Harsh Kasliwal

2003085

**EXPERIMENT 6**

# Q1 Python program

* To create class ***Student with rno, name, marks*** as instance variable and ***constructor*** to initialize these instance variables.
* Instantiate ***n*** instances of classes and save details in list.
* Create an **user defined exception** class **Fail** to raise an exception if marks is less than 40.

Display details of students and also raise exceptions for marks less than 40.

## Exception Handling in Python

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

* When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

Python Logical Errors (Exceptions)

* Errors that occur at runtime (after passing the syntax test) are called exceptions or logical errors.
* For instance, they occur when we try to open a file(for reading) that does not exist (FileNotFoundError), try to divide a number by zero (ZeroDivisionError), or try to import a module that does not exist (ImportError).
* Whenever these types of runtime errors occur, Python creates an exception object. If not handled properly, it prints a traceback to that error along with some details about why that error occurred.

If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a try: block. After the try: block, include an except: statement, followed by a block of code which handles the problem as elegantly as possible.

Syntax

Here is simple syntax of try....except...else blocks − try:

You do your operations here;

......................

except ExceptionI:

If there is ExceptionI, then execute this block. except ExceptionII:

If there is ExceptionII, then execute this block.

......................

else:

If there is no exception then execute this block.

Here are few important points about the above-mentioned syntax −

* A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.
* You can also provide a generic except clause, which handles any exception.
* After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception.
* The else-block is a good place for code that does not need the try: block's protection.

The except Clause with No Exceptions: try:

You do your operations here;

......................

except:

If there is any exception, then execute this block.

......................

else:

If there is no exception then execute this block.

This kind of a try-except statement catches all the exceptions that occur. Using this kind of try-except statement is not considered a good programming practice though, because it catches all exceptions but does not make the programmer identify the root cause of the problem that may occur.

The except Clause with Multiple Exceptions:

You can also use the same except statement to handle multiple exceptions as follows − try:

You do your operations here;

......................

except(Exception1[, Exception2[,...ExceptionN]]]):

If there is any exception from the given exception list, then execute this block.

......................

else:

If there is no exception then execute this block.

The try-finally Clause:

You can use a finally: block along with a try: block. The finally block is a place to put any code that must execute, whether the try-block raised an exception or not. The syntax of the try-finally statement is this −

try:

You do your operations here;

......................

Due to any exception, this may be skipped. finally:

This would always be executed.

......................

You cannot use else clause as well along with a finally clause.

## User-defined exceptions

Creating User-defined Exception:

Programmers may name their own exceptions by creating a new exception class. Exceptions need to be derived from the Exception class, either directly or indirectly. Although not mandatory, most of the exceptions are named as names that end in “Error” similar to naming of the standard exceptions in python. For example:

# A python program to create user-defined exception

*# class MyError is derived from super class Exception*

**class** MyError(Exception):

*# Constructor or Initializer*

**def** init (self, value): self**.**value **=** value

*# str is to print() the value*

**def** str (self): **return**(repr(self**.**value))

**try**:

**raise**(MyError(3**\***2))

*# Value of Exception is stored in error*

**except** MyError **as** error:

print('A New Exception occured: ',error**.**value) OUTPUT:

('A New Exception occured: ', 6) Deriving Error from Super Class Exception

Super class Exceptions are created when a module needs to handle several distinct errors.

One of the common way of doing this is to create a base class for exceptions defined by that module. Further, various subclasses are defined to create specific exception classes for different error conditions.

*class Error is derived from super class Exception*

**class** Error(Exception):

*# Error is derived class for Exception, but # Base class for exceptions in this module* **pass**

**class** TransitionError(Error):

*# Raised when an operation attempts a state # transition that's not allowed.*

**def** init (self, prev, nex, msg): self**.**prev **=** prev

self**.**next **=** nex

**try**:

*# Error message thrown is saved in msg*

self**.**msg **=** msg

**raise**(TransitionError(2,3**\***2,"Not Allowed"))

*# Value of Exception is stored in error*

**except** TransitionError **as** error:

print('Exception occured:',error**.**msg)

OUTPUT

('Exception occured: ', 'Not Allowed')

**CODE**

class Student:

stu\_count=0

def \_\_init\_\_(self):

self.r\_no = input("Enter the Student roll number : ")

self.name=input("Enter the name : ")

self.marks=input("Enter the marks : ")

def Fail(Exception):

'''Student scored less marks Faill!!!!'''

@classmethod

def set\_stu\_count(self):

while 1:

try:

self.stu\_count=int(input('Enter the total number of Students : '))

return self.stu\_count

except:

print("Invald input!!")

continue

n=Student.set\_stu\_count()

a=[]

print("-----------Enter the Details of Students ----------")

for i in range(0,n):

stu=Student()

a.append(stu)

print("-----------Details of Students are----------")

print("Roll Number\tName\tMarks")

for i in range(0,n):

print("{}\t\t{}\t{}".format(a[i].r\_no,a[i].name,a[i].marks))

print("------------Details after checking grades--------------")

for i in range(0,n):

try:

if(int(a[i].marks)<40):

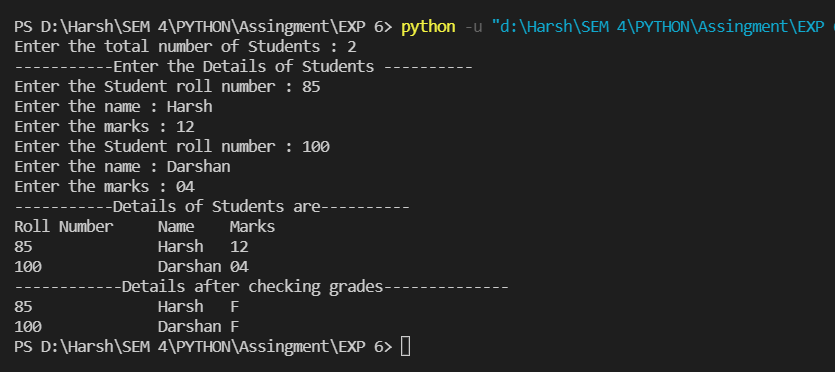
raise Fail

else:

print("{}\t\t{}\t{}".format(a[i].r\_no,a[i].name,a[i].marks))

except:

print("{}\t\t{}\tF".format(a[i].r\_no,a[i].name))



# Q2 Python program to

* create a file
* count no. of lines, words and characters in a file.
* write content of a file in a new file and read that new file.

## FILE HANDLING IN PYTHON:

1. **open():** This function accepts two arguments, ’file name’ and ‘access mode’ in which the file is accessed. This file returns a file object which can be used to perform various operations like reading,writing,etc.

ACCESS MODE:

|  |  |
| --- | --- |
| r | Read-only mode |
| rb | Read-only in binary mode |
| r+ | Read and write both |
| rb+ | Read and write both in binary mode |

|  |  |
| --- | --- |
| w | Write only mode |
| wb | Write only in binary mode |
| w+ | Write and read both |
| wb+ | Write and read both in  binary mode |
| a | Append mode |
| ab | Append in binary format |
| a+ | Append and read both |
| ab+ | Append and read both in  binary mode |

1. **write():** Inserts the string in a single line in the text file
2. **writelines():** For a list of string elements, each string is inserted in the text file. Used to insert multiple strings at a single time.
3. **append():** This function acts similar to the write() function, however, instead of overwriting the file,the append() function appends contents to the existing file. Another alternative for append can be to open file in append mode.
4. **read():** returns the read bytes in the form of a string. Read n bytes, if not specified, reads the entire file.
5. **readline():** Reads a line of the file and returns in form of string.
6. **close():** The close function clears the memory buffer and closes the file. The closed file cannot be read or written any more.

**CODE:**

def print\_details\_of\_file(file\_content):

'''function to find details of the file'''

line\_count, word\_count, char\_count = 0, 0, 0

# To find no of lines

line\_list = file\_content.split("\n")

line\_count = len(line\_list)

# To find no of words

for line in line\_list:

word\_list = line.split()

word\_count += len(word\_list)

# To find number of chars

char\_count = len(file\_content)

print("The file content:\n" + ("-" \* 30) + f"\n{file\_content}\n" + ("-" \* 30))

print(f"Line Count :\t {line\_count}")

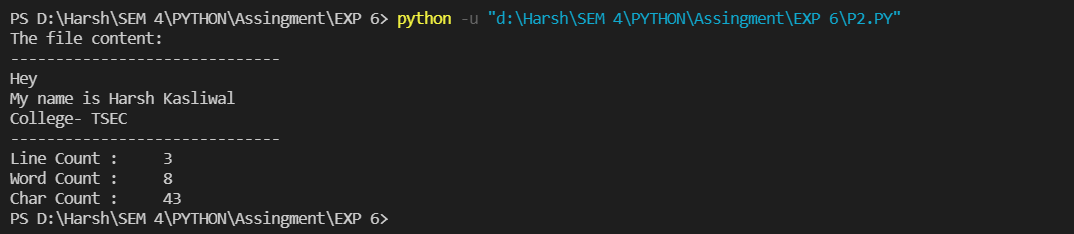
print(f"Word Count :\t {word\_count}")

print(f"Char Count :\t {char\_count}")

with open("useless.txt", 'r') as f:

file\_content = f.read()

print\_details\_of\_file(file\_content)



# Q3. Python program to

* create a ***class Customer*** with ***id, name, mobile number*** as instance variable and

***constructor*** to initialize these instance variables.

* Instantiate ***n*** instances of classes

Save details of all customer in a ***file*** and read back from that file.

**CODE:**

class Customer:

def \_\_init\_\_(self, ID, name, mobile\_num):

self.ID = ID

self.name = name

self.mobile\_num = mobile\_num

def add\_content\_to\_file(self):

with open("Details.txt", 'a') as f:

f.write(f"{self.ID}, {self.name}, {self.mobile\_num}\n")

@classmethod

def print\_content\_of\_file(cls):

with open("Details.txt", 'r') as f:

file\_content = f.read()

print("\nThe file content:\n" + ("-" \* 30) + f"\n{file\_content}" +("-" \* 30))

n = int(input('ENTER NUMBER OF CUSTOMERS: '))

for i in range(n) :

customer\_name = input('\nEnter customer name: ')

customer\_number = input('Enter customer number: ')

# Creating n Customer Instances

customer\_instance = Customer(i + 1, customer\_name, customer\_number)

# Appending content to file

customer\_instance.add\_content\_to\_file()

# Print final content of the file

Customer.print\_content\_of\_file()

