**Experiment 6.1**

**Aim :**

Write a 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 classFail to raise an exception if marks is less than 40.
* Display details of students and also raise exceptions for marks less than 40.

**Theory:**

What is Exception?

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.

**Try Except in Python:**

Try and Except statement is used to handle these errors within our code in Python. The try block is used to check some code for errors i.e the code inside the try block will execute when there is no error in the program. Whereas the code inside the except block will execute whenever the program encounters some error in the preceding try block.

* First, the try clause is executed i.e. the code between try and except clause.
* If there is no exception, then only the try clause will run, except the clause is finished.
* If any exception occurs, the try clause will be skipped and except clause will run.
* If any exception occurs, but the except clause within the code doesn’t handle it, it is passed on to the outer try statements. If the exception is left unhandled, then the execution stops.
* A try statement can have more than one except clause

**Else Clause:**

In python, you can also use the else clause on the try-except block which must be present after all the except clauses. The code enters the else block only if the try clause does not raise an exception.

**Syntax:**

try:

# Some Code

except:

# Executed if error in the

# try block

else:

# execute if no exception

**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.

**Syntax:**

try:

You do your operations here;

Due to any exception, this may be skipped.

finally:

This would always be executed.

**User-Defined Exceptions:**

Python also allows you to create your own exceptions by deriving classes from the standard built-in exceptions.

class Fail(Exception):

    def \_\_init\_\_(self):

        print("FAIL")

**………………………………………………………**

try:

        if (student.marks<40):

            raise Fail

        else:

*if no exception*

    except:

        continue

**Code:**

*class* Student:

*def* \_\_init\_\_(*self*, *rollno*, *name*, *marks*):

*self*.rollno= rollno

*self*.name = name

*self*.marks = marks

*class* Fail(*Exception*):

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

        print("FAIL")

n = *int*(input("Enter number of Students:"))

students = []

for i in range(0,n):

    rno = *int*(input("Enter roll no.: "))

    name = input("Enter name: ")

    marks = *int*(input("Enter marks: "))

    student = Student(rno,name,marks)

    students.append(student)

print("")

print("Roll No.\tName\tResult")

for student in students:

    try:

        if (student.marks<40):

            print(student.rollno,"\t",student.name,*end*="\t")

            raise Fail

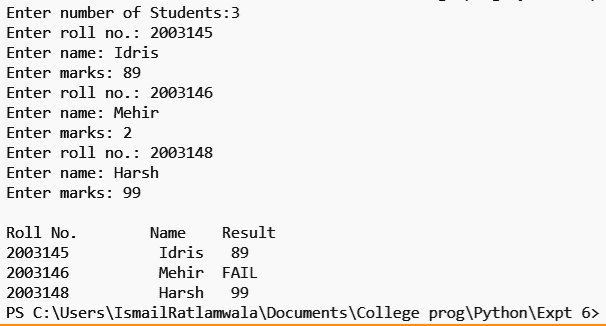
        else:

            print(student.rollno,"\t",student.name,"\t",student.marks)

    except:

        continue

**Output:**



**Experiment 6.2**

**Theory:**

**Creating an empty file**

File handling can also be used for creating a file. Even the file with different extension like .pdf, .txt, .jpeg can be created using file handling in Python. To create a file, the file must be open for writing. To open a file for writing access mode of file must be w, a, w+, a+. Access modes govern the type of operations possible in the opened file. It refers to how the file will be used once it’s opened. Below is the list of access modes for creating an empty file.

**Write Only (‘w’):** Open the file for writing. For an existing file, the data is truncated and over-written.

**Write and Read (‘w+’)**: Open the file for reading and writing. For an existing file, data is truncated and over-written.

**Append Only (‘a’)**: Open the file for writing. The data being written will be inserted at the end, after the existing data.

**Append and Read (‘a+’):** Open the file for reading and writing. The data being written will be inserted at the end, after the existing data.

**Function used:**

1. **open():** The open() function opens a file, and returns it as a file object.

Syntax:

open(file, mode)

ex.open(“a.txt”,”x”) refers to creating a file named a.txt, it returns an error if the file already exists.

1. **close():** Python file method close() closes the opened file. A closed file cannot be read or written anymore. Any operation, which requires that the file be opened will raise a ValueError after the file has been closed . Calling close() more than once is allowed.
2. **read():** By default the read() method returns the whole text, but you can also specify how many characters you want to return.

**Code:**

name = input("Enter the name of text file to be created : ")

file = open(f"{name}.txt", "x")

print(f"{name}.txt is created")

file = open("test.txt", "r")

noOfLines = 0

noOfWords = 0

noOfCharacters = 0

sentences = []

for line in file:

sentences.append(line)

line = line.strip("\n")

words = line.split()

noOfLines += 1

noOfWords += len(words)

noOfCharacters += len(line)

file.close()

print("Lines:", noOfLines, "\nWords:", noOfWords, "\nCharacters:", noOfCharacters, "\n")

new = open('new.txt', 'w')

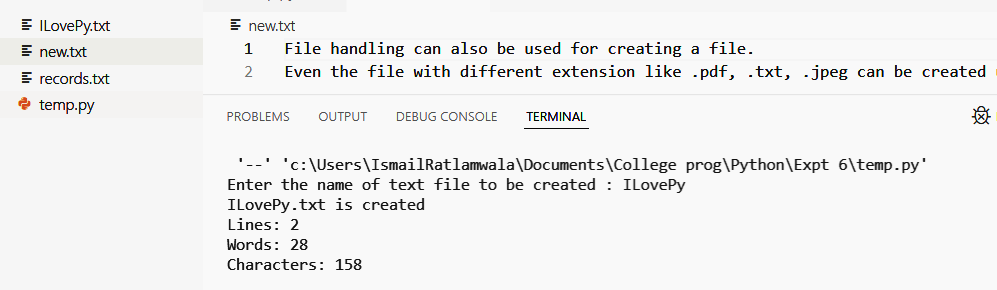
new.writelines(sentences)

new.close()

new = open('new.txt', 'r')

print(new.read())

**Output:**



**Experiment No. 6.3**

**Theory:**

**Pickle Module:**

Pickle is used for serializing and de-serializing Python object structures, also called marshalling or flattening. Serialization refers to the process of converting an object in memory to a byte stream that can be stored on disk or sent over a network. Later on, character stream can then be retrieved and de-serialized back to a Python object.

The pickle module needs to be imported for pickling and unpickling datta. Any object(list, dict, etc) in Python can be pickled so that it can be saved on disk.

**‘Pickling’ into a file**

Since a file consists of bytes of information, we can transform a Python object into a file through the pickle module. This is called pickling.

**pickle.dump()** method: converts a Python object hierarchy into a byte stream. The converted byte stream can be written to a buffer or to a disk file.

It mainly takes 3 arguments:

* Data\_to\_store ->The object to be serialized (pickled)
* Filename -> The name of the file where the data is to be stored
* Protocol\_type -> The type of the protocol version used, when None is specified it refers to the default protocol used for the Python version.

**Syntax:**

import pickle

pickle.dump(data\_to\_store, filename, protocol\_type)

**‘Unpickling’ from a file**

This is just the opposite of pickling, wherein the object is retrieved from the file. The file, which contains the serialized information of the object as a byte sequence, is deserialized into the Python object itself and the original information can be retrieved back.

**pickle.load()** method: This method converts the byte stream of a pickle Python object back to a Python object itself.

**Syntax:**

New\_object = pickle.load(filename)

#new\_object -> The object into the method stores the information into

#filename -> The file containing the serialized information

**Aim:**

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:**

import pickle

class Customer:

    def \_\_init\_\_(self, id, name, mob):

        self.id = id

        self.name = name

        self.mob = mob

customer\_list = []

def storeData(customer\_list):

    customer\_file = open('Customer\_Details' , 'ab')

    pickle.dump(customer\_list , customer\_file)

    customer\_file.close()

n = int(input("Enter the number of customers: "))

for i in range(n):

    id = int(input("\nEnter customer id: "))

    n = input("Enter customer name: ")

    m  = int(input("Enter mobile number: "))

    c = Customer(id , n , m)

    customer\_list.append(c)

storeData(customer\_list)

def loadData():

    customer\_file = open('Customer\_Details' , 'rb')

    details = pickle.load(customer\_file)

    customer\_file.close()

    return details

details = loadData()

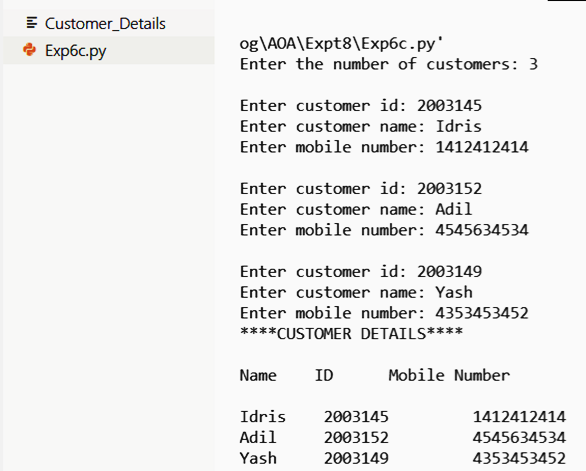
print("\*\*\*\*CUSTOMER DETAILS\*\*\*\*")

print("\nName\tID\tMobile Number\n")

for d in details:

    print(d.name, "\t" , d.id , "\t" , d.mob)

**Output:**



**Experiment 7**

Q1) WAP a program to demonstrate the use of packages

**Theory:**

* A Python module may contain several classes, functions, variables, etc. whereas a Python package can contains several module. In simpler terms a package is folder that contains various modules as files.
* Creating Package
* Let’s create a package named mypckg that will contain two modules mod1 and mod2. To create this module follow the below steps –
  + Create a folder named mypckg.
  + Inside this folder create an empty Python file i.e. \_\_init\_\_.py
  + Then create two modules mod1 and mod2 in this folder.
* \_\_init\_\_.py helps the Python interpreter to recognise the folder as package. It also specifies the resources to be imported from the modules. If the \_\_init\_\_.py is empty this means that all the functions of the modules will be imported. We can also specify the functions from each module to be made available.
* For importing, syntax is:

from packageName import file1Name, file2Name, …

**Code:**

from PyPackages.Employee import profile, qualification

from PyPackages.Account import salary

profile.ask()

qualification.ask()

salary.ask()

def ask():

basic = int(input("\nEnter basic salary:\n"))

hra = int(input("\nEnter HRA:\n"))

pf = int(input("\nEnter pf:\n"))

print("\nYour net salary is: ", basic + hra - pf)

def ask():

name = input("Enter name:\n")

age = int(input("\nEnter age:\n"))

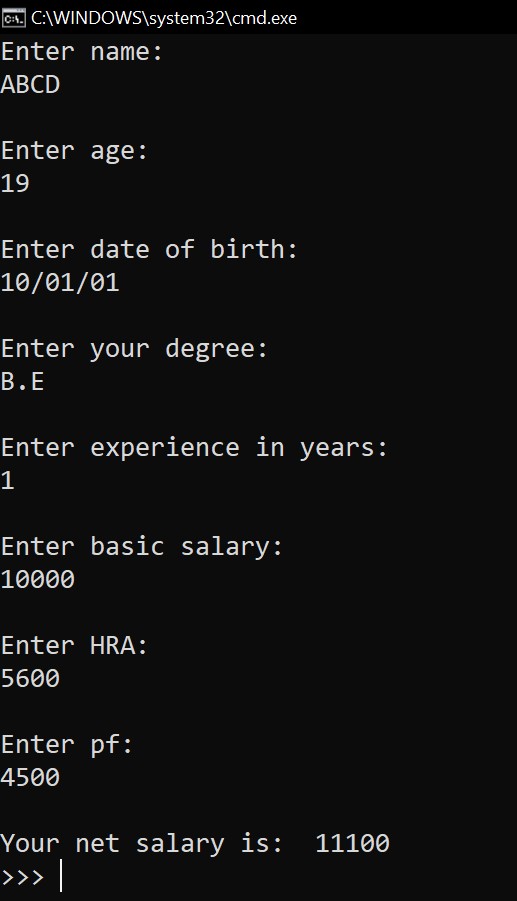
dob = input("\nEnter date of birth:\n")

def ask():

degree = input("\nEnter your degree:\n")

xp = int(input("\nEnter experience in years:\n"))

**Output:**



**Experiment 7**

**OS Module**

The OS module in Python provides functions for interacting with the operating system. OS comes under Python’s standard utility modules. This module provides a portable way of using operating system-dependent functionality. The \*os\* and \*os.path\* modules include many functions to interact with the file system.

## Handling the Current Working Directory

Consider **Current Working Directory(CWD)** as a folder, where the Python is operating. Whenever the files are called only by their name, Python assumes that it starts in the CWD which means that name-only reference will be successful only if the file is in the Python’s CWD.  
**Note:** The folder where the Python script is running is known as the Current Directory. This is not the path where the Python script is located.  
[Getting the Current working directory](https://www.geeksforgeeks.org/get-directory-of-current-python-script/)  
To get the location of the current working directory [os.getcwd()](https://www.geeksforgeeks.org/python-os-getcwd-method/) is used.

**Changing the Current Working Directory**

To change the current working directory(CWD) [os.chdir()](https://www.geeksforgeeks.org/python-os-chdir-method/) method is used. This method changes the CWD to a specified path. It only takes a single argument as a new directory path.

**Note:** The current working directory is the folder in which the Python script is operating.

## Creating a Directory

There are different methods available in the OS module for creating a directory. These are –

1. os.mkdir()
2. os.makedirs()

### **Using os.mkdir()**

os.mkdir() method in Python is used to create a directory named path with the specified numeric mode. This method raises FileExistsError if the directory to be created already exists.

### **Using os.makedirs()**

os.makedirs() method in Python is used to create a directory recursively. That means while making leaf directory if any intermediate-level directory is missing, os.makedirs() method will create them all.

## Listing out Files and Directories with Python

[os.listdir()](https://www.geeksforgeeks.org/python-os-listdir-method/) method in Python is used to get the list of all files and directories in the specified directory. If we don’t specify any directory, then the list of files and directories in the current working directory will be returned.

## Deleting Directory or Files using Python

OS module proves different methods for removing directories and files in Python. These are –

1. Using os.remove()
2. Using os.rmdir()

### **Using os.remove()**

os.remove() method in Python is used to remove or delete a file path. This method can not remove or delete a directory. If the specified path is a directory then OSError will be raised by the method.

### **Using os.rmdir()**

os.rmdir() method in Python is used to remove or delete an empty directory. OSError will be raised if the specified path is not an empty directory.

## Commonly Used Functions:

1. **os.name:** This function gives the name of the operating system dependent module imported. The following names have currently been registered: ‘posix’, ‘nt’, ‘os2’, ‘ce’, ‘java’ and ‘riscos’.
2. **os.error:** All functions in this module raise OSError in the case of invalid or inaccessible file names and paths, or other arguments that have the correct type, but are not accepted by the operating system. os.error is an alias for built-in OSError exception.
3. **os.popen():**This method opens a pipe to or from command. The return value can be read or written depending on whether the mode is ‘r’ or ‘w’.
4. **os.close():**Close file descriptor fd. A file opened using open(), can be closed by close()only. But file opened through os.popen(), can be closed with close() or os.close(). If we try closing a file opened with open(), using os.close(), Python would throw TypeError.
5. **os.rename():**A file old.txt can be renamed to new.txt, using the function os.rename(). The name of the file changes only if, the file exists and the user has sufficient privilege permission to change the file.
6. **os.remove():**Using the Os module we can remove a file in our system using the remove() method. To remove a file we need to pass the name of the file as a parameter.
7. **os.path.exists():**This method will check whether a file exists or not by passing the name of the file as a parameter. OS module has a sub-module named PATH by using which we can perform many more functions.
8. **os.path.getsize():**In this method, python will give us the size of the file in bytes. To use this method we need to pass the name of the file as a parameter.

**Code :**

import os

cwd = os.getcwd()

print("Current working directory:", cwd)

os.mkdir("Idris")

print(os.listdir())

os.rmdir("Idris")

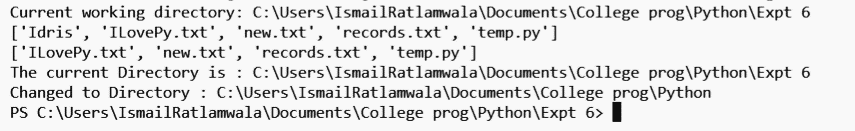
print(os.listdir())

print("The current Directory is :", os.getcwd())

os.chdir('../')

print("Changed to Directory :", os.getcwd())

**Output :**



Experiment 8.1

A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. Regular expressions are widely used in UNIX world.The Python module re provides full support for Perl-like regular expressions in Python. The re module raises the exception re.error if an error occurs while compiling or using a regular expression.

**The match Function**

This function attempts to match RE pattern to string with optional flags.

Here is the syntax for this function −

re.match(pattern, string, flags=0)

**Parameter & Description**

1.pattern

This is the regular expression to be matched.

2.string

This is the string, which would be searched to match the pattern anywhere in the string.

3.flags

You can specify different flags using bitwise OR (|). These are modifiers, which are listed in the table below.The Python module re provides full support for Perl-like regular expressions in Python. The re module raises the exception re.error if an error occurs while compiling or using a regular expression.

The re.match function returns a match object on success, None on failure. We usegroup(num) or groups() function of match object to get matched expression.

Sr.No. Match Object Method & Description

1 group(num=0)

This method returns entire match (or specific subgroup num)

2 groups()

This method returns all matching subgroups in a tuple (empty if

there weren't any)

**The search Function**

This function searches for first occurrence of RE pattern within string with optional flags.

Here is the syntax for this function −

re.search(pattern, string, flags=0)

Regular Expression Patterns

Except for control characters, (+ ? . \* ^ $ ( ) [ ] { } | \), all characters match themselves. You can escape a control character by preceding it with a backslash.

Following table lists the regular expression syntax that is available in Python

Sr.No. Pattern & Description

1

^ Matches beginning of line.

2

$ Matches end of line.

3

. Matches any single character except newline. Using m option allows it to match newline as well.

4

[...] Matches any single character in brackets.

5

[^...] Matches any single character not in brackets

6

re\* Matches 0 or more occurrences of preceding expression.

7

re+ Matches 1 or more occurrence of preceding expression.

8

re? Matches 0 or 1 occurrence of preceding expression.

9

re{ n} Matches exactly n number of occurrences of preceding expression.

10

re{ n,} Matches n or more occurrences of preceding expression.

11

re{ n, m} Matches at least n and at most m occurrences of preceding expression.

12

a| b Matches either a or b.

13

(re) Groups regular expressions and remembers matched text.

14

(?imx) Temporarily toggles on i, m, or x options within a regular expression. If in parentheses, only that area is affected.

15

(?-imx) Temporarily toggles off i, m, or x options within a regular expression. If in parentheses, only that area is affected.

16

(?: re) Groups regular expressions without remembering matched text.

17

(?imx: re) Temporarily toggles on i, m, or x options within parentheses.

18

(?-imx: re) Temporarily toggles off i, m, or x options within parentheses.

19

(?#...) Comment.

20

(?= re) Specifies position using a pattern. Doesn't have a range.

**Code :**

import re

n= int(input("Enter Total Number of People in PhoneBook: "))

f = open('PhoneBook.txt','w')

f.write("Surname \tFirstName \tPhone no\n")

for i in range(n):

    print('\nPerson '+str(i+1)+": ")

    name = input('Enter First Name: ')

    surname = input('Enter surname: ')

    phone = input('Enter Phone no.: ')

    f.write(surname+"\t\t" +name+"\t\t"+phone+"\n")

f.close()

f = open('PhoneBook.txt','r')

print('\n---Contents of PhoneBook.txt---\n')

phonelist = f.readlines()

for line in phonelist:

    print(line, end="")

pattern\_RaoJK= r'(Rao[\s]J[\w]+[\s][\d]{10})|(Rao[\s]K[\w]+[\s][\d]{10})'

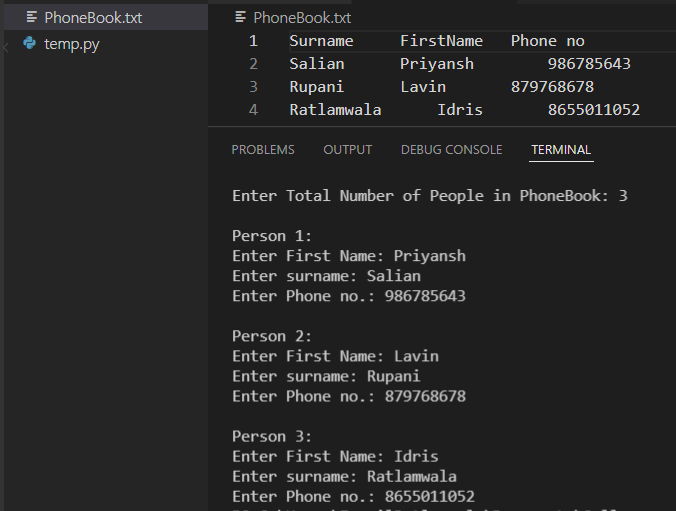
print("\nEntries with surname as 'Rao' and First Name Starting with 'J' or 'K':\n")

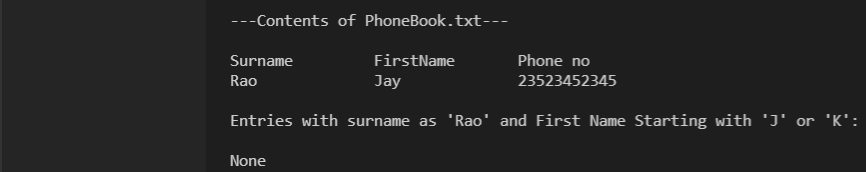
for line in phonelist:

    result = re.search(pattern\_RaoJK, line)

    print(result)

f.close()





**EXPERIMENT 8.2**

A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern.

RegEx can be used to check if a string contains the specified search pattern.

Python has a built-in package called re, which can be used to work with Regular Expressions.

Import the re module:



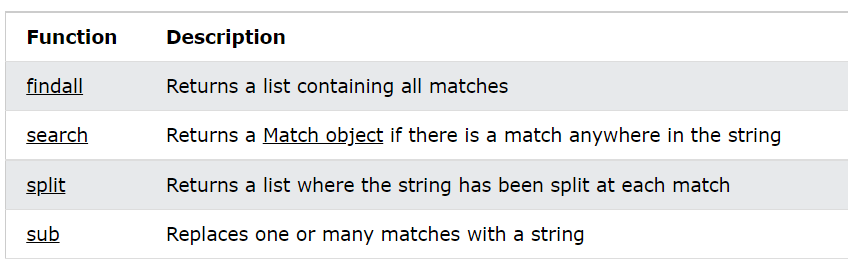
Both patterns and strings to be searched can be Unicode strings ([str](https://docs.python.org/3/library/stdtypes.html#str)) as well as 8-bit strings ([bytes](https://docs.python.org/3/library/stdtypes.html#bytes)). However, Unicode strings and 8-bit strings cannot be mixed: that is, you cannot match a Unicode string with a byte pattern or vice-versa; similarly, when asking for a substitution, the replacement string must be of the same type as both the pattern and the search string.

Regular expressions use the backslash character ('\') to indicate special forms or to allow special characters to be used without invoking their special meaning. This collides with Python’s usage of the same character for the same purpose in string literals; for example, to match a literal backslash, one might have to write '\\\\' as the pattern string, because the regular expression must be \\, and each backslash must be expressed as \\ inside a regular Python string literal. Also, please note that any invalid escape sequences in Python’s usage of the backslash in string literals now generate a [DeprecationWarning](https://docs.python.org/3/library/exceptions.html#DeprecationWarning) and in the future this will become a [SyntaxError](https://docs.python.org/3/library/exceptions.html#SyntaxError). This behaviour will happen even if it is a valid escape sequence for a regular expression.

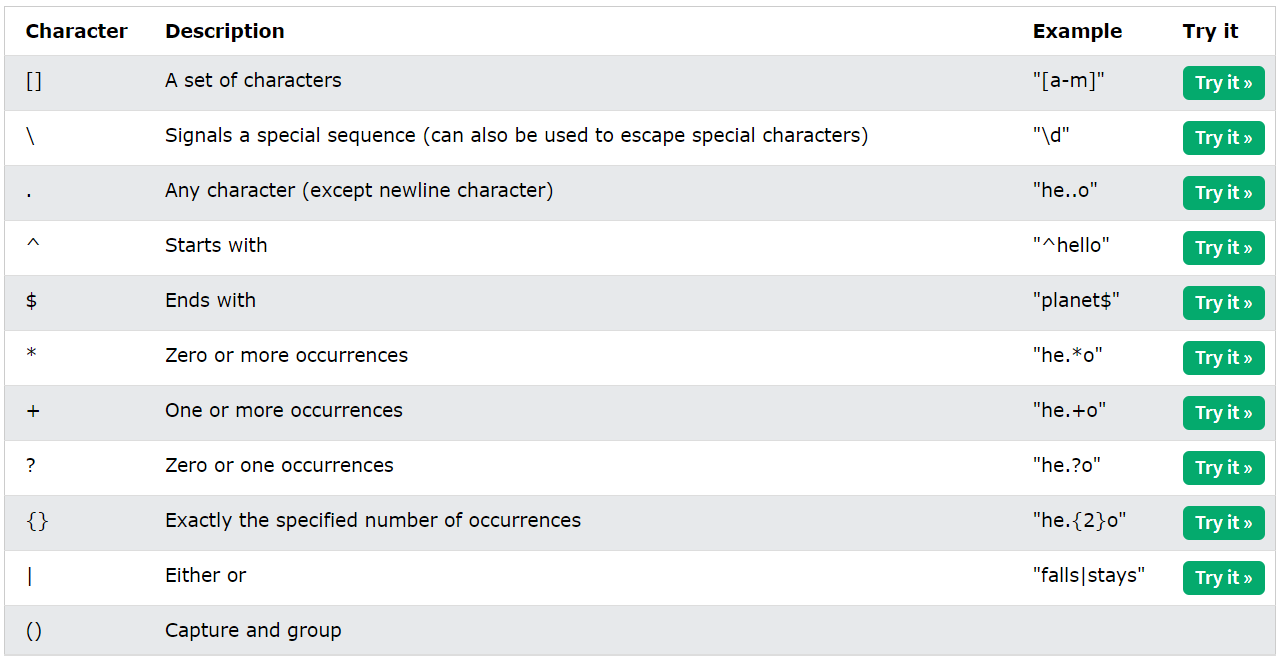
The solution is to use Python’s raw string notation for regular expression patterns; backslashes are not handled in any special way in a string literal prefixed with 'r'. So r"\n" is a two-character string containing '\' and 'n', while "\n" is a one-character string containing a newline. Usually patterns will be expressed in Python code using this raw string notation.

It is important to note that most regular expression operations are available as module-level functions and methods on [compiled regular expressions](https://docs.python.org/3/library/re.html#re-objects). The functions are shortcuts that don’t require you to compile a regex object first, but miss some fine-tuning parameters.

Functions:



Metacharacters:



**CODE:**

import re

nameofCities = input("Enter the names of cities separated with space: \n")

pattern1 = r'[\w]+ai'

result = re.findall(pattern1, nameofCities)

print("\nName of cities ending with 'ai': ")

for match in result:

    print(match, end=" ")

pattern2 = r'Mu[\w]+|Ma[\w]+'

result = re.findall(pattern2, nameofCities)

print("\n\nName of cities starting with 'Mu' or 'Ma' : ")

for match in result:

    print(match, end=' ')

pattern3 = r'\b[\w]u[\w]\*a[\w]\b'

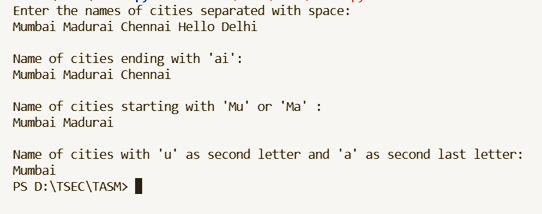
result = re.findall(pattern3, nameofCities)

print("\n\nName of cities with 'u' as second letter and 'a' as second last letter: ")

for match in result :

    print(match, end=" ")

**OUTPUT:**

****

**Experiment -09**

Python – GUI Programming (Tkinter)

Python provides various options for developing graphical user interfaces (GUIs). Most important are listed below.

* Tkinter − Tkinter is the Python interface to the Tk GUI toolkit shipped with Python. We would look this option in this chapter.
* WxPython − This is an open-source Python interface for wxWindows.
* Jpython − JPython is a Python port for Java which gives Python scripts seamless access to Java class libraries on the local machine.

# Tkinter Programming

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps − • Import the Tkinter module.

* Create the GUI application main window.
* Add one or more of the above-mentioned widgets to the GUI application.
* Enter the main event loop to take action against each event triggered by the user.

## Tkinter Widgets

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

There are currently 15 types of widgets in Tkinter, few of them are .

I. Text:

* Text widgets provide advanced capabilities that allow you to edit a multiline text and format the way it has to be displayed, such as changing its color and font.
* You can also use elegant structures like tabs and marks to locate specific sections of the text, and apply changes to those areas. Moreover, you can embed windows and images in the text because this widget was designed to handle both plain and formatted text.

## - Syntax

Here is the simple syntax to create this widget −

w = Text ( master, option, ... )

II. Button:

- The Button widget is used to add buttons in a Python application. These buttons can display text or images that convey the purpose of the buttons. You can attach a function or a method to a button which is called automatically when you click the button.

## - Syntax

Here is the simple syntax to create this widget −

w = Button ( master, option=value, ... )

III. Label:

* This widget implements a display box where you can place text or images. The text displayed by this widget can be updated at any time you want.
* It is also possible to underline part of the text (like to identify a keyboard shortcut) and span the text across multiple lines.
* Syntax

Here is the simple syntax to create this widget −

w = Label ( master, option, ... )

IV. Entry: The Entry widget is used to accept single-line text strings from a user.

* If you want to display multiple lines of text that can be edited, then you should use the Text widget.
* If you want to display one or more lines of text that cannot be modified by the user, then you should use the Label widget.

## - Syntax

Here is the simple syntax to create this widget −

w = Entry( master, option, ... )

**Program :**

from tkinter import \*

from tkinter import messagebox

def login():

    uname=rollno.get()

    pwd=password.get()

    nam=name.get()

    yea=year.get()

    bran=branch.get()

    print("Roll no\t"+"Pass\t" +"Name\t"+"Year\tBranch")

    print(uname +"\t"+ pwd +"\t"+ nam +"\t"+ yea +"\t"+ bran)

    if uname=='' or pwd==''or nam==''or yea==''or bran=='':

       messagebox.showerror('Error', 'Plese enter all details')

    else:

      if uname=="2003145" and pwd=="1234":

       messagebox.showinfo('Login success', 'You have logged in\nsuccessfully !')

      else:

       messagebox.showinfo('Login failed', 'Wrong roll no or password')

def Loginform():

    global login\_screen

    login\_screen = Tk()

    login\_screen.title("College Id Form")

    bgCol='#adfffc'

    login\_screen.geometry("400x350")

    login\_screen.configure(bg=bgCol)

    global  message

    global rollno

    global password

    global name

    global branch

    global year

    rollno = StringVar()

    password = StringVar()

    message=StringVar()

    name=StringVar()

    branch=StringVar()

    year=StringVar()

    Label(login\_screen,width="300", text="Please enter details below", bg="#56a8a5",fg="white").pack()

    yoff=20

    llogin = Label(login\_screen, text="rollno :")

    llogin.config(font=("Courier", 14),bg=bgCol)

    llogin.place(x=20,y=40+yoff)

    Entry(login\_screen, textvariable=rollno).place(

        x=150,y=42+yoff,width=200,height=25)

    lpass = Label(login\_screen, text="Password :")

    lpass.config(font=("Courier", 14),bg=bgCol)

    lpass.place(x=20,y=80+yoff)

    Entry(login\_screen, textvariable=password ,show="\*").place(

        x=150,y=82+yoff,width=200,height=25)

    lname = Label(login\_screen, text="Name :")

    lname.config(font=("Courier", 14),bg=bgCol)

    lname.place(x=20,y=120+yoff)

    Entry(login\_screen, textvariable=name ).place(

        x=150,y=122+yoff,width=200,height=25)

    lbra = Label(login\_screen, text="Branch :")

    lbra.config(font=("Courier", 14),bg=bgCol)

    lbra.place(x=20,y=160+yoff)

    Entry(login\_screen, textvariable=branch ).place(

        x=150,y=162+yoff,width=200,height=25)

    lyear = Label(login\_screen, text="Year :")

    lyear.config(font=("Courier", 14),bg=bgCol)

    lyear.place(x=20,y=200+yoff)

    Entry(login\_screen, textvariable=year ).place(

        x=150,y=202+yoff,width=200,height=25)

    but = Button(login\_screen, text="Login", width=10, height=1, bg="orange",command=login)

    but.config(font=("Calibri", 14))

    but.place(x=140,y=260+yoff)

    login\_screen.mainloop()

Loginform()