Harsh Kasliwal
2003085

EXPERIMENT 6

Q1 Python program

- To create class ***Student with rno, name, marks*** as instance variable and ***constructor*** toinitialize 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 than40.

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 thatrepresents an error.

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

Python Logical Errors (Exceptions)

- Errors that occur at runtime (after passing the syntax test) are called exceptions orlogical 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 programby 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 thetry 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-blockexecutes 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 oftry-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 namingof 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 thatmodule. 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

        # Error message thrown is saved in msg
        self.msg = msg
try:

    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))
```

```
PS D:\Harsh\SEM 4\PYTHON\Assingment\EXP 6> python -u "d:\Harsh\SEM 4\PYTHON\Assingment\EXP
Enter the total number of Students : 2
----------Enter the Details of Students ----------
Enter the Student roll number : 85
Enter the name : Harsh
Enter the marks : 12
Enter the Student roll number : 100
Enter the name : Darshan
Enter the marks : 04
----------Details of Students are----------
Roll Number     Name    Marks
85              Harsh   12
100             Darshan 04
------------Details after checking grades--------------
85              Harsh   F
100             Darshan F
PS D:\Harsh\SEM 4\PYTHON\Assingment\EXP 6>
```

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 whichthe 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

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

```
PS D:\Harsh\SEM 4\PYTHON\Assingment\EXP 6> python -u "d:\Harsh\SEM 4\PYTHON\Assingment\EXP 6\P2.PY"
The file content:
------------------------------
Hey
My name is Harsh Kasliwal
College- TSEC
------------------------------
Line Count :     3
Word Count :     8
Char Count :     43
PS D:\Harsh\SEM 4\PYTHON\Assingment\EXP 6>
```

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()
```

```
PS D:\Harsh\SEM 4\PYTHON\Assingment\EXP 6> python -u "d:\Harsh\SEM 4\PYTHON\Assingment\EXP 6\P3.PY"
ENTER NUMBER OF CUSTOMERS: 2

Enter customer name: Harsh
Enter customer number: 8698

Enter customer name: Rohit
Enter customer number: 0045

The file content:
------------------------------
1, Harsh, 8698
2, Rohit, 0045
------------------------------
PS D:\Harsh\SEM 4\PYTHON\Assingment\EXP 6>
```