**Exception Handling**

**Program 1:**

*Python Program should get the Name, Age, 6 Subjects marks as an input from the user. Then generate the dictionary as below*

*Before generating a dictionary, you need to check whether the entered age value is positive or not. If it is negative, we* ***should NOT*** *add the details to the dictionary.*

*Use User-Defined Exception Handling*

**Code:**

class InvalidAgeError(Exception):

    pass

details = {}

try:

    name = input("Enter the Name: ")

    age = int(input("Enter the Age: "))

    if age <= 0:

        raise InvalidAgeError("Age must be positive.")

    details['Name'] = name

    details['Age'] = age

    marks = []

    print("Enter marks for 6 subjects:")

    for i in range(6):

        mark = int(input(f"Enter Mark {i+1}: "))

        if mark<=0 or mark>=100:

            print("Invalid Mark!")

        else:

            marks.append(mark)

    details['Marks'] = marks

    total = sum(marks)

    average = total / len(marks)

    details['Total'] = total

    details['Average'] = average

    print("\n--- Student Details ---")

    print(details)

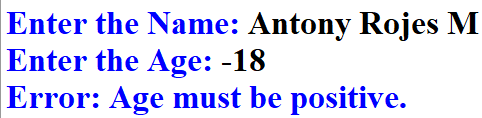
except InvalidAgeError as e:

    print("Error:", e)

except ValueError:

    print("Please enter only numeric values for age and marks.")

**Output:**





**Program 2:**

*Python Program should read the Name, Roll Number and the 6 Subject marks.*

*While entering the marks if we have entering the marks greater than 100 or less than 0 then it should throw an error (****User generated Exception****).*

*For calculating the total,average, minimum and maximum use function*

**Code:**

class InvalidMarkError(Exception):

    pass

def function\_marks(marks):

    total = sum(marks)

    average = total / len(marks)

    minimum = min(marks)

    maximum = max(marks)

    return total, average, minimum, maximum

details = {}

try:

    name = input("Enter the Name: ")

    roll = input("Enter the Roll Number: ")

    details['Name'] = name

    details['Roll Number'] = roll

    marks = []

    print("Enter marks for 6 subjects:")

    for i in range(6):

        n = int(input(f"Enter Mark {i + 1}: "))

        if n < 0:

            raise InvalidMarkError(f"Mark {n} is less than 0")

        elif n > 100:

            raise InvalidMarkError(f"Mark {n} is greater than 100")

        marks.append(n)

    details['Marks'] = marks

    total, average, minimum, maximum = function\_marks(marks)

    details['Total'] = total

    details['Average'] = average

    details['Minimum'] = minimum

    details['Maximum'] = maximum

    print("\n--- Student Mark Details ---")

    print("Name:", name)

    print("Roll Number:", roll)

    print("Marks:", marks)

    print("Total:", total)

    print("Average:", average)

    print("Minimum:", minimum)

    print("Maximum:", maximum)

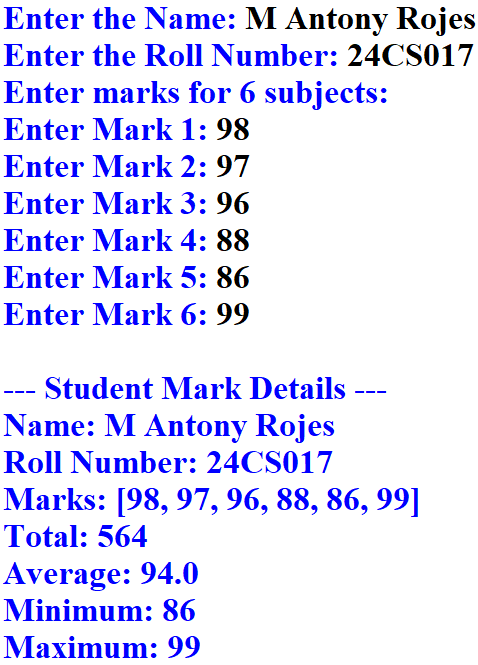
except InvalidMarkError as e:

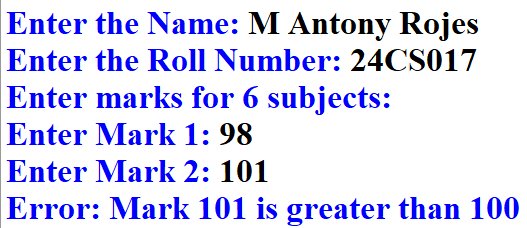
    print("Error:", e)

except ValueError:

    print("Please enter valid numeric input only.")

**Output:**





*Now Handle the above using* ***Try and Execpt*** *Block.*

*If all the marks are entered correctly then it should display the Total, Average and his minimum and maximum mark.*

*For calculating the total,average, minimum and maximum use function*

***If the entered mark is invalid then the function should "0" if it is valid then function should return total,average,minimum and maximum***

*Use User-Defined Exception*

*In addition to above add one condition when displaying the total,average,minimum and maximum.*

*if he got marks less that 50 then just display "have failed", else display the all the detials.*

*Now change the code accordingly and display the output as below*

**Updated Code 2:**

class InvalidMarkError(Exception):

    pass

def function\_marks(marks):

    try:

        for i in marks:

            if i < 0:

                raise InvalidMarkError(f"Mark {i} is less than 0")

            elif i > 100:

                raise InvalidMarkError(f"Mark {i} is more than 100")

        total = sum(marks)

        average = total / len(marks)

        minimum = min(marks)

        maximum = max(marks)

        return total, average, minimum, maximum

    except InvalidMarkError as e:

        print("Error:", e)

        return "0", "0", "0", "0"

details = {}

name = input("Enter the Name: ")

details['Name'] = name

try:

    subjects = int(input("Enter the Number of Subjects: "))

    marks = []

    for y in range(subjects):

        mark = int(input(f"Enter Mark {y + 1}: "))

        marks.append(mark)

    details['Marks'] = marks

    total, average, minimum, maximum = function\_marks(marks)

    if total == "0":

        print("Invalid mark entered. Cannot compute results.")

    else:

        if any(mark < 50 for mark in marks):

            print("Have failed")

        else:

            details['Total'] = total

            details['Average'] = average

            details['Minimum'] = minimum

            details['Maximum'] = maximum

            print("Total:", total)

            print("Average:", average)

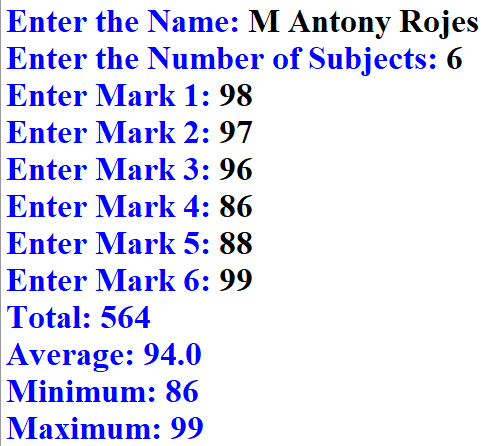
            print("Minimum:", minimum)

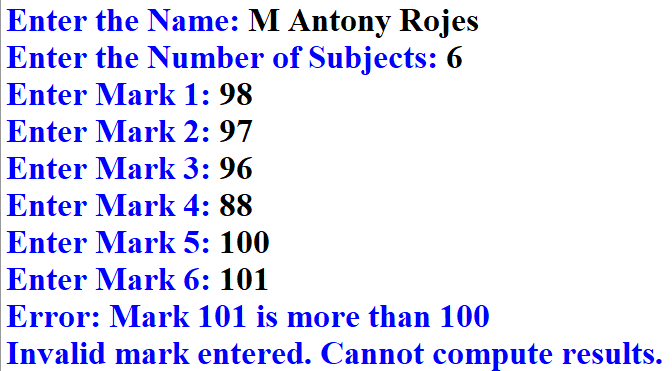
            print("Maximum:", maximum)

except ValueError:

    print("Please enter valid numeric values.")

**Output:**





**Files**

**Program 1:**

*Write a function in python to count the number of lines from a text file "proverb.txt" which is not starting with an alphabet "D".*

**Example:**

*If the file "proverb.txt" contains the following lines:*

*Many hands make light work*

*Honesty is the best policy*

*Don’t bite the hand that feeds you*

*Don’t judge a book by its cover*

*Birds of a feather flock together*

*Diamonds cut diamonds*

**Code:**

def count\_non\_D\_lines():

    try:

        count = 0

        with open("proverb.txt", "r") as file:

            for line in file:

                stripped\_line = line.strip()

                if stripped\_line and not stripped\_line.startswith("D"):

                    count += 1

        print("Number of lines not starting with 'D':", count)

    except FileNotFoundError:

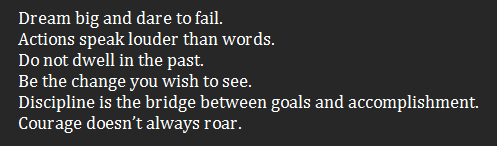
        print("File 'proverb.txt' not found.")

count\_non\_D\_lines()

**Output:**

****

**Notepad:**

****

**Program 2:**

*Python Program that display the occurance of a particular character or a string in give input file "proverb.txt"*

*If the file* ***"proverb.txt"*** *contains the following lines:*

*Many hands make light work*

*Honesty is the best policy*

*Don’t bite the hand that feeds you*

*Don’t judge a book by its cover*

*Birds of a feather flock together*

*Diamonds cut diamonds*

**Code:**

def count\_occurrences\_in\_file():

    try:

        search = input("Enter the character or string to search: ")

        with open("proverb.txt", "r") as file:

            content = file.read()

            count = content.count(search)

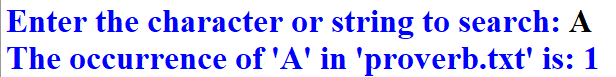
            print(f"The occurrence of '{search}' in 'proverb.txt' is: {count}")

    except FileNotFoundError:

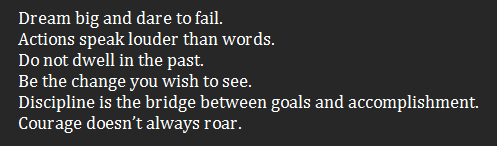
        print("File 'proverb.txt' not found.")

count\_occurrences\_in\_file()

**Output:**



**Notepad:**

****

**Program 3:**

*Write a python code to append the following proverb to the* ***"proverb.txt"*** *file.*

*"Diligence is the mother of good fortune"*

***After updating the file content must be, read the file content and display it.***

**Code:**

def append\_and\_display():

    proverb = "Diligence is the mother of good fortune"

    try:

        with open("proverb.txt", "a") as file:

            file.write("\n" + proverb)

            print("\n Proverb successfully appended.\n")

        with open("proverb.txt", "r") as file:

            print("Updated contents of 'proverb.txt':\n")

            content = file.read()

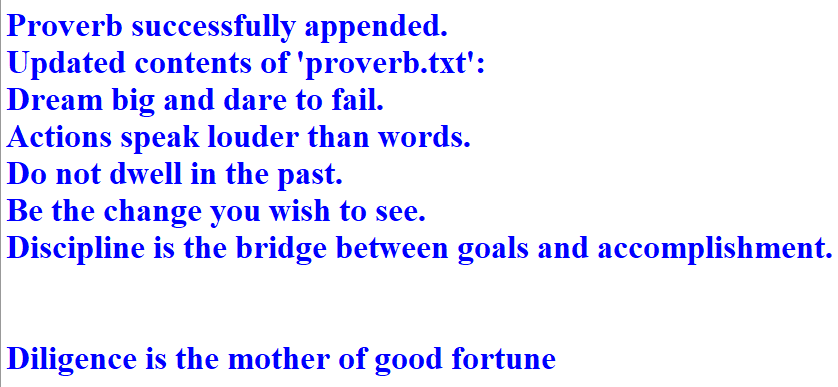
            print(content)

    except FileNotFoundError:

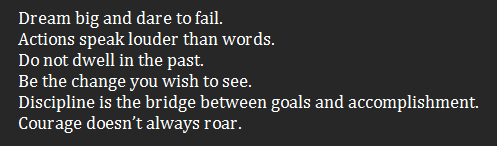
        print("File 'proverb.txt' not found.")

append\_and\_display()

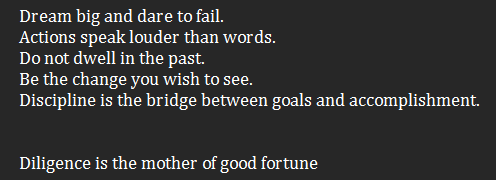
**Output:**

****

**Notepad Before:**

****

**Notepad After:**

****

**Program 4:**

*Now convert all the character in the file "proverb.txt" to uppercase and write to a new file "u\_proverb.txt" and display the new contents files also.*

**Code:**

def convert\_to\_uppercase():

    try:

        with open("proverb.txt", "r") as infile:

            content = infile.read()

        upper\_content = content.upper()

        with open("u\_proverb.txt", "w") as outfile:

            outfile.write(upper\_content)

        print("Contents converted to uppercase and written to 'u\_proverb.txt'.\n")

        print("Contents of 'u\_proverb.txt':\n")

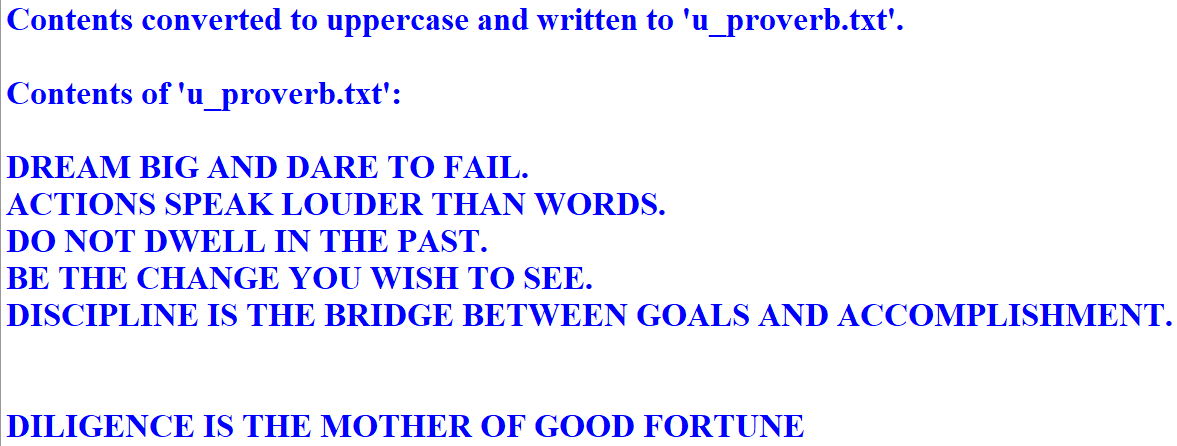
        print(upper\_content)

    except FileNotFoundError:

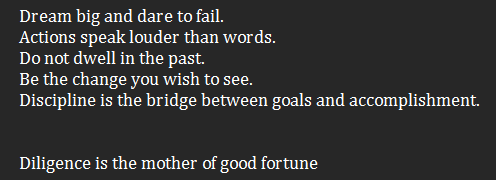
        print("File 'proverb.txt' not found.")

convert\_to\_uppercase()

**Output:**



**Notepad:**

****