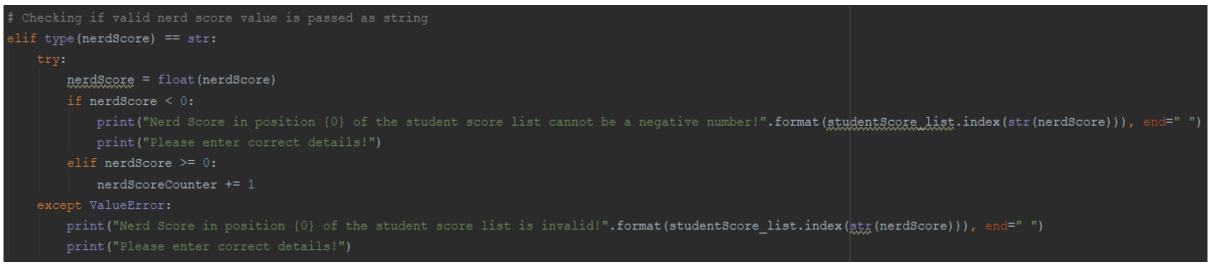
- If nerd score in the list is passed as an integer or float type and is a negative number, the program displays a message stating at which position in the list, the user has entered invalid nerd score.



```
# Validating nerd scores in the list
for nerdScore in studentScore_list:
    # Checking if int/float nerd score is passed which is less than zero
    if ((type(nerdScore) == int or type(nerdScore) == float) and nerdScore < 0):
        print("Nerd Score in position {0} of the student score list cannot be a negative number!".format(studentScore_list.index(nerdScore)), end=" ")
        print("Please enter correct details!")
```

Fig 27. Validating the nerd scores(int/float type only) in the list and displaying error message

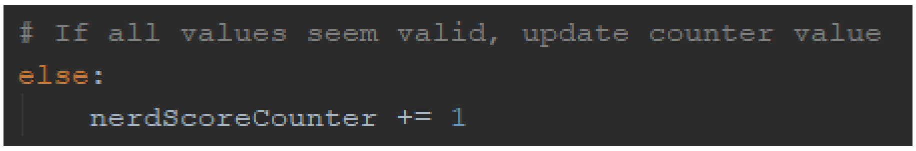
- If nerd score in the list is passed as a string type, the program converts the string into float type. If that is possible, the program checks if the nerd score is a negative number and if it is, the program displays a message stating at which position in the list, the user has entered invalid nerd score. If the nerd score after the conversion is zero or a positive number, the counter value is incremented. If the conversion fails, the program displays a message stating at which position in the list, the user has entered invalid nerd score.



```
# Checking if valid nerd score value is passed as string
elif type(nerdScore) == str:
    try:
        nerdScore = float(nerdScore)
        if nerdScore < 0:
            print("Nerd Score in position {0} of the student score list cannot be a negative number!".format(studentScore_list.index(str(nerdScore))), end=" ")
            print("Please enter correct details!")
        elif nerdScore >= 0:
            nerdScoreCounter += 1
    except ValueError:
        print("Nerd Score in position {0} of the student score list is invalid!".format(studentScore_list.index(str(nerdScore))), end=" ")
        print("Please enter correct details!")
```

Fig 28. Validating the nerd scores(string type only) in the list and displaying error message

- If nerd score in the list is valid, nerdScoreCounter is incremented.



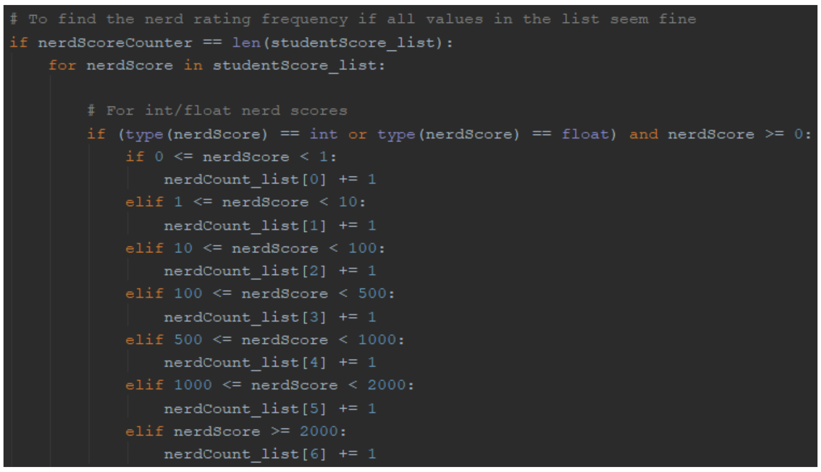
```
# If all values seem valid, update counter value
else:
    nerdScoreCounter += 1
```

Fig 29. Counter increment if nerd score is valid

## 5.2 CALCULATING THE NERD RATING FREQUENCY

As the program has validated the nerd scores in the list already, next step is to find out the nerd rating frequency. Below are few cases for how the program works:

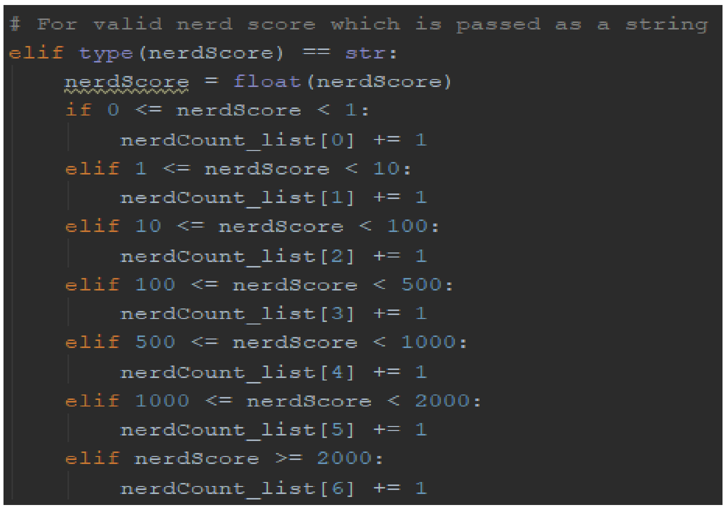
- If the nerd score in the list is of integer or float type and zero or a positive number, the program increments the nerdCount_List of a nerd class by 1 implying that the nerd score belongs to that particular nerd class. Below is the code snippet.



```python
# To find the nerd rating frequency if all values in the list seem fine
if nerdScoreCounter == len(studentScore_list):
    for nerdScore in studentScore_list:

        # For int/float nerd scores
        if (type(nerdScore) == int or type(nerdScore) == float) and nerdScore >= 0:
            if 0 <= nerdScore < 1:
                nerdCount_list[0] += 1
            elif 1 <= nerdScore < 10:
                nerdCount_list[1] += 1
            elif 10 <= nerdScore < 100:
                nerdCount_list[2] += 1
            elif 100 <= nerdScore < 500:
                nerdCount_list[3] += 1
            elif 500 <= nerdScore < 1000:
                nerdCount_list[4] += 1
            elif 1000 <= nerdScore < 2000:
                nerdCount_list[5] += 1
            elif nerdScore >= 2000:
                nerdCount_list[6] += 1
```

Fig 30. Nerd Rating Frequency for int/float type nerd scores in the list

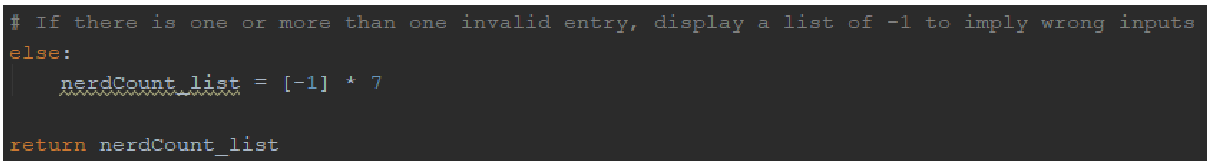
- If the nerd score in the list is of string type, the program increments the nerdCount_List of a nerd class by 1 implying that the nerd score belongs to that particular nerd class. Below is the code snippet.



```python
# For valid nerd score which is passed as a string
elif type(nerdScore) == str:
    nerdScore = float(nerdScore)
    if 0 <= nerdScore < 1:
        nerdCount_list[0] += 1
    elif 1 <= nerdScore < 10:
        nerdCount_list[1] += 1
    elif 10 <= nerdScore < 100:
        nerdCount_list[2] += 1
    elif 100 <= nerdScore < 500:
        nerdCount_list[3] += 1
    elif 500 <= nerdScore < 1000:
        nerdCount_list[4] += 1
    elif 1000 <= nerdScore < 2000:
        nerdCount_list[5] += 1
    elif nerdScore >= 2000:
        nerdCount_list[6] += 1
```

Fig 31. Nerd Rating Frequency for string type nerd scores in the list

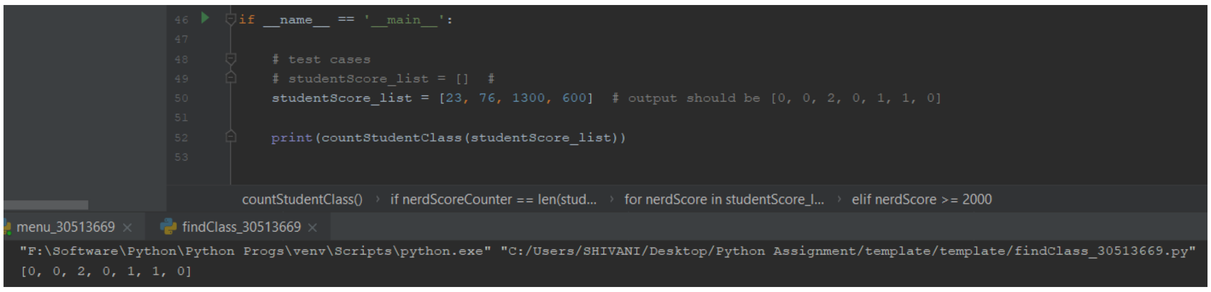
- If there is one or more than one invalid nerd scores in the list then the program returns and displays a list of seven **'-1'** to imply that the nerd scores entered by the user are invalid.



```
# If there is one or more than one invalid entry, display a list of -1 to imply wrong inputs
else:
    nerdCount_list = [-1] * 7

return nerdCount_list
```

Fig 32. Nerd Rating Frequency for string type nerd scores in the list

To test if the program is running correctly, let's try a couple of test cases. This is showed using the code snippets and screenshots of the output.



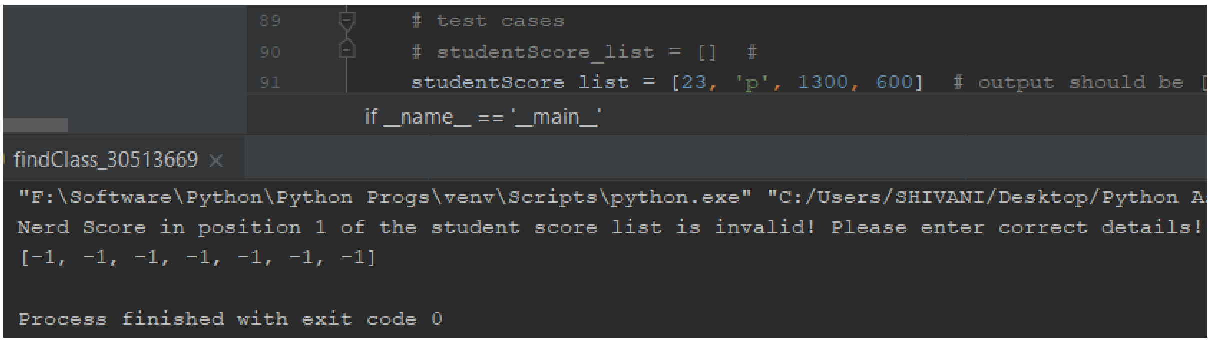
```
if __name__ == '__main__':

    # test cases
    # studentScore_list = []  #
    studentScore_list = [23, 76, 1300, 600]  # output should be [0, 0, 2, 0, 1, 1, 0]

    print(countStudentClass(studentScore_list))
```

countStudentClass() › if nerdScoreCounter == len(stud... › for nerdScore in studentScore_l... › elif nerdScore >= 2000

menu_30513669 × findClass_30513669 ×

```
"F:\Software\Python\Python Progs\venv\Scripts\python.exe" "C:/Users/SHIVANI/Desktop/Python Assignment/template/template/findClass_30513669.py"
[0, 0, 2, 0, 1, 1, 0]
```

Fig 33. Test result for input [23, 76, 1300, 600]



```
    # test cases
    # studentScore_list = []  #
    studentScore list = [23, 'p', 1300, 600]  # output should be [
```

if __name__ == '__main__'

findClass_30513669 ×

```
"F:\Software\Python\Python Progs\venv\Scripts\python.exe" "C:/Users/SHIVANI/Desktop/Python A
Nerd Score in position 1 of the student score list is invalid! Please enter correct details!
[-1, -1, -1, -1, -1, -1, -1]

Process finished with exit code 0
```

Fig 34. Test result for input [23, 'p', 1300, 600]

# 6 ASSUMPTIONS

Following assumptions have been made in the tasks.

## 6.1 TASK 1

1. The user is expected to enter numeric values only otherwise the program runs the validations and displays appropriate messages.
2. Fandom score can be a positive integer only.
3. Hobbies score can be zero or positive integer multiple of 4 only.
4. Number of sports played/ sports score can be zero or positive integer only.

## 6.2 TASK 2

1. The user is expected to give inputs in terms of variable assignment only and not via inbuilt input function which considers only string values.
2. The user can enter valid inputs in terms of integer, float or string for example, 2, 4.0, ‘3’ etc.

## 6.3 TASK 3

1. The user can enter valid inputs in terms of integer, float or string for example, 1, 50.7, ‘300.659’ etc.

# 7 BENEFITS OF THIS CODE

The following are the benefits of this code :

## 7.1 TASK 1

1. The program accepts the numbers with signs as the input for fandom, hobbies and sports score. For example, it considers +2 as a valid input for fandom/sports score. Similarly, -8 is discarded as it's a negative number.
2. The program accepts valid inputs for fandom, hobbies and sports score with initial or latter space padding. For example, <      4> is considered as a valid input for all the scores.
3. Once the nerd score is calculated, the program won't allow the user to calculate the nerd score again with the previous scores. The user has to enter new fandom, hobbies and sports score again.
4. Once the nerd class is displayed, the program won't allow the user to display the nerd class again with the previous nerd score. The user has to enter new fandom, hobbies and sports score, calculate the nerd score and then choose nerd class display option from the menu.

## 7.2 TASK 2

1. If the user enters wrong inputs, the program displays which of the inputs is invalid/wrong and prompts the user to enter valid inputs and returns the skill score as **'-1'.**
2. The return value of the function is either valid skill score or '-1'. This lets the user know if valid inputs were entered to calculate the nerd score.
3. The program accepts valid inputs in the form of string/float. It converts the string/float input into integer format, validates the value and calculates the nerd score.

## 7.3 TASK 3

1. The program accepts valid inputs in the form of integer/string/float. It converts the string input into float format, validates the value and calculates the nerd rating frequency.
2. If the user enters wrong inputs, the program displays which of the inputs is invalid/wrong and prompts the user to enter valid inputs and returns the nerdCount_List as seven **'-1'** implying that the user has entered one or more than one invalid/wrong inputs.

# 8 LIMITATIONS OF THIS CODE

In task 2 and task 3, the method returns the value but we have been asked to print appropriate display messages throughout the process. It makes sense if we pass on some return value in case of invalid inputs so that the user knows that the inputs were wrong through that returned value rather than displaying messages. This could be done out the method but as we were asked not to make changes to the template and given the space for us to write the code, this limitation couldn't be handled.

As I handled for valid inputs in the form of string/float/integer in task 2 and task 3, I didn't handle the inputs for the same in task 1. For example, a float value of 4.0 is considered as a wrong input as it is not an integer value upon string to integer conversion. It could be done by converting the string to float and checking if that converted float value is integer by using is_integer built in function.

There are many more functionalities that can be implemented in this assignment but considering what was asked, this program handles those aspects.